

University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

---

Theses, Dissertations, and Student Research  
from Electrical & Computer Engineering

Electrical & Computer Engineering, Department  
of

---

12-2019

## Pixel-Level Deep Multi-Dimensional Embeddings for Homogeneous Multiple Object Tracking

Mateusz Mittek

University of Nebraska-Lincoln, mmittek@gmail.com

Follow this and additional works at: <https://digitalcommons.unl.edu/elecengtheses>



Part of the [Computer Engineering Commons](#), and the [Other Electrical and Computer Engineering Commons](#)

---

Mittek, Mateusz, "Pixel-Level Deep Multi-Dimensional Embeddings for Homogeneous Multiple Object Tracking" (2019). *Theses, Dissertations, and Student Research from Electrical & Computer Engineering*. 113.

<https://digitalcommons.unl.edu/elecengtheses/113>

This Article is brought to you for free and open access by the Electrical & Computer Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Theses, Dissertations, and Student Research from Electrical & Computer Engineering by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

PIXEL-LEVEL DEEP MULTI-DIMENSIONAL EMBEDDINGS FOR  
HOMOGENEOUS MULTIPLE OBJECT TRACKING

by

Mateusz M. Mittek

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Electrical and Computer Engineering

Under the Supervision of Professor Lance C. Pérez

Lincoln, Nebraska

December, 2019

# PIXEL-LEVEL DEEP MULTI-DIMENSIONAL EMBEDDINGS FOR HOMOGENEOUS MULTIPLE OBJECT TRACKING

Mateusz M. Mittek, Ph.D.

University of Nebraska, 2019

Adviser: Lance C. Pérez

The goal of Multiple Object Tracking (MOT) is to locate multiple objects and keep track of their individual identities and trajectories given a sequence of (video) frames. A popular approach to MOT is tracking by detection consisting of two processing components: detection (identification of objects of interest in individual frames) and data association (connecting data from multiple frames). This work addresses the detection component by introducing a method based on semantic instance segmentation, i.e., assigning labels to all visible pixels such that they are unique among different instances. Modern tracking methods often built around Convolutional Neural Networks (CNNs) and additional, explicitly-defined post-processing steps.

This work introduces two detection methods that incorporate multi-dimensional embeddings. We train deep CNNs to produce easily-clusterable embeddings for semantic instance segmentation and to enable object detection through pose estimation. The use of embeddings allows the method to identify per-pixel instance membership for both tasks.

Our method specifically targets applications that require long-term tracking of homogeneous targets using a stationary camera. Furthermore, this method was developed and evaluated on a livestock tracking application which presents exceptional challenges that generalized tracking methods are not equipped to solve. This is largely because contemporary datasets for multiple object tracking lack properties that are specific to livestock environments. These include a high degree of visual similarity between targets, complex physical interactions, long-term inter-object occlusions, and a fixed-cardinality set of targets.

For the reasons stated above, our method is developed and tested with the livestock application in mind and, specifically, group-housed pigs are evaluated in this work. Our method reliably detects pigs in a group housed environment based on the publicly available dataset with 99% precision and 95% using pose estimation and achieves 80% accuracy when using semantic instance segmentation at 50% IoU threshold.

Results demonstrate our method's ability to achieve consistent identification and tracking of group-housed livestock, even in cases where the targets are occluded and despite the fact that they lack uniquely identifying features. The pixel-level embeddings used by the proposed method are thoroughly evaluated in order to demonstrate their properties and behaviors when applied to real data.

## DEDICATION

I dedicate this work to my mother who due to the political and economical scenery of Poland in the early 1980s could not pursue her own doctoral degree in animal husbandry despite her outstanding achievements in the field.

## ACKNOWLEDGMENTS

First and foremost, I would like to sincerely thank my adviser Lance C. Pérez for his support during my doctoral program. Lance, without you consistently motivating me this would never come to completion. You gave me the biggest opportunity of my life and for that I would like to thank you with all my heart.

It was a great honor to become a member of the Perceptual Systems Research Group and work alongside such individuals like Jędrzej Kowalczyk, who became one of my dearest friends and an older-brother-like figure. Thank you for all the support and belief you gave me.

This work could not be completed without the advisory role of Eric T. Psota who I would like to thank for the tremendous day-to-day patience and understanding. Eric, you were my teacher, adviser, and a friend. Thank you.

I would also like to thank Bertrand Clarke for all the suggestions and corrections he provided to make this work live up to the expected standards. I hope I did not fail you Dr. Clarke and I apologize for all the changes I was not able to include. Thank you once again. Without you I would not be able to ever accomplish this work.

I would also like to thank the members of my doctoral committee: Khalid Sayood, who always stood for the academic integrity, meritocracy, and transparency, and Ashok Samal who represented the department of Computer Science and contributed significantly to the ability of accomplishing this work.

I would like to thank my lab mates: Jay Carlson, who was always challenging my biases and helped me become a better person, and Yanfeng Liu, who inspired me with his *positive* demeanor in the pursuit of knowledge.

I would also like to thank my friends: Johanna Shattuck and Sartaj Chowdhury who helped me survive the stress of pursuing a doctoral degree.

Looking back I reserve the final words of gratefulness to my math teachers: Elżbieta Śledziona and Elżbieta Syska.

# Contents

<b>1: Introduction</b>	<b>1</b>
1.1 Multiple Object Tracking . . . . .	1
1.2 Motivation: Precision Livestock Farming . . . . .	3
<b>2: Background</b>	<b>7</b>
2.1 A need for novel, automated approach to Precision Livestock Farming using Machine Vision . . . . .	7
2.2 State of the Art: attempts to track livestock using cameras . . . . .	13
2.3 Multiple Object Tracking . . . . .	15
2.3.1 Object Detection . . . . .	16
2.3.2 MOT as Data Association Problem . . . . .	18
2.3.3 Motion, appearance, interaction, and affinity measures . . . . .	25
2.4 Semantic Image Segmentation . . . . .	27
2.5 Pose Estimation using Keypoints and Part Affinity Fields . . . . .	31
2.6 Embeddings . . . . .	35
2.7 Convolutional processing of images . . . . .	42
2.8 Problem Statement . . . . .	53
<b>3: Method</b>	<b>55</b>
3.1 (Big) Data collection . . . . .	58
3.1.1 Pig Detection Dataset . . . . .	61

3.2	Convention of image representation . . . . .	63
3.3	Representation of Body Part Locations (Keypoints) . . . . .	65
3.3.1	Sparse representation of keypoint locations . . . . .	66
3.3.2	Dense representation of keypoint location using heatmap images	70
3.4	Pixel-level instance identification representation . . . . .	73
3.4.1	Small manually-annotated semantic instance segmentation evaluation set . . . . .	76
3.5	Class-level representation of foreground instances . . . . .	77
3.5.1	Small manually-annotated foreground mask evaluation set . . .	80
3.5.2	Multi-view alignment and foreground mask extraction from depth images . . . . .	82
3.6	Representation of Body Part Associations (Part Affinity Fields) . . . . .	86
3.7	Image Augmentations . . . . .	89
3.7.1	Augmentations in color space . . . . .	90
3.7.2	Augmentations in pixel coordinate space . . . . .	91
3.8	Models . . . . .	94
3.8.1	OP Model: A Very Deep Multiple-Objective Convolutional Neural Network . . . . .	96
3.8.1.1	Receptive Field . . . . .	97
3.8.2	UNET: A Deep, Symmetric Architecture with Skip-Connections .	100
3.9	Instance-Level Weakly-Supervised Multi-Dimensional Embeddings . . .	101
3.9.1	Silhouette Coefficient: Cohesion and Separation of Multi-Dimensional Embeddings . . . . .	102
3.9.2	Discriminative loss function for direct silhouette score maximization	106
3.9.3	Discriminative loss function with parametric cluster margins . .	108
3.9.4	Speed of Convergence Analysis using Silhouette Score . . . . .	112
3.9.5	Speed of convergence with respect to the number of clusters . . .	114

3.9.6	Speed of convergence with respect to the number of embedding channels . . . . .	116
3.10	Training using Backpropagation . . . . .	116
3.11	Pose Estimation using Body Part Detections and Part Affinity Fields . . .	122
3.11.1	Keypoint Detection using Non-Maximum Suppression . . . . .	123
3.11.2	Bipartite Matching . . . . .	125
3.11.3	Part Affinity Fields . . . . .	126
3.11.4	Augmentations of the Cost Metric . . . . .	129
3.12	Semantic Instance Segmentation using Embeddings . . . . .	131
<b>4:</b>	<b>Results</b>	<b>134</b>
4.1	Receiver Operating Characteristics . . . . .	136
4.2	Foreground Segmentation Evaluation . . . . .	138
4.3	Evaluation of the Body Part Detector . . . . .	145
4.3.1	Keypoint Detection Threshold . . . . .	147
4.3.2	Spatial Accuracy of Keypoint Detection and Distance Threshold .	148
4.3.3	Keypoint peak detector smoothing kernel size . . . . .	149
4.3.4	Scale (target width) selection based on the keypoint detection performance . . . . .	149
4.4	Part Affinity Estimation Evaluation . . . . .	152
4.5	Embeddings Analysis . . . . .	154
4.5.1	Within-instance and between-instances embedding analysis . .	155
4.5.2	Correlation with image position, orientation, size, and color properties . . . . .	160
4.5.3	Number of Instances Estimation through Cluster Analysis of the Embedding Vectors . . . . .	164
4.6	Pose Estimation Evaluation . . . . .	170
4.6.1	Performance ceiling due to representation . . . . .	171

4.6.2	Evaluation using predictions from Deep CNNs . . . . .	172
4.7	Semantic Instance Segmentation Evaluation . . . . .	175
<b>5:</b>	<b>Conclusion</b>	<b>179</b>
5.1	Challenges . . . . .	179
5.2	Approaches and contributions . . . . .	180
5.3	Recommendations . . . . .	181
5.4	Future work . . . . .	182
	<b>References</b>	<b>184</b>
	<b>Appendices</b>	<b>205</b>
A.1	Image blending using selective reconstruction of Laplacian Pyramids . .	206

## CHAPTER 1

---

### Introduction

#### 1.1 Multiple Object Tracking

Multiple Object Tracking (MOT), or Multiple Target Tracking (MTT) is a Computer Vision problem of locating multiple objects, and keeping track of their identities along with their individual trajectories given a sequence of (video) frames.<sup>1</sup> Starting in 2012, the field of Computer Vision experienced a disruptive shift in the processing methodology due to the rise of Deep Learning.<sup>2</sup> This shift can be characterized by a few factors: 1) the use of large (millions of trainable parameters) parametric models due to improvements and increased accessibility of the computational resources - namely Graphics Processing Units (GPUs), 2) introduction of Convolutional Neural Network (CNN) architecture,<sup>3</sup> 3) release of software frameworks specifically designed for Machine Learning for Computer vision such as Caffe<sup>4</sup> or TensorFlow,<sup>5</sup> and 4) availability of large-scale, annotated image datasets such as ImageNet<sup>6</sup> and PASCAL VOC,<sup>7</sup> and more recently MS COCO,<sup>8</sup> CIFAR-10,<sup>9</sup> CityScapes.<sup>10</sup> Subjectively the most impactful factor was the introduction of the convolutional processing of images which lead to formulation of various recognizable *standard*-like model architectures and training methodology. Naturally, more and more sophisticated tasks were undertaken by researchers and eventually industry. An example of this progress can be seen in the evolution of the ILSVR challenges,<sup>6</sup> which progressed

from simple image-classification tasks to what we would know today as image segmentation and steps towards *total scene understanding*. Interest in Computer Vision for robotics and surveillance needs motivated rapid progress in tracking human locomotion in natural images which became the first applications of MOT. Alongside the modern single-image processing pipelines, came methods for tracking in video frames to address the difficulties inherent to the MOT problem. Methods like YOLO,<sup>11,12</sup> SSD,<sup>13</sup> variants of the R-CNN<sup>14,15</sup> - all based on the Convolutional Neural Networks became the standard building blocks of the custom tracking systems. Some visual applications such as pedestrian tracking or face detection already became incorporated and widely used in modern computer vision frameworks and Computer Vision libraries since.

There are two major approaches to MOT:

- 2D Tracking: focuses on tracking objects on an image plane. Recently, the most popular approach is *tracking by detection* which involves identification of *pixels* of interest by various methods of segmentation and background subtraction on a per-frame basis or optical flow using temporal windows, detecting *objects* of interest by classification, and maintaining identity of the same objects between frames by solving the *Data Association* problem - often through graphical models and bipartite matching between consecutive frames.
- 3D Tracking: where the real-world, 3-dimensional coordinates and object geometry are the primary focus.<sup>16</sup> In this category methods take advantage of multi-view geometry (e.g. stereo matching) or sensing using depth sensor like Microsoft Kinect or Intel Realsense.

Being a class of Computer Vision problems, MOT faces multiple levels of challenges which can be categorized in two major domains: 1) low-level; image-capture challenges - such as variation in light intensity, reflections, noisy images, cropping and more. Those are directly related to the properties of the scene and capture device, and 2) High-level;

tracking challenges - related to the properties of tracked objects and their interactions with each other and the environment such as: occlusions, natural object deformations, appearance changes, texture, abrupt motion, camera motion (or environment motion with respect to the camera). Problem of tracking multiple objects that are difficult to distinguish visually is here referred to as *Homogeneous* Multiple Object Tracking.

Some of these challenges can be easily addressed by using different capture spectra, such as through the use of infrared depth-sensing camera. Simultaneous capture of color and depth images provides an additional dimension (distance from the camera) to the image data and reduces ambiguities in terms of scale / distance. Consumer-grade platforms like Microsoft Kinect or Intel Real Sense have proven to be a suitable choice for a proof-of-concept applications but have inherent limitations in terms of range, the size of the sensor, and its robustness to harsh environment or high temperature. More sophisticated sensors, such as 3D lidars, address those problems but their high costs make them unrealistic for wider adoption. With the modern trends in automotive industry and rise of autonomous vehicles, it is, however, possible that robust, high quality depth sensing cameras will become readily available for custom data capture platforms.

## **1.2 Motivation: Precision Livestock Farming**

Currently, livestock farming is a manual labor-intensive industry and relies heavily on herdsmen performing both the monitoring and the intervention when walking through the facilities once or multiple times a day. The herdsmen use their senses (hearing, seeing, smelling, and feeling) to uncover and resolve potential problems in the pen or at the animal level. In large scale operations with thousands of animals and vast areas, such methodology is sensitive to the availability and capability of the herdsman, cannot ensure consistency, and precision and is prone to human bias.<sup>17</sup> Due to the overwhelmingly large set of responsibilities such personnel often simply do not have enough resources to notice

single events and maintain proper record keeping in a changing environment even when trying to provide as little as the recommended 2 seconds per day per animal.<sup>18</sup>

Additionally, recent shifts in the standards of Animal Welfare put increasingly more emphasis on the quality of human-animal interaction which requires better knowledge, skills, attitudes, and behavior when handling livestock.<sup>19</sup> New requirements, tough working conditions, expected availability, and shrinking rural population strongly indicate the need for automation and the use of technology in livestock farming.<sup>20</sup>

When considering the dynamics of the environment in the facilities, there are many components to the farming *system* that can quickly (within minutes) and drastically affect the performance of the animal. In case of hog farming, the environment controls include exhaust fans, feed and water distribution, heat mats and lamps, curtains, pit slats, lights. Being able to remotely monitor and possibly control those factors on a regular interval would reduce a chance of those controls having a catastrophic effect on the animals.

This work addresses the problem of unobtrusive, visual tracking of multiple group-housed animals in an attempt to widen the context of Precision Livestock Farming by the use of Machine Learning and Computer Vision. Data sets and annotation representations are specific to swine monitoring in the group-housed setting, using a static, over-head camera, but the findings and methodology are directly applicable to other livestock animals such as beef cattle, sheep, and potentially poultry.

It has been shown that a long-term monitoring of animal behavior has the potential of predicting health outcomes.<sup>21</sup> One of the biggest challenges to ensuring the wellbeing and efficiency of pigs is rapidly and accurately identifying compromised (sick or injured) pigs. To date, the only method available for identification of compromised pigs is via manual observation for visible indicators of sickness or illness (clinical symptoms). It is However, given the quantity of pigs in modern group-housed settings, it is a daunting task to ensure that each pig is visually inspected even as frequently as once a day. This work aims to describe the processes and methodology of designing a Multiple

Homogeneous Object Tracking System for Precision Livestock Farming using Cameras.

Being based on a long-term research project, this work also attempts to capture the dynamics of the development over the span of multiple years backed by multiple publications with the emphasis on the modern Machine Learning techniques and principles. Thus, the work is focused on the development, training, and evaluation of a deep, fully convolutional neural networks trained using back-propagation. Model transfer is accomplished by the use of a pre-trained network front-end as a deep feature extractor. Supervised learning is applied to the tasks where the annotated data was available, and the weakly supervised methodology was used elsewhere. Various ideas from the author's experience with Generative Adversarial Networks were employed to address missing data problems.

The key contributions of this work are: 1) adaptation of the pose estimation method designed for human tracking to simultaneous tracking of multiple pigs, 2) modification of said method through the use of weakly-supervised embeddings produced by a neural network, 3) a novel loss function allowing for fully convolutional neural network training to produce instance-level embedding, 4) the analysis of said embeddings in the task of semantic instance segmentation of group-housed animals, 5) method of processing color-and-depth image pairs applied to a large scale unannotated image set to improve the performance of the foreground estimation subtask, 6) detailed description of the data collection and processing system animal tracking from a fixed camera without the need for depth sensing.

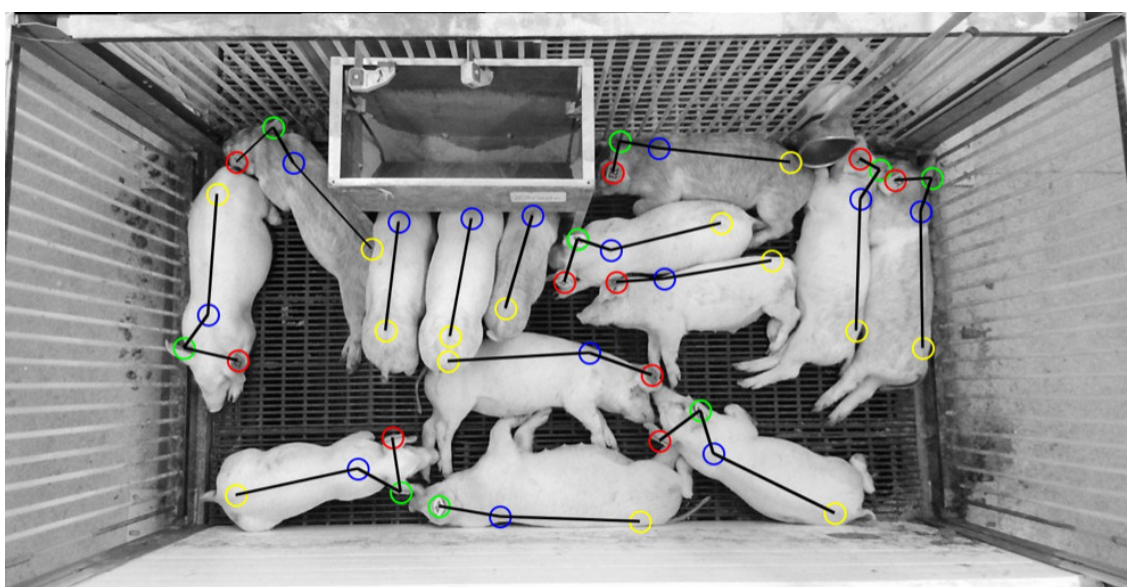


Figure 1.1: A compelling example of a pig keypoint and orientation detector from <http://psrg.unl.edu/Projects/Details/12-Animal-Tracking>.

## CHAPTER 2

---

### Background

#### 2.1 A need for novel, automated approach to Precision Livestock Farming using Machine Vision

Rapid increases in pig and cattle meat production lead to increased consumer and producer interest in animal wellbeing. Broadly speaking, health and living conditions of farm animals is referred to as *animal welfare*, which is nowadays pushing forward the standards of barn environments, food and water adequacy, and production efficiency. It has been seen, that modern technology, namely machine vision, can provide systems for real-time, automated, non-invasive animal behavior monitoring solutions. Parameters such as real-time activity (feeding, drinking, lying, locomotion, aggression, and reproductive behaviors) can be extracted from 3D and 2D image analysis. It has been established that such automated analysis can provide farmers with support much needed in the food production industry.<sup>22</sup>

Thus, the industry needs a scalable and cost-effective way of monitoring animals at an individual level and in a more continuous fashion. Supplementing the staff with precise metrics of animal and environmental performance would drastically reduce the costs and labor required to properly maintain operations. Lowering the amount of unnecessary direct interaction with animals will help reduce the animal stress level and

lowers the risk of biological contamination carried between facilities.

Precision Livestock Farming (PLF) is an engineering approach to livestock management using automated, long-term monitoring of individual livestock.<sup>23–25</sup> Systems falling under the PLF category aim to aid the producers with high-precision, low-latency (or real time) tools capable of determining the state of the facility and the animals on the individual level. PLF data collection platforms consist of cameras and sensors for measuring temperature, air speed, humidity, and gas contents (e.g. ammonia). On a facility level, vibration and current sensors help monitoring the state of equipment and microphone arrays can be deployed to keep track of the noise levels and potentially detect pen-level events.

Researchers proposed a variety of technological approaches to individual animal tracking in PLF over the last decade<sup>26,27</sup> including: wearable Ultra-Wide Band (UWB) tags,<sup>28,29</sup> GPS-enabled motes,<sup>30,31</sup> accelerometers / Inertial Measurement Units (IMUs),<sup>32–35</sup> RFID ear tags,<sup>36–38</sup> and depth-sensing cameras.<sup>39,40</sup>

Whereas the wearables offer a solution to individual animal tracking, they have a set of disadvantages when compared to surveillance-type systems with cameras.<sup>40,41</sup> Battery-powered devices also carry an additional burden on the staff due to outages and additional charge-level monitoring requirements. Animal farming facilities require the monitoring equipment to withstand harsh, humid, hot, dusty environments, need to stay attached to the observed animal, and their initial deployment is costly and scales with the size of the operation as each individual animal needs to be equipped with a uniquely identifiable sensor mote or tag.<sup>42</sup> UWB and GPS systems provide reasonable positioning accuracy in their respective environments (indoors and outdoors) but alone lack the ability to determine the animal's orientation. The use of IMUs alone allows for very accurate estimation of the orientation but due to long-term drift do not provide good positional accuracy due to magnification of the measurement errors through double integration operation when estimating position from acceleration. A combination

of positional and inertial sensing is a very common solution to this problem and has been used successfully in many applications. In land navigation, additional, potentially more sparse but precise measurements of absolute position allow for closed-loop corrections.<sup>43</sup> Literature shows successful implementations of tracking animals using IMUs but they do not inherently carry over any richer context such as social behaviors.

Cameras however, do carry over rich contextual information (both spatial via image resolution and temporal in video) and allow for unobtrusive deployments allowing for minimum human-to-animal interaction during monitoring. The video collection system has a multitude of advantages including the fact that individual frames are human readable and allow for the easy evaluation / annotation of any frame. Large-scale data sets can be easily obtained once the system such system is deployed, which leaves the most interesting aspect of this work as a still-open question of how to process such a vast amount of data with minimum human effort? As processing of the video frames of on a contextual / semantic level requires more sophisticated treatment than just mere image processing techniques, researchers started to lean more openly to the field of machine learning and artificial intelligence incorporating recent advancements in those rapidly growing fields in their work.<sup>44, 45</sup>

An image-processing pipeline of a tracking system often explicitly includes a method of determining the pixels of interest. Such a process is referred to as *segmentation* and can occur on multiply levels of abstraction. In its most basic form, it allows for distinguishing the foreground pixels from the background using a background subtraction.<sup>46</sup> When the pixel blobs obtained through this process are spatially separated, a connected-component algorithm can be used to obtain identification of individual instances. Another approach is to use a separation in the pixel intensity values via clustering if the within-cluster intensity value deviations are lower than between clusters according to a selected similarity measure. In the current application however, it is not a feasible approach as the objects (animals) are often close to one another or even on top of

each other (due to piling when animals try to stay warm) which prohibits the spatial separation, and additionally indistinguishably similar in color to one another prohibiting the clustering. Thus, a more sophisticated, rich-feature-based and context-aware method needs to be used when attempting to process images of group-housed animals.

This work is a step towards addressing the broader problem of lack of automated methods of animal behavior monitoring. Section 1.2 establishes the need for the technology to step into the agriculture, and anticipation for the potential use of visual tracking to predict animal health outcomes. A successful, widely adopted, easy to deploy, and unobtrusive long-term tracking method could yield means to better understanding of animal conditions and improve overall quality of animal handling.

Thanks to a collaboration with the Department of Animal Science of University of Nebraska-Lincoln it was possible to gain access to locations resembling swine production facilities. The main goal was to capture large amounts of visual data, while at the same time, working on the ways of processing it. At that time it was anticipated that simultaneous collection of color and depth images could be useful in the future. Use of cameras addresses the needs for an unobtrusive, asynchronous, and unattended, long-term data collection with minimal interference with animal-handling personnel.

Currently available methods produced by the research community are presented in Section 2.2. To this day however, there are no methods presented that could satisfy the needs of the industry. With such a niche field, the amount of data available for development of such method is also fairly sparse. Thus, in order to step in an attempt to tackle this problem, one had to resort to collecting their own data. Section 3.1 describes the process and methodology used to collect the data for this work.

Chapter 3 is dedicated to the description of the proposed method of tracking by detection using outputs of a deep, fully convolutional neural network. Additional sub-tasks and significant earlier attempts are briefly described as well to provide reader the broader context of work that contributed to this final presentation.

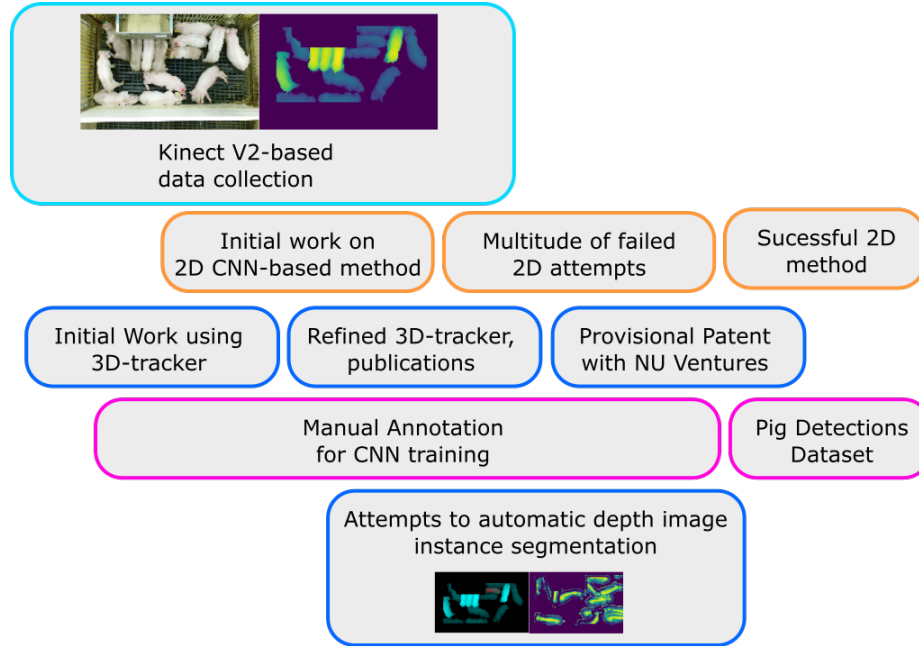


Figure 2.1: Lifetime of the *Pig Tracking* project. Lengths of the blocks represent relative timespan of various components - often done in parallel. Total work spans over the course of years 2016-2019.

As shown in Figure 2.1, the annotation ended up being one of the most time-consuming phases of this project. Section 3.3.1 describes the proposed process of annotating images in a fashion that is specific to pigs. Similar methodology however can be used for different kinds of targets.

Our initial work on this problem involved the development of a novel iterative EM-based clustering technique of elliptical objects in 3D space.<sup>39,40</sup> In spite of being very compelling at the time, and resulting in the US patent application,<sup>47</sup> this method does not seem applicable for a widespread adoption as it imposes requirement of capturing depth information as well as color images. Witnessing a huge shift in the field of computer vision towards the use of CNNs inspired the author to participate and potentially contribute to both fields: Precision Livestock Farming (PLF) and Machine Learning, in an interdisciplinary fashion. Furthermore, we decided to reinterpret the problem from iterative clustering in 3D to instance-level semantic image segmentation and pose estimation in 2D, which is described in Section 2.4.

Switching the source domain from sparse 3D point clouds to dense color images required a new model, one capable enough to accommodate necessary operations transforming the inputs  $x$  into desired outputs  $y$ , and a deep CNN became a natural choice. When using a parametric model  $y = f(x, \theta)$  one needs to define appropriate representations of the input  $x$ , output  $y$  and parameters  $\theta$ . CNNs are indifferent in that regard. Since their original versions date from the early 1990s, the community developed standard formats and conventions for representing the visual domain inputs (images), outputs, training objectives, and structure of the parameters via commonly adopted network architectures. Section 3.2 introduces the reader to the convention of data processing using CNNs in a broad sense, while Sections 3.3 through 3.4 describe the problem-specific choices of representations.

Despite being extremely successful in tackling computer vision tasks, CNNs, due to the very large number of parameters always carry a risk of over-fitting, casting a shadow on their ability to generalize and produce useful results for unseen data. We acknowledge those limitations in section 2.7, and present attempts to mitigate them.

When switching from 3D to 2D processing, we always kept in mind the anticipated potential hidden in the *depth* of depth information. First, it was identified that the process of training large, deep CNNs requires large amounts of data. Being however equipped with unannotated pairs of color and depth images, author attempted to make a good use of the sheer volume of available images by providing a reliable, high quality ground-truth information for the subtask of foreground estimation. A relatively large amount of paired data was processed to extract foreground (class level segmentation) masks. Section 3.5.2 is dedicated to description of this process including the use of the elements of multi-view geometry.

Chapter 4 presents the analysis of the method in the context of available datasets and metrics commonly used for evaluation of undertaken tasks.

## 2.2 State of the Art: attempts to track livestock using cameras

Nasirahmadi et al.<sup>48</sup> studied the relationship between group lying behavior and ambient temperature using model-based ellipse-fitting from.<sup>49</sup> They used adaptive background subtraction<sup>50</sup> to separate separate parts of the image belonging to the animals from the static elements of the scene. The number of pigs detected in the picture and their orientation were accurate approximately 95% of the time. The method is mathematically sound and well defined, however it does not address the ambiguity between the head / tail position as it only aims to determine the dominant axis. The method seems to be able to handle a partial occlusion but no explicit claims about occlusion handling are made in the paper. Also, the animals used in the trials had little to no variety in terms of appearance.

Ahrendt et al.<sup>42</sup> use a model-based approach to pig tracking with a multivariate Gaussian distribution to represent the  $x, y$  coordinates and appearance (RGB intensity). The authors adapt the model between frames in an EM-like fashion using Mahalanobis distance as instance affinity measure. The method is compact and robust but fails to separate instances when the animals are touching or in the presence of high number of instances due to limited frame rate and optics. In order to challenge the frame rate, authors had to allow for high enough variance between the frames to accommodate model adaptation, which lead to separation problems. The fish-eye optics were used to allow for larger field of view in the presence of multiple animals but the amount of introduced distortion was overwhelming for their processing algorithm. Figure 2.2 illustrates the this effect.

The introduction of a depth sensing component helps to resolve the ambiguity between scale and location (distance from the camera) and allows for fitting geometric models to the 3D point clouds in the real world coordinate space. An approach utilizing Microsoft Kinect depth-sensing camera uses an explicitly defined bounding box to isolate

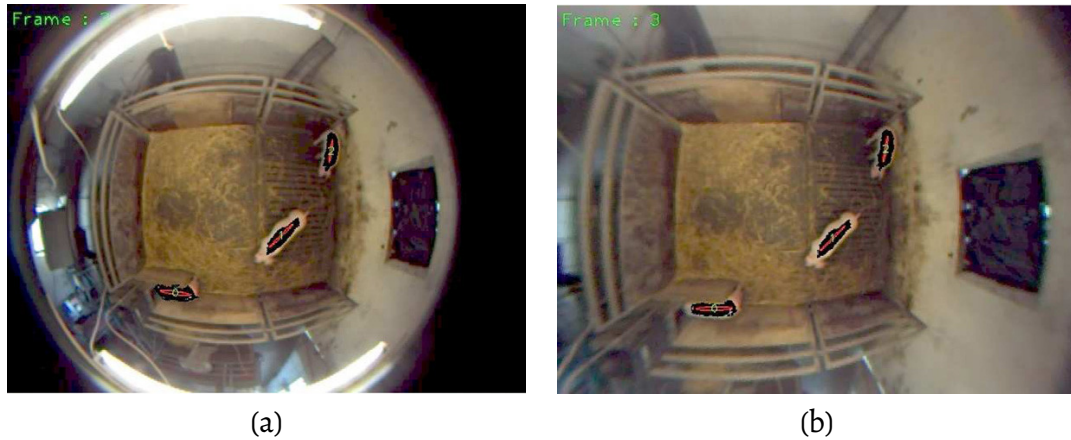


Figure 2.2: Illustration of the top-down view of the pig pen captured with the fish-eye lens in the work of Ahrendt et al.;<sup>42</sup> before the distortion compensation algorithm is used (a) and after (b).

the points of interest.<sup>40</sup> The method focuses on incremental inter-frame adaptation using EM and the authors have shown an average of 20 minutes of consistent tracking but manual initialization is required.

Matthews et al. uses depth sensing for activity monitoring of the animals.<sup>51</sup> Their method operates without the need for manual initialization and is capable of tracking individual animal instances via performing regional grouping of surface normals with the average reliable tracking duration of 22 seconds.

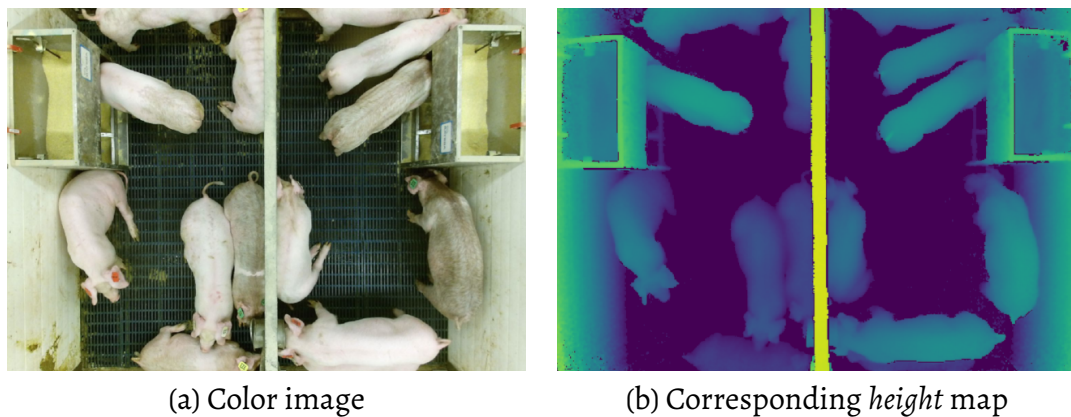


Figure 2.3: Example of color (a) and corresponding, processed depth image in form of a height map (b).

## 2.3 Multiple Object Tracking

A popular approach to MOT is *tracking by detection* which consists of two major components:<sup>52,53</sup> *object detection* and *data association*. When referring to MOT, one usually has in mind a scenario like the one shown in Figure 2.4. The image depicts a state of an MOT method for pedestrian tracking.<sup>54</sup> There is a number of pedestrians (objects) being tracked by the method. Each pedestrian is indicated in the frame using a bounding box. Each bounding box is also color-coded to display the unique identity of each tracked object. It is safe to assume that that objects are in motion between the frames. The tracker combines the detection hypothesis from each frame and combines them into likely progression of each object in the sequence of frames using a data association method. Such method often depends on appearance and motion models. The location of each object across frames is tracked by the method and displayed as individual trajectories using colored lines. It is safe to assume that the camera is stationary as the image most likely comes from a street monitoring system.

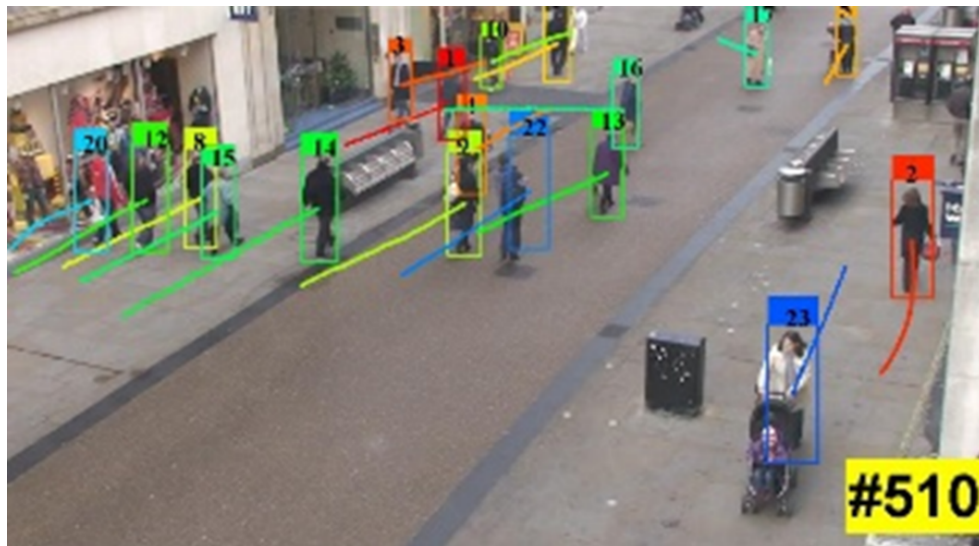


Figure 2.4: A visual depiction of an output of a Multiple Object Tracking Method applied to pedestrian tracking.<sup>54</sup>

### 2.3.1 Object Detection

Growth of availability of vast data sets containing millions of annotated images and publication of open, performance-oriented challenges can be identified as driving force in the field of Computer Vision. The most commonly known challenge (and data set) is ImageNet Large Scale Visual Recognition Challenge (ILSVRC)<sup>6</sup> focused on three fundamental tasks in the field. At this point it is important to define the terms classification, localization, and detection according to ILSVRC's definition:

- Image classification is the task of determining a single class for the entire image. In ILSVRC it is one out of a 1000 classes and the top 5 scoring classes are reported. If the ground-truth class is present among the five, it is counted as full success, failure otherwise.
- Single-object localization focuses on finding one object out of the list of ground-truth objects present in the image along with proper ( $\geq 50\%$  Intersection over Union (IoU)) bounding box position and size.
- **Object detection** is the most demanding task and involves proper classification and positioning of bounding boxes around all the objects indicated in the ground-truth with criteria as in previous tasks.

Classic approaches implemented sliding-window methods operating on multiple scales of the images and correlating pre-trained visual object representations with the content of the window, which can be unsuitable for online tracking scenario due to computational complexity increasing with the number of learned representations.<sup>55</sup> To overcome the complexity problem, some methods used two-step processing with lower-accuracy detector yielding regions of interest for more robust one.<sup>56,57</sup>

Another way of avoiding sliding-window approach is to explore the frequency-domain representation of the image.<sup>58,59</sup> Henriques et al.<sup>60</sup> use properties of

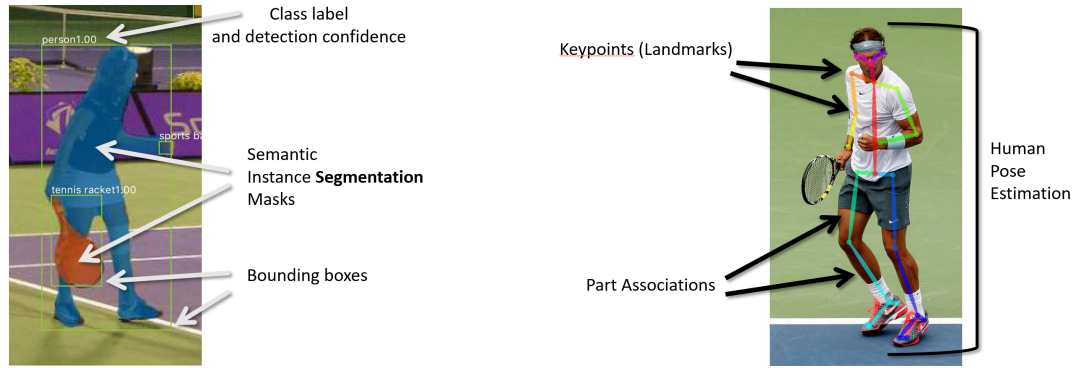


Figure 2.5: Object representations: top-down representation of objects as bounding boxes or segmentation masks<sup>64</sup> often used in tracking by detection methods (left) and a bottom-up representation of a person as an organized collection of keypoints and their associations<sup>65</sup> (right).

circular matrices and operations in Fourier domain for high performance ( $\approx 100$  frames per second) single-object tracking.

Modern hardware however allows for using more computationally expensive methods such as Neural Networks in (nearly) real time due to GPU processing and make 2D convolution operation feasible. This was exploited heavily by Sermanet et al. by using multi-scale sliding-window approach with simultaneously learned bounding box regressor and classifier.<sup>61</sup>

After obtaining hypothesis of object class membership of each pixel, additional step can be introduced to model the interaction between detected objects within the same frame. Method presented in<sup>62</sup> focuses on learning complex objects as constrained alignments of its parts and was widely used in various Computer Vision applications including MOT.<sup>63</sup>

Object detection can be related to other tasks based on representation. In Figure 2.5 we are contrasting two opposite object representation methods: top down (left) starting with region of interest proposals (bounding boxes) and bottom-up based on detecting parts (right).

Most recently, Machine Learning-based approach, namely R-CNN<sup>68, 69</sup> introduced a



Figure 2.6: Mask R-CNN extended to estimate human pose from in images selected from COCO data set and operating at 5 frames per second.<sup>66,67</sup>

multi-step Neural Network structure capable of complete Semantic Segmentation of single images. Work presented in<sup>64,67</sup> shows human pose estimation based on keypoint detection using Mask R-CNN on single color images with real time performance when running on modern hardware. It is worth to mention that this recent development combines multiple steps of the pipeline within one trainable structure yielding very promising results in terms of both accuracy and performance.

Following reasoning by Choi et al.,<sup>53</sup> having high quality object detector is essential in achieving prime tracking performance. Most recently Bergman et al.<sup>52</sup> argue that a high quality object detector is all that is necessary to achieve state-of-the-art performance in MOT.

### 2.3.2 MOT as Data Association Problem

Resolving object correspondences between frames is the essential component of MOT pipeline. Depending on the implementation and allowed output latency, it can allow for *filling the gaps* caused by occlusions and estimation of actual motion of the tracked objects while maintaining consistent distinct identities of the targets. In the scenario of 2D tracking-by-detection, resolving inter-frame correspondences is formulated as the Data Association Problem (DAP). Even though partial occlusions can be internally handled by CNN-based object detector,<sup>66</sup> proper association of object in (at least) consecutive frames

is essential for tracking.

Before diving into graphical models considering larger time windows (both past and future) it is worth to mention, that some tracking systems use Markovian assumption and maintain continuously updated *state* built upon the evidence from processing past frames and adopting the model to most recent frame. Widely used single-object tracking method known as Continuously Adaptive Mean Shift (CAM-SHIFT)<sup>70</sup> operated by continuously adopting scale and position of tracked target based on evidence of the detections. Again, it is worth to emphasize the importance of high quality detector at this point.

MOT is often formulated as Data Association Problem (DAP). This approach is commonly used in tracking-by-detection methods and heavily rely on quality of object detector and affinity measures between the detections. The goal of DAP is to find optimum (minimum cost) assignment in a graph  $G = (V, E)$  where vertices represented by set  $V$  are object detections and edges represented by set  $E$  are potential edges between vertices  $v_1, v_2 \in V$  with cost representing dissimilarity between detections corresponding to vertices  $v_1, v_2$ . This formulation is prone to major difficulties caused by false positive and negative detections.<sup>71</sup>

Depending on the application and computational constraints the correspondences between the single detections and vertices in the graph can be modeled differently as well as the global cost metric or costs associated to edges. At this point the author would like to recommend a few literature positions which, according to his belief, represents the evolution of Data Association Problem formulation in MOT.

Solving MOT as DAP is often approached using temporally local matching between each pair of consecutive frames. This class of methods simplifies the problem to *bipartite matching* which can be solved using Hungarian Algorithm.<sup>72</sup>

Temporally global methods constrain the problem around batch of frames which can better handle occlusions and provide smoother tracks.<sup>71,74–76</sup> The example difference

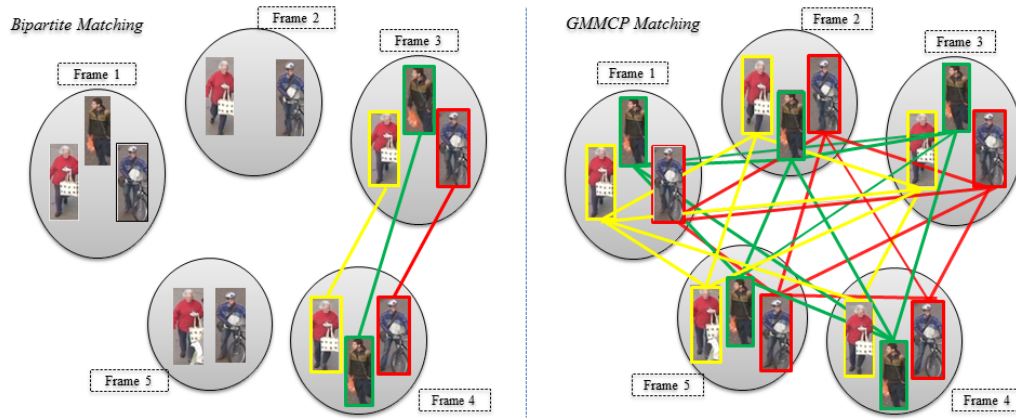


Figure 2.7: Bipartite matching between pairs of frames compared to k-partite complete graph spanned over longer time window.<sup>73</sup> It is worth to mention that k-partite complete graph is not the only formulation used in solving the DAP but is used here as a valuable example clarifying the difference.

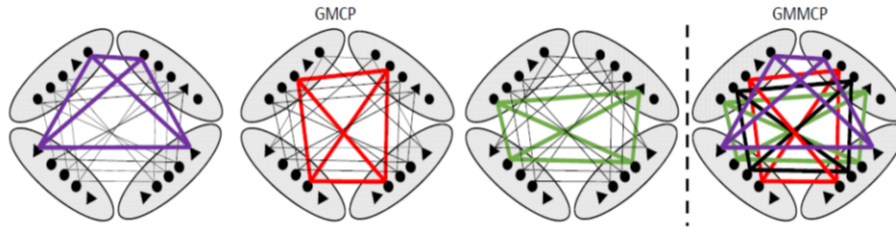


Figure 2.8: Minimum clique problem formulation used in GMCP<sup>74</sup> compared to k-partite complete graph problem.<sup>73</sup>

between temporally local and temporally global processing is presented in Figure 2.7. Global optimization is significantly more computationally demanding than local, thus some methods formulate the tracking problem in global fashion but solve it using greedy processing (solving trajectories one-by-one, often according to heuristic quality prediction based on detection scores) which is vastly criticized for its suboptimal tracking quality and yielding non-optimal tracks<sup>53</sup> caused mostly by occlusions and within-sequence variations.<sup>73</sup> The difference between greedy processing and global optimization is presented in Figure 2.7 which illustrates the major difference between presented here graphical methods: GMCP<sup>74</sup> and GMMCP.<sup>73</sup>

The rest of this section is dedicated to showing the evolution of DAP formulation for tracking. It presents *ancient* MHT recently revisited due to the use of CNN-based features. GMCP and GMMCP as examples of elegant, clever, and mathematically sound problem formulation, JMC as a the most recent reinterpretation of the problem which does not require 2-step processing allowing for more robust tracking, and MDPNN using Long Short-Term recurrent neural networks and learns motion, appearance and object interaction jointly. These methods were selected based on their high tracking scores presented in recent comparative study<sup>77</sup> (JMC, MDPNN, MHT-DAM), elegance and relevance (GMCP, GMMCP).

More recent revision of Multiple Hypothesis Tracking (MHT) using features obtained from deep convolutional neural network (MHT-DAM) presented in<sup>78</sup> is based on the initial proposal from 1979.<sup>79</sup> The major strength of this method lays in keeping multiple identity hypothesis through the tracking process and resolving the past as new information is processed. It heavily relies on optimized heuristic solver.<sup>80</sup> The most likely set of tracks is a result of solving the Maximum Weighted Independent Set (MWIS)<sup>81</sup> and incorporating appearance modeling through deep features<sup>68</sup> compressed to 256-element vectors using PCA for higher performance. Analysis of comparison between MHT and MHT-DAM using deep features shows that appearance features are significantly more important than movement features. And having *proper* appearance feature vectors makes it less sensitive to the size of temporal window. Authors also point out that simplistic motion models such as linear motion often do not represent the behavior of real tracked objects yielding suboptimal results. This method scores highly in recent comparison study<sup>77</sup> but is potentially prone to mismatches due in homogeneous target appearance applications due to its emphasis on appearance.

Zamir et al.<sup>74</sup> present a global multiple object tracking method (GMCP<sup>1</sup>) formulating DAP as minimum clique problem (where complete subgraphs span over the same identity

---

<sup>1</sup>Pleasant video presenting GMCP: <https://www.youtube.com/watch?v=f4Muul1d7NhA>

detections over multiple frames). The *globality* is achieved by incorporating 2-step approach of slicing the sequence into temporally local sub-sequences and extracting trajectories using *tracklets* and estimating linear velocity vectors and appearance representation through averaging visible appearances. Then, the problem is solved iteratively using Tabu search<sup>82</sup> using greedy method estimating *global* trajectories in sequence. Occlusions are handled by adding hypothetical nodes filled up using RANSAC. The greedy processing is the most important disadvantage of this method.

Deghan et al.<sup>73</sup> criticize GMCP for using greedy processing and introduce DAP in MOT formulated as  $k$ -partite complete graph problem which can be solved simultaneously yielding optimum track assignment. Comparison of GMCP and GMMCP is presented in Figure 2.8. On the mathematical level authors establish three major constraints incorporating all hypothetical assignments within the considered time window and use Binary Integer Programming (BIP) solver. Authors formulate tracklets to lower the computational complexity but also allow for a motion model incorporated into the cost metric.

Authors provide 3 constraints to formulate GMMCP as BIP which essentially builds a  $f$ -partite complete graph ( $f$  being the number of frames) which considers all possible associations of detections between all frames / tracklets. They introduce dummy nodes to handle occlusions and represent them as integer variables instead of binary and use a heuristic measure to determine their number, which reduces the number of constraint equations. They use<sup>83</sup> (color histogram intersection) as their affinity measure. Authors show superiority of their methods compared to GMCP.

It is worth to mention that both, GMCP and GMMCP are not well suited for large time frame formulation as the problem scales by the number of frames in the time window with respect to number of detections per frame and exponentially with respect to number of frames. The main computational complexity comes from actual memory allocation of large scale constraint matrices and filling in the values to accommodate the

constraints. Thus, for fixed number of targets and fixed size of the time window it can be done once and just solved for the cost values coming from detection scores and object affinity. This however imposes constraints of the number of detections allowed per frame, which requires either very accurate detector, raising the detection confidence threshold (which can result in increased number of FNs) or using aggregation techniques such as Non-Maximum Suppression (NMS). These observations come from the experience of the author of this proposal as he implemented the GMMCP in MATLAB and applied it to livestock tracking but did not achieve high quality results due to poorly defined visual affinity and not incorporating motion constraints.

In<sup>63</sup> authors propose JMC, a novel graphical method of solving MOT using DAP. As a natural extension of their previous work in<sup>84</sup> authors eliminate the need for intermediate local trajectory representation of tracklets as authors lower the emphasis on motion model and focus on appearance. This is aiming towards high robustness against camera motion. In their previous work<sup>84</sup> authors handled false positives using additional binary variables at nodes for masking depending on detection confidence threshold. Here, authors add post-processing step to remove small clusters of detections. The main advantage of this change is ability to use KLj-algorithm from<sup>85</sup> without any modifications. They reframe the MOT as minimum cost subgraph multicut problem.<sup>84, 86</sup> As a result all the detections corresponding to the same target can originate both inside single frame as well as across time due to the fact that solution of a problem is based on subgraphs and not paths (edges between single detections). Authors use deep matching<sup>7</sup> for affinity measures of each detection yielded by DPM<sup>62</sup> object detector. The use very efficient solver presented in.<sup>85</sup>

Finally, a very sophisticated method presented in<sup>87</sup> (MDPNN) uses Long Short-Term Memory (LSTM) Neural Networks. Authors jointly learn target representations taking into account appearance, motion and interaction after being inspired by recent work on Structural Recurrent Neural Networks<sup>88</sup>

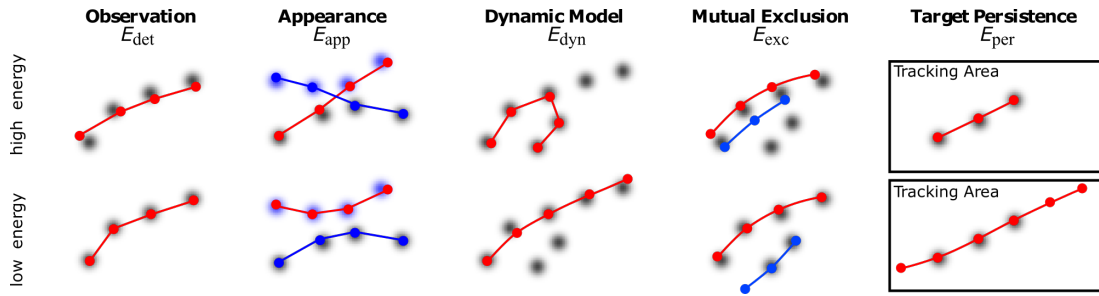


Figure 2.9: Effects on different components on the continuous energy function presented in.<sup>89</sup> Please keep in mind that this method is an energy *minimization* framework, so the *correct* associations are going to carry lower energy than the erroneous ones.

Authors point out the importance of modeling the target's motion in case of handling occlusions as it allows for predicting the location of the target in the next frame but also emphasize the importance of appearance as performance of graph-based MOT methods is bounded by design choices of similarity function.

DAP can also be solved using maximum-a-posteriori (MAP) network flow approach which is capable of handling longer time windows. Work presented in<sup>71</sup> uses min-cost flow algorithm to for pedestrian tracking scenarios. Each trajectory is modeled as Markov chain. Each detection has associated random variable with Bernoulli distribution trained on the data and modeling probability of being a true or false detection. These methods introduce additional graph vertices and edges modeling uncertainty between the hidden state (true positions of targets) and the hypothesis.

Another attempt to solve the Data Association Problem is by minimizing global energy function defined such that it captures all possible tracked targets in all frames. In<sup>89</sup> authors present Continuous Energy Minimization (CEM) framework for Multiple Object Tracking. Authors state that due to its general non-convexity the tracking problem can not be solved globally and argue that local optima of carefully formulated objective yield satisfying results in practice. Their objective function explicitly encapsulates: detection evidence, appearance, motion dynamics, persistence, and collision avoidance (Figure 2.9) in continuous space and is differentiable. Their object detection is based on

linear SVM and they use HOG<sup>90</sup> and HOF<sup>91</sup> as feature descriptors filtered by NMS.

NOMT method presented in<sup>53</sup> achieves very good performance according to,<sup>77</sup> mostly due to high emphasis on robust affinity measure between any two detections and novel affinity descriptor (AFDL) which mostly focuses on movement.

### 2.3.3 Motion, appearance, interaction, and affinity measures

Interaction between tracked objects can be modeled either explicitly or implicitly. The most common methods of explicit modeling are used in the pedestrian tracking: crowd motion patterns<sup>92</sup> and social force model (group model)<sup>93</sup> where object trajectories are augmented by two types of forces: attraction or repulsion. In<sup>89</sup> authors use repulsion based on target's volume.<sup>94</sup> These standard models were criticized for low generality (due to inflexible constraints imposed on motion such as linearity) and over-simplicity making it unable to capture more complex interactions between objects and more data-driven, LSTM-based model was proposed.<sup>87</sup> The main strength of this approach is in its ability to model long term interactions based on multiple clues. Implicit object interaction (mostly *repulsion*) is often handled by exclusive area occupancy such that no two distinct objects (or parts of those objects) can be present in the same frame at the same location (nor having bounding boxes with areas overlapping above certain threshold) which is often inherently ensured in tracking-by-detection by the detector's output (with proper segmentation).<sup>71, 73, 74</sup>

In<sup>53</sup> authors present Near-Online Multi-Target (NOMT) tracking framework heavily relying on their motion affinity measure with less emphasis on the appearance. Such choice is often motivated by the need for tracking similarly looking (or even homogeneous) objects. They introduce Aggregated Local Flow Descriptor (ALFD) which encodes relative motion pattern between two detection boxes in different time frames. Authors propose to solve the problem using temporal window  $\tau (V_1^t, \mathbb{D}_{t-\tau}^t, \mathbb{A}^{t-1}) \rightarrow \mathbb{A}^t$ . Given the previous set of points, they identify new points and reduce their number by

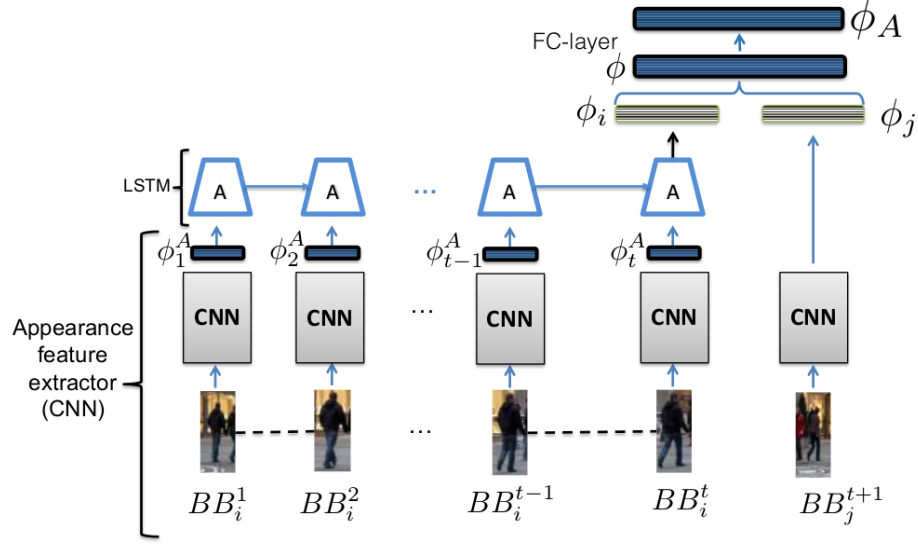


Figure 2.10: Appearance model presented in.<sup>87</sup> Bounding boxes at each time step are processed through a CNN, effectively used as a feature extractor. Those features are then passed into the LSTM-based appearance model for all time steps  $i = 1, 2, \dots, t$  and compared with the features from the most recent frame using fully connected layer.

applying a movement threshold of at least 4 pixels. Then they calculate forward and backward optical flow. Any point with high disagreement in forward-backward flow (consistency measure) is terminated.

Methods described in previous section used tracklets to describe temporally local, consistent sub-trajectories of targets. In<sup>73</sup> authors build tracklets by slicing the sequence into a fixed length sub-sequences and estimating motion models for consistent detections. They also use forward-backward consistency when merging tracklets into *smooth* (global) trajectories. It has been shown in the literature that the use of tracklets is <sup>a</sup><sup>95–97</sup>

Choi et al.<sup>53</sup> leverage the importance of pairwise affinity measure between any two detections based on motion clues for considering similarly looking targets. The most recent, previously mentioned work presented in MDPNN tracker<sup>87</sup> learns the motion model and affinity measure between tracks using LSTM neural network in very similar fashion to the way it does with respect to the appearance (Figure 2.10).

In<sup>98</sup> authors explore the relationship between spatial overlap (on an image plane)

and the upper bound of the appearance similarity, namely the appearance similarity decreases with decreasing overlap. Authors explore multiple common appearance descriptors such as bag of visual words using SURF features,<sup>99</sup> GIST,<sup>100</sup> and HOG.<sup>90</sup> Authors propose an efficient algorithm for affinity measure capable of real time operation.

In<sup>101</sup> authors propose a Siamese CNN architecture to estimate object likelihood of two pedestrian detections to belong to the same tracked identity. They combine visual affinity (pixel values) with motion (optical flow) in their measure.

Appearance-focused methods used in classic tracking systems can be often described as *weak* visual affinity measures such as: spatial affinity (bounding box overlap or euclidean distance),<sup>75, 76, 102</sup> color histogram intersection.<sup>74, 83</sup>

Dis(similarity) measure between two image regions is a fundamental problem of stereo matching. The output of the evaluation function usually produces overall *cost* value or returns disparity directly. The groundbreaking work in<sup>103</sup> uses neural network-based approach to estimate the similarity score. The network is built from two convolutional feature extractors, concatenation layer, and a stack of fully connected layers. Presented method outperformed previous attempts.

Concluding this section author would like to once again emphasize the importance of the visual object representation and affinity measure due to its broader impact in the field of Computer Vision.

## 2.4 Semantic Image Segmentation

Semantic segmentation allows systems to interpret image content in both the spatial and categorical domain. As arbitrary as it seems, it is a task beyond mere texture and color-based analysis and requires domain-specific prior knowledge stored as model coefficients. When referring to *segmentation*, it is important to specify its level starting

from *background subtraction* being a single-class foreground/background identification task, through multi-class case and eventually instance-level or sub-instance level. When an image contains multiple disjoint segments of the same category, the segments can easily be separated into unique instances. Such categories can be defined as people, animals, inanimate objects, and more.<sup>8</sup> Unfortunately, in cluttered scenes this condition is seldom satisfied; here, one can only interpret the results as a collection of ambiguous, inseparable blobs. Figure 2.11(a,b,c) illustrates various levels of segmentation and the limitations of (class-level) semantic segmentation in cluttered scenes.



Figure 2.11: Semantic segmentation and instance-level segmentation of people (Cityscapes dataset,<sup>10</sup> Hamburg image #036527): (a) original image; (b) semantic person segmentation; (c) grouping via connected components; (d) person instance segmentation.

Instance segmentation adds to the capabilities of semantic segmentation by distinguishing between objects of the same category. This task is significantly more challenging than mere semantic segmentation as it needs to produce unique identification of each instance such that there exists a clear boundary between them. Implementation carries additional challenges because the specific identifiers provided by ground truth instance labeling are permutation invariant, i.e., any permutation of ground truth instance labels (unique colors in Figure 2.11(d)) is considered a perfect solution.

Researchers have used a variety of different approaches to achieve instance

segmentation despite the inherent ambiguity of labeling. In general, they can be divided into two classes: 1) top-down and 2) bottom-up. A top-down approach begins by isolating the region of interest from the rest of the image prior to performing pixel-level segmentation<sup>66, 104</sup> while the bottom-up methods attempt to assign unique “codes” to each pixel so that clustering methods like mean-shift can separate different instances easily<sup>105, 106, 107</sup>.

Modern methods achieve impressive results on challenging datasets like COCO,<sup>8</sup> Cityscapes,<sup>10</sup> and KITTI,<sup>108</sup> however, because they are specifically trained to assign a single instance label to visible pixels, each object’s full spatial occupancy and depth ordering — two properties that humans instinctively estimate — are not represented in the image annotations. As a consequence, the common approach of training deep fully-convolutional networks (FCNs) to detect and segment objects in the image<sup>109</sup> faces the dilemma of an ambiguous target because there is no definitive ground truth to provide the network during training. In the context of tracking, the main limitation inherent to those vast instance-level segmentation datasets is the lack of temporal context as they contain random natural images and not consecutive image frames.

As mentioned above, there are two categories of approaches to achieve instance segmentation. The first begins by finding the regions (often bounding boxes) that contain each instance, and then performing pixel-wise segmentation of the dominant instance within that region.

One of the most successful and nowadays widely adopted family of top-down methods are based on the work proposed in 2014 by Girshick et al.<sup>110</sup> known as R-CNN. Its latest incarnation, Mask R-CNN, introduced by He et al.<sup>66</sup> is capable of performing instance level segmentation and keypoint detection. It extends upon Faster R-CNN<sup>15</sup> by adding a branch for segmentation mask prediction in parallel with the other branches (bounding boxes and classification). However, because it relies on a priori region proposal, it is inherently unable to separate objects with significant bounding box

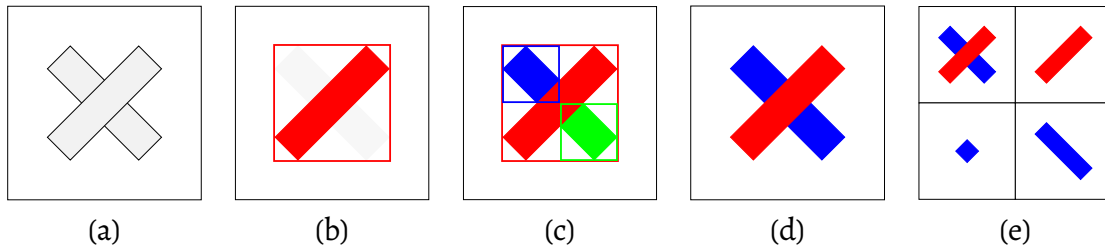


Figure 2.12: Instance segmentation output scenarios for two overlapping rectangles.

overlap—a common occurrence among group-housed animals

Li et al.<sup>111</sup> proposed a solution that uses a location-sensitive fully convolutional network that partitions bounding boxes into a  $3 \times 3$  grid, and then evaluates the likelihood that each partition contains the correct part relative to the other partitions. Alternative approaches using recurrent neural networks with attention have also been introduced to iterate through the instances while keeping track of which regions have already been processed<sup>104, 112</sup>

Methods relying on bounding box selection are inherently limited by *an priori* region selection. When instances of similar size overlap with one another, the region selection phase often experiences one of two types of error: either 1) due to non-maximum suppression, the algorithm ignores the bounding box of the occluded instance (Figure 2.12(b)), or 2) the instance will be represented as a collection of separate partial instances (Figure 2.12(c)).

As an alternative to region selection, bottom-up approaches that use pixel embedding move the high-level detection stage to the end of the process<sup>106, 107, 105</sup>. A bottom-up keypoint detection was even successfully adopted to cow tracking.<sup>44</sup> The authors identified a set of class-specific landmarks visible from a top-down view to represent each cow's location and orientation, and trained a fully-convolutional neural network to detect them in images. A post-processing network was then used to convert the annotations to per-pixel orientation classification outputs, resulting in 95% accuracy in correctly labeling all cows in a given image. In a follow-up experiment, they applied the

previously trained network to a new environment and observed that it only succeeded on only 55% of images, indicating that the network was over-fitting to a particular environment.

## 2.5 Pose Estimation using Keypoints and Part Affinity Fields

Wei et al. introduce the concept of convolutional pose machines.<sup>113</sup> Their work establishes an architecture, landmark definition and general approach to bottom-up processing suitable for human tracking. Cao et al.<sup>65</sup> extend this work and propose a real-time method of multi-person pose estimation using a combination of keypoint detection, a novel concept of *Part Affinity Fields*, and well-defined post-processing step to resolve associations between objects of interest.

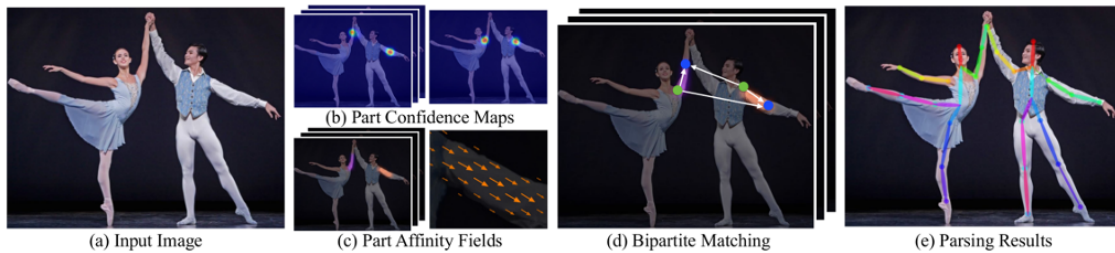


Figure 2.13: Real-time multi-person 2D pose estimation.<sup>65</sup>

This approach is still targeted to single-domain processing; namely: human pose estimation where the objects of interest are *people* visible in single images and trained on two popular datasets: MS-COCO and MPII. Their findings and methodology are however more than adequate for other object classes. Presented qualitative results of processing random video inputs (<https://www.youtube.com/watch?v=pW6nZXeWlGM>) seem very convincing, even though the presented method does not support video processing explicitly in its architecture as it operates in a state-less fashion and no consistency is enforced between video frames. Due to lack of consistency enforcement, inter-frame

object tracking (when desired) needs to be resolved using external processing. Authors use deep convolutional neural network working as a 8-step downsampling-encoder (with natural input size of  $368 \times 368 \times 3$ ) with multiple deep dense (using skip connections) refinement steps operating on the downsampled, deep features ( $46 \times 46 \times 128$ ). The final output is then produced by up-sampling back to  $368 \times 368 \times 3$  and post-processed (Figure 2.13 (d, e)).

For human pose, position of the keypoints such as wrists, knees, ankles, elbows etc. are encoded using real-valued images representing heatmap of confidence that a keypoint of certain type exists at a particular location of the image. For each type, a number of two dimensional Gaussian kernels centered at the locations of the keypoints and with a pre-assumed variance (spread factor) is drawn on the image corresponding to the keypoint's type. Those images are then stacked together using concatenation along the last index forming a multi-channel composite image (Figure 2.13 (b)).

The model is trained in an end-to-end fashion to produce detection heatmaps. The joints such as arms, legs and connections between keypoints located on the head are encoded using 2-dimensional (2 image channels per link) direction vectors defined on an image plane normalized to unit length and drawn as a thick line (Figure 2.13 (c)). The method does not explicitly encode instance-level embeddings but lays the ground work for simultaneous multiple-instance processing due to clever encoding. If however, instances are subjected to complete occlusion of joints, it may be difficult to recover their accurate placement. Although the method estimates the class-level foreground mask for background subtraction, individual instance-level masks are not explicitly produced but rather approximated in post-processing using features representing part associations.

A promising extension to this method has been applied to smaller body parts - namely hand tracking. Simon et al.<sup>114</sup> extends the capabilities of *OpenPose* by creating a separate model capable of recovering position of finger joints and links between them. The methodology and training are very similar to work on full body pose,<sup>65</sup> with the major

difference being the target class - a pair of human hands.

Due to the need of appropriate input handling and gesture recognition for augmented and virtual reality applications, hand tracking became a topic of interest - especially when approached in three dimensions. Taylor et al.<sup>115</sup> present a computationally efficient and accurate / robust method of tracking deformable model with 28 degrees of freedom in the application of hand tracking for virtual and mixed reality interaction using depth sensing. They use smooth model inferred through loop subdivision<sup>116</sup> of the triangulated reference mesh to formulate a non-linear global energy-based optimization problem which can be tackled using Levenberg method.<sup>115</sup> Authors indicate limitations present in previously used, single RGB camera-based methods mostly focused on inverse kinematics labeling them as inaccurate, thus reinforcing the use of depth information.

So far the bottom-up approaches consisted of a two-stage process. First, the images are fed to a (fully) convolutional neural network producing the keypoint heatmaps and potentially images with features allowing for keypoint association cost estimation - such as part affinity fields. Then, those dense, image-like representations are transformed into sparse, real-valued representation and an association problem is formulated and solved. Parameters like detection thresholds or size of detection window need to be put in place to constrain the problem and ensure reasonable outputs which renders those methods as very flexible, yet requiring careful parameter tuning.

In work commonly known as *PersonLab*<sup>117</sup> the authors reinterpret the idea of *Part Affinity Fields* and differentiate between the short-range, mid-range, and long-range offset vectors to produce more robust input to the post-processing stages. They additionally generate instance-level-consistent embedding vectors and segmentation (foreground / background) mask. Their publication along with *OpenPose* are the main inspiration of this work.

Mid-range (pairwise) offset vectors correspond to the *Part Affinity Fields* in *OpenPose*.

It is important to emphasize the fact that both, the mid-range and long-range offsets heavily depend on the model's ability to aggregate information over larger area of the image and will depend on the size of the *receptive field*. The short-range vectors however allow for flexibility of model selection due to their locality and make the method potentially suitable for operation with front-end networks with limited receptive fields.

Short-range vectors are drawn as discs located around each corresponding keypoint  $(x_0, y_0)$  and encoded on appropriate ground-truth image channels as pixel-coordinate distance vectors  $S_k(x, y, x_0, y_0) = [x_0, y_0] - [x, y]$  is represented as a 2-dimensional vector for each pixel within the disc (each point  $(x, y)$  that lies within the defined radius). Each of this pixels is then encoded as a two-channel image, where the first channel contains the values of the  $x$  component and the second contains the values of  $y$ . This information allows for locating the keypoint based even if the very center of it is (partially) occluded and allows for more robust position estimation. In contrast to *OpenPose*, where the keypoint position would have to be estimated using information from potentially spatially distant elements of the image, here the more local information is explicitly trained (authors only train in 32-pixel radius around the keypoint).

Long-range offsets indicate the position of each keypoint from (virtually) every pixel belonging to particular instance (indicated by the red lines in the middle of figure 2.14). This information is rarely present in the data sets used in this work and thus the most problematic. When working with actively developed and constantly expanded datasets such as COCO,<sup>8</sup> one has the comfort of using vast instance-level segmentation data. Thus, methods like OpenPose or PersonLab achieve very convincing results when visually inspected.

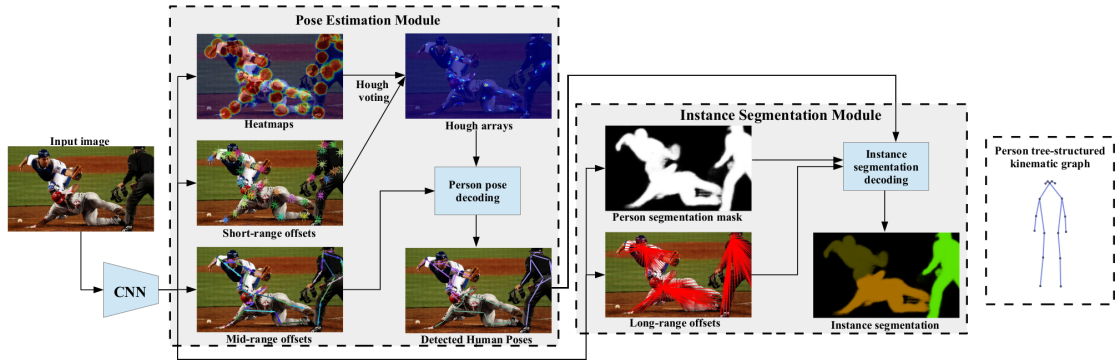


Figure 2.14: *PersonLab* processing pipeline.<sup>117</sup>

## 2.6 Embeddings

Papandreou et al. proposed a fully supervised method of resolving pixel-level instance membership for human tracking using well annotated datasets.<sup>117</sup> We consider their work an important milestone as it simultaneously addresses the human pose estimation and instance segmentation. As an alternative to linking keypoints using explicitly defined association (such as part affinity fields) for pose estimation, one can think about using dense, unsupervised features. Such approach however requires suitable, multi-dimensional dense instance representations. When produced in a per-pixel fashion, those representations are often referred to as pixel *embeddings*. This approaches is fairly new in MOT and it is not clear, what those embeddings should represent or consist of in terms of physical variables. What is known however is the fact that given two clusters, the values need to be consistent within the clusters but allow for easy differentiation between them. An engineer would be tempted to inject *hand-crafted* features like orientation parameters of the object, texture features,<sup>118</sup> position in the image etc. and train a model to properly estimate them. Convolutional Neural Networks are however capable of simultaneously learning suitable representations (and potentially solving the clustering problem) implicitly.

The use of embeddings for instance labeling was prominently featured in the

FaceNet algorithm which used a novel loss function to automatically generate unique embeddings for an individual's face — regardless of the conditions surrounding image capture.<sup>119</sup> This loss function uses three inputs simultaneously: two of the same person (the *anchor* and the *positive*), as well as a third image of a different person (the *negative*) and produces the distance value between the anchor and positive embeddings to be less than the distance between anchor and negative. This concept is referred to as the *triplet loss* due to the number of inputs, and its use for the training of a neural network is the main novelty of FaceNet.

To extend this concept to instance segmentation, each pixel is assigned an embedding such that, when clustered together, pixels in the same cluster also belong to the same instance. Figure 2.12(d) represents the ideal output of pixel embedding, where the “red” embeddings correspond to the foreground object and the “blue” embeddings correspond with the occluded object.

Fathi et al.<sup>105</sup> adopts this principle by training a network to evaluate pairwise pixel similarity. With a brute-force approach, the amount of comparisons quickly becomes unmanageable; to mitigate this issue, they train a separate model to generate seed points that represent the typicality of a pixel compared to other pixels in the area. Each seed point then generates a mask using the embedding vectors but training two models separately does not allow for joint optimization.

Formulation of the loss function and input encoding for embeddings can be challenging.<sup>119</sup> Also, when evaluating using external clustering steps one needs to resort to reinforcement learning which is a powerful but time-consuming process as the loss function gradients are propagated from a single evaluation output, which would not be a big issue if those outputs could be evaluated quickly.

This, pixel-pair-wise approach was criticized by Papandreou et al.<sup>117</sup> as sensitive to hyper parameters, and for producing hard to predict results as the output embeddings are not guaranteed to be separated enough for clustering. This criticism is very much

applicable in the context of the COCO dataset that authors used in their human-pose-targeted approach, as they had simultaneous access to the instance mask, foreground mask, and keypoint location during the training. This prior information is however not available in the context of this work as the keypoint-annotated images do not include corresponding instance masks. Their alternative encoding explores the geometric embeddings based on the relative offsets of the keypoints encoded directly in the image coordinate space which they acknowledge is a challenging task for a neural network to train for.

Training for tasks requiring larger spatial requires the model to “see” and aggregate information across substantial part of the input. Quantitatively, the measure of the size of this context is referred to as the *receptive field*. Larger receptive fields can be achieved either through downsampling / pooling, via very deep architecture, large convolutional kernels or the combinations of the above. When the feature map needs to be brought back to the size of the input image, downsampling introduces challenges related to proper spatial alignment. Training large kernels introduces additional computational complexity. Thus, the up-sampling is often realized by deep networks with small kernel sizes. Deep architectures can produce larger receptive fields at the risk of very long training times due to the nature of the backpropagation algorithm itself that applies the chain rule to calculate parameter updates at each step (layer). In a very deep networks the relationship between the output error and the input becomes progressively smaller when progressing *backwards* as most of the operations in the network are based on the inner product. Since the gradient of the output of a layer with respect to its input is equal to the linear combination of its coefficients, if those coefficients are kept at low magnitude, progressing deeper backwards leads to very small gradient updates. This problem was identified as one of the dominant difficulties in recurrent networks where it is referred to as the *vanishing gradient* problem.<sup>120</sup>

In the work of Chen et al., commonly known as Deeplab,<sup>121</sup> authors present a novel

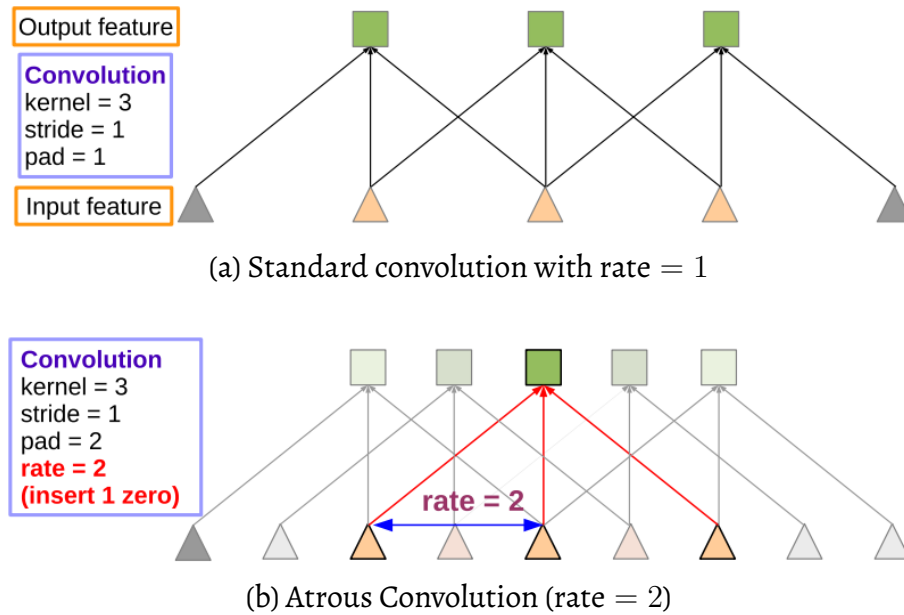


Figure 2.15: Standard (a), and Atrous Convolution (b). Figures from.<sup>121</sup>

concept of *Atrous Convolution* allowing for increasing the size of the receptive field by introducing “holes” (“à trous” in French) between the non-zero kernel coefficients. They argue that when processing large feature maps (high in resolution), the spatial context between neighboring pixels is most likely the same. In other words, in the context of semantic segmentation of images - pixels that are close together are most likely parts of the same object. The main trade-off of using Atrous Convolution is the loss in spatial accuracy. The difference between processing using standard convolution and Atrous Aonvolution is presented in Figure 2.15.

A very interesting alternative idea known as *Adversarial Training* has been introduced by Goodfellow for Generative Adversarial Networks (GANs).<sup>122</sup> Here, instead of the need for specifying an explicit loss function, one can resort to classification based on positive examples. In principle, GANs tackle two tasks simultaneously: 1) to perform the main task, and 2) to evaluate the quality of the produced output. It is done by introducing two networks: a *Generator* and a *Discriminator* trained in alternating fashion. First, a discriminator is trained to produce negative (or false) response for the generator’s output

and positive (true) for the ground truth images. Then, the generator is trained using gradients propagated from (fixed for this step) discriminator - enforcing positive response. This configuration renders the discriminator as a substitute for the task-specific loss evaluation block. GANs were widely adopted for the task of producing convincing synthetic images<sup>123, 124</sup> but their true potential lays in approximating source domain representations.<sup>125</sup> As tempting as their application is to an engineer, their training is known to be extremely difficult due to its alternating nature. Ideally, when successfully trained, components of GAN should converge to a Nash Equilibrium (state in which no weight update would improve the final result). There is however no guarantee of that being the case. The most common problem with GANs is *mode collapse* which exhibits itself via convergence to *always correct* but single generator's output..

Modern literature points out a multitude of difficulties related to the stability of GAN training and presented multitude of underlying problems and *tricks* allowing for predictable training.<sup>126</sup> Methods yielding the most visually pleasing synthetic images today employ the concepts of *Wassertein GAN*.<sup>127</sup> Authors indicate the problem of *manifold learning* and propose methodology allowing for more stable training that addresses the situation when the model produces the same output every time known as the *mode collapse*.

Kong et al.<sup>107</sup> introduce a method that maps pixels to unit-length embedding vectors on a hypersphere. For training, they randomly sample embeddings and use cosine similarity to measure the distance between the vectors. They also introduce recurrent Mean Shift<sup>128</sup> clustering into training, allowing the network to train end-to-end.

Brabandere et al.<sup>106</sup> introduce a method that does not require the embeddings to be on a hypersphere, thus relaxing the problem, while instead encouraging small-magnitude vectors via regularization. Their loss function encourages clustering without requiring the integration of mean shift in the training process, resulting in much faster training and easier implementation. The method presented in this work is directly

inspired by their work.

Learned, pixel-level embeddings encapsulate instances through similarity constraints, and potentially relieve the need for graphical post-processing. So far however, it has not been discussed how they behave in the presence of occlusions. Computer vision often aims to reverse-engineer scenes from images/video, and an assignment of all visible instance parts to a single membership is a useful but still incomplete descriptor. In contrast, the full segmentation masks and relative depth ordering prior to image projection provides a more complete input - especially when occlusion handling is of interest.

Instance segmentation extends the semantic segmentation by separating objects within the same class by identifying pixels belonging to each instance in the visible parts of the image, it does not however resolve the occluded regions. Even if the method is capable of producing correct identification masks for discontinuous regions, the shape of *occluded* parts are still unknown. Tackling this problem is referred to as *amodal segmentation*.<sup>129</sup> An illustration of such approach is presented in Figure 2.12(e), where the blue region is partially occluded by the red one, and appropriate segmentation masks are produced for the red region, the occluded part, and eventually the blue part. The main difficulty in this approach is the lack of data sets containing the proper annotations. Also, conceptually it can be difficult to assess the performance of the method, as depending on the pose, position, type of object, and amount of occlusion, it is hard to define what should be appropriate level of correctness of estimation of the occluded region. Conceptually, for humans, performing this task is intuitive, but it can be unquantifiable in the context outside of mere numerical evaluation.

Huang et al.<sup>130</sup> approach the segmentation problem via using a neural network-based deep feature extractor to produce cluster-able features for the k-means algorithm to process. They estimate the number of valid clusters using *silhouette scoring* method,<sup>131</sup> and then construct final clusters based on the best score. Cluster-based

approach of the method presented here resembles their clustering assessment methodology.

Amodal segmentation can be tackled using the concept of a *tri-state-mask*, a feature capable of indicating which pixels belong to either of the three states: the foreground, the background, and the intersection of the two that is occluded (and technically belongs to the background). This approach is limited to two levels of depth or *z-ordering*, and requires appropriately annotated ground truth segmentation masks. Again, the availability of the data set is the major limitation in this field. There are however approaches attempting to address this problem via creation of synthetically occluded regions as a composition of the existing annotated backgrounds, and instances extracted from other images, and *injected* at known locations with known tri-state masks - all based on the COCO dataset.<sup>132</sup> Unfortunately, this set contains annotations that highly vary in quality of annotations of the occluded regions, and available annotations are sparse compared to the size of the data set.

Milan et al.<sup>133</sup> in their work on multi-target tracking identify three types of occlusions:

- Inter-Object occlusions when objects of interest occlude each other,
- Scene occlusion such as caused by pillars, road signs, generally static objects in the scene,
- Self-occlusion - being very specific to the type of tracked object and involving extensive articulations, deformations, changes in orientation and other unusual transformations rendering object of interest hard to detect or track.

Yang et al. estimate layer ordering as part of instance segmentation and introduce a learned predictor based on relative detection scores, position on the ground plane, and size.<sup>134, 135</sup> They acknowledge the benefits of full spatial segmentations of visible and occluded parts, but their method focuses on the benefits of depth ordering for instance

grouping. Chen et al. attempt to fill occluded regions by selecting similar non-occluded exemplar templates from a library;<sup>136</sup> this improves instance segmentation of visible pixels.

Instance segmentation with explicit depth ordering estimation was proposed by Uhrig et al.<sup>137</sup> Their method exploits ground truth depth information provided by KITTI<sup>138</sup> and Cityscapes,<sup>10</sup> but it does not attempt to recover occluded segments. While each of these methods uses the concept of occlusions to improve instance segmentation, none of them explicitly targets the full spatial extents and depth ordering of instances. To this day, no dataset with explicit z-ordering for animal instance segmentation exists, but certain efforts can be made to limit the negative effects of occlusions such as: randomization of ground-truth instance generation and explicit cluster consistency enforcement even in the presence of partial occlusions.

Li et al.<sup>139</sup> investigate the concept of embedding stability in the video while training the model on static images. In the context of video, authors argue that foreground-background determination needs to be performed using temporal context rather than based on single frame. This improves model transferability as the network does not have to be fine-tuned for specific images.

## 2.7 Convolutional processing of images

In computer graphics images are represented as arrays of pixels such that an image can be defined by an  $m \times n$  (image height and width respectively) matrix with values corresponding to the encoding format of the color. The rectangular grid of pixels with assigned colors is referred to as *raster*. This will be the kind of images considered in this work - as opposed to the *vector* representation, which defines a sequence of mathematical operations to generate the visual shapes and colors.

For the sake of clarity, when describing raster images, a common standard to

represent monochromatic images in the black  $\rightarrow$  white spectrum is to use the 8-bit greyscale quantization (each value represents the intensity), in case of color, a 24-bit (3 bytes) color palette is a common representation with RGB order of red, green, and blue intensities.

When preparing images to being ingested by a neural network one will most likely find themselves following the standardized representation from the popular machine learning frameworks such as Caffe<sup>4</sup> and TensorFlow.<sup>5</sup> In this work the  $[B \times H \times W \times C]$  convention will be used to represent the images, where  $B$  is the number of examples fed to the neural network at each iteration (also referred to as the *batch size*),  $H$  is the height of each image in the batch in pixels,  $W$  is the image width in pixels,  $C$  is the number of channels (1 for monochromatic images, 3 for RGB images, or  $C$  for a result of concatenation of multiple feature maps along the last index). Each value is encoded using 32-bit floating-point number as both, the CPU and GPU-based operations are optimized to use this format - that includes the binary masks containing the extreme values of 0.0 and 1.0 only.

As witnessed in the literature, currently adopted conventions in the field of machine learning for computer vision have their roots in the shift of methodology which began around year 2012. That year the deep architecture proposed by Alex Krizhevsky (known as AlexNet<sup>2</sup>) outperformed most hand-crafted methods in multiple tasks of the ImageNet Large Scale Image Recognition Challenge.<sup>6</sup> Unlike previously used methods, Krizhevsky used a GPU and built upon the work originally presented by Le Cunn in the late 1980s.<sup>3</sup> His model architecture (Figure 2.17), and utilization of graphic processors became a standard of the modern architecture and consisted of various building blocks which are worth emphasizing:

- Dual-headed design - due to the limits of single-GPU memory, the network was split into two paths (*heads*).
- Use of convolutional layers allowed efficient parameter utilization.

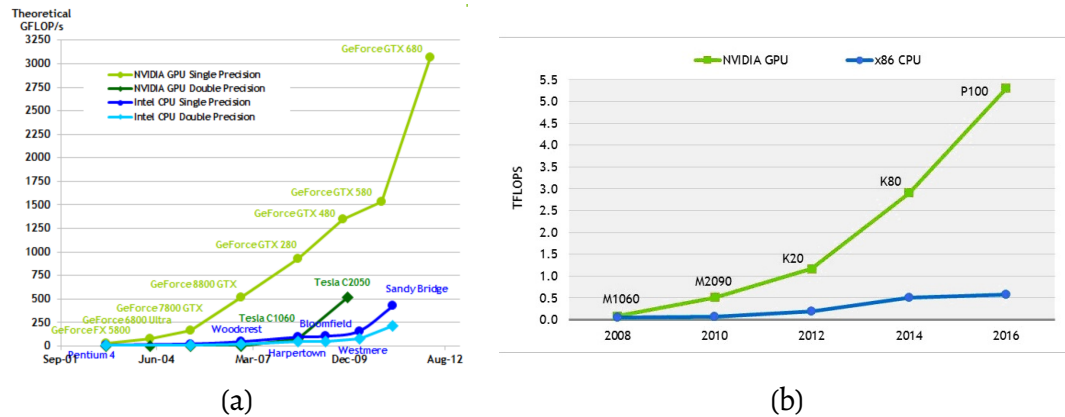


Figure 2.16: Rise of computational power of graphic cards between 2002 - 2012 (figure from<sup>140</sup>) (a) and their professional counterparts between 2008 - 2016 (figure from<sup>141</sup>) (b) when compared to a CPU performance.

- Use of Max Pooling layers (although heavily criticized by Geoffrey Hinton) allowed for robustness against small-scale image translations and larger spatial spanning.
- Use of multiple inner product (dense) layers towards the output of the network provided larger spatial context for the output via aggregating information produced by localized convolution operations.
- Reduced width and height compensated by increased *depth* allowed for creation of rich *dense features* on each level of processing.

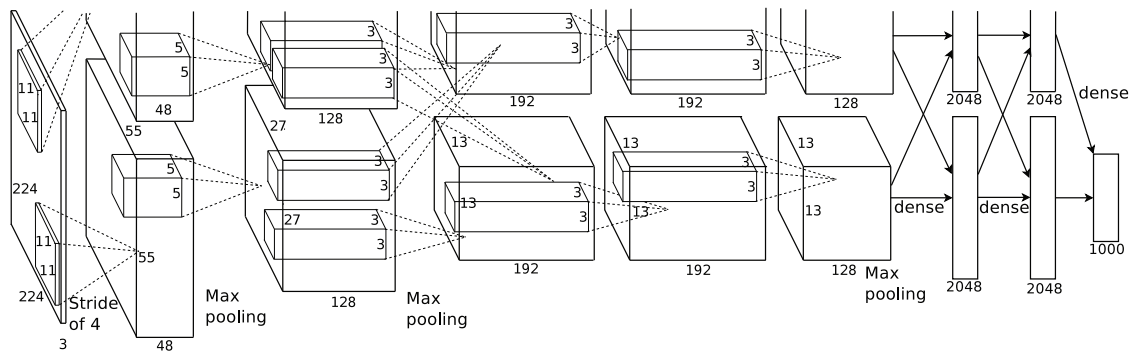


Figure 2.17: AlexNet<sup>2</sup> deep convolutional neural network architecture proposed in 2012.

Following the very useful summary by MathWorks,<sup>151</sup> one can characterize commonly used neural networks in the context accuracy - how well do they perform the

Year	Network	Depth	Size	Parameters (Millions)	Image Input Size
2012	alexnet <sup>2</sup>	8	227 MB	61.0	227 × 227
2014	vgg16 <sup>142</sup>	16	515 MB	138	224 × 224
2014	vgg19 <sup>142</sup>	19	535 MB	144	224 × 224
2015	googlenet <sup>143</sup>	22	27 MB	7.0	224 × 224
2015	inceptionv3 <sup>143</sup>	48	89 MB	23.9	299 × 299
2016	squeezenet <sup>144</sup>	18	4.6 MB	1.24	227 × 227
2017	densenet201 <sup>145</sup>	201	77 MB	20.0	224 × 224
2017	xception <sup>146</sup>	71	85 MB	22.9	299 × 299
2018	mobilenetv2 <sup>147</sup>	53	13 MB	3.5	224 × 224
2016	resnet18 <sup>148</sup>	18	44 MB	11.7	224 × 224
2016	resnet50 <sup>148</sup>	50	96 MB	25.6	224 × 224
2016	resnet101 <sup>148</sup>	101	167 MB	44.6	224 × 224
2017	inceptionresnetv2 <sup>149</sup>	164	209 MB	55.9	299 × 299
2018	shufflenet <sup>150</sup>	50	6.3 MB	1.4	224 × 224

Table 2.1: Depth, size, number of parameters, and image input size glossary of popular neural networks used for image classification tasks.<sup>151</sup>

task they were designed for, speed - relatively, how fast is the training / inference, and size - how many coefficients does the model consist of.



Figure 2.18: Illustration of data-driven low-level feature extraction: 96 convolutional kernels of size 11x11x3 learned by the first convolutional layer of AlexNet for ILSVRC2012.<sup>2</sup>

Transfer Learning is a term referring to taking advantage of model's performance to solve given task  $A$  and (often after fine tuning) making it applicable for another task  $B$ .

In the context of computer vision problems, this can be very well explained by example of Imagenet Large Scale Visual Recognition Challenge (ILSVRC),<sup>6</sup> where features learned by

neural networks on large-scale image datasets for the task of whole image *classification* were successfully used in *segmentation*. Even in the early days of deep learning, it was observed, that deep classifiers, due to their convolutional architecture, learn to decompose images into more-and-more complex structures in a bottom-up fashion. A great example of such decomposition can be illustrated when observing the first layer of convolutional kernels learned by AlexNet presented in figure 2.18. This observation allowed researchers to use pre-trained models as deep feature extractors. Long et al.<sup>109</sup> propose Fully Convolutional Networks (FCN) by converting ImageNet-trained high-performance, general purpose classifiers. Their approach became widely adopted and fully convolutional models are widely used due to their adaptation to variable image size. ImageNet competitions also had an impact on the research community in the form of defining *standard* network architectures. In the context of transfer learning, the *standardization* falls even beyond mere architecture, the entire, pre-trained networks are often used as building blocks for more complex models - most often as a source of multi-dimensional, per-pixel defined feature vectors. This work uses transfer learning in exactly that sense - both the benefit of proven, working architecture and initial values of model coefficients are *borrowed* from popular models.

Following the very useful summary by MathWorks,<sup>151</sup> one can characterize commonly used neural networks in the context accuracy - how well do they perform the task they were designed for, speed - relatively, how fast is the training / inference, and size - how many coefficients does the model consist of. Figure 2.19 illustrates accuracy as a function of complexity of the model expressed as time complexity.

When looking at the commonly used models listed in table 2.1, one needs to realize their depth and scale. Due to the high number of trained coefficients and overall number of layers, training such models is nowadays referred to as *Deep Learning*. Use of such vast models - as powerful as it is - always carries the risk of over-fitting as the data could be *stored* as model weights or biases and *recalled* upon the input. Various techniques exist to

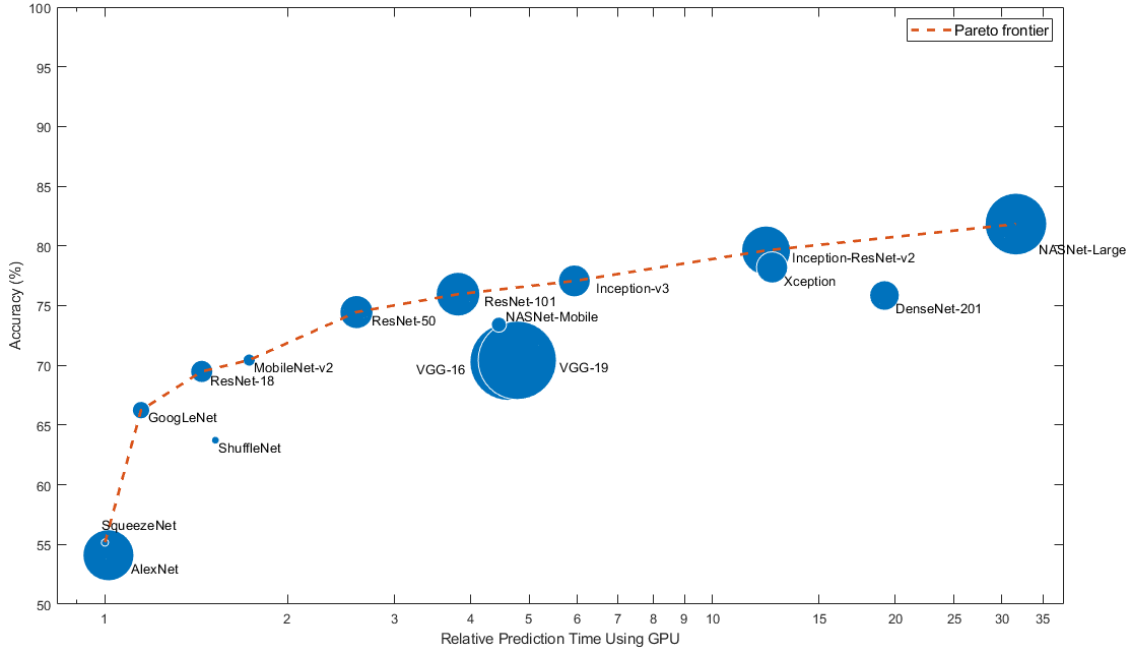


Figure 2.19: Relative Pareto efficiency of neural network models commonly used in transfer learning.<sup>151</sup>

overcome this problem, but two deserve to be emphasized: *content-preserving transformation* (augmentation) and *weight regularization*.

Broadly speaking, regularization is a term referring to imposing constraints on the behavior of the optimization solver during the weight adjustment. Whereas augmentations are introduced in the input pre-processing stages, regularizations can be included explicitly as a part of the training loss function or as explicit operations. One of the most popular regularization method used when training large CNNs is to constrain magnitude of the weights using  $L^2$ -norm penalty applied to all (or at least selected subsets of) trainable coefficients - known widely as *weight decay*. It is defined by augmenting the loss function  $L(y, y', \theta)$  as follows:

$$L(y, y', \theta)' = L(y, y', \theta) + \alpha_{\text{reg}} \frac{1}{N} \sum_{n=0}^{N-1} \|\theta_n\|, \quad (2.1)$$

where  $y$  is the desired output,  $y'$  is the model prediction,  $n \in \{0, \dots, N-1\}$  is the index

of the model coefficient,  $N$  is the number of trainable coefficients,  $\theta$ s are their values, and  $\alpha_{\text{reg}}$  is the weight decay coefficient.

**Example 1.** Let's consider a simple case of a line crossing the origin on a two-dimensional plane.

$$y' = ax, \quad (2.2)$$

and define a loss function that could be used to minimize the mean squared error of the model parametrized by single parameter  $\theta_0 = a$ , using  $M$  number of example pairs  $(x_m, y_m)$ :

$$L(y, y', \theta) = \frac{1}{M} \sum_{m=0}^{M-1} (y_m - \theta_0 x_m)^2, \quad (2.3)$$

Then, the aforementioned regularized version of the loss function would take the form of:

$$L(y, y', \theta)' = \frac{1}{M} \sum_{m=0}^{M-1} (y_m - \theta_0 x_m)^2 + \alpha_{\text{reg}} \|\theta\|, \quad (2.4)$$

When applied to the training of neural networks, this technique effectively forces the network to try to generalize intermediate representations and prevents it from creating easily-exploitable separation boundaries that would allow the network to store the training examples using bias coefficients. Although providing numerical stability, regularization can lead to the vanishing gradient problem due to its sensitivity to the hyper-parameters  $\alpha_{\text{reg}}$ . Historically this issue was tackled by heuristic estimation of additional learning rate multipliers based on the network architecture i.e. the number of layers and number of parameters at each layer. Nowadays, the optimization algorithms such as Adam attempt to maintain adequate multipliers to ensure gradient propagation.<sup>152</sup> when using standard architectures the literature often indicates standard weight decay values within the range of  $(10^{-4}, 10^{-2})$  for the multiplicative coefficients, and no penalty for the additive ones.

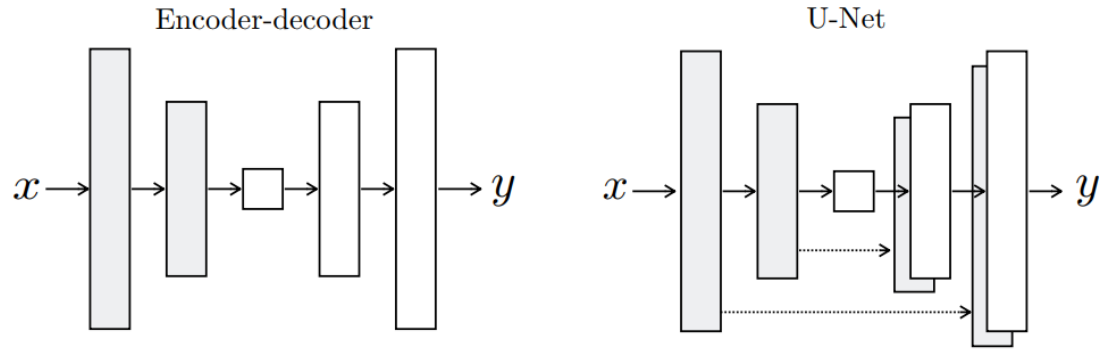


Figure 2.20: Two common model architectures: Encoder-Decoder architecture (left) and UNET (right). Figure from the work of Isola et al.<sup>153</sup>

Alongside the AlexNet-like networks striving to excel in ImageNet's *classification task*, more challenging problems such as *segmentation* or image-to-image require outputs values corresponding to every pixel in the input. In order to address it, a general *Encoder-Decoder* architecture was introduced (Figure 2.20). Such networks consist of two parts: encoder and decoder that can work independently but are trained jointly.

The role of an **encoder** is to extract features from the input images, mapping it from the color space to the feature space. This process involves multiple, subsequent two-dimensional convolution followed by sub-sampling. While the width and height of the subsequent representations decreases along the way, the model architecture compensates for it by increasing the number of channels. As a result, the outputs of the encoder, i.e. the *encoded* representation is a relatively small (a fraction of the area of the original image) but deep (high number of channels) feature map that is intended to aggregate the information across the image. The last set of operations in the encoder is often referred to as *the bottleneck layer*. The literature often loosely uses the term *compression* to describe the operation of the encoder as a means to carry across the gist of its function without much regard to the strict definition of the word.

The **decoder** addresses the mapping between the feature domain and image domain and produces the output with the same dimensions as the original input. To do so, another set of subsequent operations is employed, comprising of the convolutional

processing of the previous features, and an up-sampling operation. It is important to note the two ways of how the order and realization of those operations is performed in the field<sup>109, 154, 155</sup> as there are two commonly adopted building blocks of the decoder networks:

- Bilinear up-Sampling followed by convolution as used by Dosovitskiy et al.<sup>154</sup>  
Requires two steps but does not require large up-sampling kernel allocation,
- Convolution with fractional stride (also known as Convolution Transpose, or wrongly “deconvolution”),<sup>109</sup> immediately creates the desired number of channels but is known to generate checkerboard-like artifacts.<sup>156</sup>

Model transfer and fully convolutional processing began with the work presented in.<sup>109</sup> Proposed Fully Convolutional Network (FCN) architecture was trained for the task of foreground segmentation but proposed method of adopting a ImageNet-trained, fixed input sized classifier (particularly VGG-16<sup>142</sup>) for the segmentation task made FCN a very impactful contribution. The conversion was based on the observation, that the activations of the fully-connected layers form a pattern which can be exploited to create convolutional kernels. After that, a trainable deconvolution-based up-sampling is introduced to bring the output to the original input dimensions and solve the specific task. The number of channels in the final stage corresponds to the task for which such converted, now fully-convolutional network is designed.

An FCN is effectively a fundamental example of an encoder-decoder-style segmentation network. Its deepest version (FCN8) combines features spanning over large spatial extent due to the use of feature maps produced at  $\frac{1}{32}$ ,  $\frac{1}{16}$ , and  $\frac{1}{8}$  image sizes. Figure 2.21 depicts the model architecture and the features corresponding to each scale. The importance of accumulating features among different scales was pointed out by Maninis et al.<sup>157</sup> and inspired us to use of similar model.

To tackle problems related to gradient propagation during training at the level of network topology, architectures such as SegNet<sup>158</sup> or U-NET<sup>159</sup> introduce the concept of

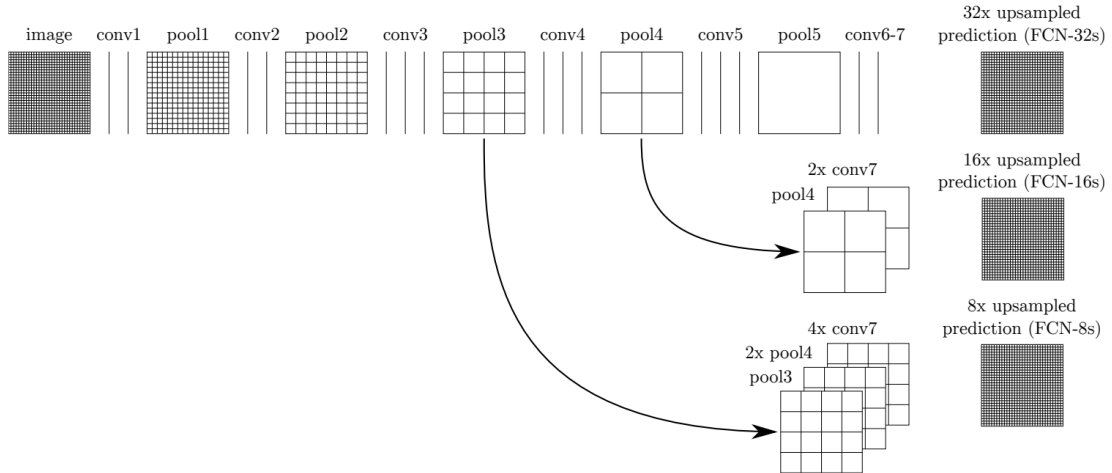


Figure 2.21: Fully Convolutional Network obtained from converting the inner product operations to convolutions.<sup>109</sup> Three spatial blocks of the FCN architectures: FCN32s, FCN16s, and FCN8s.

skip connections. Those architectures learn faster than traditional Encoder-Decoder architectures due to the presence of direct gradient propagation paths. The decoder however cannot operate independently in those architectures based entirely on the deepest feature map as the intermediate inputs are lost during encoding process outside of training.

U-NET is a tightly coupled, deep pair of encoder-decoder stacks with skip connections between the corresponding down and up-sampling steps (Figure 2.20), originally presented by Ronneberger et al.<sup>159</sup> The connections are realized using channel-wise concatenation. In its standard implementation presented in<sup>153</sup> it uses leaky ReLUs on the encoder side and regular ReLUs in the decoder. The downsampling is done using strided convolution (as oppose to pooling layers), and upsampling is realized using convolution with fractional strides (a.k.a. convolution-transpose). The main advantage of this architecture is quick gradient propagation through skip connections to the encoder layer but at the same time, due to tight coupling the decoder cannot be used outside of the network architecture as it relies on the data from the encoder. The success of the U-NET architecture encouraged us to to explore the symmetric hourglass-shaped networks with

skip connection in our previous contributions<sup>160</sup> as well as the work presented here.

Our work aims to leverage presented methods in the application to homogeneous object tracking - particularly pigs. As opposed to the general-purpose scene segmentation methods, it aims to solve a slightly different problem of locating, determining the orientation of, and differentiating the instances that not only belong to the same class and look very much alike, but also, often occupy the same space in the image due to piling.

Review of the literature reveals that the object detection is the essential component of the tracking by detection approach to multiple object tracking. Understanding that representing the tracked objects merely using bounding boxes is not sufficient in applications prone to occlusions due to the spatial overlap. Thus, we decided to propose the object representation based on semantic instance segmentation mask instead. Additionally, characterizing our application as an attempt to distinguish multiple *homogeneous* objects within the same frame, we identify that object identification using mere class-membership is not descriptive enough. Thus, in order to resolve the instance membership for each pixel within the mask we propose to use embeddings and a clustering algorithm. We start with an object detection method based on pose estimation due to the limited availability of the data for our domain and move towards semantic instance segmentation. We identify that processing images using convolutional neural networks has the benefit of simultaneous optimization for a specific task and learning the representation of the features. We use two different neural network architectures motivated by the literature and our prior experience. In order to optimize the weights of the neural networks we use two loss functions specifically designed for semantic instance segmentation and train our networks for multiple tasks simultaneously. We propose an extension of the single-frame case to a sequence of frames using augmentations along with a modified version of the loss function for embedding consistency across frames.

## 2.8 Problem Statement

Pursuits of applied engineering work in the domain of animal behavior monitoring seem justified by the needs of the industry as automated and unobtrusive solutions for animal tracking are needed and have the potential of addressing the insufficient staff problem as well as provide more detailed oversight of the operations and animal welfare as described in section 2.1.

Due to the genotypical and phenotypical similarity of group-housed pigs in production facilities, visual tracking in such scenarios can be interpreted as **Homogeneous** Object Tracking and tackled accordingly. The difficulties of processing objects hard to distinguish from one another may be however balanced by exploiting their elongated shapes.

This work follows extends on our pig tracking method based on pose estimation,<sup>160</sup> i.e. estimation of location and relative placement of keypoints to determine position and orientation of tracked instances. The extension involves the task of semantic segmentation of pigs. High accuracy and robustness however are related to the quality and availability of labor-intensive human annotations. When compared to more *popular* domains, animal tracking datasets are sparse and incomplete, thus imposing a **missing data** problem.

Both tracking-by-detection, and pose estimation methods need to solve the data association problem of joining parts of the same instance within a single image (video frame). Thus, requiring a **representation** and **similarity metric** for evaluation. Explicitly defined representation generation optimized in the supervised learning regime was successfully applied to the animal tracking, semi-supervised, implicit methods are often approach with very high dose of skepticism with no further exploration.<sup>117</sup>

Work shown in<sup>106</sup> provides an example of discriminatively trained model producing pixel-level representations (**embeddings**) allowing for membership determination and

clustering of instances within a single image. Their method relies on a formulation of a heuristic loss function with adjustable margins but authors do not address the analogy to the **silhouette score** commonly used in clustering assessment.

To our knowledge, an in-network, pixel-level semantic instance segmentation for pig tracking using color images has not been tackled effectively yet. As opposed to more popular problems, no dataset containing foreground or instance-level masks exists for this application.

## CHAPTER 3

---

### Method

This chapter is dedicated to a bottom-up approach of processing large scale multi-domain image data sets using modern computer vision and machine learning techniques for the task of multiple object tracking. Specifically, we are addressing relatively easy to obtain long sequences of unannotated video captured from a static overhead-viewing camera. We are presenting a novel method for automating the process of generating high-quality annotations from relatively sparse dataset of static frames with annotated keypoints. No target identity persistence among the images is ensured outside of indication of associations between keypoints and instances within each image. The dataset collected and presented here contains images of group-housed pigs but the method is directly applicable to any objects that can be geometrically defined using keypoints. The properties of the selected animals (pigs) make the method specifically attractive for practical applications due to the visually-homogeneous nature of the tracked instances.

Keeping in mind practical deployments in barns, relying on depth information cannot always be assured due to the equipment costs. As for the inexpensive sensors, the field of view required for a production facility deployment extends beyond the capabilities of Kinect v2 used in this work. Long-range infrared depth sensors allowing for reliable mapping of distances greater than 8 meters are still unjustifiably expensive. Recent work on human pose estimation also indicates that depth information may not

even be necessary for reliable bottom-up tracking.<sup>65,113,117</sup> Thus, we shifted the focus toward tracking in 2D using deep neural networks. It is worth mentioning, that conversion from two-dimensional pixel coordinates to three dimensional can still be performed (to some extent) as long as it is possible to estimate the floor plane and camera is static. Our method relies on both of these assumptions.

We chose to assume a down-facing, static camera setup for our work on MOT for group-housed pigs. This setting has two key advantages. First, animals are usually not occluded from the top-down perspective unless they are crawling over or piling on one another for heat preservation. Second, the size of the animals stays relatively constant with respect to position and orientation in the image. Thirdly, with the top-down view, the artifacts can be easily mitigated if a reprojection to and from three-dimensional coordinate system is desired. The ability to align the depth information with their color counterparts could be successfully used in the context of this work due to that ability.

The main component of a 2D-tracking method based on pose estimation is the deep, convolutional neural network producing outputs allowing for keypoint position and instance orientation estimation. We are exploring two different model architectures. First one resembles the successful model for real time human pose estimation by Cao et al. and was re-implemented mostly for the purpose of reference as its substantial depth was proven to provide high performance.<sup>161</sup> Second one is a U-NET model, similar to our previous work<sup>160</sup> but differently implemented skip-connections and different depth. The second model was anticipated to produce more consistent results and be more stable during the training stage.

It was already indicated in<sup>117</sup> that training separate models for each tasks yields *better* per-task results rather than using a single, joint model. Here however, multi-task models are presented. This choice is motivated by the memory limitations of consumer-grade GPUs. A cost-effective, small form factor on-site computer would most likely use such GPU for near real-time operation.

Section 3.1 describes the equipment, format and process of our data collection with the emphasis of the recently published<sup>160</sup> publicly available subset that has been carefully annotated using keypoints to indicate the position and orientation of each target.

In Section 3.2 we familiarize the reader with the convention used to represent the images in the context of processing using neural networks and common image processing libraries.

Sections 3.3 and 3.6 describe the keypoint and part association representations (respectively). They are specific but not limited to the task of pose estimation like in the work of Cao et al.<sup>161</sup> We present the task-specific small evaluation datasets produced in the respective subsections as well.

Our extension to the pose estimation method by incorporation of the embeddings begins with the representation of per-pixel instance membership described in Section 3.4.

In Section 3.7 we briefly introduce image augmentations as a method of increasing diversity of the training data for the purpose of increasing model's ability to generalize. We describe both model architectures used in this work: OP and UNET in Section 3.8.

Section 3.9 addresses the main topic of this work - the deep multi dimensional embeddings trained under weak supervision. We explain the concepts of *cohesion* and *separation* of clusters and attempt to tackle the problem of training a neural network for their maximization in an end-to-end fashion. To do so we present two loss functions  $L_{\text{margin}}$  and  $L_{\text{ssmax}}$  for training and analyze their relative speed of convergence with respect to the number of instances and number of encoding channels.

In Section 3.10 we describe conditions and the procedure used to train our deep neural networks.

Section 3.11 describes the pose estimation method we used in our work to produce object descriptors for the tracked targets (pigs). In Section 3.11.4 we propose changes that can be made to the baseline method to improve part-to-instance membership matching.

Finally in Section 3.12 we describe our approach to semantic instance segmentation for homogeneous targets based on processing of the multi-dimensional embeddings.

### 3.1 (Big) Data collection

Modern consumer-grade real-time depth cameras can serve as the backbone of an enhanced visual tracking system.<sup>162</sup> A popular choice for such a camera is the Kinect v2 gaming peripheral developed by Microsoft to track human movement. The Kinect v2 comes equipped with a high-definition color image sensor, an infrared illuminator, and a time of flight depth sensor that produces color, infrared, and depth frames, as illustrated in Figure 3.1. In addition to facilitating depth measurement, the infrared illuminator makes it capable of tracking day and night without the need for visible light. Due to its low cost and high availability Kinect v2 became a capture platform for majority of the data sets used in this work. The main weakness of the Kinect v2 platform is its limited to about 7 meters depth sensing range. This drawback did not impose many challenges during the collection of the data for this work but it does render the platform insufficient for static deployments in large commercial barns. Vast data sets containing color, infrared, and depth image frames listed in Table 3.2 were collected using custom-designed software / hardware solution based around Intel NUC low-footprint computer running Windows operating system.

A multi-threaded application was developed in C++ to ingest images captured by the camera at a rate of 30 Frames Per Second (FPS) and based on the amount of motion estimated when compared to the the previous frame, the frame was either discarded or stored. Such approach allowed for continuous operation in remote locations which do not provide the Internet connectivity as the files were stored on high-capacity hard drives and swapped when full.

Each file was stored in its respective folder structure build using chronological

	Color	Infrared	Depth
Resolution:	$1920 \times 1080$	$512 \times 424$	$512 \times 424$
Encoding:	JPG	JPG	16-bit PNG
Typical image size:	$\approx 350kb$	$\approx 30kb$	$\approx 200kb$
Number of channels:	3	1	1
Data type:	RGB	Monochromatic	Distance from camera [mm]

Table 3.1: Image formats of our large scale data sets captured using Microsoft Kinect v2.



Figure 3.1: Color, infrared, and depth frames captured by the Kinect v2 camera.

hierarchy such as: 'YEAR/MONTH/DAY/HOUR/MINUTE/' with file named using 'YEAR\_MONTH\_DAY\_HOUR\_MINUTE\_SECOND\_MILLISECOND.EXTENSION' pattern. This convention is mostly consistent throughout this work and the data sets used for training are named after date and time of the first frame occurring in the set (as shown in Table: 3.2).

Since the files were stored directly on a filesystem, an additional index database was created at the end of deployment for easy traversal using date and time-based indices or frame numbers without the need of recursive directory search. It consisted of a single table storing the core of the file name, date extracted from the name, and the frame number in the dataset. The total superset of all images used in this work is listed in Table 3.2.

Name	Dates	# Samples	Data type	Creation
2017-03-07-17-16-25	Mar 7 2017 - Mar 23 2017	4, 329, 965	Color, Depth, IR	Automatic
2017-03-07-17-09-59	Mar 7 2017 - Mar 23 2017	4, 268, 908	Color, Depth, IR	Automatic
2017-04-19-21-30-25	Apr 19 2017 - Apr 28 2018	2, 967, 837	Color, Depth, IR	Automatic
2017-04-06-15-53-13	Apr 06 2017 - Apr 19 2017	4, 561, 632	Color, Depth, IR	Automatic
2017-05-05-21-51-54	May 05 2017 - May 14 2017	863, 904	Color, Depth, IR	Automatic
2017-05-18-20-43-28	May 18 2017 - May 31 2017	969, 933	Color, Depth, IR	Automatic
2017-05-18-20-48-17	May 18 2017 - May 31 2017	2, 762, 064	Color, Depth, IR	Automatic
2017-05-18-20-43-28	May 18 2017 - May 31 2017	969, 933	Color, Depth, IR	Automatic
2017-05-18-20-55-21	May 18 2017 - May 31 2017	2, 035, 596	Color, Depth, IR	Automatic
2017-06-01-15-54-51	Jun 01 2017 - Jun 09 2017	1, 583, 260	Color, Depth, IR	Automatic
2017-06-01-16-03-15	Jun 01 2017 - Jun 09 2017	626, 182	Color, Depth, IR	Automatic
2017-06-01-16-10-41	Jun 01 2017 - Jun 09 2017	1, 084, 931	Color, Depth, IR	Automatic
2017-07-10-21-35-15	Jul 10 2017 - Jul 28 2017	2, 992, 284	Color, Depth, IR	Automatic
2017-07-10-21-47-45	Jul 10 2017 - Aug 14 2017	2, 650, 732	Color, Depth, IR	Automatic
PIGS500	Oct 2016 - Apr 2017	500	Color, 4 Keypoints	Annotated
PIGS1300	Oct 2016 - May 2018	1315	Color, 4 Keypoints	Annotated
PIGS1600	Oct 2016 - May 2018	1600	Color, 4 Keypoints	Annotated
PDD2019 <sup>160</sup>	Oct 2016 - Sep 2018	2000	Color, 4 Keypoints	Annotated
MFG110EVAL	Oct 2016 - May 2018	110	Color, Foreground	Annotated
PIGSEG96	Apr 2017 - Jun 2017	96	Color, Segmentation	Annotated

Table 3.2: Image data sets used in this work.

### 3.1.1 Pig Detection Dataset

The main source of annotated training data used in this work is contained in the Pig Detections Dataset 2019<sup>160</sup> (last row in Table 3.2). It contains the total of 2000 **manually annotated** images of group housed pigs. The dataset is divided into two main partitions: one containing 1600 images for model training (referred to as *train* subset), and another, containing 400 images for testing. Additionally, the testing partition was subdivided into two subsets: *test:seen* and *test:unseen*. The *test:seen* subset contains images similar, but not identical to the ones in the *train* as they originate from the same deployments. The second testing partition, *test:unseen* contains novel images with environment and lighting conditions that are noticeably different than the ones from the training collection.

Most of the images originated from the massive superset of images collected using Microsoft Kinect v2 but the depth information was not preserved. Annotations consist of the pixel positions of 4 types of landmarks: left ear, right ear, shoulder, and tail. Visualization of such annotation is presented in Figure 3.6 (a). To author's knowledge this set is the most complete, publicly available source of training data for the tasks of pose estimation and semantic instance segmentation of group-housed pigs.

In the context of the spatially-challenging tasks presented in this work, it is worthwhile to mention the relationship between the size of the objects of interest (pigs) with respect to the entire visible area (image size). Distributions of the sizes of the instances of interest are presented in figure 3.2. Assuming each instance has to be defined as a pair of shoulder and tail like in,<sup>160</sup> the the number of instances properly annotated in each subset is presented in figure 3.3. At this point it is important to notice that the training set is dominated by rather small instances (around 220 pixels long) and images contain rather substantial number of instances - especially compared to a small number of large instances present in *test:unseen* subset.

When looking at histograms in Figures 3.2 (a) and (b), it is visible that the means are

only 10 pixels apart between the *train* and *test:seen* subsets, the spread of instance size is also similar. The *test:unseen* subset seems to almost contain instances with sizes that could be represented by a uniform distribution in the range of 300 – 550 pixels on top of some samples with size similar to the other two subsets.

When considering the number of pigs annotated in the images visible in Figure 3.2, it is apparent that the *training* subset mostly contains images with the number of instances around 13 – 15, while neglecting the range of 22 – 27 animals per picture. When compared to part (b) indicating the statistics for *test:seen*, this absence is also visible. Also, the range of small number of instances  $\leq 7$  however is covered in *test:seen* at higher rate than in the test set. The distribution of instances in *test:unseen* does not match the histograms for the other two subsets. We recognize those properties as a sampling bias and anticipate potential performance drawbacks in the range of small number of instances and instances larger than 350 pixels while expecting the peak performance for images with number of instances within the 13 – 15 and instances with about 180 – 200 pixels shoulder to tail distance.

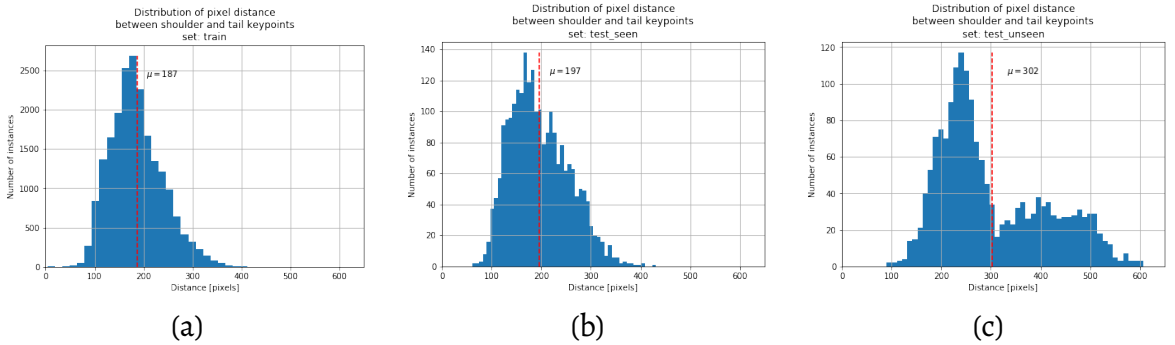


Figure 3.2: Histograms representing the distributions of the pixel-level shoulder-to-tail distances of all instances containing both annotated keypoints over the entire subsets: train (a), test:seen (b), and test:unseen (c).

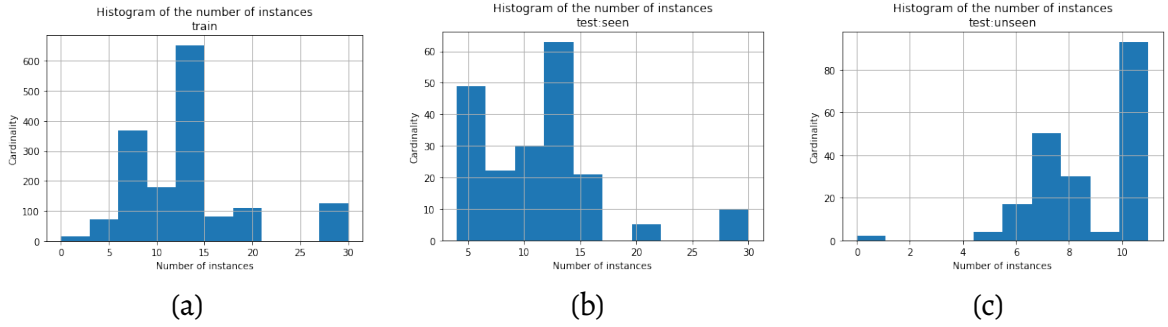


Figure 3.3: Histograms representing the distributions of number of instances the entire subsets: train (a), test:seen (b), and test:unseen (c).

### 3.2 Convention of image representation

Following the commonly used Machine Learning frameworks supporting the 2D-convolution operation, such as TensorFlow or Caffe, it became a standard to represent input data in the generalized form of multi-dimensional (particularly 4-dimensional) arrays often referred to as *tensors* or *blobs*. TensorFlow, being the framework mostly used in this work uses the  $B \times H \times W \times C$  convention, where  $B$  is the number of elements in the *batch*,  $H$  is the height (or number of rows),  $W$  is the width (or number of columns), and  $C$  is the number of channels (or number of dimensions representing a single instance - in case of images - pixel).

**Example 1.** To disambiguate the meaning of the word *channel* among other terms and clarify the representation of the images, let's consider two RGB images  $A$  and  $B$  that are 5 pixels wide and 4 pixels tall as presented in Figure 3.4. According to the Electrical Engineering Dictionary by Laplante,<sup>163</sup> the word *channel* refers to “the medium along which data travel between the transmitter and receiver in a communication system”. In the context of processing images, the word *channel* is generally used to refer to a specific component of a pixel such as color intensity value or more generally, a *channel number* is a specific dimension of a vector encoded at each discrete image location (pixel).

There are three, fully saturated distinct colors present in the image  $A$ : blue, green,

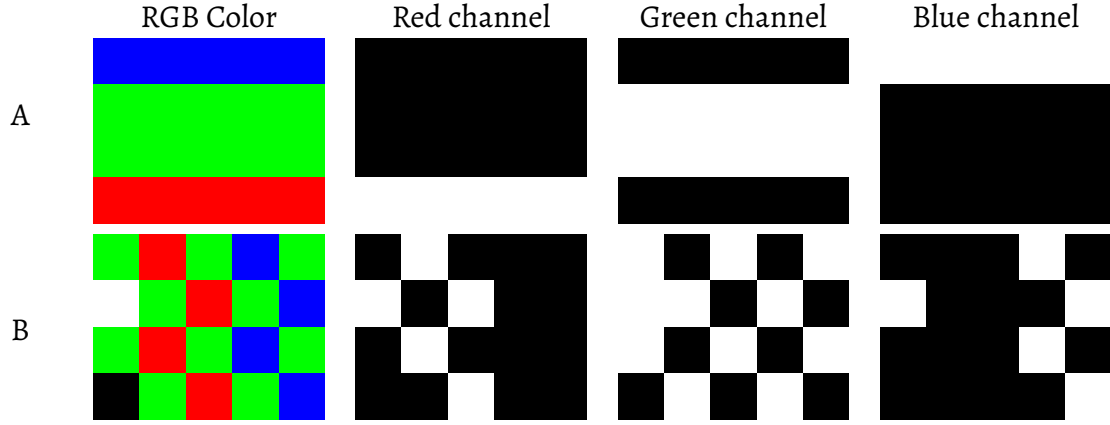


Figure 3.4: Two example images and their RGB decompositions with the content of each channel visualized as white for the pixel value of 255 and black for 0.

and red. Image *B* introduces a single white and a single black pixel. Using RGB encoding and 8-bit palette, those colors can be described as three-dimensional vectors:

$$p_{\text{red}} = [255, 0, 0]$$

$$p_{\text{green}} = [0, 255, 0]$$

$$p_{\text{blue}} = [0, 0, 255]$$

$$p_{\text{white}} = [255, 255, 255]$$

$$p_{\text{black}} = [0, 0, 0]$$

Our work adopts the convention of representing images as arrays of  $1 \times H \times W \times C$ . Thus, each of the images *A* and *B* would be represented by  $1 \times 4 \times 5 \times 3$  arrays. Note the row-first sizing like when using standard matrices as opposed to the usual column-first description when referring to images. When simultaneous processing of multiple images is desired, the input can be formatted into a *batch* containing multiple images. A batch, containing images *A* and *B* would then be represented as a  $2 \times 4 \times 5 \times 3$  array as the first index is used to indicate example in the batch. Literature refers to batches with

small number of examples ( $1 - 32$ ) as “minibatch”. In our work we feed single images (batch size  $B = 1$ ) to the models due to size variation between the training examples.

Since no recurrent models are used in this work, both neural networks used here can be represented as directed acyclic graphs. There are two directions of traversal through such model: **forward** (also referred to as the “forward pass”), and **backward**. The lone pass of the input data through the model in order to obtain the output is referred to as *inference*. The backward direction is used only during the *training* stage and requires a prior forward pass to occur to estimate the **loss**. Loss is defined using a differentiable function that produces the value of the residual to be minimized by the **solver**. The solver is a program that implements an optimization algorithm capable of adjusting the model’s trainable parameters to minimize the loss. At this stage it is important to explain what the *roles* of data inputs are with their respective representations / encodings. The explanatory variables (**input** to the model) are going to consist only of the pre-processed, RGB color images of pigs in the pen organized in multi-dimensional arrays. The neural network is responsible for producing the following **outputs**: 1) body part locations (described in section 3.3), 2) body part association vectors (section 3.6), 3) foreground binary mask (section 3.5), and 4) embedding vectors (section 3.9).

### 3.3 Representation of Body Part Locations (Keypoints)

A bottom-up approach to visual object tracking is via breaking down the problem into simpler sub-problems and focusing on localizing the parts that those tracked objects are composed of. In the field of computer vision, those parts are called *keypoints* and are also known as *landmarks* in statistics. They are specific to the kind of object being tracked. Each keypoint is defined as an inherent part of the *instance* that may or may not be visible in the frame. In case of human tracking, popular datasets such as COCO<sup>8</sup> and MPII<sup>164</sup> establish encoding conventions for human instances using landmarks such as: head,

shoulders, elbows, wrists, ankles, knees, hips etc. with various levels of granularity. An example of keypoints encoded by a heatmap is presented in Figure 2.14.

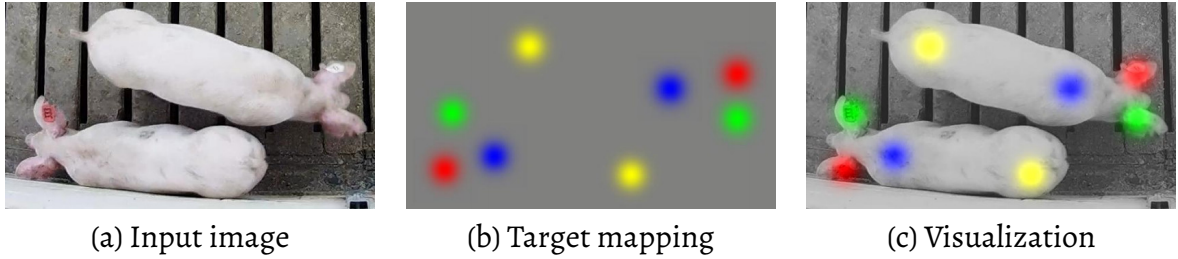


Figure 3.5: Part of the original image (a) is mapped to four-dimensional encoding of the keypoints of interest (b), where the location of each landmark is represented by Gaussian kernel on a separate image channel. The colors used in the figure are merely for visualization purposes. Superimposed combination presented in (c) shows the alignment of the keypoints with respect to the true body parts visible in the image.

### 3.3.1 Sparse representation of keypoint locations

Following the experience from our work on the 3D-tracking with downward-facing cameras, it seemed natural to describe and represent animal's orientation using a line along its back. In our work we adopted a convention of encoding animal's position and orientation using two-dimensional image coordinates of shoulder and tail location. Thus, assuming the presence of  $N$  animals in the scene in the pen, the tail and shoulder position of animal  $n \in 0, \dots, N - 1$  is denoted as  $s_n = (x_{shoulder_n}, y_{shoulder_n})$  and  $t_n = (x_{tail_n}, y_{tail_n})$ . The main advantages of this representation are 1) ease of annotation with just two points for animal location and 2) simple but robust way determining animal's unambiguous orientation. Terms *tail* refers to a surface point along the center ridge of the back that is between the left and right ham. Analogically, *shoulder* refers to the center ridge of the back between the shoulder blades.

Our work on re-identification of tracked animals involved reliance on the presence of colored and numbered ear-tags, thus creating a natural set of two additional keypoints to be detected. Including the location of the ears, complete representation of animal

keypoints now also include  $l_n = (x_{le_n}, y_{le_n})$  and  $r_n = (x_{re_n}, y_{re_n})$  corresponding to the left and right ear respectively. Those features are treated with secondary priority due to the fact that ears are often not visible in the frame - especially when occluded by the enclosure of the feeder (when eating) or when the animal is lying down on its side (thus self-occluding one of the ears). They are still useful however, for more exact orientation determination, and when animals are ear-tagged, they can allow for unique identification.

In to order to provide *ground-truth* data for the training, an interactive tool implemented in python to allow for easy and relatively quick image annotation. For the lack of a better name, it will be called Pig Keypoint Annotation Tool (PKAT). It contains graphical user interface with an interactive preview of the annotated images. The user can define an instance (pig) by sequentially clicking on image locations representing four keypoints (left ear, right ear, shoulder, and tail). This operation can be repeated for each animal visible in the scene. After completing the annotation process the script generates output in the following format:

$$\begin{aligned}
 &[image\_path, [[x_{le_0}, y_{le_0}], \dots, [x_{le_{n-1}}, y_{le_{n-1}}]], \\
 &[[x_{re_0}, y_{re_0}], \dots, [x_{re_{n-1}}, y_{re_{n-1}}]], \\
 &[[x_{s_0}, y_{s_0}], \dots, [x_{s_{n-1}}, y_{s_{n-1}}]], \\
 &[[x_{t_0}, y_{t_0}], \dots, [x_{t_{n-1}}, y_{t_{n-1}}]],
 \end{aligned} \tag{3.1}$$

where  $[a, b, c]$  operator represents the *list* containing elements  $a, b, c$  the  $x$  and  $y$  coordinates indicate the absolute pixel position in the image, and subscripts correspond to the particular type of keypoint. Note that the subscripts allow us to keep track of the membership of each keypoint. The data is saved using MATLAB file format (.mat) using the scipy.io library. Even though the native MATLAB library is not used, the data exchange between files saved using MATLAB and the ones saved using scipy is seamless.

PKAT draws convenient blue lines between the ears and along the back for each annotated target. It also displays white circles with text indicating which keypoint is being placed. The user enters the annotation using the left mouse button and can remove the last one using a right click. Pressing *space* moves to the next image, randomly selected from the entire superset of the images. If at least one full annotation is provided (i.e. four keypoints are indicated) before pressing *space*, the annotation index is augmented with the new entry. Figure 3.6 (a) shows the interface with fully annotated sample scene.

For tracking annotation, only shoulder and tail locations are indicated and a separate but very similar tool had to be designed. The Pig Tracking Annotation Tool (PTAT) is different from PKAT as it preserves the identity of each pig and traverses images by a regular interval. The main in the user interface difference is the per-instance previous state slightly visible in the bottom left corner of figure 3.6. User indicates position of shoulder followed by the tail using left mouse button. Right mouse button removes previously marked location, and allows for correction. After the entire image is annotated, the user can press the spacebar button to move to the next frame. The tracking annotation entries are formatted as follows:

$$[image\_path, [[x_{s_0}, y_{s_0}], \dots [x_{s_{n-1}}, y_{s_{n-1}}]], \\ [[x_{t_0}, y_{t_0}], \dots [x_{t_{n-1}}, y_{t_{n-1}}]],$$

When annotating images, one can follow a set of general-practice rules to avoid introducing erroneous data. The rules followed in this work apply to both keypoint and tracking annotations and can, be summarized as follows:

- Always try to place the keypoint exactly on the actual physical part of the target (seems obvious but needs to be emphasized).
- Use the relative orientation with respect to the target - e.g. the *left ear* is the one located on the left hand side of the animal's body - regardless of the orientation of

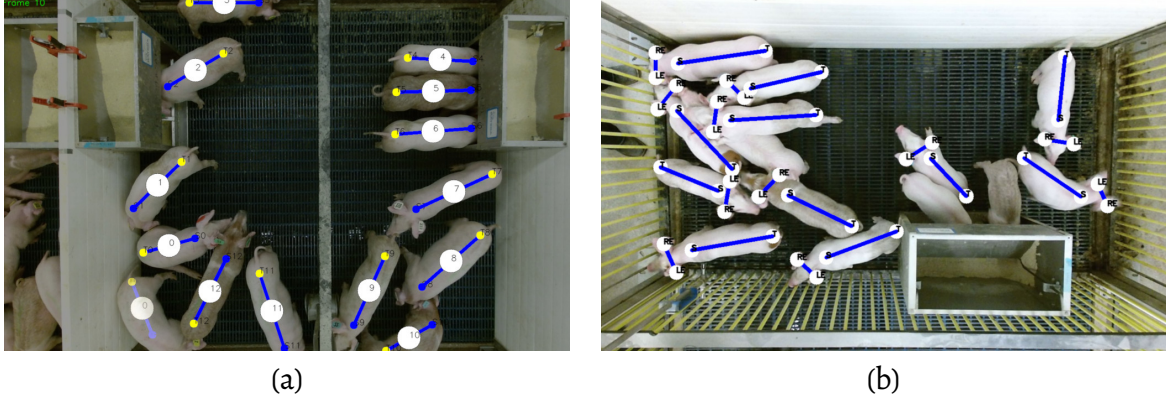


Figure 3.6: PTAT (a), and PKAT (b). Note the animal with label 0 in the left image - its state in the previous frame is indicated by semi-transparent line and previously marked placements of shoulder and tail.

the image.

- Only proceed if more than one target can be annotated in the image.
- Always mark all keypoints or do not indicate the target.
- Only indicate the *visible* parts - unless it could be *clearly* determined. Avoid marking keypoints on the fixed parts of the environment.

Figure 3.6 (b) represents a proper example of following the annotation rules in PTAT. Two animals by the feeder (bottom, right-hand side of the image) were not annotated because their ears would have to be placed on the feeder itself which could lead to making the trained model learn the visual representation of the ears to be on the metal body of the feeder. It is worth noting that *PIGS500*, *PIGS1300*, and *PIGS1600* (Table 3.2) were annotated using the same methodology but using MATLAB environment instead.

Formally, keypoint locations can be represented by a matrix. Consider an image that is  $w$  pixels wide and  $h$  pixels tall and a set of keypoint annotations like described above represented as an  $N \times 2k$  matrix  $P$ , where  $N$  is the number of annotated objects and  $k$  is the number of keypoints used to annotate each object. In the case of selected application

$k = 4$  as we chose to encode 4 types of keypoints: left ear, right ear, shoulder, and tail. The format of matrix  $P$  is presented in Equation 3.2.

Each row represents 4 pairs of coordinates such that columns 1 – 2 represent the  $x, y$  position of the left ear, 3 – 4 represent the position of the right ear, 5 – 6 represent the position of the shoulder and finally 7 – 8 represent the position of the tail. Per-object annotations contained in each row will be referenced using  $P_i$ , where  $i = 0, 1, \dots, N - 1$  is the index of individual object (pig).

$$P = \begin{bmatrix} x_{le_0} & y_{le_0} & x_{re_0} & y_{re_0} & x_{s_0} & y_{s_0} & x_{t_0} & y_{t_0} \\ x_{le_1} & y_{le_1} & x_{re_1} & y_{re_1} & x_{s_1} & y_{s_1} & x_{t_1} & y_{t_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{le_{N-1}} & y_{le_{N-1}} & x_{re_{N-1}} & y_{re_{N-1}} & x_{s_{N-1}} & y_{s_{N-1}} & x_{t_{N-1}} & y_{t_{N-1}} \end{bmatrix} \quad (3.2)$$

Extension of this representation to tracking assumes that the index  $i$  uniquely identifies the object within the entire sequence and not only within the frame.

### 3.3.2 Dense representation of keypoint location using heatmap images

Even when following the strictest rules, there always exist uncertainties inherent to the manual annotation of body part locations introduced by the human error. To increase robustness and allow for reasonable tolerance, the keypoint locations were encoded using 2-dimensional Gaussian kernels instead of single-pixel binary values or sharp-edged discs. This decision was made because to remain consistent with our previous methods of keypoint encoding<sup>160</sup> which were motivated by Cao et al.<sup>65</sup>

While a bivariate Gaussian kernel is described by a mean vector  $\mu = [x, y]$  and a  $2 \times 2$  covariance matrix, here the off-diagonal elements are zeroed-out, and the same parameter  $\sigma_{kp}^2$  is used to describe the spread along  $x$  and  $y$  direction. This represents the assumption that the annotator does not *favor* any particular direction of error, no bias is

introduced along the diagonal directions. Thus, generation of ground-truth heatmaps as visible in Figure 3.5 only requires center coordinates and predefined uncertainty expressed as  $\sigma_{kp}^2$ . The covariance matrix is then obtained by  $\sigma_{kp}^2 \cdot I$ , where  $I$  is the  $2 \times 2$  identity matrix. The value of  $\sigma_{kp}^2 = 5$  was chosen arbitrarily and does not vary with respect to the image resolution or the size of the tracked objects. This decision is motivated by reduction of the number of hyper-parameters. Note that in our prior work we varied the standard deviation in keypoint encoding in a per-animal fashion depending on its size.<sup>160</sup> Fixed spreading factor has been however used successfully in the literature.<sup>65, 113, 114, 161</sup>

The resulting encoding is represented as four  $h \times w \times 1$  *heatmaps*, which are *channel-wise* concatenated (stacked) producing a  $h \times w \times 4$  ground truth input for training - similar to the mapping presented in Figure 3.5 (b). Each major color is placed on a separate channel. It is worth mentioning, that such encoding allows us to indicate different body parts in a very similar (or even identical) location. This ability is motivated by the physical closeness of the pigs visible in multitude of examples in the dataset, where one animal's tail can be very close to another's ear. Taking into account variability in the annotation, our method makes the encoding of such situations possible.

To create the dense image-like representation of the aforementioned keypoint location using heatmaps from the sparse representation from annotations like presented in Equation 3.2 we use bivariate Gaussian kernel. For the general case, the multi-variate Gaussian kernel density is expressed as follows:

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right), \quad (3.3)$$

where  $x$  is the  $n$ -dimensional vector for which the density is estimated,  $\mu$  is the  $n$ -dimensional vector representing the center of the distribution, and  $\Sigma$  is a positive-definite  $n \times n$  covariance matrix of the distribution.

In our encoding we decided to have the peaks of the distributions to assume value 1, thus we normalize by the value of the peak:

$$p_{\text{peak}}(\mu, \Sigma) = p(\mu; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}}. \quad (3.4)$$

We arrive at the radial function for estimating the heatmap value at location  $x$  given assumed  $\Sigma$  and keypoint position  $\mu$ :

$$h(x; \mu, \Sigma) = \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right). \quad (3.5)$$

Procedure described by Algorithm 1 shows the process of producing a  $h \times w$  heatmap image given the annotations in a form of a row  $P_i$  of a matrix  $P$  for a keypoint  $k_i \in [0, 1, 2, 3, 4]$ , and the assumed  $\sigma^2 = \sigma_{\text{kp}}^2$ .

---

**Algorithm 1** Create per-instance keypoint heatmap image

---

```

1: procedure KEYPOINTLOCATIONHEATMAP( $P_i, k_i, \sigma^2, w, h$ )
2:    $H = \text{zeros}(h, w)$ 
3:    $\mu = [P_i[2k_i], P_i[2k_i + 1]]$ 
4:    $\Sigma = \text{diag}(\sigma^2, \sigma^2)$ 
5:   for  $x \in [\mu_x - 3\sigma, \mu_x - 3\sigma + 1, \dots, \mu_x + 3\sigma]$  do
6:     for  $y \in [\mu_y - 3\sigma, \mu_y - 3\sigma + 1, \dots, \mu_y + 3\sigma]$  do
7:        $H[y, x] = h([x, y], \mu, \Sigma)$ 
   return  $H$ 

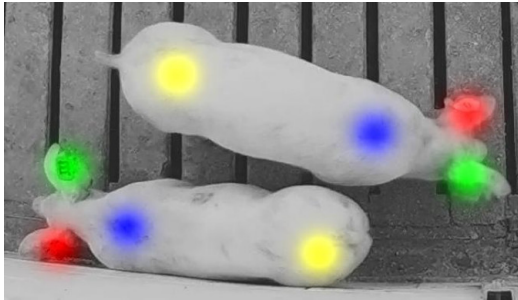
```

---

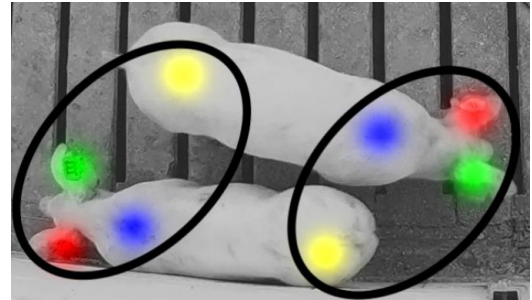
In bottom-up tracking by detection, even if body parts are detected correctly, they must be associated with one another in order to identify individual object instances. A naive approach relying only on the locations of the body part detections would associate each body part with the nearest neighbor in terms of Euclidean distance using bipartite matching such as in the Hungarian algorithm or a greedy algorithm ordered by the strength of detections. However, due to elongated shapes of pigs, such approach would fail in cluttered environments as illustrated in Figure 3.7.

To address this problem, additional features are introduced to the ground-truth

instance representations in the form of body part association vectors, implemented here as Part Affinity Fields.<sup>65</sup> We also explore matching keypoints using cost calculated using embeddings.



(a) Annotated image



(b) Nearest-neighbor euclidean distance grouping

Figure 3.7: Properly annotated two object instances of pigs are depicted in (a). A failure case of association using naive nearest-neighbor matching based on euclidean distance between coordinates is shown in (b).

### 3.4 Pixel-level instance identification representation

It is important to mention that even though the data sets with instance-level annotations exist for variety of problems such as pedestrian tracking, human pose estimation, and various segmentation tasks for natural images, there is yet to be published a suitable training set for MOT, pose estimation or image segmentation for pigs. Thus, we attempted to generate a synthetic dataset. This section describes the transformation of keypoint and body part association annotations into representations for individual pigs first, and then the entire image.

Here, per-instance masks are drawn and labeled based on the annotations from the PDD2019 dataset in an attempt to overcome the lack of ground-truth masks. The goal is to generate a map of desired pixel membership; i.e. such an image, in which each pixel contains an integer value of the index of the instance (pig) to which such pixel belongs. Please refer to figure 3.8.

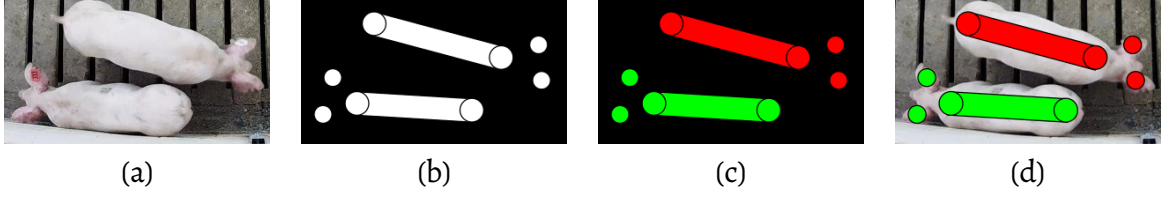


Figure 3.8: Stages of generating pixel-id image  $I_{id}$  with the input image (a), union of  $I_{mask_0}, I_{mask_1}$  (b),  $I_{id}$  with different values assigned to each instance (c), overlay for visualization purposes (d). Strokes around the circles and lines are presented only for visualization purposes, also, instances like in (b) are never merged together prior to the  $combine\_ids(IDs)$  operation.

The mask can be generated for any image  $I$  of  $H \times W \times C$  dimensions when accompanied with a set of annotations for  $N$  pigs  $P = \{P_0, \dots, P_{N-1}\}$ , where each  $P_i$  is a  $4 \times 2$  matrix containing  $x, y$  coordinates of all four keypoints.

$$P_i = [x_{le_i}, y_{le_i}, x_{re_i}, y_{re_i}, x_{shoulder_i}, y_{shoulder_i}, x_{tail_i}, y_{tail_i}] \quad (3.6)$$

In order to generate the approximate mask  $M_i$  for each annotated target, the dimensions of the image  $I$  need to be known and referenced as  $w, h$  for width and height respectively. For each tracked object indexed by  $i$ , its identification image is generated as presented in Algorithm 2.

---

**Algorithm 2** Create approximate semantic instance segmentation mask from keypoints.

---

- 1: **procedure** APPROXIMATEINSTANCEMASK( $P_i, h, w, r_{EAR}, r_{SHOULDER}, r_{TAIL}, t$ )
  - 2:    $M_i \leftarrow zeros(h, w)$
  - 3:    $(x_{le_i}, y_{le_i}) = (P_i[0], P_i[1])$
  - 4:    $(x_{re_i}, y_{re_i}) = (P_i[2], P_i[3])$
  - 5:    $(x_{s_i}, y_{s_i}) = (P_i[4], P_i[5])$
  - 6:    $(x_{t_i}, y_{t_i}) = (P_i[6], P_i[7])$
  - 7:    $M_i \leftarrow drawFilledCircle(ID_i, (x_{le_i}, y_{le_i}), r_{ear})$
  - 8:    $M_i \leftarrow drawFilledCircle(ID_i, (x_{re_i}, y_{re_i}), r_{ear})$
  - 9:    $M_i \leftarrow drawFilledCircle(ID_i, (x_{s_i}, y_{s_i}), r_s)$
  - 10:    $M_i \leftarrow drawFilledCircle(ID_i, (x_{t_i}, y_{t_i}), r_t)$
  - 11:    $M_i \leftarrow drawThickLine(ID_i, (x_{s_i}, y_{s_i}), (x_{t_i}, y_{t_i}), t)$  **return**  $M_i$ ,
- 

where  $w, h$  are the width and height of the image to be drawn (in pixels),  $zeros(h, w)$  instantiates the  $h \times w$  matrix filled with zeros,  $r_{ear}$  is the radius of the circle indicating

left and right ear,  $t$  is the thickness of the line drawn along the back, and  $drawFilledCircle(image, point, radius)$  draws the circle with unit color, and  $drawThickLine(image, point_{from}, point_{to}, thickness)$  draws a line with unit color.

Formally, for each pixel with coordinates  $x, y$  the  $M_i$  image is defined as:

$$M_i[y, x] = \begin{cases} 1, \text{ if } \sqrt{(x - x_{le_i})^2 + (y - y_{le_i})^2} \leq r_{ear}, \text{ or} \\ 1, \text{ if } \sqrt{(x - x_{re_i})^2 + (y - y_{re_i})^2} \leq r_{ear}, \text{ or} \\ 1, \text{ if } \sqrt{(x - x_{shoulder_i})^2 + (y - y_{shoulder_i})^2} \leq r_{shoulder}, \text{ or} \\ 1, \text{ if } \sqrt{(x - x_{tail_i})^2 + (y - y_{tail_i})^2} \leq r_{tail}, \text{ or} \\ 1, \text{ if } d_{line}(x, y, x_{shoulder}, y_{shoulder}, x_{tail}, y_{tail}) \leq t \\ 0, \text{ otherwise} \end{cases}, \quad (3.7)$$

where  $d_{line}(x, y, x_0, y_0, x_1, y_1)$  is the distance between the point  $(x, y)$  and a straight line drawn between  $(x_0, y_0)$  and  $(x_1, y_1)$  expressed as:

$$d_{line}(x, y, x_0, y_0, x_1, y_1) = \frac{|(y_1 - y_0)x - (x_1 - x_0)y + x_1y_0 - y_1x_0|}{\sqrt{(x_{shoulder_i} - x_{tail_i})^2 + (x_{tail_i} - x_{shoulder_i})^2}}. \quad (3.8)$$

Then having an ordered collection of  $IDs = [M_0, \dots, M_{n-1}]$  a composite image representing the approximate ground truth labels for semantic instance segmentation can be generated using procedure like the one presented in Algorithm 3:

---

**Algorithm 3** Create a single-channel, integer-encoded composition of approximate semantic instance segmentation masks.

---

```

1: procedure COMBINEIDS( $IDs, height, width$ )
2:    $I_{id} \leftarrow \text{zeros}(height, width)$ 
3:   for  $i = 0, \dots, n - 1$  do
4:      $I_{mask_i} \leftarrow \text{true}(ID_i > 0)$ 
5:      $I_{id} \leftarrow I_{id} \cdot (1 - I_{mask_i}) + (i + 1) \cdot ID_i$ 
   return  $I_{id}$ ,

```

---

where the  $true(condition)$  operation writes value of 1 to all pixels satisfying the  $condition$  and 0 everywhere else.

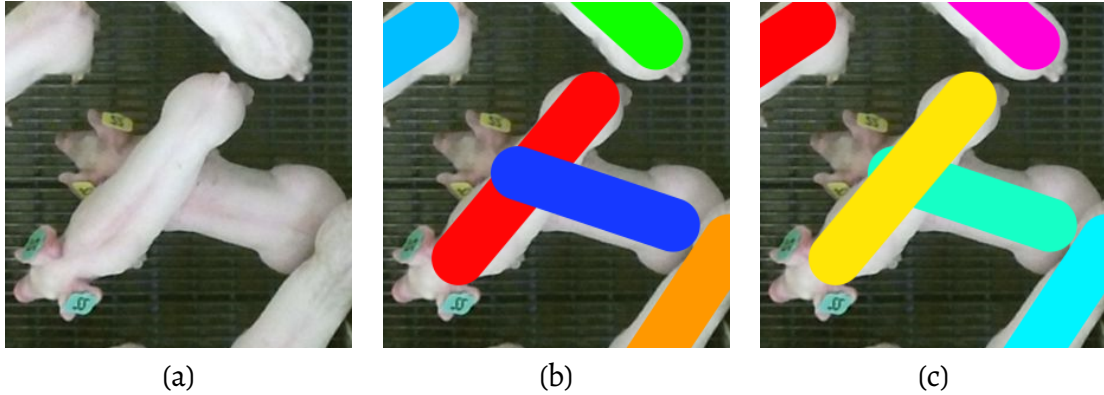


Figure 3.9: Illustration of the effect of randomization of the order of masks prior to generation of the composite ground-truth for semantic instance segmentation. A cropped image showing two overlapping animals (a), a case in which the horizontally oriented animal is drawn on top of the diagonally oriented one (b) and another case where the diagonally oriented animal is drawn on the top (c). The colors are incidental and merely indicate the relative membership of pixels (same color - same instance). Image was captured in March 2017.

As a result an image with values  $0, \dots, N$  is produced, where the background pixels (pixels which do not belong to any object of interest) have the value of 0, and each instance  $i$  is labeled with  $i + 1$  value. With the z-ordering being unavailable in the used data sets, an additional operation is introduced to shuffle the order of instances in the *IDs* list *before* invoking the *CombineIds()* procedure. The effect of randomization is depicted in Figure 3.9. This is motivated by the fact that in most cases, instances are not occluded. It is anticipated that randomization would yield outputs that correspond to the true z-order of instances presented in the image at the cost of small positive lower-bound bias of the loss value.

### 3.4.1 Small manually-annotated semantic instance segmentation evaluation set

In Section 3.9 we describe the concept of multi-dimensional embeddings produced by the neural network for the task of semantic instance segmentation of pigs. As stated in Section 2.8, the main challenge of approaching this problem is the lack of ground-truth

data containing instance segmentation masks. We overcome this problem by generating a synthetic representation based on the annotations available in PDD2019 dataset for the task of pose estimation.

Section 3.4 describes our method of generating a synthetic representation of the ground truth of the semantic instance segmentation labels for pigs. We decided that the automatic process described in Section 3.4 is appropriate for training as the labels can be generated quickly. We also use those approximate labels when analyzing the performance of our semantic instance segmentation method in Section 4.7 of Chapter 4.

Additionally, in order to provide more *accurate* ground-truth instance labels for *evaluation*, we constructed a small dataset of 96 images. Let's call it PIGSEG96. The data set consists of three subsets, each containing 32 examples. Each example contains a  $1920 \times 1080$  color image captured using Microsoft Kinect v2 camera paired with a manually created  $1920 \times 1080$  image indicating desired ground-truth labels for each pixel. To produce this dataset we traced the outlines of pigs in color images using GIMP - similarly to the method used to produce MFG110EVAL dataset described in Section 3.5.1.

Sample images from the three subsets of PIGSEG96 dataset are presented in Figure 3.10. Please note that the *actual* colors representing the labels are incidental.

### 3.5 Class-level representation of foreground instances

To determine the area of interest within each image, one needs to identify which pixels belong to the elements of the environment and which belong to the tracked or segmented objects. Such determination is a common subtask of pose estimation and segmentation methods as it limits the number of pixels required to process by removing the background and defines the *useful* image region i.e. parts of the image containing objects of interests. Use of the word *class* is intentional and refers to a binary classification interpretation with two available alternatives *pig* and *no-pig*.

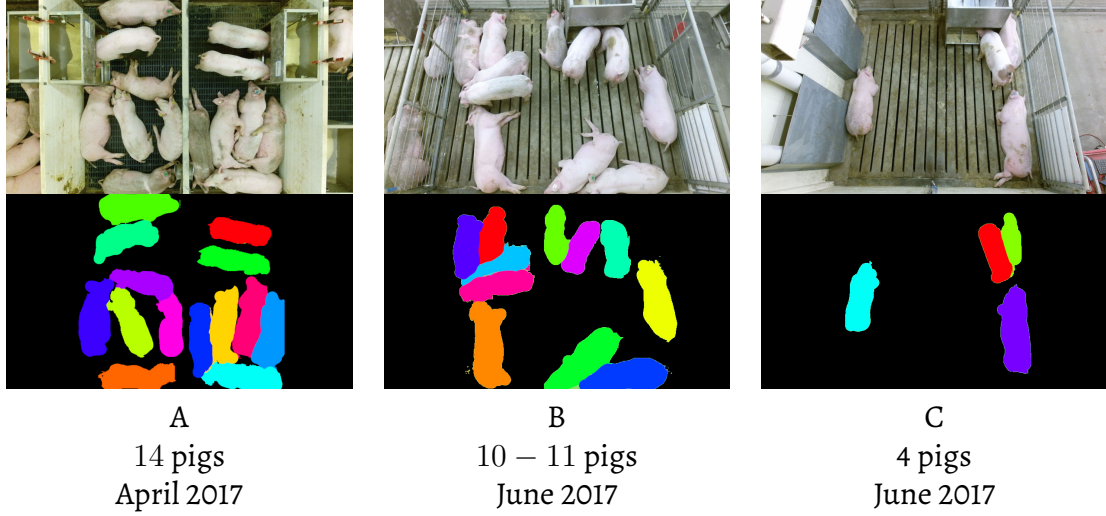


Figure 3.10: Sample images from PIGSEG96 dataset for semantic instance segmentation of pigs evaluation. Each image represents a single sample from one of the three subsets (A, B, C in columns). Color images are presented in the top row. Corresponding labeling images are presented underneath (with black background). Number of pigs and date of collected in presented underneath.

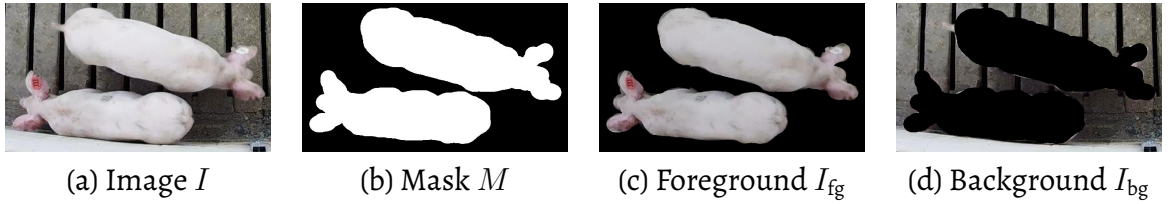


Figure 3.11: Image composition in the context of equation 3.9. Input image shown in (a), a corresponding example of a foreground binary mask  $M$  in (b), foreground image obtained by  $I_{fg} = I \cdot M$  in (c), and background image  $I_{bg} = I \cdot (1 - M)$  in (d).

After obtaining proper coordinates of object keypoints and solving the instance associations, one would like to be equipped with a binary mask allowing for single-instance isolation. Thus, an attempt was made to approximate such mask using background subtraction. Given image  $I$  consisting of foreground  $I_{fg}$  and background  $I_{bg}$  pixels an image can be understood as a composition of foreground and background as:

$$I = I_{fg} \cdot M + I_{bg} \cdot (1 - M), \quad (3.9)$$

where  $M$  is the binary mask.

Such (de)composition is depicted in (Figure 3.11). In this work, the binary foreground segmentation mask  $M$  is encoded a single image channel as visible in figure 3.11 (b), with unit value indicating the pig and 0 indicating the background.

As previously mentioned, the PDD2019 set does not contain foreground masks for the pigs. Thus, the ground-truth images containing binary masks indicating the pixels belonging to the pigs in the images need to be additionally annotated or estimated. We considered three ways to provide ground-truth data for foreground estimation training:

1. An entirely manual annotation method using a image manipulation program was used to produce a small test dataset described in Section 3.5.1 with an overview depicted in Figure 3.12. Although capable of achieving arbitrarily high level of *quality* (based entirely on the visual inspection), we considered this method to be too time consuming and labor intensive and proceeded only up to the number of 110 images;
2. An automated method using available keypoint annotations to draw thick lines along the backs of the animals (and optionally circles covering areas of ears) as estimates of the foreground masks. This method is described in Section 3.4 and was used to produce ground-truth for the both class, and instance-level masks. The main disadvantage of this method is its inability to accurately cover the area of animals that are not oriented straight;
3. A semi-automated method of converting the depth images into foreground masks from available image data sets captured using Microsoft Kinect v2, and pairing it with the corresponding color images. This method relies on our knowledge about each, specific pen environment such as: which parts of the image should be excluded from processing, and coefficients of a plane equation defining the surface of the floor in the pen; This method is described in Section 3.5.2.

### 3.5.1 Small manually-annotated foreground mask evaluation set

To build a test data set for the segmentation task, a sample of 110 images was constructed. 10 images were uniformly sampled and annotated for 11 data collection system deployments we conducted over the span of more than 2 years (from October 2016 to May 2018). The annotation was performed manually by drawing white shapes using *pencil* tool in GIMP resulting in 110 binary masks. The purpose of this set is to inspect the suitability of trained models across different pig pen environments using Receiver Operating Characteristics (ROC) analysis. Thus, let's call it **MFG110EVAL** for **Manually-annotated set of ForeGround masks containing 110 images for EVALuation**. Results for our models are presented in Section 4.2 in Chapter 4. An overview of this set is presented in Figure 3.12.

Subsets (a)-(i) were all collected using Kinect v2 camera and contain paired color and depth images. Subset labeled by (a) contains images with 15 pigs from one of our earliest deployments of the Kinect v2 systems in 2016 at Union Farms in Ulysses, NE. Subsets (b) and (c) were collected in near-ideal conditions as the camera orientation was facing directly down with the image centered in the middle of the pen. Subsets (d)-(h) contain images of larger animals and the cameras had to be installed at an angle due to mounting on the wall instead of the ceiling. Subset (i) contains pictures of 6 small pigs during a viral challenge. Most recent subsets (j) and (k) represent images captured without the depth information using a camera with heavily distorting optics.

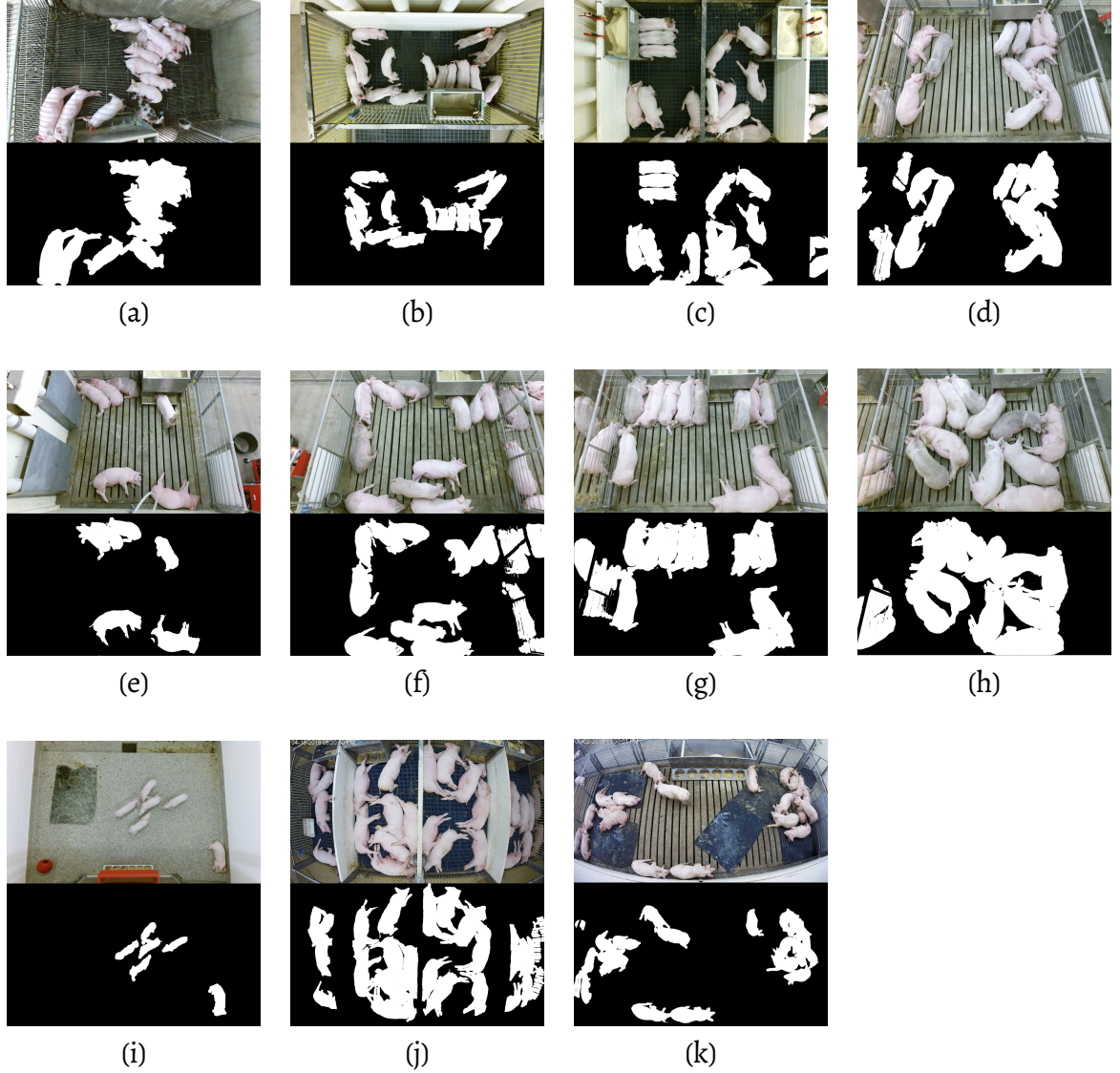


Figure 3.12: Overview of the nature of images in the MFG110EVAL set. Color images on the top with respective, hand-annotated segmentation masks on the bottom. Deployment dates: October 2016 (a), March 2017 (b), March 2017 (c), April 2017 (d), May 2017 (e), May 2017 (f), June 2017, (g), July 2017 (h), October 2017 (i), April 2018 (j), May 2018 (k).

### 3.5.2 Multi-view alignment and foreground mask extraction from depth images

Availability of the datasets containing annotated image masks for semantic segmentation of group-housed animals is extremely limited. Thus, in order to provide data for our we attempted to construct a synthetic sets of images with data that approximates foreground masks. We chose to use the color-and-depth image pairs captured by Microsoft Kinect v2. Table 3.2 provides a list of subsets containing depth information.

We picked 120000 color and depth image-pairs from 12 subsets, 10000 images pairs each. The images were selected randomly (uniformly) from each deployment. Color images were mapped (transformed) to the depth image coordinate spaces, and the depth images were processed as described below. The main goal of this set was to provide medium-scale training data at relatively low cost without the need for manual annotation. This aspect of presented work constitutes the attempt to leverage the use of massive data available to the author.

It is advised that the reader refers to Figure 3.13. The described method produces a foreground mask (f) from the depth image (b) given the pen mask (d) and parameters describing the pen floor, such that the obtained foreground mask is aligned with the color image (a) and is adequate (according to human judgment) to determine which pixels in the color image belong to *pigs* and which do not.

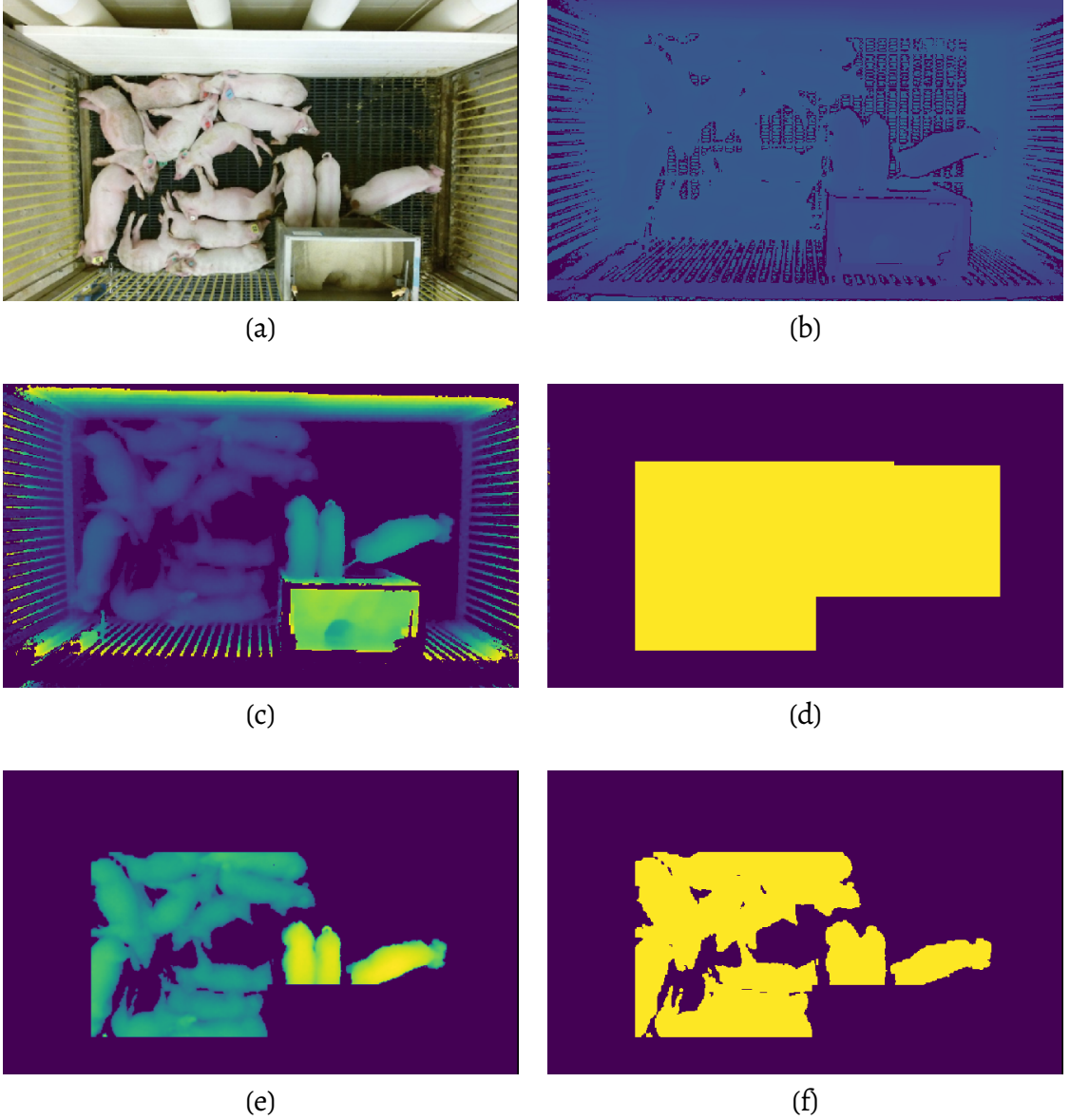


Figure 3.13: Intermediate steps of the depth to binary mask conversion process: color image for reference (a), input depth image (b), height map with aligned coordinates (c), deployment-specific extraction mask (d), masked height map (e), output instance segmentation mask (f).

Given a data set of  $N$  pairs of color (rgb) and depth images  $x^i = \{x_{rgb}^i, x_d^i\} \in X$ , where  $i = 1 \dots N$  under the assumption that each image  $x^i$  contains objects of interests that could be extracted using background subtraction, the goal is to estimate the function  $M = f(x_{rgb}^i, \theta | x_d^i)$  using depth information available in  $x_d^i$  where  $\theta$  is the set of parameters obtained using plane fitting process.

The process of extraction of the foreground mask from depth images is visualized in the figure 3.13. The first step is conversion from raw depth image (3.13 a) to a height map (3.13 b). It is done by subtracting the depth data from the floor height. The floor height image is deployment-specific and is estimated by solving an approximation function  $d \approx f(x, y)$ . Plane fitting is formulated as an optimization problem in the form of  $d = HA$ , where  $H$  is the transformation matrix and  $A$  is the row-concatenation of manually sampled fixed floor points (unoccupied by the objects of interest) points in the form of  $p_i = [x_i^2, x_i, y_i^2, y_i, 1]$ . The quadratic form is used to accommodate lens imperfections. The  $H$  matrix can be then obtained by using the Least Squares solution:

$$H = (AA^T)^{-1}A^Td, \quad (3.10)$$

where

$$A = \begin{bmatrix} x_1^2 & x_1 & y_1^2 & y_1 & 1 \\ x_2^2 & x_2 & y_2^2 & y_2 & 1 \\ \dots & & & & \\ x_n^2 & x_n & y_n^2 & y_n & 1 \end{bmatrix}, \text{ and } d = \begin{bmatrix} d_1 \\ d_2 \\ \dots \\ d_n \end{bmatrix} \quad (3.11)$$

This approximation works acceptably well for the images with camera facing down and the floor height can be estimated for every pixel on an image plane as  $f' = [x^2, x, y^2, y, 1]H$ . Doing so for all pixels results in an image representation of the floorplane (distance of the floor from the camera). After subtracting the values in the depth image from floorplane image, the result looks like in figure 3.13 (c) - which should now contain all objects of interest above zero.

Due to different optical parameters of the color and depth sensor, an alignment step needs to be introduced to match the pixel coordinates of objects visible by both cameras. It is done using properties of projective geometry. Each camera's intrinsic parameters can be briefly characterized by a  $K$  matrix. Depending on the lens and physical realization of the sensors, each particular device only roughly follows the general

specifications and thus, the camera matrix *should* be adjusted for each application.

The popular convention describes the intrinsic camera parameters as follows:<sup>165</sup>

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.12)$$

, where  $f_x, f_y$  is the focal length in pixels,  $s$  is the axis skew factor (usually 0), and  $x_0, y_0$  are the coordinates of the principal point offset (usually the middle of the image).

The sensor used to collect most images in datasets listed in table 3.2 was Microsoft Kinect V2. This sensor contains two cameras: 1080p color camera and a  $512 \times 424$  infrared camera. Kinect also contains embedded processor performing the analysis of a structured light pattern projected onto a scene. The analysis is based on the observation of the relative spatial shift (disparity) of the pattern which allows for estimation of distance to the camera (depth). Hence the images post processed by Kinect sensor are commonly referred to as *depth images*.

The base camera parameters used in this work are:

$$K_{\text{RGB}} = \begin{bmatrix} 1081.37 & 0 & 959.530 \\ 0 & 1081.37 & 507.5 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.13)$$

$$K_{\text{IR}} = \begin{bmatrix} 365.402802 & 0 & 261.696594 \\ 0 & 365.402802 & 202.522202 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.14)$$

, and

The rest of the pipeline involves binary masking by the area of interest which allows avoiding structures and obstacles present in the picture which could occupy the same

height in space. The actual mask needs to be manually created in a per-deployment fashion, similarly to the the selection of points belonging to the floor-plane. Example of such mask s presented in figure 3.13(d), where area containing the objects of interest is marked as a polygon with unit values. Result of binary masking is presented in figure 3.13 (e). The final segmentation mask (presented in figure 3.13 f) is then obtained by simple thresholding within the range of interest (e.g. everything above the floor and less than maximum height of the object).

The above process requires a manually defined binary mask and involves a relatively small number of deployment-specific hyper-parameters. Those include: the  $K$  matrix of the specific Kinect v2 unit capturing the image pairs, and height bounds defining the range of  $z$  coordinate above the floor within which the pigs instances are contained. Assuming static camera placement and orientation, the required transformations have to be computed only once per subset. Extraction of binary masks using the described process allows for productions of much needed ground-truth for the foreground estimation tasks but due to its automatic nature can introduce small artifacts caused by unwanted camera motion or presence of unexpected objects in the scene (such as people).

### 3.6 Representation of Body Part Associations (Part Affinity Fields)

Following the work on the bottom-up pose estimation method referred to as the *OpenPose* method,<sup>65, 113, 114, 161</sup> the underlying neural network model can be trained to produce additional features allowing for easier aggregation of the instance graph. Researchers behind OpenPose used landmarks and proposed the concept of *Part Affinity Fields* (PAF) as a way to encode connections between landmarks using additional image channels. Being developed for the human pose tracking, PAF were naturally defined along the human limbs but here the concept is applied to pigs and defined over arbitrarily selected

keypoints, following our prior work.<sup>160</sup> The word *field* corresponds to the fact, that PAF can be understood as two-dimensional vector fields defined for a pair pair keypoint types on a discrete space of image pixels.

Joining the keypoints along the part affinity field vectors outside of the neural network will be regarded here as a form of post-processing.

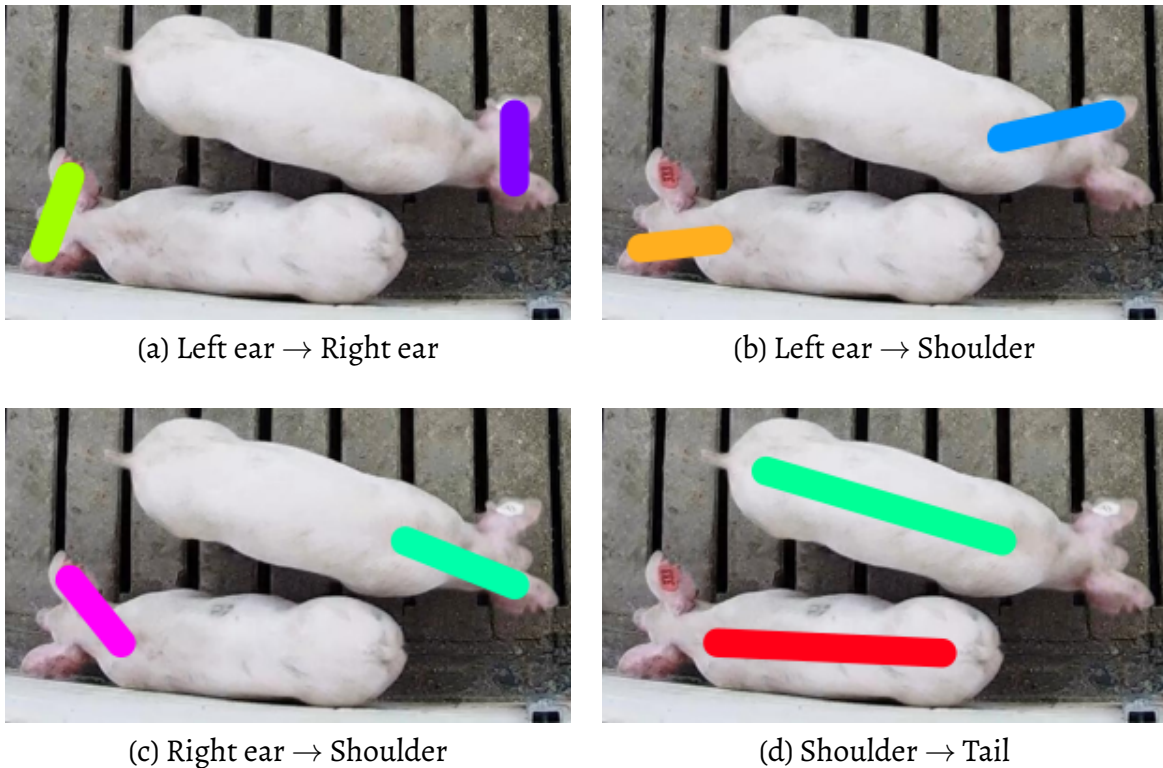


Figure 3.14: Visualization of the encoding of 2-dimensional Part Affinity Fields using color-mapped lines.

While when using 4 keypoints, the total number of PAF that can be defined is  $\binom{4}{2} = 6$ , i.e. if we encoded an association between each type of keypoint. Following our prior work<sup>160</sup> we decided to use only four associations to avoid redundancy. Each type of association is encoded using a pair of images representing the the  $x$  and  $y$  components of the association vectors - thus forming a representation known as Part Affinity Fields. Selected associations consist of:

- Images 0-1: Left Ear → Right Ear (Figure 3.14a.),

- Images 2-3: Left Ear  $\rightarrow$  Shoulder (Figure 3.14b.),
- Images 4-5: Right Ear  $\rightarrow$  Shoulder (Figure 3.14c.),
- Images 6-7: Shoulder  $\rightarrow$  Tail (Figure 3.14d.).

For each pair of body parts denoted  $L_{i,j} = [[x_i, y_i], [x_j, y_j]]$ , the corresponding part affinity is calculated as a 2-dimensional unit vector representing a displacement direction in the image space:

$$\vec{d}_{i,j} = \frac{1}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}} \cdot \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix} \quad (3.15)$$

To produce ground-truth inputs for training of a deep CNN,  $\vec{d}_{i,j}$  needs to be formatted as image-like inputs. Thus, a 2-channel map is created for each represented association. Manually indicated body part location annotations (as described in section 3.3) are used as the data source (figure 3.7 (a)), and *approximate* masks are produced by drawing thick lines between the annotated body parts. Each association can be visualized as in Figure 3.14 using angle  $\rightarrow$  color mapping like in Figure 3.15.

We use dense representations like the one presented in Section 3.4. Our procedure to generate the values of the 2-channel image representing the Part Affinity Fields between two keypoints is presented below:

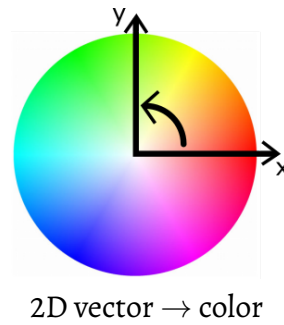


Figure 3.15: Color-wheel used for visualization of the 2D directional vectors.

As visible in equation 3.15, we normalize the values by the magnitude of the vector - thus encoding only the direction of the association and not the magnitude. Papandreou et al.<sup>117</sup> used similar method but preserved the magnitude of the vectors to encode the exact value of the distance between keypoints in the image coordinate space. Our previous contribution used such encoding with additional scaling with respect to the image / patch size successfully.<sup>160</sup> It is worth noting that a special care needs to be taken when using features sensitive to scale. Certain techniques of image augmentation and preprocessing steps (particularly image rotation) distort such encodings. On the other hand, the unit direction vectors are scale-invariant and their coordinates are bound within the  $(-1, 1)$  range.

Another difference from our previous approaches is the lack of additional reverse associations.<sup>160</sup> In other words, we do not encode the Left Ear  $\leftarrow$  Right Ear in addition to the Left Ear  $\rightarrow$  Right Ear. Previously we considered associations between: Left Ear  $\leftarrow$  Right Ear, Left Ear  $\leftarrow$  Shoulder, Right Ear  $\leftarrow$  Shoulder, and Shoulder  $\leftarrow$  Tail. Their goal was to increase robustness via consistency enforcement. Upon further inspection it was determined that they were identical to their reversed counterparts. Dropping their support aims to reduce the number of model parameters required to encode the associations and decrease the training complexity.

### 3.7 Image Augmentations

When training large scale (millions of parameters) models, there exists a significant risk of *over-fitting*, which can be observed when model approximates the training data very well but does not generalize, thus generates poor estimates for previously unseen inputs. When processing natural images and generalization is of interest, it is important to attempt making the model immune to content-independent variations such as camera angle, lighting conditions, etc. This is often explicitly handled using *augmentation*

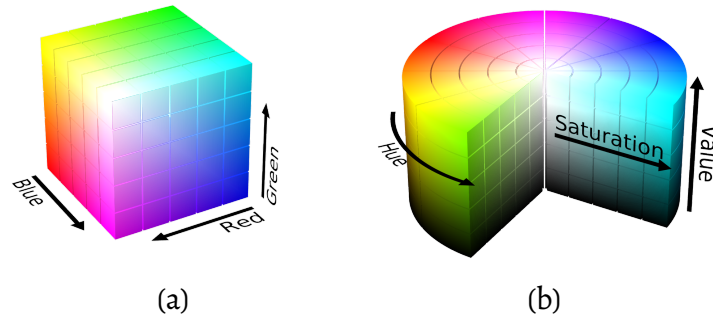


Figure 3.16: RGB and HSV Color models<sup>166</sup>

techniques. It can be understood as introduction of artificially generated content-preserving transformations into the input such that they stay within the boundaries of plausibility.

### 3.7.1 Augmentations in color space

To address different lighting conditions, exposure settings, image sensor's color capture accuracy and different noise properties of capture equipment, a color-space, content-preserving transformations can be applied. A common method is to introduce perturbations in Hue, Saturation, and lightness (Value) in the HSV color space.

When capturing an image using standard image sensor, obtained pixel colors are encoded using RGB (Red, Green, Blue) color scheme. There exist however, other popular methods of encoding pixel colors such as CMYK (Cyan, Magenta, Yellow, black - commonly used when preparing images for printing as the color encoding corresponds to the used ink cartridges) or HSV.

Here, due to the use of images captured from various data collection deployments over long periods of time, no additional variations in color space are introduced except for random greyscale conversion. The main reason behind the need for the network to perform in greyscale is the presence of colored ear-tags. In each deployment a set of differently colored and numbered ear tags was used, and it suspected that it would be

easy for a neural network to recognize colored, round objects in the image and train for those as ear representations explicitly. To mitigate this potential problem, color information is reduced when converting images to greyscale. It is argued, that such transformation is *content-preserving* as the presence of the ear is not eliminated.

Conversion of sRGB-encoded image  $C_{\text{srgb}}$  to greyscale is a one-way operation commonly defined as transformation to the encoding of linear luminance  $Y_{\text{linear}}$ . OpenCV - the library used in this work uses that common conversion defined as follows:

$$Y_{\text{linear}} = 0.2126R_{\text{linear}} + 0.7152G_{\text{linear}} + 0.0722B_{\text{linear}}, \text{ where} \quad (3.16)$$

$$\begin{bmatrix} R_{\text{linear}} \\ G_{\text{linear}} \\ B_{\text{linear}} \end{bmatrix} \leftarrow \begin{cases} \frac{C_{\text{srgb}}}{12.92} & \text{for } C_{\text{srgb}} \leq 0.04045 \\ \left( \frac{C_{\text{srgb}} + 0.055}{1.055} \right)^2 .4 & \text{for } C_{\text{srgb}} > 0.04045 \end{cases} \quad (3.17)$$

In our example preprocessing procedure we randomly convert the input image to greyscale with the probability of 50% using the exact formula presented above.

### 3.7.2 Augmentations in pixel coordinate space

Our goal is to produce a method applicable for scenarios beyond the exact ones from our data set. Particularly, we are using a relatively small dataset of 1600 annotated training images in an attempt to train a robust object detector for homogeneous objects for multiple object tracking. Due to the lack of frame-by-frame data with preserved instance identity, we are decided to introduce small perturbations to the available annotated images to simulate diversity using augmentations. The key factors to success when applying augmentations are: plausibility and content-preservation.

In order to not waste the capacity of the model to learn to handle improbable transformations (Figure 3.17 (d)) one should define a set of possible transformations that

are applicable and correspond to the effects observed in the data. Content preservation constraints the selection of available transformations to the ones that do not disturb the **content** of the image. In our case transformations drastically changing the appearance of animals would be considered not content-preserving.

We decided to resort to the set of affine transformations such as: translation, rotation, scaling, shearing, and warping. They are expressed in the form of  $3 \times 3$  transformation matrices, and operate on homogeneous pixel coordinate vectors  $[x, y, 1]$ . Additionally, common transformations include horizontal and vertical flipping (mirroring). These techniques accommodate variations in camera parameters and capture angles along with deformations of objects of interest.

$$T(\Delta_x, \Delta_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta_x & \Delta_y & 1 \end{bmatrix} \quad (3.18)$$

$$R(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.19)$$

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.20)$$

$$SH(sh_x, sh_y) = \begin{bmatrix} 1 & sh_y & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.21)$$

$$(3.22)$$

A single transformation encapsulating all desired augmentations can be calculated

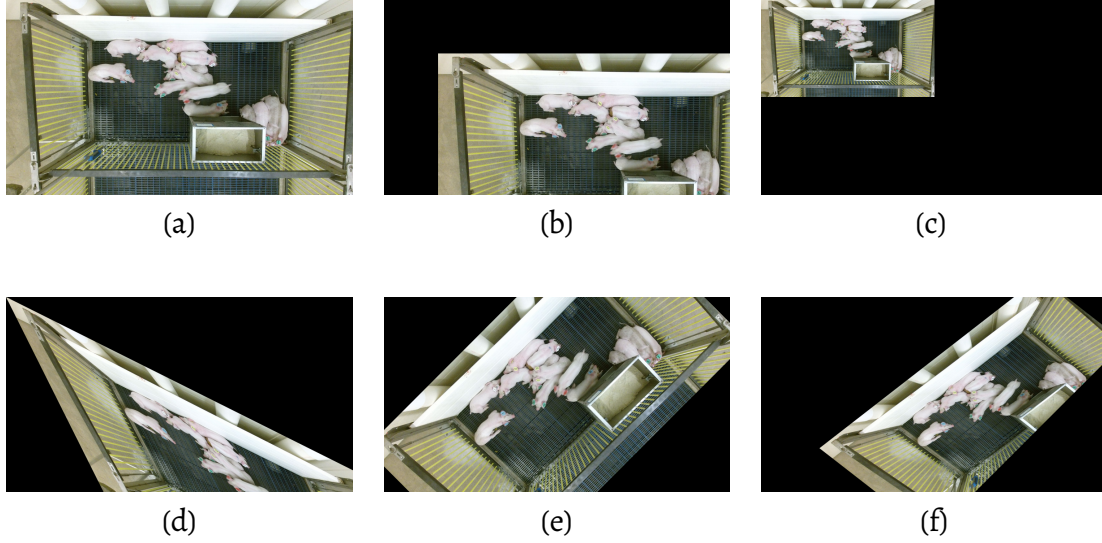


Figure 3.17: Extreme examples of augmentation transformations: original image from 2017-03-07-17-16-25 deployment (a), image translated by  $\Delta_x = 300, \Delta_y = 300$  (b), image scaled by  $s_x = 0.5, s_y = 0.5$  (c), image sheared by  $sh_x = 0.5, sh_y = 0.5$  (d), image rotated by  $\alpha = \frac{\pi}{4}$  (e), image augmented by random combination of transformations (f).

as the matrix in-order multiplication. Thus, the applied final transformation often looks as follows:

$$H(s_x, s_y, sh_x, sh_y, \alpha, \Delta_x, \Delta_y) = S(s_x, s_y)SH(sh_x, sh_y)R(\alpha)T(\Delta_x, \Delta_y) \quad (3.23)$$

Such representation allows for easy application by inner product of  $n \times 3$  homogeneous pixel coordinates matrix and the transformation matrix  $H$ ; e.g.  $\hat{x} = xH$ .

Those methods were however widely used in the literature and had proven effective as a tool to increase model's ability to generalize. Author's previous experience with training neural networks also had proven augmentations as a useful tool to overcome challenges imposed by limited amount of training data.<sup>160</sup>

Availability of the dataset containing frame-by-frame tracking information would allow for estimation of the necessary parameters. Unfortunately due to lack of thereof, here we decided to select the parameters arbitrarily according to our judgement. The chosen values are presented in Table 3.3. Note that both the input image and

Parameter	Value / Range	Description
$p_{\text{grey}}$	0.5	Probability of converting the image to greyscale
$\Delta_x, \Delta_y$	$U(-5, 5)$	Random shift in $x, y$ pixel coordinates
$\alpha$	$U(-0.5^\circ, 0.5^\circ)$	Random rotation angle (in degrees)
$s_x, s_y$	$U(0.99, 1.01)$	Random image size scaling factor
$sh_x, sh_y$	$U(-0.05, 0.05)$	Random shearing coefficient
$p_{\text{flip}l}$	0.5	Probability of flipping the image along the x-axis
$p_{\text{flip}u}$	0.5	Probability of flipping the image along the y-axis

Table 3.3: Values of the hyper-parameters used for augmentation of the training examples during training.  $U(a, b)$  defines sampling from a uniform distribution with the lower bounded between  $< a, b >$  (inclusive).

ground-truth information are exposed to the same transformations.

The shearing (warping) is the most questionable transformation among the ones presented and is not deeply explored. The selected shearing coefficients are kept low in our experiments as visible in Table 3.3.

### 3.8 Models

This section is dedicated to the description of the two deep, convolutional neural network architectures used in this work. The first one will be referred to as *OP* as it is directly inspired by the line of work presented in the Convolutional Pose Machines,<sup>113</sup> and the OpenPose<sup>65,161</sup> human pose estimation method. Decision to include this architecture in this work was motivated by the author’s fascination with the method’s performance in simultaneous multi-target human pose estimation task which could be attributed to the model’s properties. The training methodology presented in the related work also attempts to leverage model transfer using pre-existing image classification network in an attempt to lower the required training time. While being very deep, the OP model was shown to be capable of producing high quality results. However, the number of trainable parameters may end up being detrimental to the overall performance due to training difficulties.

We would like to point the reader to “*A guide to convolution arithmetic for deep learning*” by Dumoulin et al.<sup>167</sup> It contains the description of the building blocks of modern deep convolutional neural networks such as two-dimensional convolution, pooling and deconvolution (convolution with fractional stride). Provided description is complete we did not find the need to duplicate their description here.

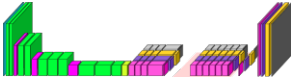
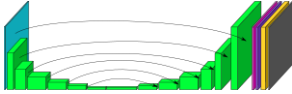
		
Main inspiration:	OpenPose (OP) <sup>161</sup>	UNET <sup>159</sup>
Description:	Section 3.8.1	Section 3.8.2
Visualization:	Figure 3.18	Figure 3.20
Embeddings:	Weakly-supervised	Weakly-supervised
Batch normalization:	No	Yes
Max downsampling:	$8\times$	$64\times$
Input size ( $W$ ):	$368 - 1024$	Fixed 512
Input context:	$368 \times 368$ patches	Entire image
Weight normalization:	$L2$	None
Skip connections:	Dense-only	Size-matched feature maps
Pre-trained front-end:	VGG-16 <sup>142</sup>	None
Downsampling:	Max-pooling	Strided convolution
Up-sampling:	Fixed, bilinear	Trained, convolution transpose

Table 3.4: Summary of two presented architectures: very deep, OpenPose-inspired network and a  $64\times$  downsampling UNET model with skip connections.

The second model is based on the UNET architecture<sup>159</sup> and resembles the network used in author’s prior contributions.<sup>160</sup> The main strength of this architecture is its robustness against common problems in training such as *vanishing gradient*. Author explores the residual approach, recently adopted batch normalization operations, and decided on fixed input size which are intended allow for easier training.

Due to substantial differences between the OP and UNET models, the author does not attempt to make a point of evaluating superiority of one model family over the other but rather present their performance alongside their advantages and drawbacks.

### 3.8.1 OP Model: A Very Deep Multiple-Objective Convolutional Neural Network

We decided to follow the model design used in the work on the Convolutional Pose Machines.<sup>113</sup> We base our design on the model introduced by Cao et al.<sup>65</sup> We extend on their network by introducing blocks responsible for producing dense, multi-dimensional pixel embeddings. Thus, our model produces the following outputs: 1) keypoint heatmaps (described in Section 3.3), 2) part affinity fields (described in Section ??), 3) foreground mask (described in Section 3.5), and 4) pixel embeddings (described in Section 3.9) allowing for cluster-based processing.

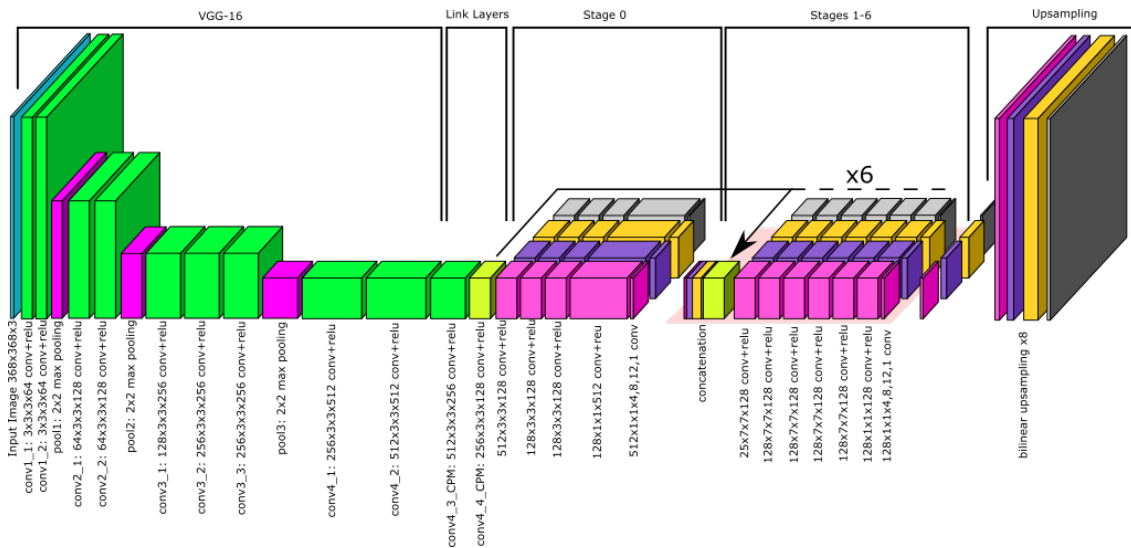


Figure 3.18: Deep Fully Convolutional Neural Network Architecture for simultaneous multiple instance estimation of body part location, part affinity fields, multi-dimensional pixel embeddings, and foreground segmentation mask based on VGG-16 and 7 (6+1) stage approach inspired by OpenPose.

The network presented in Figure 3.18 was designed to process 3-channel RGB input images and estimate 25-channel output representing the estimates of:

- Channel 0-3: Body Part Detection Heatmaps,
- Channel 4-11: Part Affinity Fields,

- Channel 12-23: Pixel Embeddings,
- Channel 24: Foreground Segmentation Mask.

The network was designed around the VGG-16<sup>142</sup> architecture as opposed to the use of SegNet-based hour-glass model in previous contributions.<sup>160</sup> Both approaches use a max pooling operation in processing eventually producing deep, but spatially small representations. Here however, most of the processing is performed using very deep, convolutional blocks operating on  $8\times$  downsampled representations as opposed to performing simultaneous upsampling and feature refinement.

To remove the *burden of upsampling* from the network, here a bilinear method is used and the upsampling kernels are not trainable.

Our model also uses *skip connections* as a means to improve backpropagation performance due to more pronounced gradient access and feature-reuse without the need for increasing the number of network coefficients.<sup>145</sup> The use of the skip connections is limited to reusing the outputs of *Stage 0* after the VGG-16 blocks.

### 3.8.1.1 Receptive Field

For spatially challenging tasks it is important to consider the size of the *receptive field*. The receptive field can be defined as the area that the network “sees” when creating an output pixel and corresponds to the window of aggregation of information that can be used to produce this output. More formally, it represents the width of a square region of the input image that affects the output pixel.<sup>160</sup>

After the input layer, the output of each convolution operation is called the *feature map*. To derive the size of the receptive field in a layered architecture containing downsampling operations such as strided convolutions and pooling layers, one needs to consider the effective stride length of each layer.

Following the convention from Psota et al.,<sup>160</sup> we define the following properties required to calculate the size of the receptive field:  $s_l$ ,  $s_{l_{\text{effective}}}$ ,  $d_l$ ,  $w_l$ , and  $r_l$ :

$$s_{l_{\text{effective}}} = s_{l-1} \cdot s_l, \quad (3.24)$$

where  $l$  is the layer index,  $s_l$  is the stride length at the layer  $l$ , and  $s_{l_{\text{effective}}}$  is the stride length between the adjacent coordinates in the feature map. For the input “layer”,  $s_0 = 1$ . All convolutional kernels in this network have the  $s_l = 1$ , all pooling layers have the  $s_l = 2$ , and the final upsampling layers have  $s_l = 0.125$  as in our model they up-sample feature maps by the factor of 8.

$$r_l = r_{l-1} + (w_l \cdot d_l - 1 - 1)/2 \cdot s_{l-1_{\text{effective}}}, \quad (3.25)$$

where  $w_l$  is the width of the convolutional kernel at layer  $l$ ,  $d_l$  is the atrous dilation rate,<sup>121</sup> and  $r_l$  is the size of the receptive field at layer  $l$  (calculated recursively). In the proposed network, convolutional kernels have size  $w_l = 3$  in the vgg, link, and first 3 parts of the stage0 stages,  $w_l = 1$  in the last 2 operations in stage0 and last two operations in each later stage (1-5), and  $w_l = 7$  in the first 5 operations of stages (1-5).

Calculation of the receptive field for all layers of the proposed network is presented in Table 3.5. Please note the elements in the *Type* column are: I - being the input,  $C_{h \times w, s}$  being the 2D convolution operation with  $h \times w$  kernels strided by  $s$  pixels, and  $US_8$  is the final upsampling step. As visible in Table 3.5, due to large depth and the use of  $7 \times 7$  kernels, the network exhibits large receptive field - particularly when compared to the model by Psota et al.<sup>160</sup> This choice is motivated by the need for spatially-intensive embedding estimation, which is intended to aggregate information from large portion of the image in order to produce distinct values for each *cluster*. Both the spatial accuracy for the task of keypoint detection, and large receptive field are highly desired. T we decided not to use atrous convolution and resort to deep architecture instead.

When comparing the spatial extent of the network’s visibility represented as the receptive field to the distributions of the sizes of the instances of interest presented in figure 3.2 (section 3.1.1), it was determined that the receptive field is of sufficient size to tackle the semantic instance segmentation task using available training data.



Figure 3.19: Visualization of the proposed network’s final stage receptive field.

$l$	Type	$s_l$	$s_{l_{\text{effective}}}$	$w_l$	$r_l$
0	I	1	1	1	1
1	$C_{3 \times 3, 1}$	1	1	3	3
2	$C_{3 \times 3, 2}$	2	2	3	4
3	$C_{3 \times 3, 1}$	1	2	3	6
4	$C_{3 \times 3, 2}$	2	4	3	8
5	$C_{3 \times 3, 1}$	1	4	3	12
6	$C_{3 \times 3, 1}$	1	4	3	16
7	$C_{3 \times 3, 2}$	2	8	3	20
8	$C_{3 \times 3, 1}$	1	8	3	28
9	$C_{3 \times 3, 1}$	1	8	3	36
10	$C_{3 \times 3, 1}$	1	8	3	44
11	$C_{3 \times 3, 1}$	1	8	3	52
12	$C_{3 \times 3, 1}$	1	8	3	60
13	$C_{3 \times 3, 1}$	1	8	3	68
14	$C_{3 \times 3, 1}$	1	8	3	76
15	$C_{1 \times 1, 1}$	1	8	1	84
16	$C_{1 \times 1, 1}$	1	8	1	84
17	$C_{7 \times 7, 1}$	1	8	7	84
18	$C_{7 \times 7, 1}$	1	8	7	108
19	$C_{7 \times 7, 1}$	1	8	7	132
20	$C_{7 \times 7, 1}$	1	8	7	156
21	$C_{7 \times 7, 1}$	1	8	7	180
22	$C_{1 \times 1, 1}$	1	8	1	204
23	$C_{1 \times 1, 1}$	1	8	1	204
24	$C_{7 \times 7, 1}$	1	8	7	204
25	$C_{7 \times 7, 1}$	1	8	7	228
26	$C_{7 \times 7, 1}$	1	8	7	252

$l$	Type	$s_l$	$s_{l_{\text{effective}}}$	$w_l$	$r_l$
27	$C_{7 \times 7, 1}$	1	8	7	276
28	$C_{7 \times 7, 1}$	1	8	7	300
29	$C_{1 \times 1, 1}$	1	8	1	324
30	$C_{1 \times 1, 1}$	1	8	1	324
31	$C_{7 \times 7, 1}$	1	8	7	324
32	$C_{7 \times 7, 1}$	1	8	7	348
33	$C_{7 \times 7, 1}$	1	8	7	372
34	$C_{7 \times 7, 1}$	1	8	7	396
35	$C_{7 \times 7, 1}$	1	8	7	420
36	$C_{1 \times 1, 1}$	1	8	1	444
37	$C_{1 \times 1, 1}$	1	8	1	444
38	$C_{7 \times 7, 1}$	1	8	7	444
39	$C_{7 \times 7, 1}$	1	8	7	468
40	$C_{7 \times 7, 1}$	1	8	7	492
41	$C_{7 \times 7, 1}$	1	8	7	516
42	$C_{7 \times 7, 1}$	1	8	7	540
43	$C_{1 \times 1, 1}$	1	8	1	564
44	$C_{1 \times 1, 1}$	1	8	1	564
45	$C_{7 \times 7, 1}$	1	8	7	564
46	$C_{7 \times 7, 1}$	1	8	7	588
47	$C_{7 \times 7, 1}$	1	8	7	612
48	$C_{7 \times 7, 1}$	1	8	7	636
49	$C_{7 \times 7, 1}$	1	8	7	660
50	$C_{1 \times 1, 1}$	1	8	1	684
51	$C_{1 \times 1, 1}$	1	8	1	684
52	US $_{8 \times}$	0	1	8	684
53	O	1	1	1	687

Table 3.5: Receptive field of each layer of proposed network.

In order to verify the size of the receptive field presented in Table 3.5 via visual and numerical inspection, the OP network was initialized with random weights and fed with two input images:  $I_0, I_1$  initialized with zeros, with dimensions  $512 \times 1024$  for height

and width respectively.  $I_1$  was augmented by setting a value of 1.0 at the 0, 0 pixel coordinates. Both images were fed through the neural network and their difference was calculated as  $I_{\text{diff}} = \|I_0 - I_1\|$ . This resulting image (after normalization and cropping) is presented in Figure 3.8.1.1. The maximum  $x$  coordinate with non-zero value is  $x_{\text{max}} = 687$  and a red, dashed line was drawn marking the boundary. This operation can be understood as calculating gradient of the output with respect to the input and finding the right-most pixel with non-zero value.

### 3.8.2 UNET: A Deep, Symmetric Architecture with Skip-Connections

After preliminary evaluation of the OP model presented in Section 3.8.1, an alternative UNET architecture resembling the model we used previously<sup>160</sup> was trained to address the long training time of the OP model. Its main purpose is to provide a context to the criticism of the use of unsupervised embeddings stated in<sup>117</sup> and relate the findings in this work to our previous contribution.<sup>160</sup>

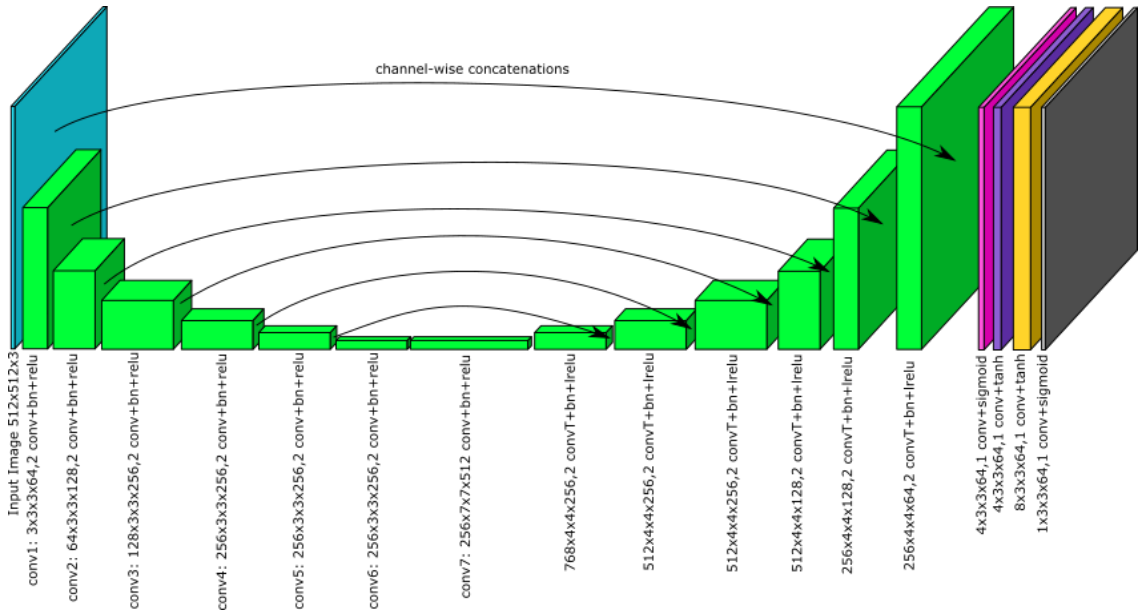


Figure 3.20: UNET-based Multi-Objective Model Architecture with skip connections and up to  $64\times$  downsampling deep features.

As visible in figure 3.20, presented model significantly downsamples the feature

maps, eventually producing  $8 \times 8 \times 512$  deep representation spanning vast spatial extent. The model then up-samples the deep features in a multi-stage fashion while re-using previously computed feature maps at matching scales. We use convolution with fractional stride for upsampling. The intermediate feature maps are brought to the original size of  $512 \times 512$  pixels and concatenated with the input. The final outputs are generated using  $3 \times 3$  convolution followed by activation functions matching the tasks as follows: *sigmoid* for foreground and keypoints, *tanh* for embeddings and part affinity fields. The reasoning behind this selection follows the output domain of the activation functions:  $0 - 1$  for *sigmoid*, and  $-1 - 1$  for *tanh*. We train this UNET model to produce the same types of outputs as the OP model described in Section 3.8.1.

### 3.9 Instance-Level Weakly-Supervised Multi-Dimensional Embeddings

We identified the need for encoding homogeneous object instances such that they can be easily detected in the images based on the per-pixel instance membership. We decided to explore the use of embeddings due to their independence from geometric representations and potential robustness against partial occlusions. We are using said embeddings to perform semantic instance segmentation and pose estimation of the group-housed homogeneous animals.

The main purpose of generating embedding vectors is to use them for instance-level segmentation through clustering. Generally speaking clustering is an unsupervised method of pattern-based classification of feature values into clusters. It can be understood as discovering natural structure in the data and appropriate grouping.<sup>168</sup> We are exploring creation of such easy to cluster features (embeddings) using neural networks.

We desire certain properties of those embeddings with respect to their closeness

(cohesion) and separability. Ideally, the produced feature values could easily be grouped into distinct sets, cohesive within the instances and easily separable from the others. Like previously mentioned, the prior value of the desired embedding values is unknown, but certain constraints can be enforced given the ground-truth data and appropriate loss function. This section describes our weakly-supervised approaches to implicit generation of such cluster-able embeddings using deep CNNs through the formulation of differentiable objective functions. Section 3.9.1 provides formal explanation of the cohesion and separation metrics. Section 3.9.3 describes a successful, heuristic, parametric method adopted in the literature for the task of semantic instance segmentation while in section 3.9.2 we present a different implementation, directly maximizing the silhouette score of the clustering.

### 3.9.1 Silhouette Coefficient: Cohesion and Separation of Multi-Dimensional Embeddings

In cluster analysis, a common instance labeling evaluation method is through the interpretation of consistency properties of produced clusters using Silhouettes.<sup>131</sup> Among other scoring measures, Silhouette score (also known as Silhouette width) are known to perform well in literature.<sup>168, 169</sup> Silhouette measures the similarity (cohesion) of values within the cluster when compared among the clusters (separation). The silhouette score of features can be calculated for the output of virtually any clustering algorithm (e.g. *k-means*). The *silhouette coefficient* of point  $i$  takes values in the  $[-1, 1]$  interval, and is defined as:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i), \\ 0, & \text{if } a(i) = b(i), \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i), \end{cases} \quad (3.26)$$

or equivalently:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_i| > 1, \quad (3.27)$$

$$a(i) = \frac{1}{|C_n| - 1} \sum_{j \in C_n, i \neq j} d(i, j), \quad (3.28)$$

where  $a(i)$  represents the average dissimilarity, i.e. mean distance between point  $i$  and every other point  $j \neq i$  within the same cluster,  $n$  is the cluster index,  $C_n$  is the set of all points belonging to cluster  $n$ ,  $|C_n|$  is the cardinality of that set, and values in  $d(i, j)$  are calculated using a dissimilarity measure between points  $i$  and  $j$ .

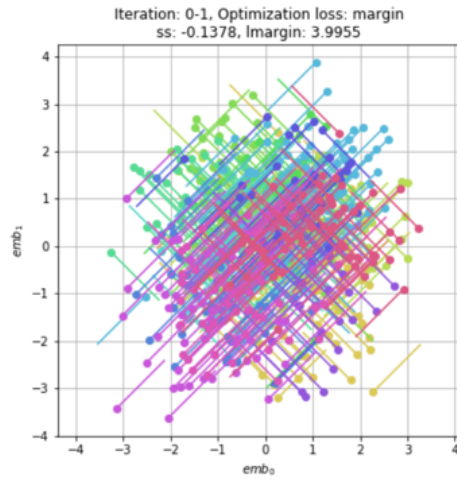
$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j), \quad (3.29)$$

where  $b(i)$  represents the smallest average distance of point  $i$  to all points in all other clusters, and  $k$  is the cluster index that  $i$  does not belong to.

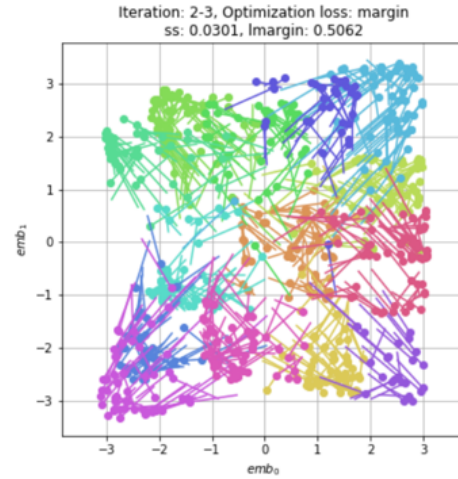
Values of  $s(i)$  close to 1 indicate membership to a well separated cluster, coefficients around 0 indicate that the point may belong to multiple clusters (or any other cluster without making cohesion or separation any worse<sup>169</sup>), while  $-1$  indicates wrong labeling of the point.

**Example 1.** To gain intuition behind the separation and cohesion properties and their relationship to the silhouette score, please consider the examples presented in Figure 3.21. Figure shows a 2-dimensional feature space centered around the origin. There are few hundreds of points being displayed. The ground-truth labels are indicated by distinct colors. The progression is presented in the order from top-left to bottom-right with the increase in silhouette score at each step.

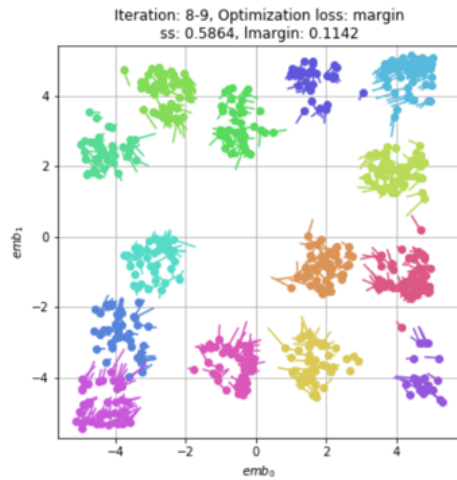
In this work, the silhouette scores are used to evaluate the clustering based on the embedding vectors produced by a neural network. It is important to mention the



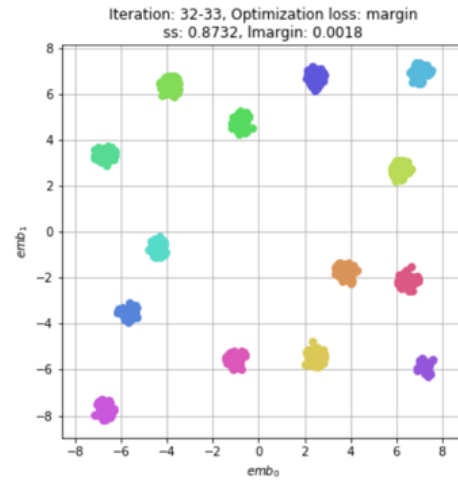
Poor cohesion  
Poor separation  
 $ss_{avg} < 0$



Poor cohesion  
Better separation  
 $ss_{avg} \approx 0$



Better cohesion  
Good separation  
 $ss_{avg} \approx 0.6$



Very good cohesion  
Very good separation  
 $ss_{avg} \geq 0.8$

Figure 3.21: Examples of four clustering situations with respect to cohesion, separation and average silhouette score  $s_{avg}$ .

feasibility of this metric when working with image data. Since, the calculation of per-point measures  $a(i)$ ,  $b(i)$  requires a distance measure  $d(i, j)$  defined between any two points, let's consider the storage needs for distance matrix for few of the image sizes

presented in Table 3.6. The growth of presented values with respect to the image size suggests the need for a way of determining which parts of the image could be excluded from evaluation due to memory constraints.

Image size ( $H \times W$ )	$d$ Matrix Size	# of Elements	Memory @float32
$46 \times 46$	$2116 \times 2116$	4,477,456	$\approx 17$ MBytes
$64 \times 64$	$4096 \times 4096$	16,777,216	64 MBytes
$128 \times 128$	$16384 \times 16384$	268,435,456	1,024 MBytes
$256 \times 256$	$65536 \times 65536$	4,294,967,296	16,384 MBytes
$512 \times 512$	$262144 \times 262144$	68,719,476,736	262,144 MBytes

Table 3.6: Minimum memory requirement assessment for a complete distance matrix  $d$  calculation for typical image sizes.

Consider using the Euclidean Distance between the embedding vectors as an underlying (dis)similarity metric when computing  $d(i, j)$ . Then, to produce an array containing distances between each pixel pair, one can explore the matrix representation of the Euclidean distance between two vectors. Given two vectors  $\vec{A}, \vec{B}$ , the matrix representing the squared distance can be computed as:

$$D^2(\vec{A}, \vec{B}) = \sum_{\text{columns}} \vec{A}\vec{A} - 2 \cdot \vec{A}\vec{B}^T + \left( \sum_{\text{columns}} \vec{B}\vec{B} \right)^T, \quad (3.30)$$

which comes from the expansion of:

$$(x - y)^2 = x^2 - 2xy + y^2 \quad (3.31)$$

It is worthwhile to mention a peculiar implementation detail regarding numerical stability of this method. It was observed, that due to limited precision of 32-bit floating point arithmetics,  $D^2$  can contain extremely small magnitude values for near-exact points. Thus, computing squared root and attempting to proceed with further processing of those values generates numerical errors such as infinities or “not-a-number” warnings. To overcome this, the values are often bottom clamped to 0 or a small bias term in the range of  $(10^{-10}, 10^{-6})$  is added before taking a square root.

### 3.9.2 Discriminative loss function for direct silhouette score maximization

Inspired by the work of De et al.<sup>106</sup> on semantic instance segmentation using embeddings, we decided to explore the idea of training a neural network to produce features that maximize the Silhouette index directly. This section describes formulation of a loss function for direct Silhouette score maximization for the purpose of training a deep CNN to produce easily cluster-able embedding vectors for the task of semantic instance segmentation.

Given the distance matrix  $D \approx \sqrt{D^2(y'_{\text{emb}}, y'_{\text{emb}})}$  defined for all embedding vectors produced by the neural network, and the ground truth cluster assignment image  $ID$ , one could formulate a differentiable loss function attempting to maximize the silhouette score of the clusters produced by the network. Since the value of 1 represents the best per-pixel score, a loss function component maximizing it could be as simple as:

$$L_{\text{ssmax}}(y'_{\text{emb}}, y_{\text{id}}) = \frac{1}{H \cdot W} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} (1 - s[y, x])^2, \quad (3.32)$$

where  $W, H$  are the width and height of the image respectively,  $x, y$  are the coordinates in the image, and  $s[y, x]$  is the silhouette score calculated for each pixel as presented above in (3.27).

The actual calculation of the  $a(i), b(i)$  parts required to obtain the score however is not trivial to implement. However it can be accomplished using matrix algebra as follows (please consider the following equations as a form of pseudo-code):

$$M = [1(ID[i] == ID[j])]_{W \cdot H \times W \cdot H}, \forall i \neq j \in [0, \dots, W \cdot H] \quad (3.33)$$

$$W_a = 1 / \left( \sum_{\text{rows}} M - 1 \right) \quad (3.34)$$

$$V_b = \sum_{\text{rows}} M \quad (3.35)$$

$$W_b = 1 / ([V_b, V_b, \dots, W \cdot H \text{ times}]) \odot (1 - M) + b_{\max}^+ \odot M \quad (3.36)$$

$$a = (D \odot (1 - I) \odot M) \times W_a \quad (3.37)$$

$$b = (D \odot W_b) \times M, \quad (3.38)$$

where  $M$  is the binary matrix indicating which distance comparisons are within the same clusters (1), and which are between clusters (0) - based on the ground truth cluster assignment image  $ID$ ,  $W_a$  are the weights representing the inverse of the number of points in a cluster in (3.28),  $V_b$  is the vector representing the number of elements in each cluster for each point that it contains,  $W_b$  are the weights representing the inverse of the elements in every *other* cluster in (3.29),  $b_{\max}^+$  is the artificially introduced absolutely maximum value of  $b$  introduced to never activate using min function in (3.27),  $I$  is the identity matrix, and finally  $a, b$  are the per-pixel intermediate values for (3.27).

This formulation was implemented using *TensorFlow*<sup>5</sup> machine learning framework, where all of the operations are defined along with accompanying derivative function. This allows for automatic differentiation using the chain rule and training directly using back-propagation. The method was compared to silhouette scoring function available in *sklearn.metrics*<sup>170</sup> python package and validated using side-by-side comparison. Is is however important to again mention certain aspects of numerical stability.

First, the memory requirements for estimation of both  $a(i)$  and  $b(i)$  over a moderately sized image, say  $512 \times 512$  are currently beyond reasonable expectations, as visible in Table 3.6. Thus, a (random) sub-sampling of the ground truth ( $y_{\text{id}}$ ) and

embedding feature maps ( $y'_{\text{emb}}$ ) can be successfully introduced at the price of the speed of convergence. One can focus on evaluating the scores only for the area of interest, if the appropriate mask is available. A parameter  $p_{\text{ssmax}} \in (0, 1)$  represents the ratio of number of pixels processed over the number of pixels belonging to the foreground. It is important to note that number of elements in each cluster is required to be greater or equal to 1, which is not necessarily always satisfied when sub-sampling without additional care in pre-processing of images. These operations must be carefully implemented to avoid divisions by zero.

Presented loss function does not enforce any particular value structure on the produced clusters aside from the fact that the mean intra-distance must be smaller than the smallest inter-distance. As a contrast, the loss function presented in<sup>106</sup> introduces a minimum margin component to their hinged loss allowing for potentially more predictable separation of clusters enforced to spread no more than the value of that margin.

To summarize, presented implementation of the silhouette score, and the following loss function  $L_{\text{ssmax}}$  provide differentiable, direct, per-pixel values allowing for training using back propagation.

### 3.9.3 Discriminative loss function with parametric cluster margins

To contrast with the (non-parametric) clustering score maximization described in Section 3.9.2, a loss function based on the work presented in<sup>106</sup> is presented in this section. Their weakly-supervised approach to representation learning via discriminative loss function is based on a concept of triplet loss by Schroff et al.<sup>119</sup> They address the face recognition problem using deep learning by referencing a dataset of positive and negative examples for person identification, and train a model to be able to generate *same* and *different* responses. When provided with two images, the network can assess the if they are of the same person in a regardless of the pose and lighting conditions. The key

ingredient of their success is the use of *triplet-loss* function used for training.

The *triplet loss* (conceptually depicted in Figure 3.22b) is defined as:<sup>119</sup>

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2, \quad (3.39)$$

where each training example  $x_i = \{x_i^a, x_i^p, x_i^n\}$  consists of three elements: the anchor node  $x_i^a$ , the positive match with respect to the anchor node  $x_i^p$ , and a negative match  $x_i^n$ , and  $\alpha$  is a margin enforced between the positive and negative pairs.

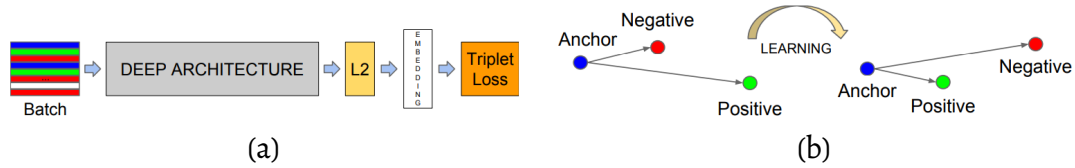


Figure 3.22: Model architecture (a) and conceptual representation of the triplet loss<sup>119</sup>(b).

For multiple object detection, it would not be sufficient to simply rely on a simple triplet loss due to the fact that it is defined to differentiate two opposite examples with an anchor node. The idea however, can be extended to multiple instances using the same principles. What is desired is the cost function that uses a reference image indicating which pixels belong to which instance and not be limited by number of instances in any other way than the memory required for intermediate representations. Such extensions was successfully accomplished by De et al.<sup>106</sup> They expand the concept of the triplet loss from a binary to a multi-category case for the task of semantic instance segmentation. The rest of this section is dedicated to the formulation and implementation of such loss function.

Presented discriminative loss has three components: 1) the intra-cluster cost  $L_{\text{margin\_var}}(y'_{\text{emb}}, y_{\text{id}})$  associated with distance between each member and cluster mean, 2) inter-cluster cost  $L_{\text{margin\_dist}}(y'_{\text{emb}}, y_{\text{id}})$  penalizes the *closeness* of the means of different clusters, and 3) embedding magnitude penalty  $L_{\text{margin\_reg}}(y'_{\text{emb}})$ .

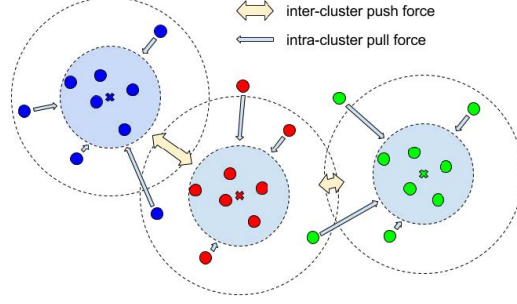


Figure 3.23: Graphical representation of the pointer-to-cluster-mean attraction component of the  $L_{\text{margin\_var}}$  as *intra-cluster pull force* (cohesion), and the *inter-cluster push force* represented by  $L_{\text{margin\_dist}}$  (separation). Boundaries depicted by the dotted lines are the hinged margins  $\phi_{\text{var}}$  (smaller radii) and  $\phi_{\text{dist}}$  (larger radii). Figure from De et al.<sup>106</sup>

$$\begin{aligned}
 L_{\text{margin}}(y'_{\text{emb}}, y_{\text{id}}) &= L_{\text{margin\_var}}(y'_{\text{emb}}, y_{\text{id}}) \\
 &+ L_{\text{margin\_dist}}(y'_{\text{emb}}, y_{\text{id}}) \\
 &+ \alpha_{\text{reg}} L_{\text{margin\_reg}}(y'_{\text{emb}}, y_{\text{id}}),
 \end{aligned} \tag{3.40}$$

The first part of the loss represents the within-cluster variance:

$$L_{\text{margin\_var}}(y'_{\text{emb}}, y_{\text{id}}) = \frac{1}{K} \sum_{k=1}^K \frac{1}{n_k} \sum_{i=0}^{n_k-1} [\|\mu_k - y'_{\text{emb}}[y_{k,i}, x_{k,i}]\| - \phi_{\text{var}}]_+^2, \tag{3.41}$$

where  $K$  is the total number of clusters,  $k$  is the instance number as defined in  $y_{\text{id}}$ ,  $n_k$  is the total number of pixels belonging to cluster  $k$  (number of pixels in  $y_{\text{id}}$  having value of  $k$ ),  $\mu_k$  is  $C$ -dimensional vector representing the mean value of embedding vectors calculated over all pixels in  $y'_{\text{emb}}$  corresponding to pixels in  $y_{\text{id}}$  with value of  $k$ ,  $y_{k,i}$ ,  $x_{k,i}$  are lists of all  $y$  and  $x$  (respectively) image coordinates corresponding to  $i \in 0 \dots n_k$  pixel number belonging to cluster  $k$ .

The mean cluster representation's value on channel  $c$  given output embedding  $y'_{\text{emb}}$

and reference cluster membership  $y_{id}$  is obtained as:

$$\mu_k[c] = \frac{1}{n_k} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} y'_{emb}[y, x, c], \text{ if } y_{id}[y, x] == k, \quad (3.42)$$

where  $n_k$  is the number of  $k$ -valued pixels in  $y_{id}$ ,  $W, H$  are the width and height of the image, and  $k$  is the cluster index.

The second component represents the average distance between cluster means:

$$L_{margin\_dist}(y'_{emb}, y_{id}) = \frac{1}{K(K-1)} \sum_{k=1}^K \sum_{l=1, l \neq k}^K [2\phi_{dist} - \|\mu_k - \mu_l\|]_+^2 \quad (3.43)$$

which is calculated pair-wise for each pair of clusters, where  $K$  is the total number of clusters,  $k, l$  are the indices of the first and second cluster in considered pair respectively, and  $\mu_k, \mu_l$  are cluster means.

Finally, the embedding magnitude penalty is defined as:

$$L_{margin\_reg}(y'_{emb}) = \frac{1}{K} \sum_{k=1}^K \|\mu_k\|, \quad (3.44)$$

which penalizes the magnitudes of cluster means.

The  $\phi_{var}$  and  $\phi_{dist}$  are the cohesion and separation margins. In<sup>106</sup> authors advise using  $\phi_{dist} > 2\phi_{var}$  to encourage each embedding to be closer to all embeddings of its *own* cluster than to any *other* cluster. The conceptual depiction of this loss function is presented in Figure 3.23.

It is important to mention the  $[x]_+$  hinge operator (positive part) used in the above description, which is equivalent and can be implemented as  $\text{ReLU}(x)$  operation available in machine learning frameworks. The purpose of that component in this context is to avoid activating for elements which already satisfy certain criteria, such as falling below

certain margin  $\phi$ .  $\text{ReLU}(x)$  is defined as:

$$\text{ReLU}(x) = \begin{cases} x, & \text{for } x > 0 \\ 0, & \text{for } x \leq 0 \end{cases} \quad (3.45)$$

The main advantage of this formulation is a fairly low memory footprint as it does not require per-pixel-pair distance. Instead, it uses per-pixel distance to the cluster mean which requires  $H \cdot W \times C \times K$  array at most. Table below shows the memory requirements for the largest array used in the estimation of described loss function for number of embedding channels  $C = 12$  and number of cluster  $K = 32$ , which represent the values used in this work.

Image size ( $H \times W$ )	# of Pixels	C	K	# of elements	Size @float32
$46 \times 46$	2116	12	32	812,544	$\approx 3$ MBytes
$64 \times 64$	4096	12	32	1,572,864	6 Mbytes
$128 \times 128$	16384	12	32	6,291,456	24 Mbytes
$256 \times 256$	65536	12	32	25,165,824	96 Mbytes
$512 \times 512$	262144	12	32	100,663,296	384 Mbytes

Table 3.7: Memory requirement for per-pixel, per-cluster storage for 32 instances with 12-dimensional embeddings for common image sizes.

De et al.<sup>106</sup> suggest that having a controllable margin, the instance segmentation algorithm can be as simple as picking a random point, thresholding around the  $\phi_{\text{var}}$  and assigning the first label to all points falling within the threshold. Then we can remove the labeled points and proceed with another randomly selected point until samples are exhausted.

### 3.9.4 Speed of Convergence Analysis using Silhouette Score

In Section 3.9.2 we are proposing a novel loss function for training multi-dimensional embeddings for the task of semantic instance segmentation. We contrast that loss function with the approach by De et al.<sup>106</sup> who proposed a discriminative loss function for semantic instance segmentation described in Section 3.9.3. The PIGSEG96 dataset

(described in Section 3.4.1) is used for the analysis and it contains images of pig-shaped clusters.

We are using the *single example over-fitting* approach presented by De et al.<sup>106</sup> to illustrate the convergence properties of the two loss functions. We are using hyper-parameters listed in Table 3.8 and run our experiments for the same initial values for each loss function. We use silhouette score with respect to the number of optimization iterations as our metric. Figure 3.24 (a) illustrates that  $L_{\text{margin}}$  converges significantly faster for the same learning rate of the optimizer. When observing the values of  $L_{\text{margin}}$  during optimization using  $L_{\text{ssmax}}$  (red curve in Figure 3.24b) it is visible that at the early stages of optimization the loss functions are *in agreement*.

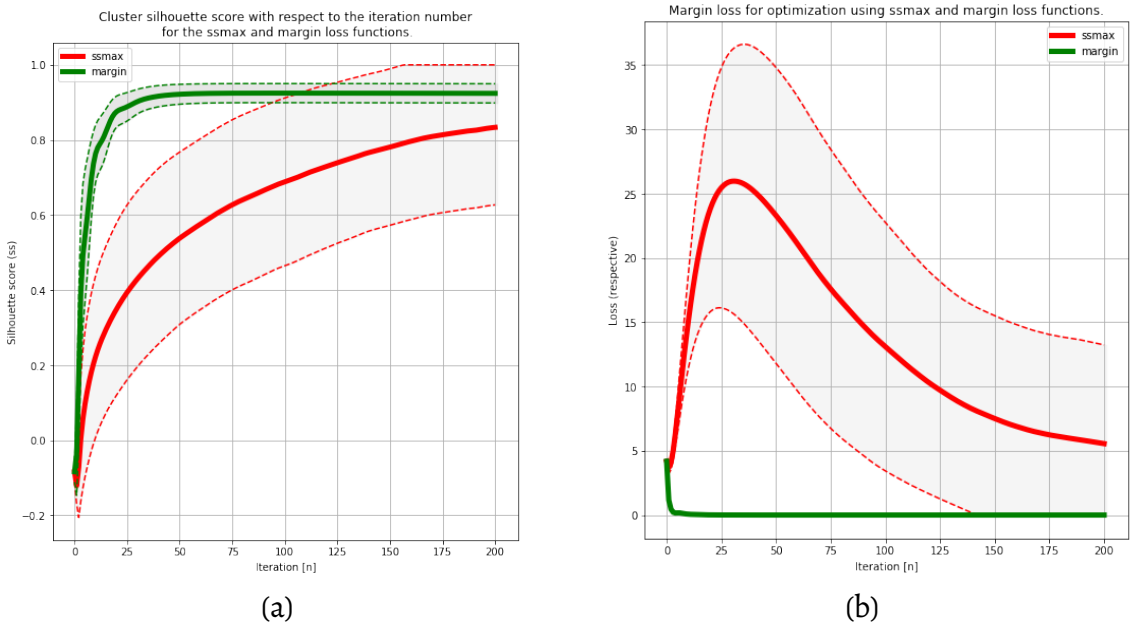


Figure 3.24: Depiction of the speed of convergence of the two considered loss functions:  $L_{\text{ssmax}}$  and  $L_{\text{margin}}$  when using the over-fitting method for all examples in PIGSEG96 dataset. Silhouette score with respect to the number of optimization iterations calculated for all examples when training using both losses (a), and the value of  $L_{\text{margin}}$  with respect to optimization iteration for all examples when trained using both losses (b).

In order to determine the cause of such significant difference between  $L_{\text{margin}}$  and  $L_{\text{ssmax}}$  in terms of convergence, we also captured the values of the  $L_{\text{margin\_var}}$  and

$L_{\text{margin\_dist}}$  components of the  $L_{\text{margin}}$  loss function while optimizing using either of the losses. The results are presented in Figure 3.25.

When looking at Figure 3.25 (b) representing the comparison of the  $L_{\text{margin\_dist}}$  when optimizing using either of the loss functions, it is visible that both functions are *in agreement* and the component responsible for *separation* is correctly implemented in  $L_{\text{ssmax}}$ . When investigating Figure 3.25 which depicts the  $L_{\text{margin\_var}}$  component of the  $L_{\text{margin}}$  loss, it is visible that the *cohesion* is not handled correctly by  $L_{\text{ssmax}}$ .

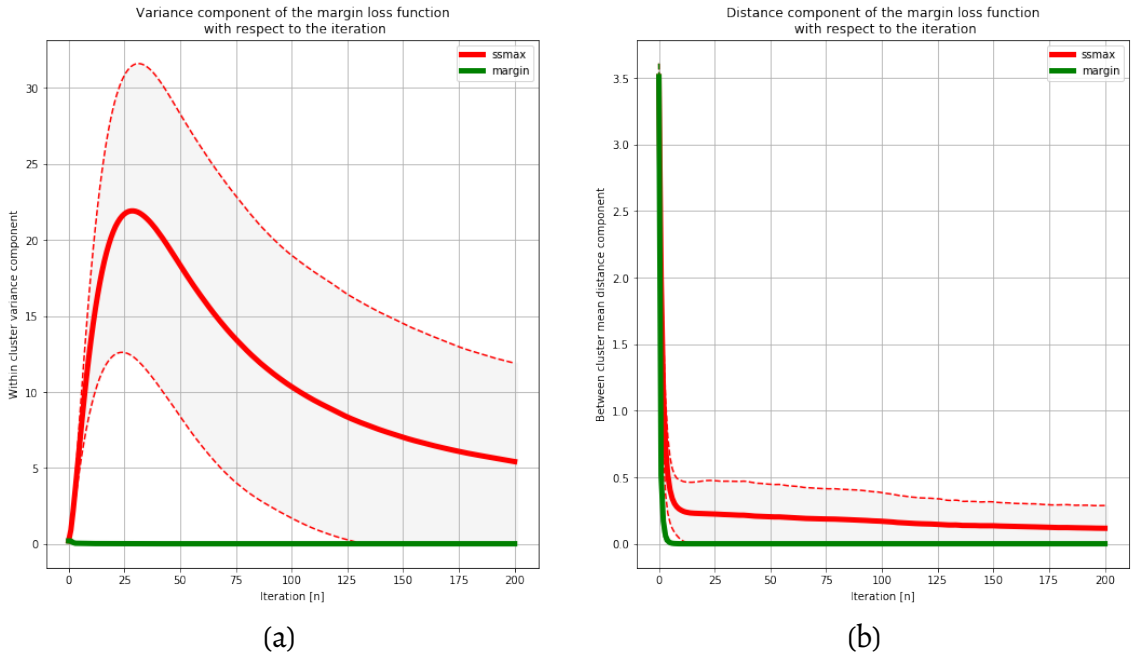


Figure 3.25: Values of the components of the  $L_{\text{margin}}$  with respect to the optimization iterations of  $L_{\text{ssmax}}$  (red curves) and  $L_{\text{margin}}$  (green curves); Within-cluster variance component  $L_{\text{margin\_var}}$  (a), and between-cluster means distance component  $L_{\text{margin\_dist}}$  (b).

### 3.9.5 Speed of convergence with respect to the number of clusters

To complete the analysis and determine the properties of the  $L_{\text{ssmax}}$  and  $L_{\text{margin}}$  loss functions, we varied the number of clusters present in the population and performed the optimization like in Section 3.9.4. Table 3.9 contains the numerical results of this analysis

Parameter	Value	Description
$W$	64	Target width of the input images
$C$	2	Number of embedding channels
$it_{\max}$	200	Maximum number of iterations
$k$	$\in \{4, 10, 11, 14\}$	Number of clusters
$ss_{\text{converged}}$	0.85	Minimum silhouette score to consider convergence
Solver	Adam <sup>152</sup>	Optimization solver
$\lambda_{\text{margin}}$	1.0	Learning rate for $L_{\text{margin}}$
$\phi_{\text{dist}}$	1.0 (A), 2.0 (B)	Between-clusters margin for $L_{\text{margin}}$
$\phi_{\text{var}}$	0.4 (A), 0.0 (B)	Within-cluster margin for $L_{\text{margin}}$
$\alpha_{\text{reg}}$	$10^{-4}$	Weight of the regularization term for $L_{\text{margin}}$
$\lambda_{\text{ssmax}}$	1.0	Learning rate for $L_{\text{ssmax}}$
$p_{\text{ssmax}}$	1.0	Proportion of foreground points to be processed by $L_{\text{ssmax}}$

Table 3.8: Hyper-parameters used for optimization of  $L_{\text{margin}}$  and  $L_{\text{ssmax}}$  using the over-fitting method.

for the number of clusters  $k = 2, 3, \dots, 14$ . Figures 3.26 (a) and (b) show the optimization progress measured by the silhouette score with respect to the iteration number for  $L_{\text{ssmax}}$  and  $L_{\text{margin}}$  respectively. Figure 3.26 (c) indicates that the  $L_{\text{margin}}$  always exhibits *higher* rate of convergence when compared to  $L_{\text{ssmax}}$  - regardless of the number of clusters  $k$ .

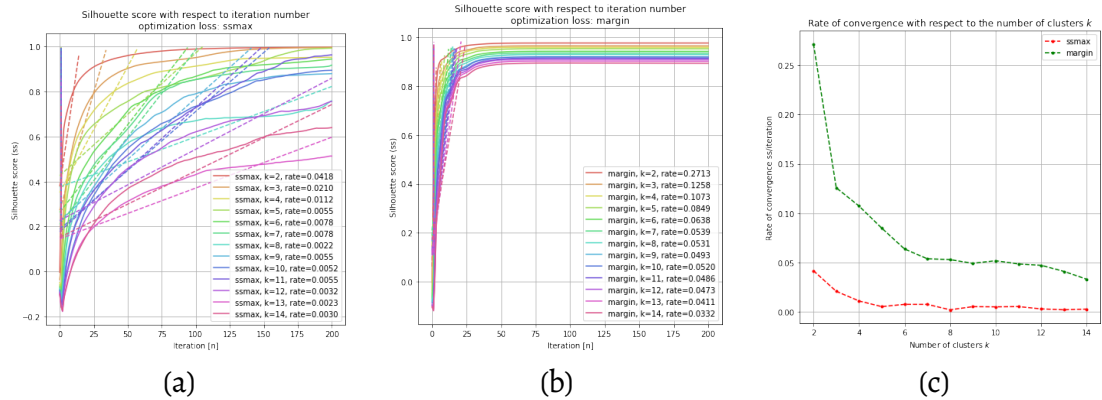


Figure 3.26: Evaluation of the speed of convergence expressed as silhouette score with respect to the number of clusters  $k$ . Curves of the silhouette score with respect to the iteration for optimization using  $L_{\text{ssmax}}$  (a), and  $L_{\text{margin}}$  (b). Comparison of the convergence rates for with respect to the number of clusters  $k$  (c) for  $L_{\text{ssmax}}$  (red curve), and  $L_{\text{margin}}$  (green curve).

Number of clusters $k$	Rate for $L_{ssmax}$	Rate for $L_{margin}$
2	0.0418	0.2713
3	0.0210	0.1258
4	0.0112	0.1073
5	0.0055	0.0849
6	0.0078	0.0638
7	0.0078	0.0539
8	0.0022	0.0531
9	0.0055	0.0493
10	0.0052	0.0520
11	0.0055	0.0486
12	0.0032	0.0473
13	0.0023	0.0411
14	0.0030	0.0332

Table 3.9: Rate of convergence expressed in terms of silhouette score for both loss functions for number of clusters  $k = 2, 3, \dots, 14$ .

### 3.9.6 Speed of convergence with respect to the number of embedding channels

We also varied the number of embedding channels  $C$  in order to determine the convergence properties of both loss functions. Numerical results of this analysis are listed in Table 3.10 for  $C = 2, 3, \dots, 62$ . We initially found a case for  $L_{ssmax}$  converging faster than  $L_{margin}$  (for mode A with  $\phi_{var} = 0.4, \phi_{dist} = 2.0$ ) but reduction of the margins to mode B with  $\phi_{var} = 0.0, \phi_{dist} = 1.0$  quickly indicated that  $L_{margin}$  is always a preferred choice over  $L_{ssmax}$  from the speed of convergence point of view. This finding is illustrated in Figure 3.27.

Figures 3.27 (a) and (b) present the optimization curves for  $L_{ssmax}$  and  $L_{margin}$  (mode A) respectively.

## 3.10 Training using Backpropagation

All models presented in this work have the benefit of being end-to-end trainable which allows for training using backpropagation. The end-to-end trainability can be defined as the situation in which the loss can be calculated immediately after the forward pass given the ground-truth data. It is possible due to the fact, that the loss functions is

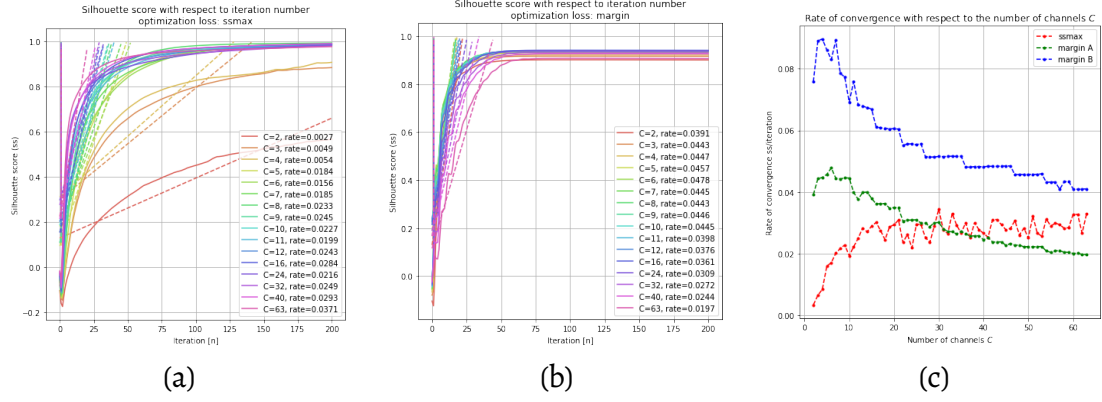


Figure 3.27: Evaluation of the speed of convergence expressed as silhouette score with respect to the number of channels  $C$ .

$C$	Rate for $L_{ssmax}$	Rate for $L_{margin A}$	Rate for $L_{margin B}$	$C$	Rate for $L_{ssmax}$	Rate for $L_{margin A}$	Rate for $L_{margin B}$
2	0.0034	0.0391	0.0758	33	0.0330	0.0271	0.0516
3	0.0064	0.0443	0.0891	34	0.0291	0.0264	0.0515
4	0.0084	0.0447	0.0897	35	0.0266	0.0272	0.0516
5	0.0159	0.0457	0.0861	36	0.0300	0.0264	0.0481
6	0.0170	0.0478	0.0830	37	0.0251	0.0257	0.0480
7	0.0201	0.0445	0.0893	38	0.0300	0.0257	0.0482
8	0.0216	0.0443	0.0785	39	0.0277	0.0258	0.0482
9	0.0228	0.0446	0.0773	40	0.0266	0.0244	0.0482
10	0.0193	0.0445	0.0692	40	0.0266	0.0244	0.0482
11	0.0222	0.0398	0.0759	41	0.0253	0.0252	0.0483
12	0.0250	0.0376	0.0683	42	0.0310	0.0238	0.0483
13	0.0283	0.0398	0.0679	43	0.0310	0.0238	0.0483
14	0.0271	0.0400	0.0674	44	0.0294	0.0238	0.0484
15	0.0289	0.0379	0.0669	45	0.0277	0.0226	0.0484
16	0.0301	0.0361	0.0612	46	0.0282	0.0233	0.0484
17	0.0274	0.0362	0.0608	47	0.0307	0.0227	0.0456
18	0.0245	0.0363	0.0606	48	0.0260	0.0228	0.0457
19	0.0287	0.0347	0.0605	49	0.0283	0.0222	0.0456
20	0.0293	0.0350	0.0606	50	0.0251	0.0223	0.0457
21	0.0309	0.0348	0.0604	51	0.0321	0.0222	0.0457
22	0.0236	0.0305	0.0552	52	0.0266	0.0223	0.0457
23	0.0262	0.0308	0.0555	53	0.0256	0.0223	0.0459
24	0.0218	0.0309	0.0556	54	0.0312	0.0208	0.0431
25	0.0296	0.0310	0.0553	55	0.0289	0.0204	0.0432
26	0.0295	0.0299	0.0557	56	0.0298	0.0209	0.0432
27	0.0253	0.0299	0.0515	57	0.0294	0.0209	0.0409
28	0.0236	0.0288	0.0513	58	0.0282	0.0204	0.0433
29	0.0289	0.0300	0.0514	59	0.0284	0.0204	0.0434
30	0.0345	0.0301	0.0515	60	0.0327	0.0201	0.0410
31	0.0276	0.0281	0.0515	61	0.0327	0.0201	0.0408
32	0.0265	0.0272	0.0516	62	0.0266	0.0197	0.0410

Table 3.10: Rate of convergence for  $L_{ssmax}$  and  $L_{margin}$  (configuration variants A and B, please refer to Table 3.8) for variable number of clusters  $C = 2, 3, \dots, 62$ .

differentiable and no external post-processing is required to assess the residual with respect to the output and ground truth.

As our networks are designed to estimate multiple different types of outputs, it's



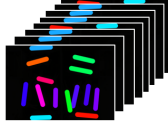


				
Image	Keypoints	Part Affinity Fields	Foreground Mask	Pixel ID
$x$	$y_{kps}$	$y_{paf}$	$y_{fg}$	$y_{id}$
3 Channels	4 Channels	8 Channels	1 Channel	1 Channel
Section 3.2	Section 3.3.2	Section 3.6	Section 3.5	Section 3.4

Table 3.11: Input  $x$  and the corresponding ground-truth information provided for training using backpropagation (in columns). First row shows a small visual representation of the data. Names of the features are provided in the second row, followed by the mathematical symbols used in the rest of this section. Fourth row indicates the number of channels used by the feature. The Section number indicating the description of particular feature is provided in the last row.

objective consists of minimizing a composite loss function (Equation 3.47). Generally, for the model  $f(x|\theta) = y'$  the training procedure is aimed to adjust the model coefficients  $\theta$  in the presence of ground-truth data  $y$ :

$$\theta = \arg \min L(f(x|\theta), y, \theta), \quad (3.46)$$

where  $L(y', y)$  is the loss function. Here, the loss function  $L(y', y)$  consists of 5 components:

$$\begin{aligned}
 L(y', y, \theta) = & \alpha_{kps} L_{kps}(y'_{kps}, y_{kps}) \\
 & + \alpha_{paf} L_{paf}(y'_{paf}, y_{paf}) \\
 & + \alpha_{emb} L_{emb}(y'_{emb}, y_{id}) \\
 & + \alpha_{fg} L_{fg}(y'_{fg}, y_{fg}) \\
 & + \alpha_{reg} L_{reg}(\theta),
 \end{aligned} \quad (3.47)$$

where  $\alpha_{kps}$ ,  $\alpha_{paf}$ ,  $\alpha_{emb}$ ,  $\alpha_{fg}$ ,  $\alpha_{reg}$  are the learning rate coefficients for the body part detections, part affinity fields, embeddings, foreground, and the regularization term.

The loss functions are responsible for defining error in:  $L_{kps}$  - body part detection,  $L_{paf}$  - part affinity fields,  $L_{emb}$  - embeddings,  $L_{fg}$  - foreground mask, and  $L_{reg}$  - weight regularization. Table 3.11 provides a visualization of the input and ground-truth data provided for the training.

Body part detections, part affinity fields and foreground mask are all trained using masked mean squared error as:

$$L_{mMSE}(y', y, m) = \frac{1}{\sum m} \sum m \odot ((y - y')^2), \quad (3.48)$$

where  $m$  is the binary mask defined in the area where training needs to occur, thus  $\sum m$  is the effective number of pixels to which the loss is applied, and  $\odot$  is the element-wise multiplication operator.

**Example 1.** Consider the case of training the foreground mask like presented in Figure 3.28. Equation 3.48 provides the means to train the model only with respect to the specific, selected part of the image. Figure 3.28 (c) shows the desired confident ground truth value  $y_{fg}$  that is only valid within the region defined by a mask  $m$  (Figure 3.28 b). Use of selective loss function allows to only penalize the errors within the area for which the data is certain.

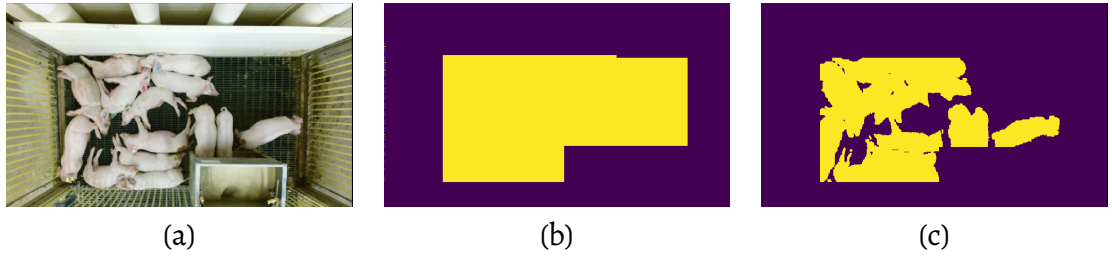


Figure 3.28: Example of *selective* foreground mask estimation training with the input image  $x$  (a), training mask  $m$  (b), and expected  $y_{fg}$  (c).

Embedding loss is defined using different loss as the actual target values in  $y_{id}$  are represented to only indicate the cluster membership and not the specific values of the

embedding vectors. Loss functions for the embedding training are described in sections 3.9.2 and 3.9.3 and provide error values related to the separation and cohesion properties of the embedding clusters and not the actual values. Thus, we consider our approach to training such embeddings as a weakly-supervised approach.

Proposed model are Directed Acyclic Graphs, thus they can be represented as a *superposition* or its layers as:

$$y' = f_{l-1}(\dots f_1(f_0(x_0, \theta_0), \theta_1) \dots, \theta_{l-1}), \quad (3.49)$$

where  $y'$  is the output produced by the model,  $l$  is the number of operations,  $\theta$  contains trainable model parameters, and  $f_l(x_l, \theta_l)$  are the operations performed by the model (loosely speaking *layers* or nodes in DAG). Given ground truth data  $y$ , and the loss function  $L(y, y')$ , the role of the optimizer is to adjust parameters  $\theta$  along the direction yielding decrease in the loss function. Due to the complexity of the model, and its non-linearity, this cannot be done in a single step and needs to be performed iteratively. Generally, model adjustment using gradient descent algorithm is represented as:

$$\theta_{t+1} = \theta_t - \alpha_t \frac{\partial L}{\partial \theta_t}, \quad (3.50)$$

where  $t$  is the time step or number of iteration,  $\alpha_t \in \mathbb{R}_+$  is the learning rate,  $\theta$  contains model parameters,  $\frac{\partial L}{\partial \theta_t}$  represents the positive gradient of the loss function  $L$  with respect to parameters  $\theta$  at time  $t$ .

The backpropagation algorithm then uses the *superposition* interpretation of the model and applies *chain rule* to adjust model weights - backwards - from the loss function  $L(y, y')$  all the way to the first subset of coefficients  $\theta_0$ , while for each operation  $f_l(x_l, \theta_l)$  estimating approximate residual terms with respect to both, model coefficients and inputs.

Modern machine learning frameworks provide a variety of optimization solvers

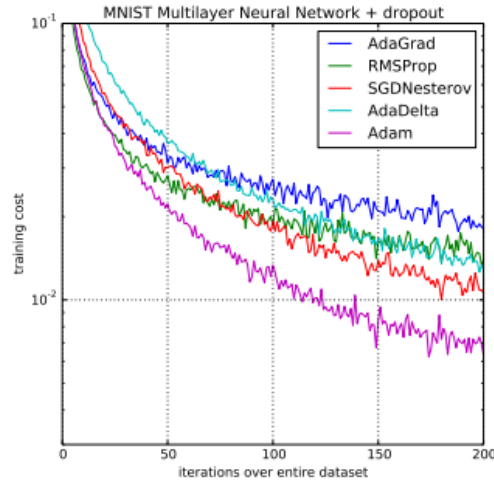


Figure 3.29: Speed of convergence of popular optimization methods over the MNIST dataset indicating superior performance of Adam optimizer. Figure from.<sup>152</sup>

based on the Stochastic Gradient Descent method. The *stochasticity* comes from the fact, that adjustments are performed based on small subsets of examples and the ensemble. Thus, a learning rate  $\alpha_t$  is generally small when training neural networks and the procedure spans over extended periods of time. The main differences between them involve the use of *momentum* and adaptability of the learning rate  $\alpha_t$  depending on the training progress. Adaptation of the learning rate is a research topic on its own. We used the starting learning rate  $\alpha = 10^{-4}$ .

Among many optimizers available in the modern machine learning frameworks, Adam,<sup>152</sup> AdaGrad<sup>171</sup> and RMSProp<sup>172</sup> are often selected by researchers as they maintain adaptive, per-parameter learning rates, which improves stability in the presence of sparse gradients based on the mean of recent weight gradients when compared to base Stochastic Gradient Descent. We selected the Adam optimizer to train all presented models as it was shown in the literature to combine the best properties of AdaGrad and RMSProp and is well-suited for problems with sparse gradients (such as the ones produced by selective masking or loss functions defined only in the part of the image). A comparison of popular optimization methods using the MNIST<sup>173</sup> dataset is presented in

figure 3.29.

We trained each of our models until *convergence* which we define as the state in which the loss with respect to the training examples  $L_{\text{train}}$  is lower or equal than the loss with respect to the testing  $L_{\text{test}}$  examples. We used samples from the *test:seen* to calculate  $L_{\text{test}}$  using a different, randomly selected example after every training iteration. We used batch size of 1 and obtained curves for  $L_{\text{train}}$  and  $L_{\text{test}}$  and applied a windowed-average to determine the stop conditions. The time necessary to train each model varied between days to about a week. The extensive period of time required for training is mostly attributed to the on-the-fly example generation. We used augmentation techniques as described in Section 3.7.

### 3.11 Pose Estimation using Body Part Detections and Part Affinity Fields

We present two main object detection methods in this work: one using pose estimation and another using semantic instance segmentation. Pose Estimation is a task of defining and object through the position and arrangement of its parts in the image. We perform this based on the body part locations (keypoints / landmarks) and the associations between them based on the outputs of our neural networks. This section describes the performance of our models' ability to determine the location of pig body parts (left ear, right ear, shoulder, and tail) based solely on the input images of pigs in a pen with respect to manually annotated ground-truth.

We are presenting two methods of pose estimation for pigs based on the work of Cao et al.<sup>161</sup> referred to as *OpenPose*, and our prior contributions.<sup>160</sup> Those methods are 1) our reimplement of the OpenPose method for pigs and 2) a Hybrid method that still relies on keypoints but the matching is done based on the similarity in the embedding space.

Both of our pose estimation methods start with inferring the location of keypoints from the color images. The inference is performed using either of our neural networks OP or UNET. Each of our networks produces a real-valued image  $y'_{\text{kps}}$  with values between 0 and 1 (keypoint detection heatmap). This heatmap is a *dense* representation of the predicted locations of keypoints in the form of a two-dimensional estimate of a likelihood conditioned of the input image. The *location* in the image (in pixels) corresponds to the spatial component, and the *value* of the pixel corresponds to the likelihood of the presence of a keypoint at that location. In order to produce a *sparse* representation of the detected keypoint locations, we perform non-maximum suppression parametrized by the threshold  $th_{\text{kp}}$ .

Following the methodology presented in,<sup>160,161</sup> tracking by detection begins with processing body part location heatmaps obtained by passing an image through the neural network. Let's agree, that a set  $y' = \{y'_{\text{kps}}, y'_{\text{paf}}\} = f(x)$  are respectively, the body part location heatmap and part affinity fields estimations. In order to use the bipartite assignment method to determine which keypoint belongs to which instance, first, each part location needs to be extracted from the  $h \times w \times 4$  dense representations in  $y'_{\text{kps}}$ . Precise estimates of the keypoints are represented by the peaks in the heatmaps. A visualization of keypoint location heatmaps is presented in Figure 3.30.

### 3.11.1 Keypoint Detection using Non-Maximum Suppression

The goal of this step is to convert the dense, image-like encoding of the likelihood of the presence of the sub-instance landmarks into a sparse, image coordinate-based representation allowing for further processing. To find the sparse coordinates of the peaks in the heatmaps produced by the network, a local max response filter (Equation 3.51) can be applied to the image - this method is known as *non-maxima suppression* and can be described as determination of the set of  $x, y$  coordinate pairs for which the

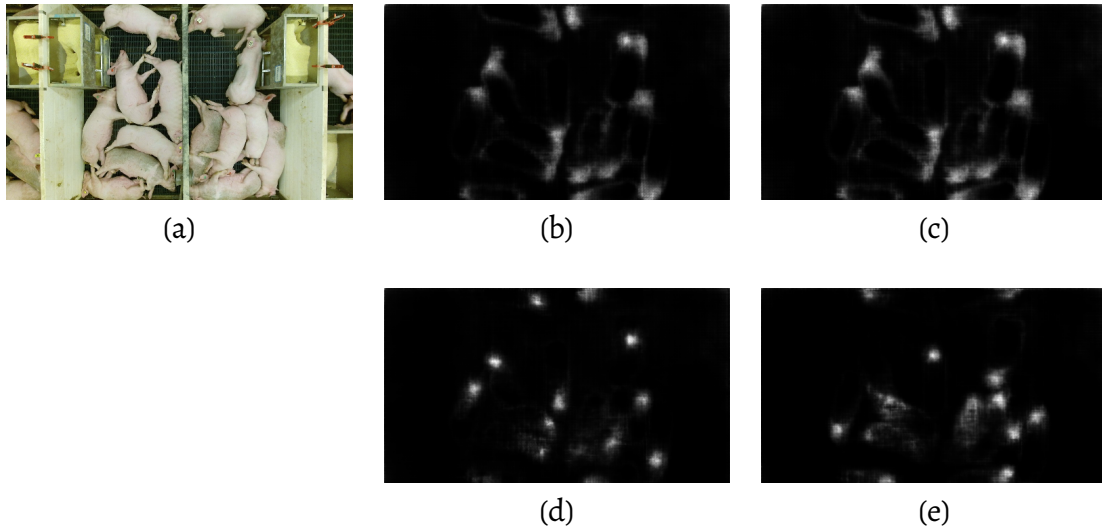


Figure 3.30: Visualization of the keypoint heatmaps produced by the  $\text{UNET}_{\text{ssmat}}$  network based on the example 2 of the PIGSEG96 dataset. Input image (a), heatmap for the left ear (b), right ear (c), shoulder (c), and tail (d). Greyscale encoding of the values: 0: black, 1: white.

heatmap value is spatially local maximum:

$$\{\text{peaks}_p\} = \left\{ \{x, y\} \text{ if } y'_{\text{kps},p}[y, x] == \max(R(x, y, y'_{\text{kps},p}, r)) \right\}, \quad (3.51)$$

where  $\{\text{peaks}_p\}$  is the resulting set of coordinates,  $p \in \{l, r, s, t\}$  is the label of the body part (corresponding to the left ear, right ear, shoulder, and tail respectively),  $x, y$  are the coordinates in the image,  $R(x, y, I, r)$  is the region extraction operation defined as follows:

$$R(x, y, I, r) = I \left[ y - \frac{r-1}{2}, \dots, y + \frac{r-1}{2}, x - \frac{r-1}{2}, \dots, x + \frac{r-1}{2} \right] \quad (3.52)$$

$$\forall x \in \left[ \frac{r-1}{2}, \dots, w - \frac{r-1}{2} \right], \forall y \in \left[ \frac{r-1}{2}, \dots, h - \frac{r-1}{2} \right],$$

where  $w, h$  are the width and height of the image  $I$ , and  $r$  is the odd-number representing the size of the sliding window of the peak detector. Confidence of the detection is determined by the height of the peak  $\max(R(x, y, I'_{\text{kps}}, r))$ .

**Example 1.** To better understand the concept of the non-maximum-suppression

consider the following example. Let's assume a  $21 \times 21$  image containing a heatmap as visible in Figure 3.31 (a). There are two smooth hot spots with the centers (peaks) at the pixel coordinates of  $(5, 5)$  and  $(17, 17)$ . In order to obtain the sparse positions of those peaks, let's apply a sliding-window-based non-maximum suppressor like the one described in Equation 3.51. The output (Figure 3.31 (b)) is a binary image and each non-zero pixel is considered as a position of the detected peak.

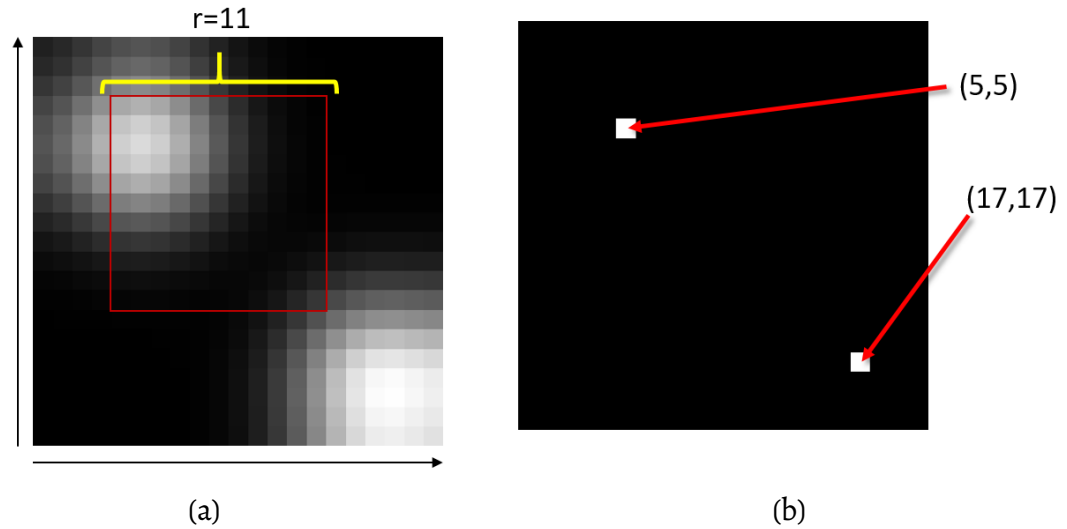


Figure 3.31: Visualization of the input (a) and the output (b) of the local max response filter (non-maximum-suppressor) with the size of the sliding window  $r = 11$ . Dimensions of the images are  $21 \times 21$ . Red square represents the sliding window operation.

First stage of processing is complete when peaks for all body parts are estimated according to equation 3.51 with the resulting superset

$$\text{peaks} = \{\text{peaks}_l, \text{peaks}_r, \text{peaks}_s, \text{peaks}_t\}.$$

### 3.11.2 Bipartite Matching

Like in the work of Cao et al.,<sup>65,161</sup> the presented pose detection method revolves around greedy, sequential bipartite matching of the detected keypoints. In our approach we only resolve the matching between the shoulders and tails. Thus, our we only use the  $\text{peaks}_s$  and  $\text{peaks}_t$  subsets of the detected keypoint locations.

To match all detected shoulders to all detected tails we formulate a linear sum assignment problem and solve it using Hungarian Algorithm. We define two partitions, one for the shoulders and another for the tails. We obtain keypoint location estimates using the neural network like described in Section 3.11.1. We use the matching costs based on the features from the neural networks using Part Affinity Fields (Section 3.11.3) and Embeddings (Section 3.11.4). The following paragraph describes the general case of the bipartite matching problem.

Having two partitions  $A$  and  $B$  with their respective number of instances  $n_A, n_B$ , a problem instance is described by a cost matrix  $C$ , where  $C[i, j]$  is the cost of assigning element  $i$  from partition  $A$  to element  $j$  from partition  $B$ . The goal is to find a complete assignment of all elements  $A_i \forall i \in [0, \dots, n_A - 1]$  to all elements  $B_j \forall j \in [0, \dots, n_B - 1]$  with minimum total cost. Formally, a boolean matrix  $X$  is estimated such that  $X[i, j] = 1$  if the algorithm determined that  $A_i$  should be associated with  $B_j$ , and  $X[i, j] = 0$  if not. The optimal assignment then has the cost:

$$\min \sum_{i=0}^{n_A-1} \sum_{j=0}^{n_B-1} C_{i,j} X_{i,j} \quad (3.53)$$

### 3.11.3 Part Affinity Fields

Next step involves processing based on the part affinity fields estimates from  $y'_{\text{paf}}$  to estimate the matching cost of keypoint matching using bipartite matching. The output  $y'_{\text{paf}}$  is formatted in a dense image-like fashion as a channel-wise concatenation of the images composed  $x$  and  $y$  coordinates of the of the  $\vec{d}_{i,j}$  vectors as presented in figure 3.14. The associations estimated by the neural network are:  $\{\{l \rightarrow r\}, \{l \rightarrow s\}, \{s \rightarrow t\}\}$  and should correspond to the direction of vectors drawn between the estimated keypoint locations.

First, the assignments between shoulders and tails are resolved as a solution of

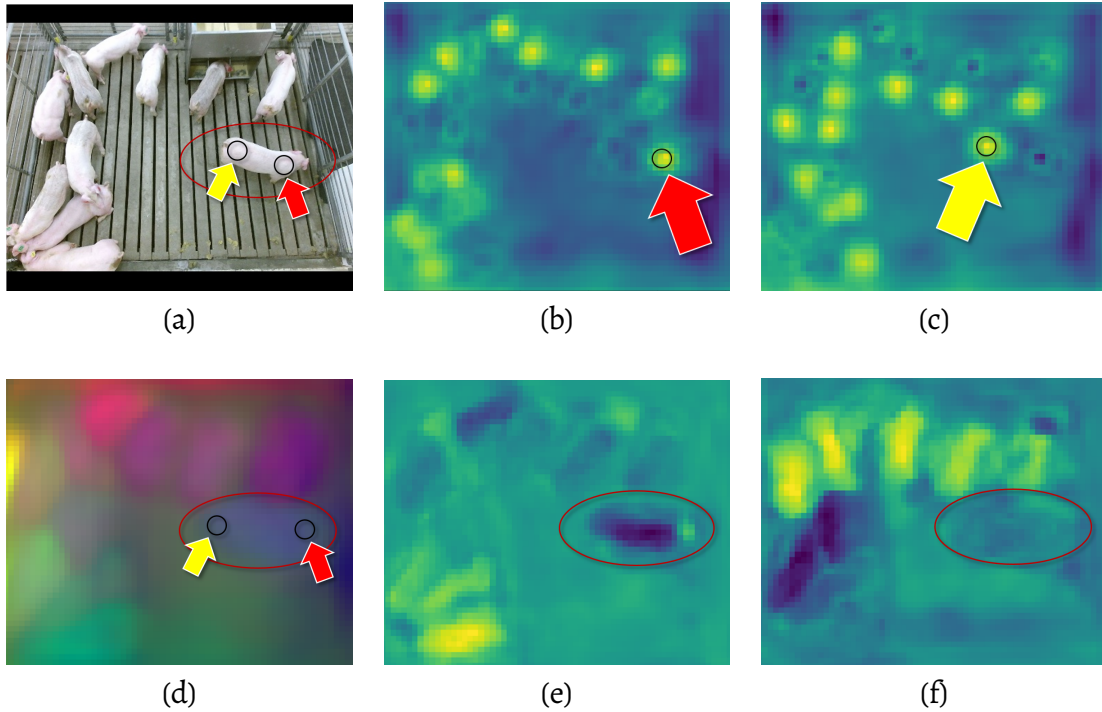


Figure 3.32: Input image with overlaid annotations (a), heatmap of the head keypoint (b), heatmap of the tail keypoint (c), PCA visualization of the embeddings  $y'_{\text{emb}}$  (c),  $x$  component of the part affinity field between shoulder and tail (e),  $y$  component of the part affinity field between shoulder and tail (f). True position of the shoulder is indicated by the red arrow and the true position of the tail is indicated using the yellow arrow. Outputs generated by the  $\text{OP}_{\text{margin}}$  network. Image was captured in May 2017.

linear sum assignment problem using Hungarian Algorithm like described in section 3.11.2. To provide values for the matrix  $C$ , the part affinity field estimates  $y'_{\text{paf}}$  need to be sampled, and a dissimilarity metric needs to be defined, such that it provides low values (or close to 0) for a proper match, and high values for improper assignment. Figure 3.32 presents an example of the data used in the processing.

For each assignment defined as  $\{p, q\}$ , the location estimates of corresponding parts  $\text{peaks}_p, \text{peaks}_q$  are considered in descending order of confidence. For each pair of considered source and destination keypoints

$\vec{k}_0 = [x_0, y_0] \in \text{peaks}_p, \vec{k}_1 = [x_1, y_1] \in \text{peaks}_q$ , the part affinity field is sampled at

locations:

$$\mathbf{s}\vec{p}_s = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \frac{s}{s_{\max} \cdot \|\vec{k}_1 - \vec{k}_0\|} \cdot \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \end{bmatrix}, \quad (3.54)$$

where  $\mathbf{s}\vec{p}_s$  is a 2-dimensional vector with elements corresponding to the  $x, y$  coordinates in  $y'_{\text{paf}}$ ,  $s$  is the step number  $\in [0, \dots, s_{\max}]$ , and  $s_{\max}$  is the number of sampling points along the vector  $\vec{d}_{i,j} = \vec{k}_1 - \vec{k}_0$ .

After obtaining the sampling coordinates, the part affinity estimate image is sampled at each location indicated by the  $\mathbf{s}\vec{p}_s$  resulting in a set of:

$$D_{i,j} = \{ [y'_{\text{paf}}[\mathbf{s}\vec{p}_0[1]], y'_{\text{paf}}[\mathbf{s}\vec{p}_0[0], l]] , \dots , [y'_{\text{paf}}[\mathbf{s}\vec{p}_{s_{\max}}[1]], y'_{\text{paf}}[\mathbf{s}\vec{p}_{s_{\max}}[0], m]] \}, \quad (3.55)$$

where  $i, j$  correspond to indices in  $\text{peaks}_p$ , and  $\text{peaks}_q$  respectively, and  $l, m$  are the channel numbers in  $y'_{\text{paf}}$  corresponding to corresponding to  $x$  and  $y$  coordinates of the vector for affinity between parts  $p$  and  $q$ ,

Normalized dot product (*cosine similarity*) was selected as a metric of similarity between the detected displacement of peak  $i$  and  $j$ . Thus, the *similarity* between the estimated part affinity fields and a vector indicating the direction of those keypoints is expressed as:

$$\text{cosine}(\vec{d}_{i,j}, \vec{D}_{i,j}[s]) = \frac{\vec{d}_{i,j} \cdot \vec{D}_{i,j}[s]}{\|\vec{d}_{i,j}\| \|\vec{D}_{i,j}[s]\|}, \quad (3.56)$$

which takes values  $\text{cosine}(\vec{v}_0, \vec{v}_1) \in [-1, \dots, 1]$  with 0 being the value for orthogonal vectors,  $-1$  for vectors of the same orientations but opposed directions, and 1 for a pair of vectors with exactly the same direction and orientation.

To transform cosine similarity to a *dissimilarity* metric, a mapping to angle in degrees was chosen, such that:

$$\text{dissimilarity}_{\text{cosine}}(\vec{d}_{i,j}, \vec{D}_{i,j}[s]) = \frac{1}{\pi} \arccos \left( \text{cosine}(\vec{d}_{i,j}, \vec{D}_{i,j}[s]) \right), \quad (3.57)$$

which relies on the fact, that  $\arccos(\alpha) \in \{0, \dots, \pi\}$ , thus the final dissimilarity metric is bounded between 0, 1.

The cost for all sampled points can be obtained by averaging:

$$C[i, j] = \frac{1}{s_{\max}} \sum_{s=0}^{s_{\max}} \text{dissimilarity}_{\cosine}(\vec{d}_{i,j}, \vec{D}_{i,j}[s]) \quad (3.58)$$

Now, the cost matrix  $C_{i,j}$  can be constructed and solved for optimal assignment, producing  $X_{i,j}$ . The entire procedure is first performed to determine all optimal assignments in the following order: shoulder  $\rightarrow$  tail assignments, followed by left ear  $\rightarrow$  shoulder, right ear  $\rightarrow$  shoulder, and finally left ear  $\rightarrow$  right ear. This order was selected arbitrarily in a fashion similar to the original OpenPose method.<sup>161</sup> We identified that shoulder and tail are keypoints that are very pronounced and fully define the position and orientation of the animal in our application. The other two selected keypoints: left ear and right ear are often occluded and thus we treat them with a secondary priority.

### 3.11.4 Augmentations of the Cost Metric

The complexity of the described procedure for producing cost matrix entries for all pairwise combinations of detected body parts may result in extensive processing time mostly due to the multi-step part affinity fields sampling procedure for each pair. If the network is trained to produce the foreground mask estimates  $y'_{fg}$ , it can be used for multiple purposes: first, all keypoints should belong to the foreground, second, all part affinities must not pass through background areas as in our case, the instances are defined as continuous regions in the image. The only exception for that rule is the presence of partial occlusion creating discontinuous regions. In that situation however, the occluder should also belong to the foreground. To address both proposed improvements, each heatmap and part affinity field estimate can be element-wise

multiplied by the estimated foreground binary mask.

$$y_{\text{kps},c_k}^* = y'_{\text{fg}} \odot y_{\text{kps},c_k} \quad \forall c_k \in [0, \dots, C_k - 1] \quad (3.59)$$

$$y_{\text{paf},c_p}^* = y'_{\text{paf}} \odot y_{\text{paf},c_p} \quad \forall c_p \in [0, \dots, C_p - 1], \quad (3.60)$$

where  $y_{\text{kps},c_k}^*, y_{\text{paf},c_p}^*$  are the resulting, masked estimates of the body part location heatmaps and part affinities respectively,  $\odot$  is the element-wise multiplication operation,  $c_k, c_p$  are the channel indices for the body part location and part affinity maps respectively, and  $C_k, C_p$  are the numbers of channels in each map.

It is important to stress that such method heavily relies on the network's capabilities of estimating the foreground mask and is particularly sensitive to false-negative values as the element-wise multiplication effectively removes potential candidates. Special caution is advised due to that sensitivity.

Additional terms can be introduced to the bipartite matching cost metric  $C$ . We propose the dissimilarity based on the embedding vectors produced by the neural network in  $y'_{\text{emb}}$ . Since the network is trained to produce embedding vectors that are consistent within a single instance but different among instances, a distance metric could be derived from those embeddings directly. A sampling procedure at points obtained as presented in equation 3.54 can be used for this task. Please note, that if sampling only at the estimated locations of keypoints and not across the line joining them, the value of  $s_{\text{max}} = 2$  can be used. Substitution of the part affinity field-based cost for the one based on the embedding vectors is used in this work and referred to in chapter 4 as the *Hybrid* method. Please refer to Figure 3.32 (d) which depicts the output of the multi-dimensional embeddings used for matching in this scenario.

### 3.12 Semantic Instance Segmentation using Embeddings

This section focuses on the use of embeddings in an attempt to segment images of homogeneous group-housed animals into groups of pixels containing individual animals given the input images. As described before in Section 3.9 those embeddings are a result of image transformation modeled by a neural network allowing for *easy* clustering. K-means was selected as the underlying clustering mechanism due to its relatively high performance to complexity ratio and wide adoption. We used the implementation available in *scikit.cluster* python package.<sup>174</sup>

There are however a few drawbacks of this clustering method: 1) it requires the number of cluster  $k$  to be known prior to clustering, and 2) its clustering performance heavily relies on the initialization of the cluster centroids<sup>175</sup> (even with correct  $k$  provided). To address concerns regarding the first issue, an attempt of guessing the number of clusters is presented in section 4.5.3, but it is argued that for the described application of pig tracking, it is safe to assume that the number of instances (clusters) will be known. Cluster initialization issue however is not addressed directly as it is out of scope of this research.

Figure 3.33 shows the input (a) and intermediate outputs generated during the described process. First, the preprocessed image is fed forward through a neural network to obtain the foreground mask estimate  $y'_{fg}$  and the embedding vectors  $y'_{emb}$ . Then, the foreground mask is thresholded based on the  $th_{fg}$  parameter. The foreground-masked embedding vectors are then clustered using the k-means algorithm producing labeling used to separate pig instances.

When instance image extraction is of interest, additional smoothing of the instance mask can be applied. Four examples of what could be considered a successful extraction and four for the failure are presented in figure 3.34. When looking at successful examples in the context of presented processing pipeline, it is visible, that pigs that are isolated and

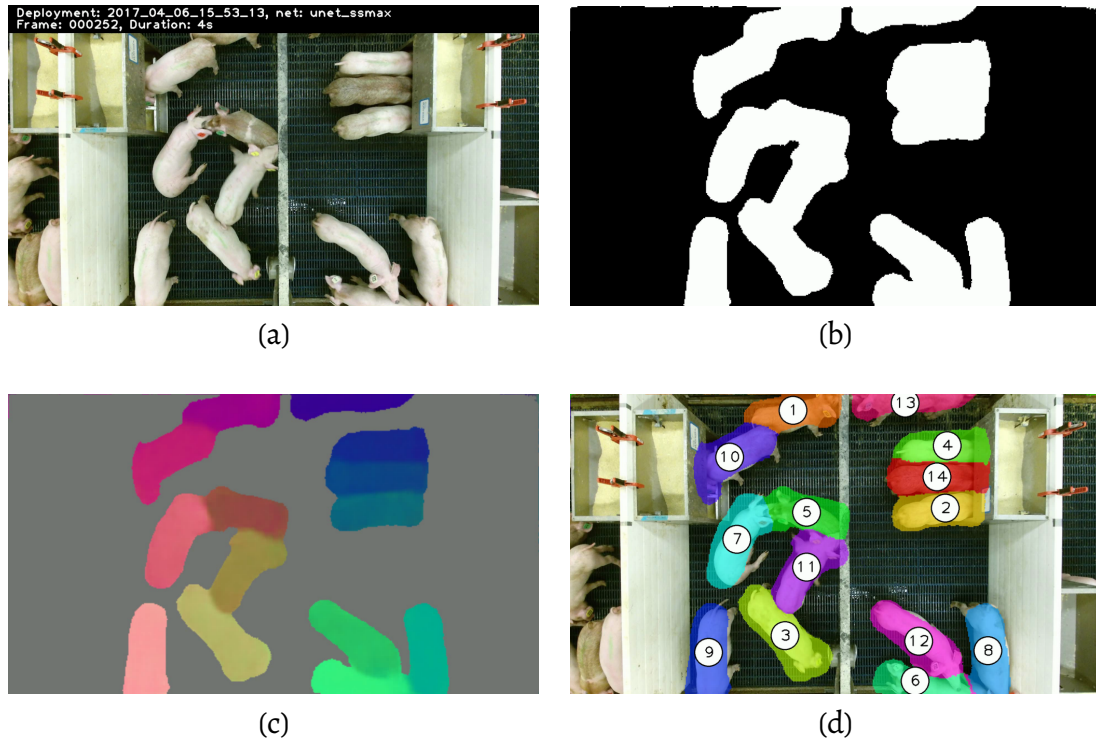


Figure 3.33: Visualization of the intermediate representations of our approach to semantic instance segmentation using embeddings. Input image (a), estimate of the semantic (foreground) segmentation mask (b), PCA visualization of the embeddings masked by the foreground (c), labels assigned to each out of the 14 segmented pigs (d).

oriented such that their backs are straight are more likely to produce proper outputs. On the other hand, pigs laying down in a group or an unlikely textured one will more likely fail.

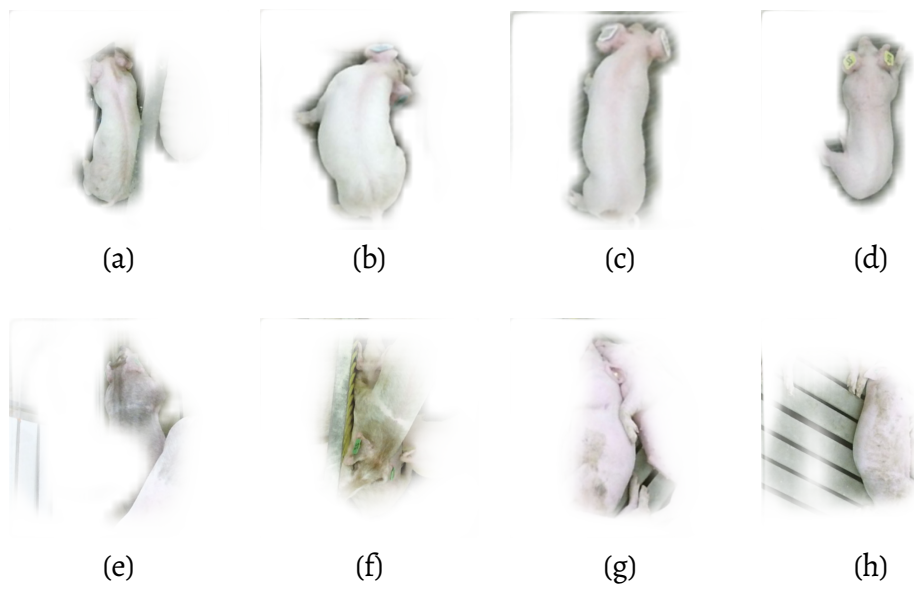


Figure 3.34: Examples of successful (a-d) and unsuccessful (e-h) attempts to instance segmentation using proposed model.

## CHAPTER 4

---

### Results

Our method can be understood as a sequence of *tasks*, each with a set of hyper-parameters. Conceptually, those tasks can be considered in the following order: foreground mask estimation, body part detection, part association estimation, pose estimation, and instance segmentation. This chapter contains results of qualitative and quantitative evaluation of our method performing those tasks. To reduce the search space of hyper-parameters controlling how methods we take a *greedy* approach to discover the best performing set of parameters at each step through appropriate performance analyses. Selected parameters then get fixed and used in consecutive tasks.

We are using two neural network architectures in this work: OpenPose-like network (OP) motivated by the systems developed for human pose estimation, and a UNET as it belongs to the family of segmentation-friendly networks which has been successfully used in our previous contributions.<sup>160</sup>

Section 4.1 is dedicated to recalling the concept of Receiver Operating Characteristics curves and statistics used for per-instance or per-pixel performance assessment. These metrics appear throughout the chapter and are heavily emphasized in Section 4.2 - which assesses the performance of the estimated foreground masks. The ability to determine which pixels belong to the objects of interest (pigs in this work, i.e. the foreground) and which are part of the (static) background in a per-image fashion is a major factor

affecting most of the sub-tasks that contribute to the operation of the method.

Being inspired by and heavily borrowing from methods used in human tracking, our methods detect keypoints as one of the sub-tasks in order to (at least) identify the number of instances visible in the picture. Evaluation of the keypoint detection is presented in Section 4.3.

Like in the work of Cao et al.<sup>161</sup> and our previous approaches,<sup>160</sup> the associations between keypoints (Part Affinity Fields) are explicitly modeled using the outputs of the neural network and constitute the essential component of the pose estimator. In Section 4.4 we present the evaluation of Part Affinity Fields produced by each of our neural networks: OP and UNET.

Our networks also produce pixel-level embedding vectors. Those are used in two scenarios: 1) as an alternative to Part Affinity Fields in our pose estimation method (the *Hybrid* pose estimation method), and 2) as features used in Semantic Instance Segmentation using k-means clustering. Section 4.5 contains the analysis of the properties and performance of the embedding vectors produced by our neural networks: OP and UNET. We used two loss functions to produce those embeddings: 1) a discriminative loss function with parametric cluster margins<sup>106</sup> (method described Section 3.9.2), and 2) our novel approach using silhouette score maximization (method described in Section 3.9.3).

Culmination of this chapter is in the final two sections: 4.6 and 4.7. The former presents the analysis of the pose estimation in three scenarios: 1) study of the effects of the selected representations, 2) pose estimation using keypoints and part affinity fields (OpenPose method) for the comparative performance baseline, and 3) our *Hybrid* pose estimation method using keypoints and embedding vectors. We use evaluation metrics established in our previous work<sup>160</sup> and present the performance of four neural networks:  $OP_{\text{margin}}$ ,  $OP_{\text{ssmax}}$ ,  $UNET_{\text{margin}}$ , and  $UNET_{\text{ssmax}}$ .

## 4.1 Receiver Operating Characteristics

Mask estimation and keypoint detection can be interpreted as binary classification tasks. Thus, a useful metric for assessment of their performance is through the Receiver Operating Characteristic (ROC) curve. The ROC curve represents detection performance given the applied decision threshold. Given that the output of the classifier is a real valued number between 0 and 1, where 0 represents *false* or lack of detection and 1 represents *true* or presence of detected object, and those values estimate the probability distribution of the detection, one can apply ROC methodology.

Let's consider a binary constant  $Y \in \{0, 1\}$  representing the ground-truth, a real-valued variable  $h$  representing the output of the probability model, and a real-valued threshold variable  $t$ , the detector's output  $Y'$  assumes the following values:

$$Y' = \begin{cases} P, & \text{if } h \geq t \\ N, & \text{otherwise} \end{cases} \quad (4.1)$$

Informally, the ROC analysis consists of varying the threshold  $t$  between 0 and 1 to determine the number of agreements and disagreements between  $Y$  and  $Y'$ . Table 4.1 shows all possible outcomes for the binary case. The  $TP$  and  $TN$  indicate two types of agreement (successfully detected and successfully rejected respectively). The disagreements are represented by  $FP$  (failed to reject) and  $FN$  (failed to detect).

	Y	
	P	N
Y' / P	TP	FP
N	FN	TN

Table 4.1: Four considered detection outcomes for true values  $Y$ , and predicted values  $Y'$ .

Outside of the Table 4.1, symbols  $TP$ ,  $FP$ ,  $FN$ , and  $TN$  will refer to the number of events. Statistical metrics required for this analysis are: *sensitivity* and *specificity*.

Sensitivity, also known as True Positive Rate (TPR) or *recall* is defined as:

$$TPR = \frac{TP}{TP + FN}, \quad (4.2)$$

where  $TP$  is the number of true positives, i.e. the elements that have been properly classified,  $FN$  is the number of incorrectly classified positives.  $FN$  corresponds to Type II statistical error and is also referred to as *a miss*.

Specificity, also referred to as True Negative Rate (TNR) or *specificity* is defined as:

$$TNR = \frac{TN}{TN + FP}, \quad (4.3)$$

where  $TN$  is the number of properly classified negatives, and  $FP$  is the number incorrectly classified negatives.

An ROC curve is defined by plotting the False Positive Rate (FPR) on the x-axis and true positive rate on the y-axis. FPR is defined as:

$$FPR = 1 - TNR = \frac{FP}{FP + TN} \quad (4.4)$$

Another metric commonly reported in literature is the Positive Predictive Value (PPV) referred to as *precision*:

$$PPV = \frac{TP}{TP + FP} \quad (4.5)$$

The  $F_1$  statistic, also referred to as *F-score* or *F-measure* is a harmonic mean of precision and recall and assumes 1 for perfect precision and recall, and 0 for worst.  $F_1$  is defined as:

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN} \quad (4.6)$$

## 4.2 Foreground Segmentation Evaluation

Both of our neural networks (OP and UNET) were trained to produce multiple outputs: keypoint location heatmaps, part affinity fields, embedding vectors and foreground masks. Foreground mask indicate which pixels belong to the objects of interest (pigs) and which pixels belong to the background (elements of the environment, floor, pen, walls etc.). The analysis presented in this section is trying to answer four questions: 1) is there a benefit to the use of foreground masks extracted from depth images?; 2) which model produces better foreground mask estimations under ideal conditions, and 3) what are the optimal decision threshold values for foreground detection for each model?; and 4) at which scale does the OP model yields best foreground mask estimates.

To answer those questions we decided to resort to the ROC curve analysis, the  $F_1$  statistic (both explained in section 4.1), and the manually annotated evaluation dataset (MFG110EVAL) described in Section 3.5.1. This data set was created to represent the deployment of the data collection systems used in this work accurately. We evaluated the performance foreground estimation in three cases:

1.  $OP_0$ : In this case we trained the foreground estimator part of the OP model (Section 3.8.1) using only the synthetic foreground masks created from instance annotations as described in Section 3.5,
2.  $OP_1$ : In this case we trained the foreground estimator part of the OP model using both the synthetic masks created from instance annotations and the ground truth foreground masks extracted from depth images as described in Section 3.5.2.
3. UNET: In this case we trained the UNET model (described in Section 3.8.2) using ground truth foreground masks extracted from depth images just like in the  $OP_1$  case.

For the OP model in  $OP_0$  and  $OP_1$  cases, the images were fed-forward at four scales, each determined by the ratio of the parameter  $W$  representing the desired width of the image at the model's input to the *width* of the input image. The height  $H$  is adjusted per-example based on the original image size and the determined scale as presented in the example below.

**Example 1.** Consider a color image that is  $width = 1920$  pixels wide and  $height = 1080$  pixels tall represented, and a desired, target width  $W = 1024$ . Then the scale  $s$  and inferred height  $H$  are calculated as proportions:

$$s = \frac{W}{width} = \frac{1024}{1920} \approx 0.53 \quad (4.7)$$

$$H = height \cdot s = \frac{height \cdot W}{width} = \frac{1080 \cdot 1024}{1920} = 576 \quad (4.8)$$

A set of target widths was chosen to be:  $\{W\} = \{368, 512, 736, 1024\}$  which was motivated by the following rationale. First,  $W = 368$  was the size of the training patch,  $W = 512$  is a medium-resolution standard image size that is divisible by 8,  $W = 736$  is twice the size of the training patch, and  $W = 1024$  represents the largest, high resolution image size that can be processed on an a single GPU with 8GB of graphic memory given the proposed architecture. UNET was trained using  $W = 512$  and is tested with it as well. Each curve is estimated using a set of 100 threshold values between  $0 \leq \text{threshold} \leq 1$ , spaced by 0.01. This granularity allows for smooth curve visualization and optimal threshold estimation.

The corresponding ROC curves are presented in Figure 4.1. For each target width  $W$ , the optimal threshold  $th_{\text{optim}}$  was determined as the point on the ROC curve that has the smallest distance to the point  $(0, 1)$ . This provides the best balance between the ability to accurately detect, and reject a foreground membership hypothesis. Additionally, the Area

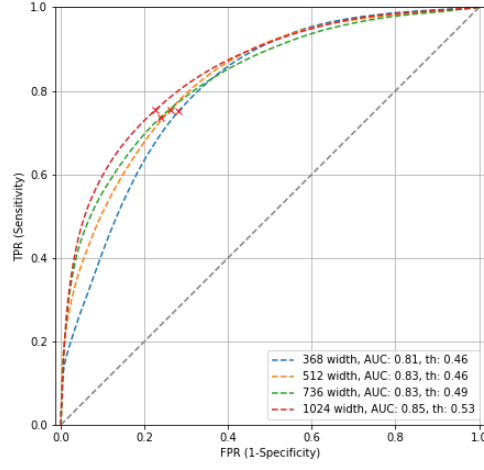
under Curve (AUC) was calculated to assess the general performance of the detector. A perfect detector would have  $AUC = 1$ .

When comparing the AUC values reported in Figures 4.1 (a) and (b) for the  $OP_0$  and  $OP_1$  cases respectively, it is visible, that even the worst case for  $OP_1$  (0.92 at  $W = 368$ ) outperforms the best case for  $OP_0$  (0.85 at  $W = 1024$ ). Based on this finding we determined that it is beneficial in our case to use foreground masks extracted from the depth images. The following analyses were performed using models trained using foreground masks like in the  $OP_1$  case. This means that we used both the synthetic ground truth masks generated from annotations and the masks extracted from depth images.

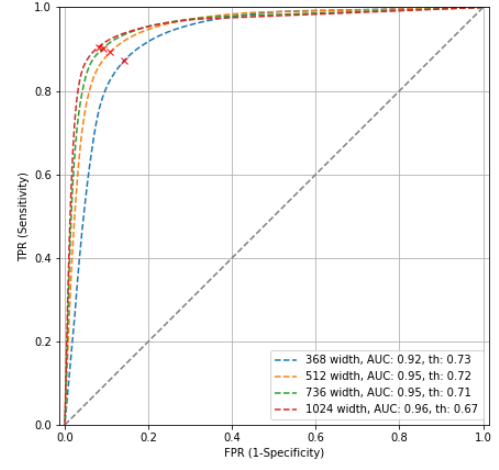
When looking at Figure 4.1 (b) it is apparent that the best operating conditions of the OP model trained like in the  $OP_1$  case determined by the highest AUC of 0.96 are the target width  $W = 1024$  and detection threshold  $th = 0.67$ .

When comparing Figures 4.1 (b) and (c) based on the AUC, it is visible that the OP model trained in  $OP_1$  case provides higher AUC (0.96 at  $W = 1024$ ) than the UNET model (0.91) for the task of foreground mask estimation. Thus, it is determined that under ideal conditions of optimal scale and threshold selection, the OP model trained like in  $OP_1$  case outperforms the UNET model in the task of foreground mask estimation.

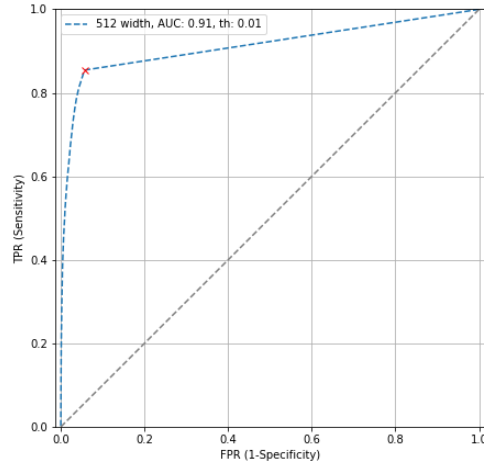
Figure 4.1 also indicates that the best detection threshold for our UNET network is 0.01.



(a)



(b)



(c)

Figure 4.1: Foreground mask extraction evaluation using ROC curves for the three considered cases:  $OP_0$  (a),  $OP_1$  (b) and UNET (c). Evaluated using MFG110EVAL dataset.

Subset	W	$th_{\text{optim}}$	Precision	Recall	$F_1$
A	368	0.59	0.43	0.81	0.56
A	512	0.59	0.47	0.80	0.59
A	736	0.56	0.52	0.84	0.64
A	1024	0.60	0.57	0.86	<b>0.69</b>
B	368	0.39	0.18	0.75	0.29
B	512	0.49	0.22	0.82	0.35
B	736	0.72	0.35	0.81	0.49
B	1024	0.76	0.41	0.84	<b>0.55</b>
C	368	0.42	0.29	0.77	0.42
C	512	0.58	0.42	0.81	0.56
C	736	0.59	0.46	0.84	0.59
C	1024	0.67	0.47	0.83	<b>0.60</b>
D	368	0.35	0.62	0.75	0.68
D	512	0.39	0.65	0.79	0.71
D	736	0.44	0.69	0.83	<b>0.75</b>
D	1024	0.55	0.68	0.83	<b>0.75</b>
E	368	0.52	0.19	0.78	0.31
E	512	0.55	0.22	0.77	<b>0.34</b>
E	736	0.59	0.20	0.70	0.32
E	1024	0.63	0.22	0.69	0.33
F	368	0.47	0.59	0.81	<b>0.69</b>
F	512	0.33	0.54	0.81	0.65
F	736	0.38	0.55	0.73	0.63
F	1024	0.44	0.56	0.74	0.64
G	368	0.47	0.77	0.87	<b>0.82</b>
G	512	0.40	0.71	0.82	0.76
G	736	0.41	0.65	0.78	0.71
G	1024	0.53	0.69	0.77	0.72
H	368	0.52	0.78	0.80	0.79
H	512	0.49	0.76	0.80	0.78
H	736	0.54	0.79	0.80	0.80
H	1024	0.58	0.80	0.82	<b>0.81</b>
I	368	0.41	0.06	0.79	0.12
I	512	0.54	0.09	0.80	0.15
I	736	0.66	0.15	0.84	0.25
I	1024	0.69	0.18	0.85	<b>0.30</b>
J	368	0.30	0.51	0.71	0.60
J	512	0.33	0.58	0.76	<b>0.66</b>
J	736	0.36	0.58	0.73	0.65
J	1024	0.40	0.59	0.73	0.65
K	368	0.36	0.39	0.82	0.53
K	512	0.37	0.45	0.87	0.59
K	736	0.39	0.49	0.83	<b>0.61</b>
K	1024	0.36	0.47	0.81	0.59
TOTAL	368	0.46	0.43	0.75	0.55
TOTAL	512	0.46	0.45	0.76	0.56
TOTAL	736	0.49	0.47	0.74	0.57
TOTAL	1024	0.53	0.49	0.75	<b>0.59</b>

Table 4.2: Foreground detector performance of the OP model when trained only using synthetic masks generated from annotations - OP<sub>0</sub> case. MFG110EVAL dataset was used in the evaluation.

Subset	W	$th_{\text{optim}}$	Precision	Recall	$F_1$
A	368	0.67	0.62	0.86	0.72
A	512	0.74	0.72	0.89	0.79
A	736	0.76	0.78	0.91	0.84
A	1024	0.72	0.82	0.94	<b>0.88</b>
B	368	0.48	0.33	0.78	0.46
B	512	0.61	0.48	0.91	0.63
B	736	0.74	0.66	0.94	0.78
B	1024	0.75	0.73	0.95	<b>0.82</b>
C	368	0.59	0.55	0.88	0.67
C	512	0.66	0.65	0.91	0.76
C	736	0.73	0.74	0.93	0.82
C	1024	0.72	0.76	0.95	<b>0.85</b>
D	368	0.52	0.80	0.90	0.85
D	512	0.53	0.86	0.94	0.89
D	736	0.58	0.87	0.95	0.90
D	1024	0.58	0.87	0.94	<b>0.91</b>
E	368	0.79	0.32	0.91	0.47
E	512	0.79	0.42	0.93	0.58
E	736	0.81	0.49	0.94	0.64
E	1024	0.81	0.59	0.95	<b>0.73</b>
F	368	0.75	0.69	0.88	0.77
F	512	0.79	0.73	0.85	0.78
F	736	0.78	0.76	0.86	<b>0.81</b>
F	1024	0.75	0.75	0.85	0.80
G	368	0.78	0.79	0.92	<b>0.85</b>
G	512	0.73	0.81	0.88	0.84
G	736	0.74	0.83	0.86	0.84
G	1024	0.62	0.80	0.85	0.83
H	368	0.87	0.87	0.94	<b>0.90</b>
H	512	0.86	0.85	0.92	0.89
H	736	0.84	0.84	0.93	0.89
H	1024	0.80	0.87	0.91	0.89
I	368	0.27	0.13	0.87	0.22
I	512	0.65	0.38	0.94	0.54
I	736	0.66	0.49	0.95	0.65
I	1024	0.65	0.55	0.96	<b>0.70</b>
J	368	0.73	0.67	0.84	0.74
J	512	0.61	0.78	0.89	0.83
J	736	0.55	0.80	0.91	0.85
J	1024	0.50	0.82	0.92	<b>0.86</b>
K	368	0.69	0.56	0.90	0.69
K	512	0.70	0.68	0.93	0.79
K	736	0.73	0.72	0.92	0.81
K	1024	0.73	0.75	0.93	<b>0.83</b>
TOTAL	368	0.73	0.64	0.87	0.74
TOTAL	512	0.72	0.70	0.89	0.78
TOTAL	736	0.71	0.74	0.90	0.81
TOTAL	1024	0.67	0.76	0.91	<b>0.83</b>

Table 4.3: Foreground detector performance of the OP model when trained on both the synthetic foreground masks generated from annotations and the foreground masks extracted from the depth images - OP<sub>1</sub> case. MFG110EVAL dataset was used in the evaluation.

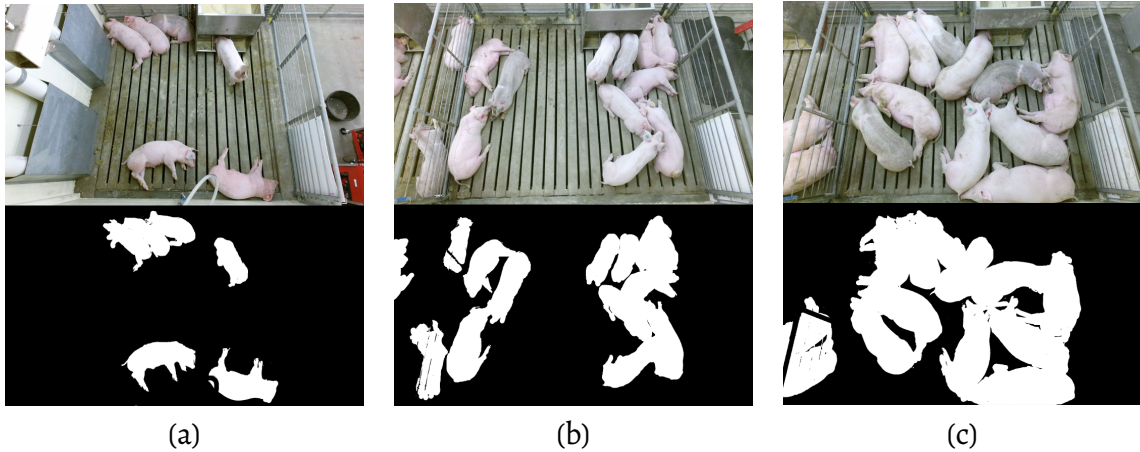


Figure 4.2: Interesting cases of foreground masks depicted by sample images (top) and their corresponding ground-truth masks(bottom) from the MFG110EVAL data set: subset E for which performance improved most drastically (a), subset D for which the OP model in OP<sub>1</sub> case achieved the highest  $F_1$  score (b), and subset H for which all cases achieved high  $F_1$  score.

Subset	W	$th_{\text{optim}}$	Precision	Recall	$F_1$
A	512	0.01	0.83	0.88	0.85
B	512	0.02	0.73	0.97	0.83
C	512	0.02	0.76	0.95	0.84
D	512	0.01	0.76	0.77	0.77
E	512	0.01	0.63	0.85	0.72
F	512	0.01	0.75	0.65	0.69
G	512	0.01	0.82	0.74	0.77
H	512	0.01	0.83	0.84	0.84
I	512	0.01	0.50	0.96	0.65
J	512	0.01	0.74	0.60	0.66
K	512	0.01	0.74	0.85	0.79
TOTAL	512	0.01	0.75	0.76	0.75

Table 4.4: Performance of the foreground detector of the UNET model when trained on both the synthetic foreground masks generated from annotations and the foreground masks extracted from the depth images - UNET case.

Numerical results for all cases are presented in Tables 4.2 and 4.3, and 4.4 for the OP<sub>0</sub>, OP<sub>1</sub>, and UNET cases respectively.

It is important to remember we used the MFG110EVAL data set in this section which was created from images captured across 11 deployments, where each deployment can be considered as a separate *subset*. Please refer to Figure 3.12 in Section 3.5.1 for more detail.

Both, the overall and per-subset statistics are provided in order to determine the applicability of the models for specific types of deployments. In order to determine the optimal (or at least best given the test set) scale selection and decision threshold, the tables also contain those parameters.

When comparing the per-subset results in Tables 4.2 and 4.3 for the  $OP_0$  and  $OP_1$  respectively it is visible that the performance was increased among all subsets. The most drastic change occurred for subset E at  $W = 1024$  as the  $F_1$  score of 0.33 increased to 0.73 with the main reason being the increase in precision from 0.22 to 0.59. A sample image from subset E is presented in Figure 4.2 (a).

### 4.3 Evaluation of the Body Part Detector

We determine the location and orientation of pigs in the images using two *pose estimation* methods described in Section 3.11 1) our reimplementation of OpenPose by Cao et al.<sup>161</sup> for pigs, and 2) our Hybrid method involving the use of the embeddings. Both of these methods begin with the estimation of the locations of pigs body parts (shoulder and tail, but also left ear and right ear) - referred to as *keypoints*. Conceptually, elements of our method involved in locating the keypoints in images are referred to as *keypoint detector* or *body part detector* interchangeably.

It is important to note that the pose estimation methods will **fail** immediately if no keypoints are detected in the image. This imposes a desire to minimize the number of false negative detections. On the other hand, the method's complexity is lower-bound by  $O(n_{\text{shoulder}} \cdot n_{\text{tail}})$  as the method always considers all possible combinations of pairing detected shoulders with tails. This imposes a desire to minimize the number false positive detections due to performance. Similarly to the analysis of the foreground detector in Section 4.2, we are using  $F_1$  statistic as a metric that represents a compromise between detection and rejection.

This section is dedicated to performance evaluation of the keypoint detector using PDD2019 data set containing three subsets *train*, *test:seen*, and *test:unseen*. Forward inference of the model is ran for each image in *test:seen* and *test:unseen* at multiple multiple target widths  $W$ , threshold values  $th_{kp}$ , and smoothing parameter of the peak detector  $\sigma$ . Model's keypoint output heatmaps  $y'_{kps}$  are then processed by the peak detector parameterized by  $th$  and  $\sigma$  to produce output formatted like the ground truth annotation data described in section 3.3. The parameters used in evaluation are listed in Table 4.5. The non-maximum-suppression-based processing of the network's output is described in section 3.11.1.

Parameter	Tested values	OP Model	UNET Model	Description
$\sigma$	1, 3, ..., 13, 15	1	1	Peak detector smoothing
$W$	368, 512, 736, 1024	1024	512	Target width
$d_{max}$	0, 1, ..., 50	45	45	Max match distance
$th_{fg}$	0, ..., 1, with 0.01 step	0.67	0.01	Foreground detection threshold
$th_{kp}$	0, ..., 1, with 0.01 step	0.34	0.45	Keypoint detection threshold

Table 4.5: Keypoint / inference evaluation parameters tested and determined experimentally.

Section 4.3.2 shows the evaluation of the spatial error in the produced keypoint location estimates for both models. Since the processing involves the non-maximum suppression operation parametrized by the smoothing kernel width  $\sigma$ , section 4.3.3 shows the effect of  $\sigma$  on the PPV of the detector. For the OP model architecture, it is important to determine the optimal input size parametrized by  $W$ , section 4.3.4 presents a short evaluation of PPV with respect to variable  $W$ . Finally, in Section 4.3.1, author presents the analysis with respect to the variable keypoint detection threshold that leads to the selection of  $th_{kp}$  for each model used in the further analysis.

### 4.3.1 Keypoint Detection Threshold

Similar to the process presented in Section 4.2 we used statistics defined in Section 4.1, namely precision, recall, and  $F_1$  to determine the *optimal* threshold value  $th_{kp}$ . To do so we fixed the values of other parameters and varied only  $th_{kp} \in \{0, 0.01, \dots, 1\}$  to determine the value that maximizes the  $F_1$  score. We performed this evaluation for both OP and UNET models. The fixed parameter values are presented in table 4.5. Results for the OP and UNET models are presented in Figure 4.3. Numerical results are presented in Tables 4.6 and 4.7 for the OP and UNET networks respectively.

OP model performs better on the keypoint detection task on both sets. We chose to use the  $th_{kp} = 0.4$  for this model based on the fact, that  $F_1$  score will still be greater than 0.7 when using it on the images like in *test:seen* images, and will reach peak performance against inputs like from *test:unseen* images. Looking at very low score of the UNET model against test:unseen image subset (d), we decided to use  $th = 0.45$  based on the highest-ranking threshold for the test:seen subset.

Keeping the settings fixed as listed in table 4.5, a per-keypoint evaluation was ran for OP and UNET models. Results are presented in tables 4.6 and 4.7 respectively. Observing presented values it is safe to draw a conclusion that model C over-fit to the training data as it performs *significantly* poorer against the unseen images than the seen ones.

Subset	Keypoint	Precision	Recall	$F_1$
Test:seen	Left ear	0.73	0.70	0.71
Test:seen	Right ear	0.72	0.68	0.70
Test:seen	Shoulder	0.82	0.84	0.83
Test:seen	Tail	0.86	0.85	0.85
Test:seen	Left ear	0.84	0.51	0.64
Test:seen	Right ear	0.81	0.48	0.60
Test:seen	Shoulder	0.86	0.72	0.78
Test:seen	Tail	0.87	0.75	0.81
Test:unseen	Left ear	0.66	0.34	0.45
Test:unseen	Right ear	0.68	0.36	0.47
Test:unseen	Shoulder	0.75	0.45	0.56
Test:unseen	Tail	0.69	0.46	0.55

Table 4.6: OP model keypoint detection performance with respect to the type of keypoint with fixed  $\sigma = 1$ ,  $W = 1024$ ,  $d_{\max} = 45$ , and  $th = 0.34$ .

Subset	Keypoint	Precision	Recall	$F_1$
Train	Left ear	0.89	0.61	0.75
Train	Right ear	0.87	0.56	0.74
Train	Shoulder	0.92	0.66	0.86
Train	Tail	0.79	0.94	0.86
Test:seen	Left ear	0.48	0.61	0.54
Test:seen	Right ear	0.47	0.56	0.51
Test:seen	Shoulder	0.66	0.66	0.66
Test:seen	Tail	0.63	0.71	0.67
Test:unseen	Left ear	0.13	0.16	0.15
Test:unseen	Right ear	0.18	0.22	0.20
Test:unseen	Shoulder	0.17	0.12	0.14
Test:unseen	Tail	0.15	0.17	0.16

Table 4.7: UNET model keypoint detection performance with respect to the type of keypoint with fixed  $\sigma = 1$ ,  $W = 512$ ,  $d_{\max} = 45$ , and  $th = 0.45$  for three subsets of PDD2019 data set: train, test:seen, and test:unseen.

### 4.3.2 Spatial Accuracy of Keypoint Detection and Distance Threshold

Evaluation starts with overview of ability to detect keypoints when matched to ground truth using linear assignment. The goal of this step is to determine useful distance bounds for keypoint matching for further evaluation. Figures 4.4 show the numerical results for the union of sets *test:seen* and *test:unseen* for OP and UNET models. Observing the figure leads to the following observations:

- Neither of the models provide reliable sub-pixel detection overall, as not even 1% of keypoints is detected when distance threshold of 1 pixel is selected.
- The OP model outperforms the UNET model in the overall ability to detect keypoints. Caution is however advised as the analysis does not show ability to reject.
- In both cases, selection of a distance threshold of 45 *pixels* seems justifiable. It is however important to note, that spatial accuracy of described method is *lower* than in,<sup>160</sup> which in case of the OP model can be justified by  $8\times$  upsampling.

### 4.3.3 Keypoint peak detector smoothing kernel size

In case of sharp responses generated by the neural networks - particularly in scenarios when a single-pixel indicators are used instead as ground-truth of gaussian kernels, it may be necessary to post-process keypoint heatmaps with additional Gaussian smoothing parametrized by  $\sigma$ . Here, an effect of such smoothing is presented over entire test subset of PDD2019 to determine if such smoothing is necessary.

Results presented in Figures 4.5 (a) and (b) show the ability to detect keypoints based on PPV with respect to varying  $\sigma$  for two models: OP and UNET respectively. It is seen, that using a post-processing smoothing parameter  $\sigma > 1$  does not increase the performance. Thus the value of  $\sigma = 1$  is used in further evaluation.

### 4.3.4 Scale (target width) selection based on the keypoint detection performance

Presented UNET model operates with fixed target width parameter  $W = 512$ , but the OP model is designed to accept images adjust to different widths  $W$ . Please note that we used a greedy approach to estimate the best parameters for our method. In Section 4.2 we determined optimal  $W = 1024$  for the OP model based on the performance evaluated using  $F_1$  in the task of foreground mask estimation. Given how important the keypoint estimation is for the overall performance of the method, a cautious choice needs to be made regarding the choice of parameter  $W$ . In this section we are evaluating

Result is presented in Figure 4.6 and the highest PPV was achieved at  $W = 1024$ , which will be selected for further analysis of models relying on the OP architecture.

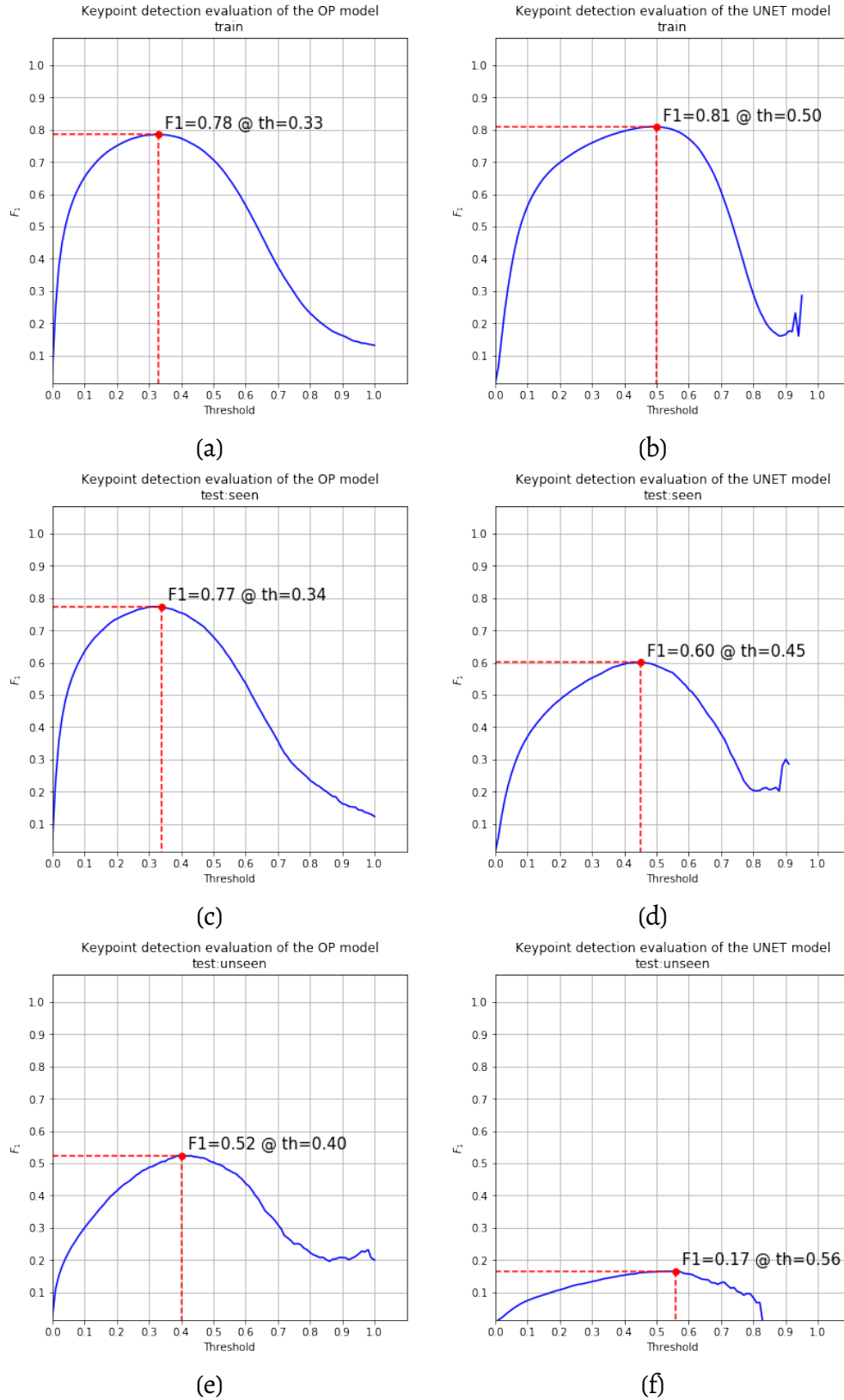


Figure 4.3: *Optimal* keypoint detection threshold based on  $F_1$  score curves for two models: OP (left column) and UNET (right column) for subsets of PDD2019 data set: train (top row), test:seen (middle row), and test:unseen (bottom row).

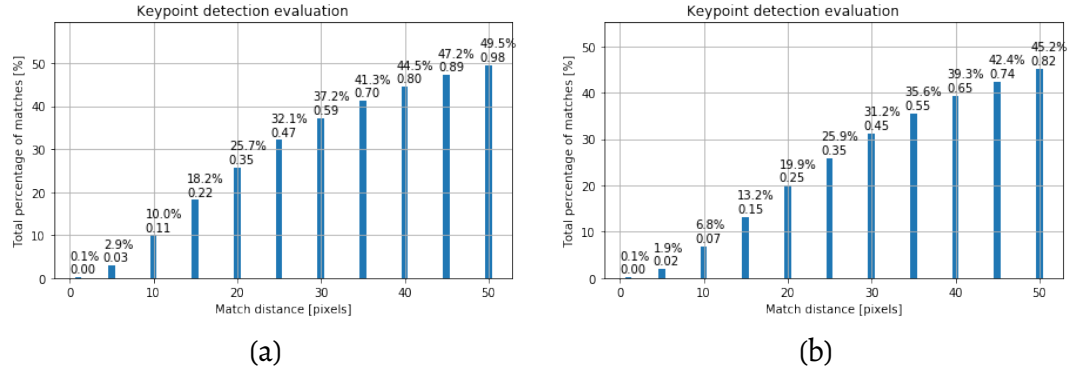


Figure 4.4: Percentage of total matches and PPV (precision) for OP (a), and UNET (b) architecture.

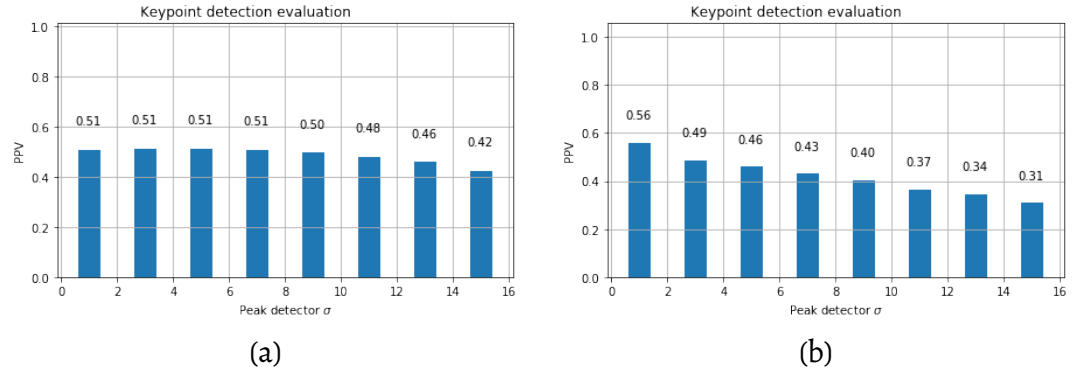


Figure 4.5: PPV as a function of image smoothing parameter  $\sigma$  in peak detector with distance threshold  $d_{\max} = 45$  pixels (determined in previous section) for OP (a), and UNET (b) models.

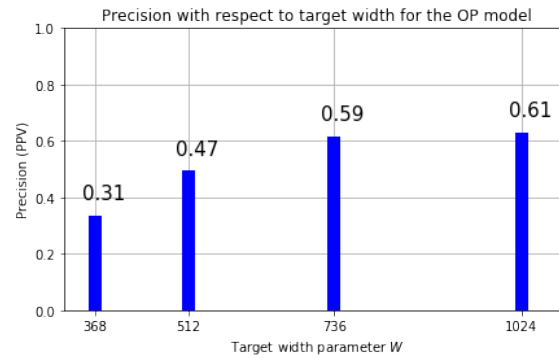


Figure 4.6: PPV as a function of  $W \in \{368, 512, 736, 1024\}$  for OP model with  $d_{\max} = 45$  pixels.

## 4.4 Part Affinity Estimation Evaluation

Using the inference parameters for the foreground and keypoint detection established in previous sections and listed in Table 4.5, both models were evaluated using dissimilarity metric presented in Section 3.11, Equation 3.57, but here, the considered vector pairs are consisting of: 1) the ground-truth orientation between keypoints, and 2) sampled (part affinity fields,  $y'_{\text{paf}}$ ) feature maps produced by the neural network. We follow the same sampling method as presented in Section 3.11.3.

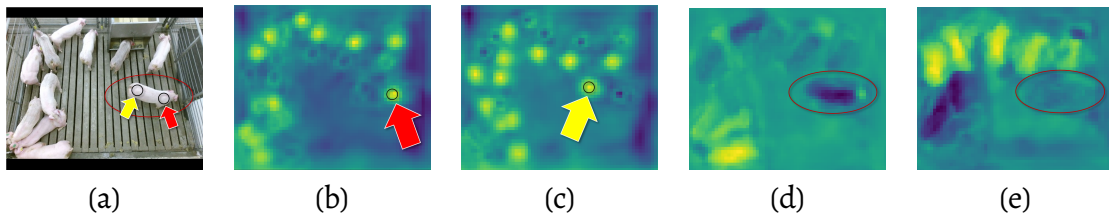


Figure 4.7: Depiction of the part affinity field sampling inputs: original image (a), shoulder keypoint heatmap (b), tail keypoint heatmap (c),  $x$  component of the part affinity field corresponding to the Shoulder  $\rightarrow$  Tail part association (d),  $y$  component of the part affinity field corresponding to the Shoulder  $\rightarrow$  Tail part association.

Our models produce localized outputs that are expected to only be valid between the keypoints of interest without the penalty elsewhere. Thus, when sampling  $y'_{\text{paf}}$  on two channels corresponding to  $x, y$  components of the direction vector, it is expected, that all samples along the entire path between the keypoints will have *similar* values. Figures 4.7 (d) and (e) present an example of the two channels representing the  $x$  and  $y$  components of the part affinity field between the shoulder and tail keypoints.

Thus, the similarity measure is computed between the ground truth, normalized direction vector, and normalized sampled values. The ground truth vector is obtained as the difference in pixel positions of the two considered keypoints as visible in Figures 4.7 (a), (b) and (c). Results of our evaluation are presented in Table 4.8. The average dissimilarity and standard deviation statistics are calculated over all images in the subset, and all sampled points within those images, for all instances with annotated

keypoint pairs. We used the PDD2019 dataset for evaluation in this Section.

It is important to point out, that the cosine similarity (as presented in Equation 3.56) produces values bounded by  $-1$  for colinear vectors with opposite direction, and  $1$  for colinear vectors with equal direction with  $0$  for perpendicular vectors. When comparing the results in Tables 4.8 and 4.9, particularly assignments different than *Shoulder*  $\rightarrow$  *Tail*, the OP model achieves higher performance - even for the *test:unseen* dataset. The *Shoulder*  $\rightarrow$  *Tail* assignment was expected to be problematic due to high spatial extend and ease of confusion between those two keypoints - particularly when pig heads are hidden in the feeder during eating.

It is also important to note, that the cardinality of *Left ear*  $\rightarrow$  *Right ear* assignments is the smallest among all in the training set, which corresponds to its lower score when compared to *Left ear*  $\rightarrow$  *Shoulder* and *Right ear*  $\rightarrow$  *Shoulder*.

The UNET model was also trained to produce part affinity fields. Evaluation results for the UNET model are presented in table 4.9. Unfortunately, the UNET model performs significantly poorer than the OP model in this task as the standard deviation of the similarity is significant or higher with respect to the mean.

Subset	Keypoint pair	Average similarity	Standard deviation	Sample size
Train	Left ear $\rightarrow$ Right ear	0.91	0.24	11851
Train	Left ear $\rightarrow$ Shoulder	0.95	0.18	15273
Train	Right ear $\rightarrow$ Shoulder	0.95	0.18	15227
Train	Shoulder $\rightarrow$ Tail	0.86	0.47	20497
Train	Total	0.91	0.32	62848
Test:seen	Left ear $\rightarrow$ Right ear	0.91	0.23	1532
Test:seen	Left ear $\rightarrow$ Shoulder	0.95	0.19	1848
Test:seen	Right ear $\rightarrow$ Shoulder	0.94	0.21	1845
Test:seen	Shoulder $\rightarrow$ Tail	0.84	0.51	2367
Test:seen	Total	0.91	0.34	7592
Test:unseen	Left ear $\rightarrow$ Right ear	0.78	0.42	950
Test:unseen	Left ear $\rightarrow$ Shoulder	0.81	0.46	1266
Test:unseen	Right ear $\rightarrow$ Shoulder	0.77	0.50	1244
Test:unseen	Shoulder $\rightarrow$ Tail	0.52	0.77	1723
Test:unseen	Total	0.70	0.60	5183

Table 4.8: Part Affinity Fields Prediction evaluation results for the OP model with  $W = 1024$  and  $s_{\max} = 10$ .

Subset	Keypoint pair	Average similarity	Standard deviation	Sample size
Train	Left ear $\rightarrow$ Right ear	0.56	0.70	11851
Train	Left ear $\rightarrow$ Shoulder	0.60	0.69	15273
Train	Right ear $\rightarrow$ Shoulder	0.60	0.69	15227
Train	Shoulder $\rightarrow$ Tail	0.65	0.70	20497
Train	Total	0.61	0.70	62848
Test:seen	Left ear $\rightarrow$ Right ear	0.38	0.67	1532
Test:seen	Left ear $\rightarrow$ Shoulder	0.44	0.68	1848
Test:seen	Right ear $\rightarrow$ Shoulder	0.44	0.68	1845
Test:seen	Shoulder $\rightarrow$ Tail	0.67	0.61	2367
Test:seen	Total	0.54	0.65	7592
Test:unseen	Left ear $\rightarrow$ Right ear	0.15	0.70	950
Test:unseen	Left ear $\rightarrow$ Shoulder	0.09	0.72	1266
Test:unseen	Right ear $\rightarrow$ Shoulder	0.05	0.73	1244
Test:unseen	Shoulder $\rightarrow$ Tail	0.19	0.76	1723
Test:unseen	Total	0.18	0.73	5183

Table 4.9: Part Affinity Prediction evaluation results for the UNET model with  $W = 512$  and  $s_{\max} = 10$ .

## 4.5 Embeddings Analysis

Both, the OP and UNET architectures were trained to produce embedding vectors using two different loss functions: 1) direct-silhouette score maximization described in Section 3.9.2 and referenced using  $_{ss\max}$  subscript, and 2) discriminative loss with parametric margin described in Section 3.9.3 and referenced using  $_{\text{margin}}$  subscript.

In both cases the embeddings were trained in a weakly-supervised fashion based on pixel-level membership assignments represented as image (as described in Section 3.4) with custom loss functions penalizing inter-instance mean closeness, and minimizing the intra-instance spread of embedding values. It was expected, that such method would yield multi-channel feature maps suitable for instance-level segmentation based on clustering alone. To be able to additionally bridge the gap between the pose estimation and semantic segmentation tasks in the context of animal tracking, the embedding vectors were anticipated to have the potential to augment the components of part matching cost (as described in Section 3.11.4) or become the sole contributor to it.

First, in Section 4.5.1 we present the assessment of our models' ability to produce embedding features suitable for clustering using cohesion and separation metrics. Then, in Section 4.5.2 we explain the learned embedding vectors in terms of intuitively

engineered features based on visual clues. We perform a correlation-based analysis while acknowledging the homogeneous nature of the tracked objects (pigs). In Section 4.5.3 we evaluate our method's ability to estimate the number of foreground objects in the image based on the embedding vectors using k-means clustering and silhouette score.

To finalize the performance evaluation of the embedding vectors, Section 4.6 presents the effect of incorporating them in the *hybrid* pose estimation method.

#### 4.5.1 Within-instance and between-instances embedding analysis

The first step of the analysis is to determine our loss functions described in Section 3.9 produce results in accord with our expectations; particularly if the embedding variation within the instance (Equation 4.9) is smaller than Euclidean distances between instance cluster means (Equation 4.10).

For all instances in the subsets *test:seen* and *test:unseen* we calculated the average per-instance mean of the difference between the cluster mean and the average inter-instance distances. Use of those statistics is motivated by the *silhouette score* commonly used in statistics to validate the number of clusters present in the data<sup>176</sup> and the parts of the  $L_{\text{margin}}$  loss function described in Section 3.9.3.

The following metrics are used in this analysis:

$$d_{\text{within}} = \frac{1}{K} \sum_{k=0}^{K-1} \frac{1}{P_k} \sum_{j=0}^{n_k-1} \|\mu_k - \hat{y}_{\text{emb},j}\|, \quad (4.9)$$

where  $K$  is the number of ground-truth instances,  $n_k$  is the number of pixels belonging to the instance  $k$ ,  $\mu_k$  is the average predicted embedding vector for instance  $k$ ,  $\hat{y}_{\text{emb},j}$  is the pixel of predicted embedding feature map indexed by  $j$  such that it belongs to instance  $k$ .

$$d_{\text{between}} = \frac{1}{K(K-1)} \sum_{k=0}^{K-1} \sum_{j=0}^{K-1} 1(j \neq k) \cdot \|\mu_k - \mu_j\|, \quad (4.10)$$

where  $K$  is the number of instances,  $k, j$  are the indices of the instances, with corresponding mean embedding vectors  $\mu_k, \mu_j$ .

Those metrics are then calculated for each image in the PDD2019 dataset and averaged to produce values in table 4.10. Small values of  $d_{\text{within}}$  indicate good cohesion of the produced clusters as the cluster members are close to the mean. High values of  $d_{\text{between}}$  indicate good separation as the means of the clusters are far apart. Additionally, the average silhouette score  $ss$  (described in Section 3.9.1) is provided as an additional measure for the quality of the clustering.

When observing the values in Table 4.10, it is visible, that the UNET model performs remarkably better than our OP model in the task of producing multi-dimensional embeddings suitable for easy clustering. It is worth to note, that OP model was able to produce clusters with small  $d_{\text{within}}$  in relation to  $d_{\text{between}}$  but the low silhouette score indicates that the clusters are poorly formed.

Model	Subset	avg. $d_{\text{within}}$	avg. $d_{\text{between}}$	avg. $ss$
OP <sub>ssmax</sub>	train	0.18	1.83	0.45
OP <sub>ssmax</sub>	test:seen	0.19	1.64	0.43
OP <sub>ssmax</sub>	test:unseen	0.22	1.12	0.30
OP <sub>margin</sub>	train	0.58	4.02	0.46
OP <sub>margin</sub>	test:seen	0.61	3.76	0.44
OP <sub>margin</sub>	test:unseen	0.62	2.77	0.36
UNET <sub>ssmax</sub>	train	0.08	3.11	0.89
UNET <sub>ssmax</sub>	test:seen	<b>0.09</b>	<b>3.21</b>	<b>0.88</b>
UNET <sub>ssmax</sub>	test:unseen	0.24	2.87	0.70
UNET <sub>margin</sub>	train	0.21	3.51	0.82
UNET <sub>margin</sub>	test:seen	0.22	3.54	0.82
UNET <sub>margin</sub>	test:unseen	0.42	3.19	0.63

Table 4.10: Average  $d_{\text{within}}$ ,  $d_{\text{between}}$  and silhouette scores of the embeddings produced by OP and UNET models over all instances in PDD2019 dataset.

To get a better understanding of the embedding vectors produced by both models, additional evaluation step was performed to observe the relationship between the number of instances presented in the image and presented performance metrics. Their correlation with respect to the number of instances is presented in Table 4.11.

Model	Subset	$r_{\text{within}}$	$r_{\text{between}}$	$r_{\text{ss}}$
OP <sub>ssmax</sub>	Train	−0.07	0.54	−0.71
OP <sub>ssmax</sub>	Test:seen	0.19	0.74	−0.63
OP <sub>ssmax</sub>	Test:unseen	−0.93	0.70	0.91
OP <sub>margin</sub>	Train	−0.07	0.54	−0.71
OP <sub>margin</sub>	Test:seen	0.19	0.74	−0.63
OP <sub>margin</sub>	Test:unseen	−0.93	0.70	0.91
UNET <sub>ssmax</sub>	Train	−0.04	0.20	−0.74
UNET <sub>ssmax</sub>	Test:seen	−0.11	0.12	−0.79
UNET <sub>ssmax</sub>	Test:unseen	−0.94	−0.13	0.84
UNET <sub>margin</sub>	Train	−0.68	0.73	−0.83
UNET <sub>margin</sub>	Test:seen	−0.52	0.44	−0.81
UNET <sub>margin</sub>	Test:unseen	−0.87	0.43	0.80

Table 4.11: Pearson’s correlation coefficients between number of instances in the image and within cluster variation  $d_{\text{within}}$  ( $r_{\text{within}}$ ), distance between cluster means  $d_{\text{between}}$  ( $r_{\text{between}}$ ), and silhouette score ( $r_{\text{ss}}$ ) for OP and UNET models.

Ideally, the  $d_{\text{within}}$  would not be correlated with the number of instances. This would mean, that the given method can produce embeddings with high cluster cohesion for each instance without overlapping with another instance. Also, it would be beneficial to increase the spread among the cluster means as the number of instances in the image increases.

This would be confirmed by positive correlation between  $d_{\text{inter}}$  and the number of instances. Silhouette score should be unaffected or it should not decrease as the number of instances increases.

When observing values presented in Table 4.11 it is visible that the  $r_{\text{within}}$  assumes mostly low values for the train and test:seen subsets for the OP model. Values for the UNET architecture indicate that UNET produces clusters that decrease  $r_{\text{within}}$  as the number of instances increases which is the desired behavior. When looking at the  $r_{\text{between}}$  models tend to increase the separation between the clusters as the number of clusters increases which is the desired behavior. When looking at the silhouette score it seems like it does decrease as the number of clusters increases which indicates the difficulty in separating instances when more of them are present.

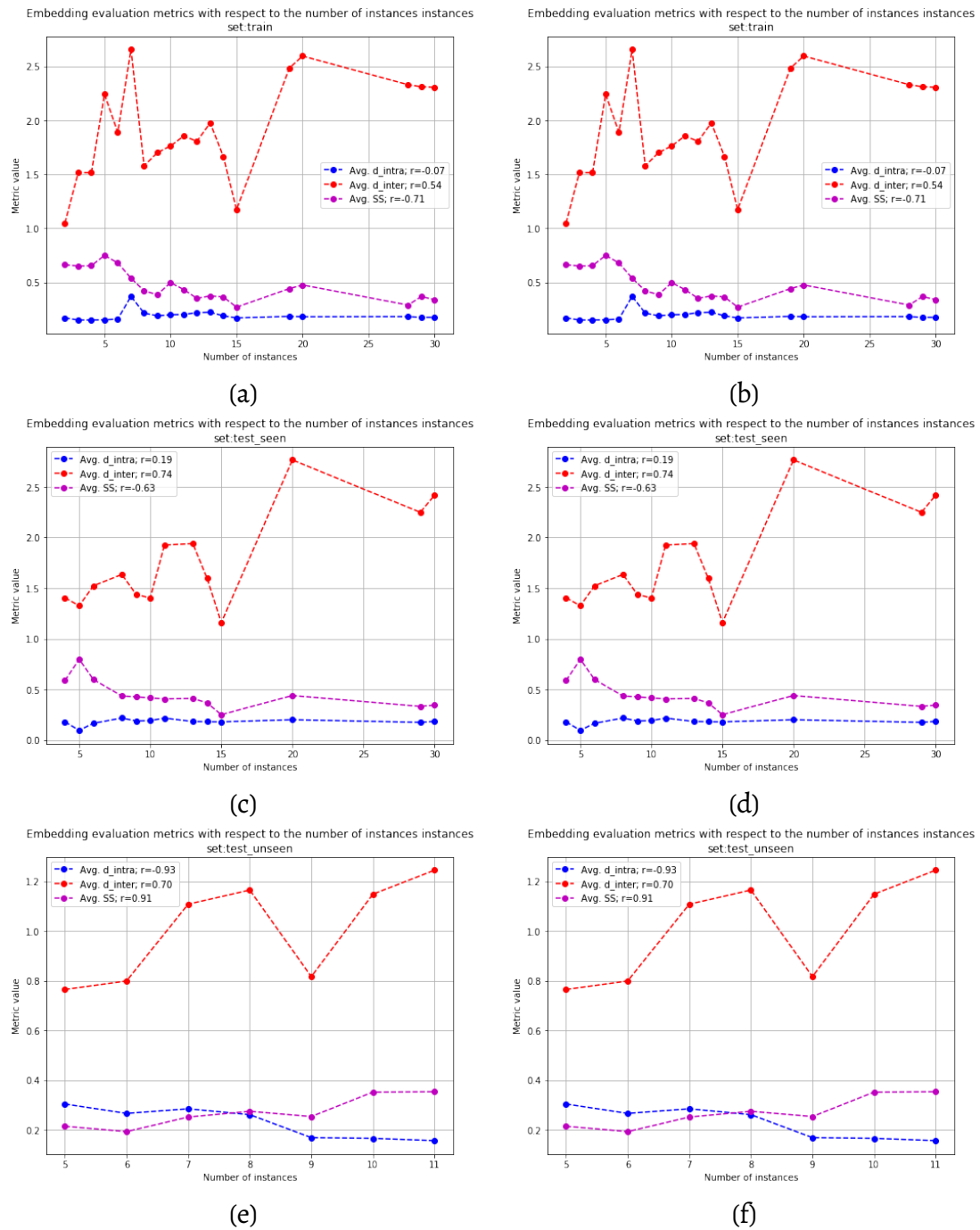


Figure 4.8: Performance of the OP models indicated by averages of  $d_{intra}$ ,  $d_{inter}$ , and  $ss$  with respect to the number of instances visible in the annotated images for all images (a), (b), test:seen (c), (d), and test:unseen (e), (f).  $OP_{ssmax}$  presented in (a), (c), and (e), and  $OP_{margin}$  in (b), (d), and (f).

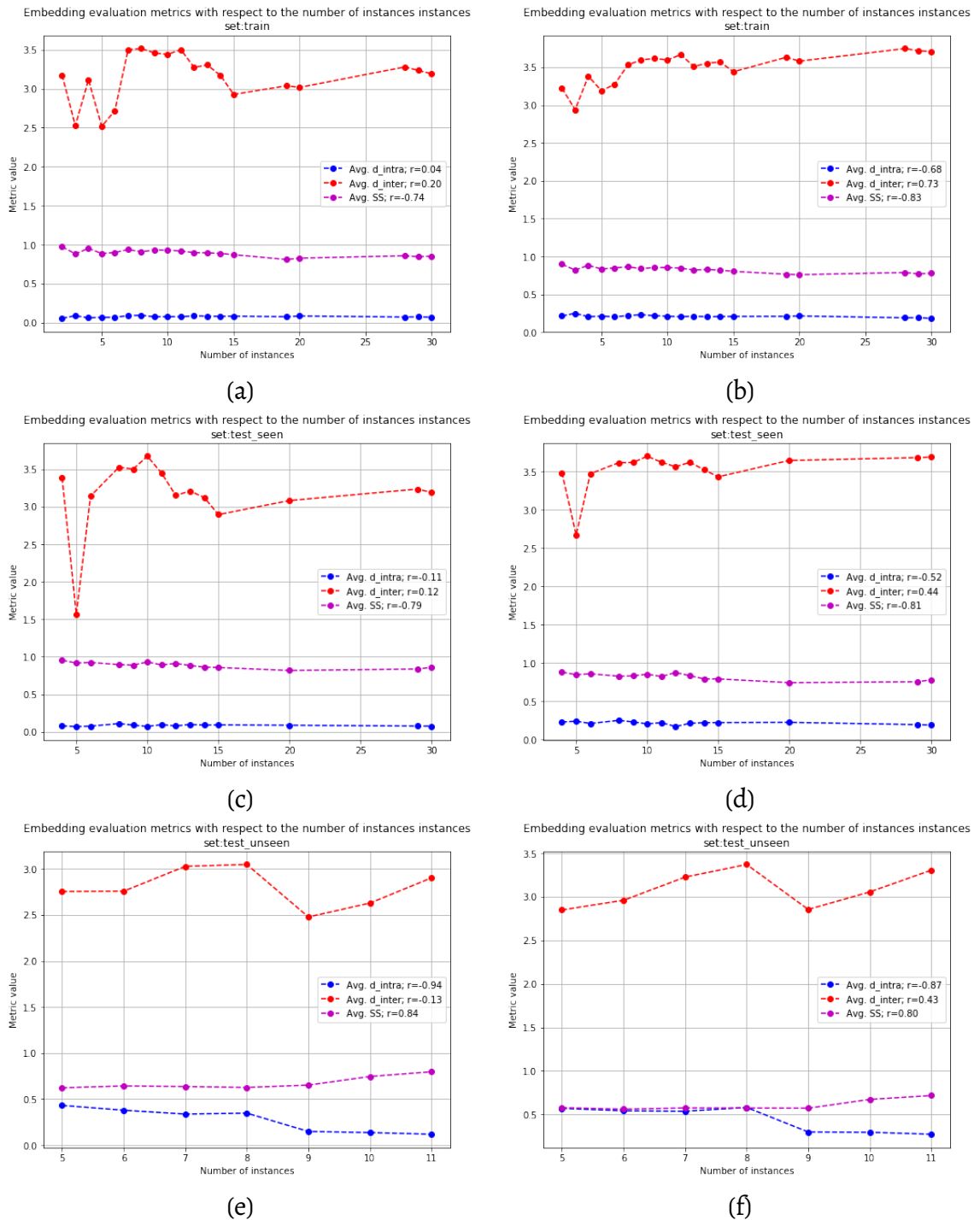


Figure 4.9: Performance of the UNET models indicated by averages of  $d_{intra}$ ,  $d_{inter}$ , and  $ss$  with respect to the number of instances visible in the annotated images for all images (a), (b), test:seen (c), (d), and test:unseen (e), (f). UNET<sub>ssmax</sub> presented in (a), (c), and (e), and UNET<sub>margin</sub> in (b), (d), and (f).

### 4.5.2 Correlation with image position, orientation, size, and color properties

During preliminary evaluation, it was noticed that the embeddings trained in a weakly-supervised fashion exhibit a tendency to capture the image position of the represented instance. This observation was one of the reasons of training the UNET model as a different architecture than OP for which this property was observed.

Ideally, only some portion of the embedding vectors would be correlated to the position and orientation and the rest would grasp other properties inherent to the instance but potentially different among instances. Separation of the spatial properties (location) from more subtle ones (specific intricate features of the object itself) could potentially yield a set of features allowing for distinguishing instances that appear visually identical and provide a solution for tracking of *homogeneous* objects. In this section we correlation coefficient to understand the properties of the per-instance mean embedding vectors. We correlate the the mean embedding vector with the following (arbitrarily selected) *engineered* features:

- $pos_x, pos_y$  - Image position represented by x,y pixel coordinates scaled by the inverse of image size to produce values between 0 and 1,
- $ori_x, ori_y$  - 2-dimensional normalized instance orientation vector (like described in Section 3.6),
- $size$  - Size of the instance represented by the norm of the vector between shoulder and tail of the instance,
- $col$  - Average *color* of the instance represented as average *hue* component of the HSV-encoded part of the image image containing the instance,
- $\sigma_H, \sigma_S, \sigma_V$  - standard deviations in the HSV color space as an attempt to capture texture-dependent correspondences.

Choice of HSV encoding was motivated by its property of encapsulating chromatic properties within a single property: *hue*, while separating it from the light intensity: *value*, and *saturation*, which is not the case when using RGB encoding. Since this step of evaluation explores values of the correlation coefficient, the scale of properties is not of concern as it is taken into account during calculation. Thus, no additional scaling is required or applied.

Tables 4.12 through 4.15 contain the Pearson’s correlation coefficients extracted from the correlation matrices formulated from concatenating the embedding vectors for OP and UNET models’ outputs and properties listed above. In all cases it is apparent that the embedding vectors correlate strongly with the position in the image. The values with maximum magnitude are highlighted in the presented Tables. Additionally, the correlation matrices are presented in Figures 4.10 though 4.11. The only exception is visible in table 4.13, where  $emb_0$ ,  $emb_2$ , and  $emb_{10}$  vectors correlate with the *color* feature for the test:unseen dataset.

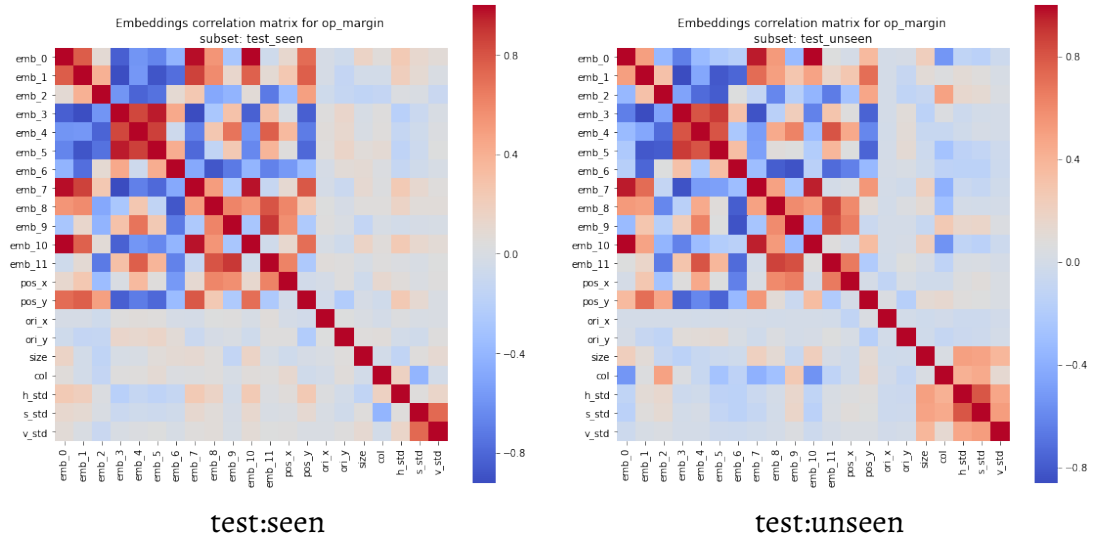


Figure 4.10: Correlation matrices of the embedding vectors concatenated with engineered properties described in Section 4.5.2 for the OP<sub>margin</sub> model two test subsets of the PDD2019 dataset (test:seen, test:unseen).

We offer a visual inspection presented in Figure 4.12. We calculated the embeddings

	$pos_x$	$pos_y$	$ori_x$	$ori_y$	$size$	$col$	$\sigma_H$	$\sigma_S$	$\sigma_V$
$emb_0$	0.12	<b>0.71</b>	-0.01	-0.07	0.15	0.05	0.24	0.11	0.07
$emb_1$	0.25	<b>0.76</b>	-0.02	-0.13	-0.04	-0.04	0.20	0.09	0.00
$emb_2$	-0.36	<b>0.48</b>	-0.06	-0.12	-0.15	-0.13	0.02	0.00	-0.10
$emb_3$	0.00	<b>-0.84</b>	0.04	0.13	-0.01	0.04	-0.20	-0.10	-0.00
$emb_4$	0.35	<b>-0.71</b>	0.06	0.12	-0.00	0.05	-0.13	-0.06	0.02
$emb_5$	0.05	<b>-0.80</b>	0.05	0.14	0.07	0.08	-0.16	-0.08	0.03
$emb_6$	<b>-0.57</b>	-0.37	-0.02	0.07	0.09	0.02	-0.13	-0.06	-0.01
$emb_7$	0.11	<b>0.78</b>	-0.02	-0.09	0.10	0.03	0.23	0.11	0.05
$emb_8$	<b>0.60</b>	0.24	0.03	-0.02	0.05	0.05	0.15	0.07	0.06
$emb_9$	<b>0.57</b>	-0.25	0.04	0.00	-0.14	-0.01	-0.04	-0.02	-0.01
$emb_{10}$	0.12	<b>0.70</b>	-0.01	-0.06	0.16	0.06	0.24	0.11	0.08
$emb_{11}$	<b>0.62</b>	-0.26	0.05	0.04	-0.01	0.04	0.02	-0.00	0.03

Table 4.12: Section of the correlation matrix between embeddings produced by the  $OP_{margin}$  model and image properties of the instances for the *test:seen* subset.

	$pos_x$	$pos_y$	$ori_x$	$ori_y$	$size$	$col$	$\sigma_H$	$\sigma_S$	$\sigma_V$
$emb_0$	0.01	0.36	-0.00	0.01	0.22	<b>-0.54</b>	-0.12	-0.14	-0.04
$emb_1$	0.28	<b>0.71</b>	0.00	-0.08	0.10	0.05	0.11	0.08	0.02
$emb_2$	-0.27	0.47	-0.00	-0.11	-0.08	<b>0.49</b>	0.15	0.13	0.04
$emb_3$	0.07	<b>-0.76</b>	-0.00	0.09	-0.14	0.05	-0.04	-0.01	0.00
$emb_4$	0.46	<b>-0.61</b>	-0.00	0.10	-0.07	-0.07	-0.02	0.01	0.00
$emb_5$	0.07	<b>-0.76</b>	0.00	0.12	-0.04	-0.27	-0.13	-0.10	-0.04
$emb_6$	<b>-0.63</b>	-0.33	0.00	0.03	-0.03	-0.13	-0.14	-0.14	-0.04
$emb_7$	0.03	<b>0.55</b>	-0.00	-0.02	0.21	-0.39	-0.06	-0.09	-0.02
$emb_8$	<b>0.60</b>	0.10	-0.01	0.05	0.11	-0.29	-0.00	-0.01	-0.00
$emb_9$	<b>0.66</b>	-0.06	-0.00	0.00	-0.08	0.27	0.15	0.17	0.05
$emb_{10}$	0.01	0.35	-0.00	0.01	0.23	<b>-0.55</b>	-0.12	-0.15	-0.05
$emb_{11}$	<b>0.67</b>	-0.21	-0.01	0.07	0.02	-0.13	0.03	0.04	0.00

Table 4.13: Section of the correlation matrix between embeddings produced by the  $OP_{margin}$  model and image properties of the instances for the *test:unseen* subset.

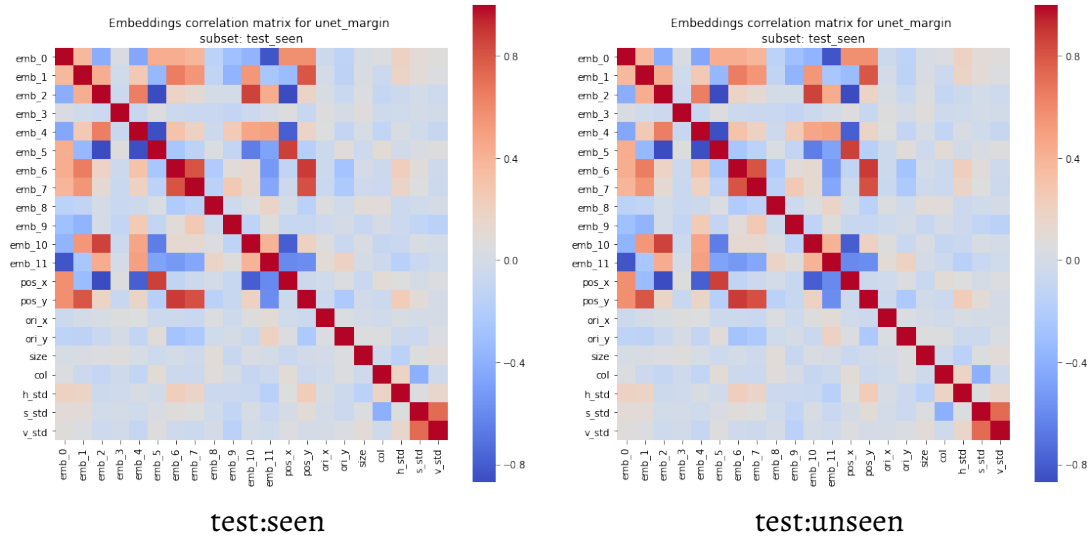


Figure 4.11: Correlation matrices of the embedding vectors concatenated with engineered properties described in Section 4.5.2 for the  $UNET_{margin}$  model two test subsets of the PDD2019 dataset (test:seen, test:unseen).

	$pos_x$	$pos_y$	$ori_x$	$ori_y$	$size$	$col$	$\sigma_H$	$\sigma_S$	$\sigma_V$
$emb_0$	0.58	<b>0.58</b>	-0.05	-0.13	0.02	0.04	0.19	0.10	0.08
$emb_1$	-0.33	<b>0.80</b>	-0.01	-0.13	0.03	-0.05	0.18	0.10	0.04
$emb_2$	<b>-0.87</b>	0.18	0.02	-0.07	0.04	-0.09	-0.05	-0.01	-0.04
$emb_3$	0.07	<b>-0.08</b>	0.07	0.01	0.07	-0.03	-0.03	-0.01	-0.02
$emb_4$	<b>-0.79</b>	0.17	0.06	-0.10	0.02	-0.11	0.02	-0.03	-0.09
$emb_5$	<b>0.87</b>	-0.17	-0.05	0.09	-0.04	0.11	-0.02	0.03	0.06
$emb_6$	-0.11	<b>0.88</b>	-0.05	-0.28	-0.01	-0.05	0.22	0.08	-0.03
$emb_7$	-0.05	<b>0.82</b>	-0.06	-0.23	-0.03	-0.05	0.17	0.06	-0.05
$emb_8$	-0.01	<b>-0.15</b>	0.04	-0.03	0.09	0.09	-0.06	-0.03	0.00
$emb_9$	-0.08	-0.08	-0.04	0.01	-0.08	0.01	-0.05	-0.11	<b>-0.15</b>
$emb_{10}$	<b>-0.79</b>	0.20	0.07	-0.06	0.03	-0.08	-0.07	0.03	-0.01
$emb_{11}$	<b>-0.60</b>	-0.58	0.10	0.19	0.01	-0.05	-0.17	-0.07	-0.03

Table 4.14: Section of the correlation matrix between embeddings produced by the UNET<sub>margin</sub> model and image properties of the instances for the *test:seen* subset.

	$pos_x$	$pos_y$	$ori_x$	$ori_y$	$size$	$col$	$\sigma_H$	$\sigma_S$	$\sigma_V$
$emb_0$	<b>0.67</b>	0.56	-0.02	-0.05	0.06	0.08	0.13	0.11	0.09
$emb_1$	-0.30	<b>0.76</b>	0.07	-0.09	0.12	0.06	0.05	0.05	0.13
$emb_2$	<b>-0.89</b>	0.14	0.14	-0.05	0.06	-0.02	-0.05	-0.07	-0.00
$emb_3$	0.17	0.02	0.03	0.05	0.01	<b>-0.19</b>	-0.12	-0.11	-0.14
$emb_4$	<b>-0.83</b>	0.12	0.08	-0.02	0.01	0.02	-0.05	-0.08	-0.05
$emb_5$	<b>0.89</b>	-0.14	-0.12	0.03	-0.05	-0.01	0.03	0.06	0.01
$emb_6$	-0.14	<b>0.87</b>	0.05	-0.16	0.04	0.10	0.03	0.02	-0.03
$emb_7$	-0.07	<b>0.82</b>	0.04	-0.18	0.03	0.09	0.02	-0.00	-0.07
$emb_8$	-0.10	-0.05	-0.02	-0.03	-0.02	<b>0.20</b>	0.12	0.13	0.18
$emb_9$	<b>-0.19</b>	-0.08	-0.02	-0.03	-0.11	0.07	-0.02	-0.05	-0.13
$emb_{10}$	<b>-0.79</b>	0.14	0.11	-0.07	0.07	-0.03	-0.07	-0.07	0.04
$emb_{11}$	<b>-0.68</b>	-0.58	0.06	0.07	-0.03	-0.04	-0.08	-0.07	0.01

Table 4.15: Section of the correlation matrix between embeddings produced by the UNET<sub>margin</sub> model and image properties of the instances for the *test:unseen* subset.

for all animal instances in all images in the PIGSEG96 dataset using UNET<sub>margin</sub> network. We concatenated the obtained embedding vectors forming the  $n \times C$  matrix  $A$  with  $n$  being the number of all instances and  $C = 12$  being the number of embedding channels. We treated matrix  $A$  as an ensemble of all embeddings in the set and performed Principal Component Analysis using the covariance matrix. We used those principal components to reduce the number of representation dimensions from 12 to 3 to encode it using RGB color images and display the middle row of Figure 4.12.

The goal was to illustrate what happens to the embeddings as we change the position in the image. We introduce the change in position by rotating the image around its center by  $0^\circ$ ,  $45^\circ$ , and  $90^\circ$ . As visible in Figure 4.12 the embeddings corresponding to the animal indicated with the green arrow change color from light green to light pink to more pronounced pink. This indicates that the same instance is not globally encoded using our

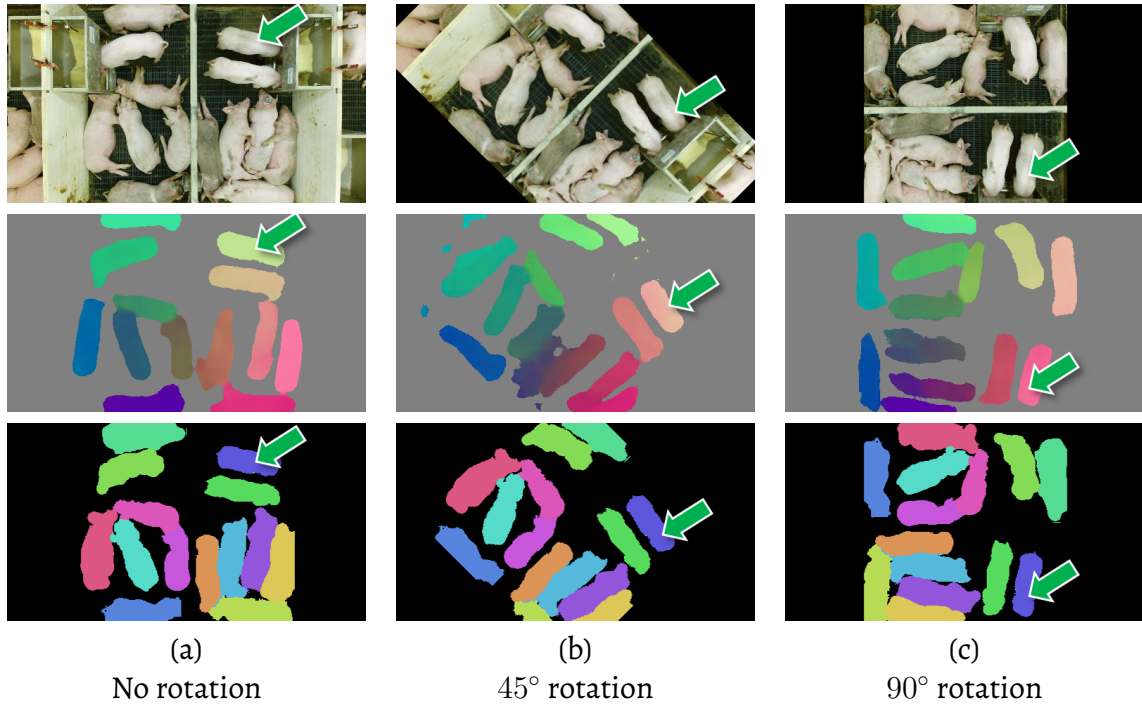


Figure 4.12: Illustration of the embedding vectors following the position in the image using Principal Component Analysis and example #1 from the PIGSEG96 dataset. In rows: the input image (top), visualization of the embeddings using RGB color components via dimensionality reduction using PCA (middle), ground-truth labels from the PIGSEG96 dataset (bottom). In columns: original pen scenario with no clockwise rotation applied (a), image and ground truth with applied 45° clockwise rotation (b), and image and ground truth with applied 90° degree rotation. We use green arrows to indicate the same instance in all images.

method but rather expressed in relation to the image position or the environment. This finding confirms the previous observations based on Tables 4.12 through 4.15.

### 4.5.3 Number of Instances Estimation through Cluster Analysis of the Embedding Vectors

When considering total reliance on embedding vectors produced by our neural networks (OP and UNET) for instance segmentation, it is important to determine if those embedding vectors can be properly clustered into the number of instances visible in the images. Findings in Section 4.5.1 indicate the potential of the produced outputs to

generate separable instances using clustering. We decided to use k-means clustering in our work due to its simplicity. Finding an optimal number of cluster centers ( $k$ ) prior to performing the actual clustering is a challenging and necessary task for certain applications.<sup>175</sup> It is important to note that however accurate the automatic instance counting is, the number of animals held in the pen was usually constant during the trials for which the data used here was collected. Thus, one could set the number of tracked instances as a deployment-specific, fixed parameter. The presence of occlusions however affects the number of *visible* instances, thus making the automatic number of instances estimation useful.

By design however, the method should be (at least partially) able to estimate the number of clusters present in the embedding map  $y'_{\text{emb}}$ . To determine if that is the case, a post-clustering *silhouette score*-based method is used to determine the *best* number of clusters. Four tiers are considered: Tier 0 requires the number of instances to be exactly equal to the ground truth, Tier 1 allows for a difference of a single instance, Tier 2 allows 2, and Tier 3 allows mistakes of 3 instances. The total number of samples is presented to provide more context to the performance metric.

The accuracy is assessed by the ratio of the number of images for which the estimated number of instances lies within each tier to the total number of images containing that true number of instances.

The results are presented in Table 4.16 through 4.19. The results are also presented in Figures 4.13 through 4.14. Both models trained using  $L_{\text{margin}}$  achieve higher tier 0 accuracy as visible in Figures 4.13 (c) and 4.14 (c). Results for the number of instances higher than 15 for the test:seen subset indicate that the performance follows the distribution of the number of instances in the dataset shown in Figure 3.3.

Our semantic instance segmentation method relies on how exact the estimation of the number of instances is. When observing results presented in this Section we conclude that we can rely on the selected method of estimation number of clusters reliably.

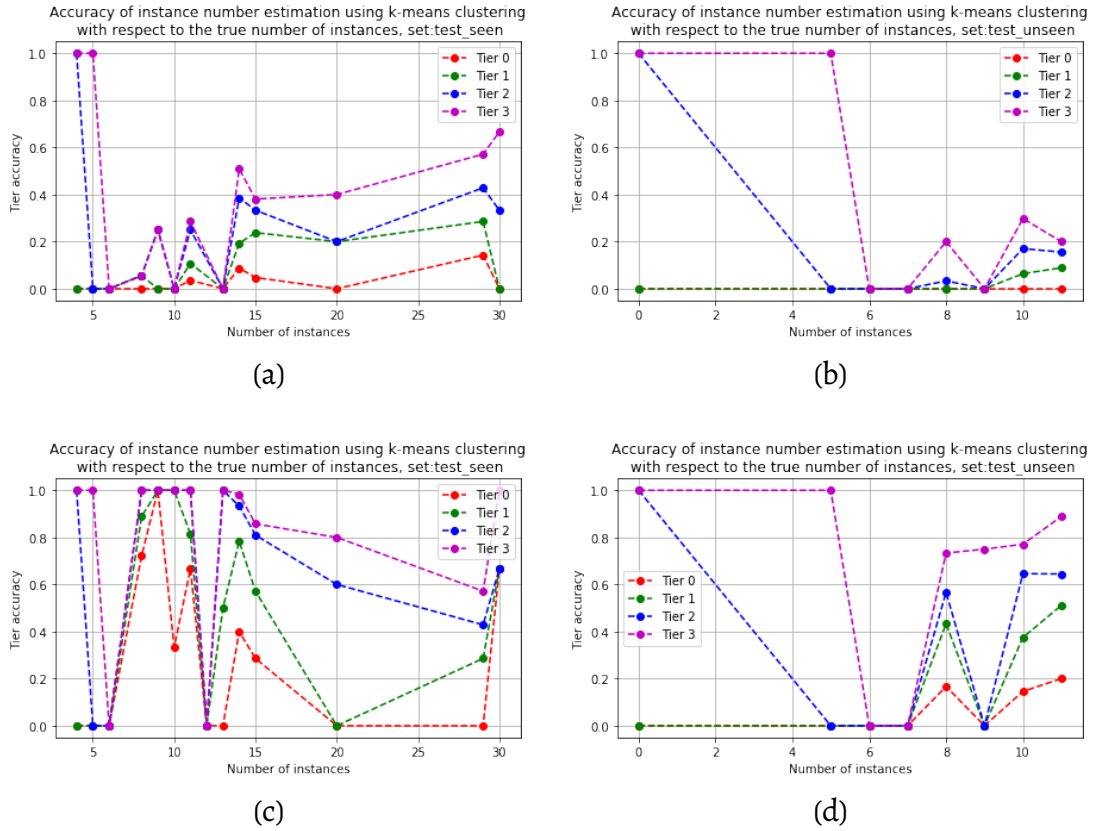


Figure 4.13: Accuracy of instance number estimation based on the silhouette score of the labeling produced by applying the k-means algorithm to the embedding outputs of the OP models: OP<sub>ssmax</sub> on test:seen (a), OP<sub>ssmax</sub> on test:unseen (b), OP<sub>margin</sub> on test:seen (c), and OP<sub>margin</sub> on test:unseen (d).

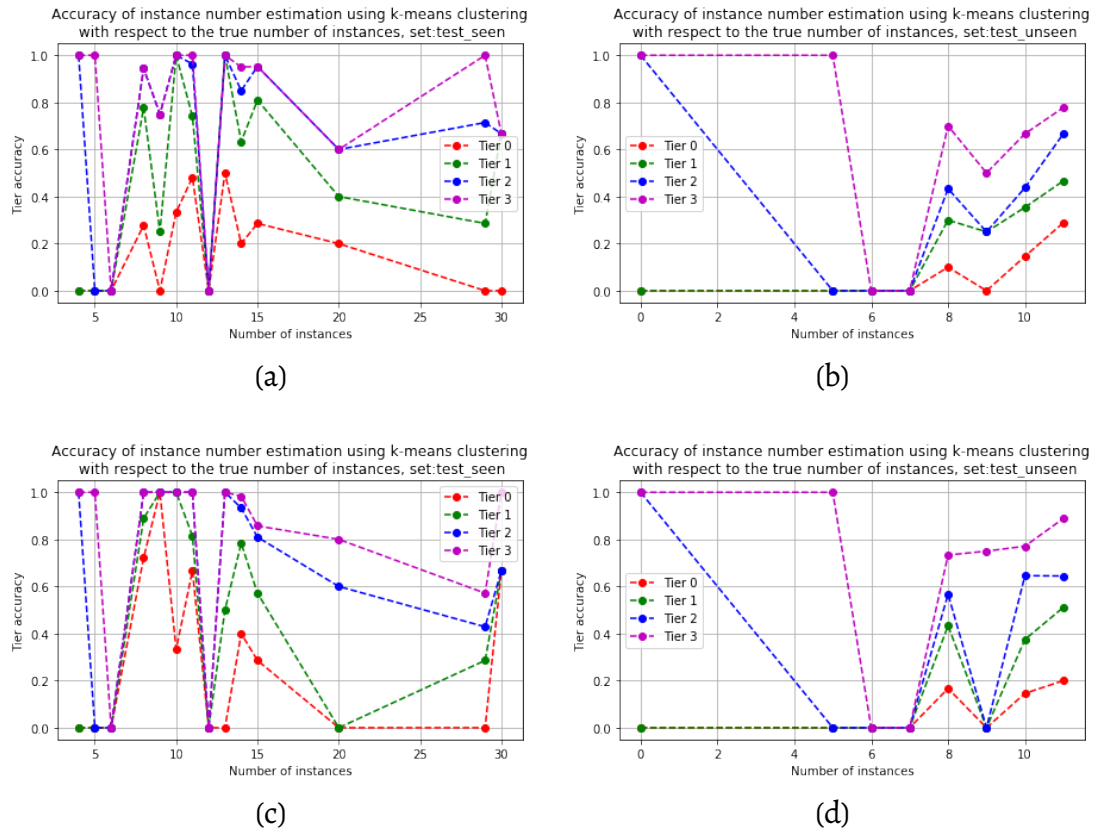


Figure 4.14: Accuracy of instance number estimation based on the silhouette score of the labeling produced by applying the k-means algorithm to the embedding outputs of the UNET models: UNET<sub>ssmax</sub> on test:seen (a), UNET<sub>ssmax</sub> on test:unseen (b), UNET<sub>margin</sub> on test:seen (c), and UNET<sub>margin</sub> on test:unseen (d).

Subset	# of instances	Tier 0	Tier 1	Tier 2	Tier 3	# of samples
Test:seen	14	0.09	0.19	0.39	0.51	57
Test:seen	6	0.00	0.00	0.00	0.00	36
Test:seen	11	0.04	0.11	0.25	0.29	28
Test:seen	15	0.05	0.24	0.33	0.38	21
Test:seen	8	0.00	0.06	0.06	0.06	18
Test:seen	4	0.00	0.00	1.00	1.00	12
Test:seen	29	0.14	0.29	0.43	0.57	7
Test:seen	20	0.00	0.20	0.20	0.40	5
Test:seen	9	0.00	0.00	0.25	0.25	4
Test:seen	30	0.00	0.00	0.33	0.67	3
Test:seen	13	0.00	0.00	0.00	0.00	2
Test:seen	10	0.00	0.00	0.00	0.00	1
Test:seen	5	0.00	0.00	0.00	1.00	1
Test:unseen	7	0.00	0.00	0.00	0.00	48
Test:unseen	10	0.00	0.06	0.17	0.30	47
Test:unseen	11	0.00	0.09	0.16	0.20	45
Test:unseen	8	0.00	0.00	0.03	0.20	30
Test:unseen	6	0.00	0.00	0.00	0.00	17
Test:unseen	5	0.00	0.00	0.00	1.00	4
Test:unseen	9	0.00	0.00	0.00	0.00	4

Table 4.16: Accuracy of estimation of the number of instances present in the image with respect to the number of ground-truth instances separated into 4 tiers - sorted in the descending order of the number of annotated samples. Clustering performed on embeddings produced by  $OP_{ssmax}$  model.

Subset	# of instances	Tier 0	Tier 1	Tier 2	Tier 3	# of samples
Test:seen	4	0.00	0.00	1.00	1.00	12
Test:seen	5	0.00	0.00	0.00	1.00	1
Test:seen	6	0.00	0.00	0.00	0.00	36
Test:seen	8	0.00	0.00	0.00	0.28	18
Test:seen	9	0.00	0.25	0.50	0.50	4
Test:seen	10	0.00	0.00	1.00	1.00	1
Test:seen	11	0.07	0.10	0.10	0.41	29
Test:seen	13	0.00	0.00	0.00	0.00	2
Test:seen	14	0.05	0.20	0.27	0.47	60
Test:seen	15	0.05	0.14	0.48	0.71	21
Test:seen	20	0.20	0.40	0.60	0.60	5
Test:seen	29	0.00	0.00	0.14	0.29	7
Test:seen	30	0.00	0.00	0.00	0.33	3
Test:unseen	5	0.00	0.00	0.00	1.00	4
Test:unseen	6	0.00	0.00	0.00	0.00	17
Test:unseen	7	0.00	0.00	0.00	0.00	49
Test:unseen	8	0.00	0.10	0.20	0.40	30
Test:unseen	9	0.00	0.25	0.25	0.25	4
Test:unseen	10	0.06	0.06	0.11	0.21	47
Test:unseen	11	0.18	0.20	0.27	0.47	45

Table 4.17: Accuracy of estimation of the number of instances present in the image with respect to the number of ground-truth instances separated into 4 tiers - sorted in the descending order of the number of annotated samples. Clustering performed on embeddings produced by  $OP_{margin}$  model.

Subset	# of instances	Tier 0	Tier 1	Tier 2	Tier 3	# of samples
Test:seen	14	0.20	0.63	0.85	0.95	60
Test:seen	6	0.00	0.00	0.00	0.00	36
Test:seen	11	0.48	0.74	0.96	1.00	27
Test:seen	15	0.29	0.81	0.95	0.95	21
Test:seen	8	0.28	0.78	0.94	0.94	18
Test:seen	4	0.00	0.00	1.00	1.00	12
Test:seen	29	0.00	0.29	0.71	1.00	7
Test:seen	20	0.20	0.40	0.60	0.60	5
Test:seen	9	0.00	0.25	0.75	0.75	4
Test:seen	10	0.33	1.00	1.00	1.00	3
Test:seen	30	0.00	0.67	0.67	0.67	3
Test:seen	13	0.50	1.00	1.00	1.00	2
Test:seen	5	0.00	0.00	0.00	1.00	1
Test:seen	12	0.00	0.00	0.00	0.00	1
Test:unseen	7	0.00	0.00	0.00	0.00	50
Test:unseen	10	0.15	0.35	0.44	0.67	48
Test:unseen	11	0.29	0.47	0.67	0.78	45
Test:unseen	8	0.10	0.30	0.43	0.70	30
Test:unseen	6	0.00	0.00	0.00	0.00	17
Test:unseen	5	0.00	0.00	0.00	1.00	4
Test:unseen	9	0.00	0.25	0.25	0.50	4

Table 4.18: Accuracy of estimation of the number of instances present in the image with respect to the number of ground-truth instances separated into 4 tiers - sorted in the descending order of the number of annotated samples. Clustering performed on embeddings produced by UNET<sub>ssmax</sub> model.

Subset	# of instances	Tier 0	Tier 1	Tier 2	Tier 3	# of samples
Test:seen	14	0.40	0.78	0.93	0.98	60
Test:seen	6	0.00	0.00	0.00	0.00	36
Test:seen	11	0.67	0.81	1.00	1.00	27
Test:seen	15	0.29	0.57	0.81	0.86	21
Test:seen	8	0.72	0.89	1.00	1.00	18
Test:seen	4	0.00	0.00	1.00	1.00	12
Test:seen	29	0.00	0.29	0.43	0.57	7
Test:seen	20	0.00	0.00	0.60	0.80	5
Test:seen	9	1.00	1.00	1.00	1.00	4
Test:seen	10	0.33	1.00	1.00	1.00	3
Test:seen	30	0.67	0.67	0.67	1.00	3
Test:seen	13	0.00	0.50	1.00	1.00	2
Test:seen	5	0.00	0.00	0.00	1.00	1
Test:seen	12	0.00	0.00	0.00	0.00	1
Test:unseen	7	0.00	0.00	0.00	0.00	50
Test:unseen	10	0.15	0.38	0.65	0.77	48
Test:unseen	11	0.20	0.51	0.64	0.89	45
Test:unseen	8	0.17	0.43	0.57	0.73	30
Test:unseen	6	0.00	0.00	0.00	0.00	17
Test:unseen	5	0.00	0.00	0.00	1.00	4
Test:unseen	9	0.00	0.00	0.00	0.75	4

Table 4.19: Accuracy of estimation of the number of instances present in the image with respect to the number of ground-truth instances separated into 4 tiers - sorted in the descending order of the number of annotated samples. Clustering performed on embeddings produced by UNET<sub>margin</sub> model.

## 4.6 Pose Estimation Evaluation

Two strategies for Pose Estimation are presented in this work: 1) OpenPose-based baseline method relying on body part detections and part-affinity fields, and 2) a *hybrid* method based on OpenPose but substituting the Part Affinity Field-based cost for embedding vector matching.

Let us recall that in the task of pose estimation the goal is to determine the location and orientation of each instance (pig) in each given image. Although the models used in this work were designed to detect four body parts: left ear, right ear, shoulder and tail, only the last two are required to completely determine the necessary instance parameters. This decision is motivated by the results in Section 4.3.1 confirming that shoulder and tail are the highest scoring keypoints and also, as mentioned in previous work establishing the dataset used here,<sup>160</sup> special care was taken to ensure the quality of the annotation of those keypoints specifically. Using such simplification, each (pig) instance is represented by a pair of shoulder and tail coordinates, such that the instance matching is performed over a set of  $M$  estimated shoulder-tail pairs  $\{(\hat{s}_0, \hat{t}_0), \dots, (\hat{s}_{M-1}, \hat{t}_{M-1})\}$ , and a ground truth set of  $N$  annotations  $\{(s_0, t_0), \dots, (s_{M-1}, t_{M-1})\}$ . An association method is necessary here as the predicted pixel coordinates will not likely contain the exact same values as the ground truth due to spatial inaccuracies as presented in section 4.3.2.

Statistics based on both Hungarian Matching (HM) and Cross Check Matching (CCM) methods are used in the evaluation of the hybrid and baseline methods. Unconstrained Hungarian Matching (Section 3.11.2) allows for assignment between far-away instances in order to minimize the global cost of the entire solution.<sup>160</sup> One way to mitigate that would be to constrain the maximum allowed distance of matching based on prior statistics but this would depend image scale and introduce additional parameters. To avoid that, the CCM is used as in the work of Psota et al.<sup>160</sup> CCM is a strict consistency metric, where the match between the ground truth instance and predicted

instance is established if and only if they are each others' (respective) minimum cost matches. Formally, two instances  $n$  and  $m$  match if and only if:

$$\begin{aligned} m &= \arg \min_{m \in \{0, \dots, M-1\}} (\|s_n - \hat{s}_m\| + \|t_n - \hat{t}_m\|), \\ n &= \arg \min_{n \in \{0, \dots, N-1\}} (\|s_n - \hat{s}_m\| + \|t_n - \hat{t}_m\|), \end{aligned} \quad (4.11)$$

where  $\|x\|$  operation denotes the L2 norm.

#### 4.6.1 Performance ceiling due to representation

To establish the numerical upper-bounds, and determine if the data representations themselves contribute negatively to the methods' performance, the OpenPose and Hybrid methods were evaluated using ground-truth inputs instead of the predictions first.

We encode the keypoint locations as described in Section 3.3 and part affinity fields like described in Section 3.6. We obtain the results by processing encoded keypoints using non-maximum-suppression as described in Section 3.11.1 and part affinity fields using the sampling procedure described in Section 3.11.3. We combine the shoulder and tail keypoints using bipartite matching like described in Section 3.11.2 to obtain object detections. We match the keypoints using two methods: 1) using sampled part affinity fields (PAF) and using equivalence of the identity labels (IDs). We use the identity labels to simulate the behavior of the *Hybrid* method in the best case scenario. We then match those detections to the ground-truth using Hungarian Matching and CCM. We run this analysis for the three subsets of PDD2019 dataset: train, test:seen and test:unseen. The results for all three data sets are presented in Table 4.20.

Selected failure cases were picked to illustrate the sources of non-perfect performance presented in table 4.20 and presented in figure 4.15. The cases were selected using visual inspection from the set of images contributing to the number of False

Set	Matching	TP	FP	FN	Precision	Recall	$F_1$
train	OpenPose (baseline w. PAF), CCM	19831	181	511	0.991	0.975	0.983
train	OpenPose (baseline w. PAF), HM	19991	21	351	0.999	0.983	0.991
train	OpenPose (baseline w. IDs), CCM	19429	583	913	0.971	0.955	0.963
train	OpenPose (baseline w. IDs), HM	19991	21	351	0.999	0.983	0.991
test:seen	OpenPose (baseline w. PAF), CCM	2265	20	38	0.991	0.983	0.987
test:seen	OpenPose (baseline w. PAF), HM	2284	1	19	1.000	0.992	0.996
test:seen	OpenPose (baseline w. IDs), CCM	2232	53	71	0.977	0.969	0.973
test:seen	OpenPose (baseline w. IDs), HM	2284	1	19	1.000	0.992	0.996
test:unseen	OpenPose (baseline w. PAF), CCM	681	0	7	1.000	0.990	0.995
test:unseen	OpenPose (baseline w. PAF), HM	681	0	7	1.000	0.990	0.995
test:unseen	OpenPose (baseline w. IDs), CCM	680	1	8	0.999	0.988	0.993
test:unseen	OpenPose (baseline w. IDs), HM	681	0	7	1.000	0.990	0.995

Table 4.20: Performance upper-bound due to encoding of representations as described in chapter 3. Evaluation using ground-truth only.

Positives or False Negatives in table 4.20. In all cases the animals that are separated from the group seem to be resolved with no problems, and the mismatches tend to occur in the crowded areas of the image. Regardless if the Part Affinity Fields or IDs are used, the method is still prone to failure in cluttered scenes due to the non-maximum-suppressor (Section 3.11.1) combining detections as visible in the bottom line of images in figure 4.15.

Increasing image resolution and assuming less uncertainty thanks to higher quality annotations would mitigate the problem at the level of ground-truth generation. Currently our networks were processing images with the number of pixels in the longer edge of  $W = 512$  for the UNET networks, and  $W = 1024$  for the OP networks. Increase in the amount of memory available on the GPUs will contribute positively to the ability of processing significantly larger images.

#### 4.6.2 Evaluation using predictions from Deep CNNs

The final result for both model architectures (OP and UNET) with embeddings trained using two different criteria (Silhouette Score Maximization and Margin method), and two different matching methods are presented table 4.21 in descending order of the  $F_1$  score.

When looking at the top positions for all three subsets, it becomes apparent, that the use of Hungarian Matching method tends to yield higher  $F_1$  values for all subsets. This

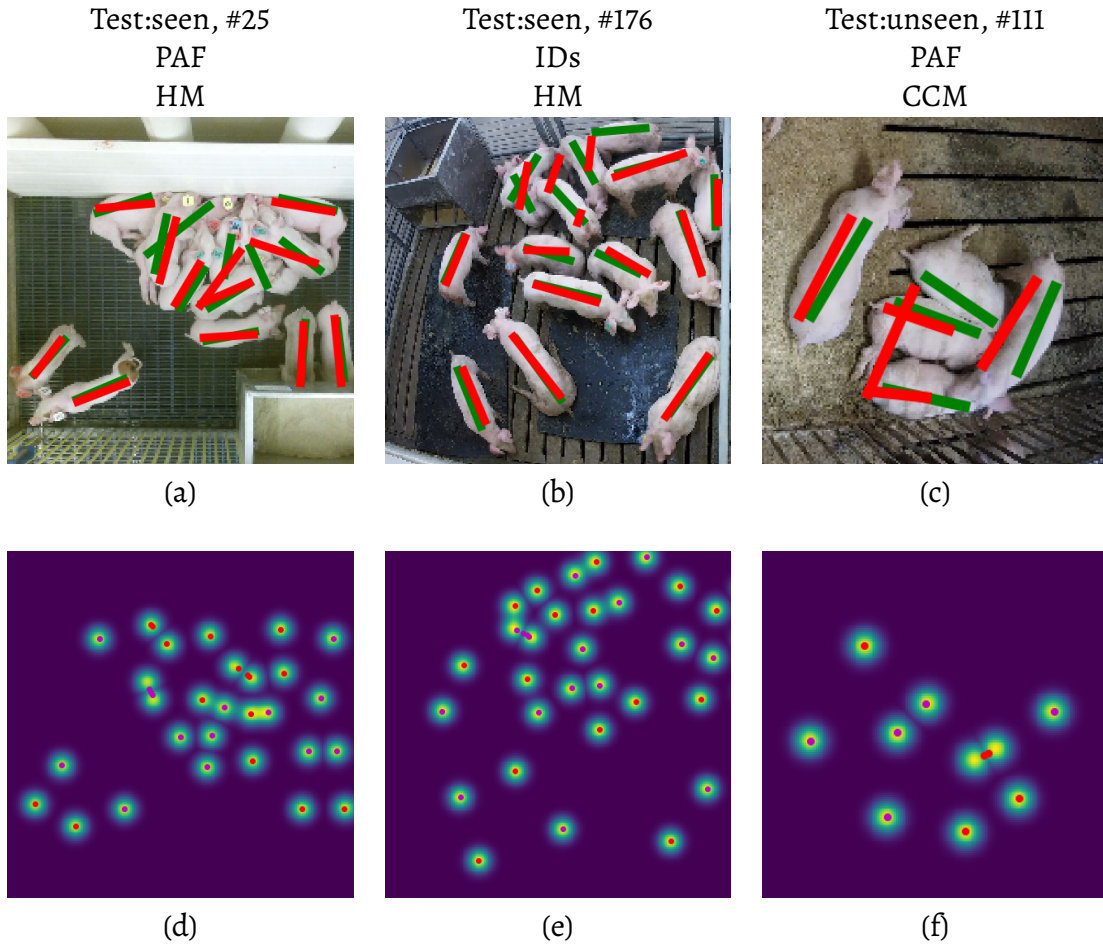


Figure 4.15: Illustrations of problematic situations in which the selected method of ground-truth encoding produces inputs causing the pose estimation method to **fail**. In the color images (a, b, c), instances are represented using lines drawn across the back. Green color represents the lines drawn using annotated data, and the red is reserved for *predictions*. Bottom images (d, e, f) illustrate the corresponding keypoint encodings via gaussian kernels, and the outputs of the non-maximum-suppressor by the red dots.

confirms the initial concerns presented in.<sup>160</sup> Thus, the rest of the analysis will be presented with respect to the CCM score.

Ignoring other factors than network architecture, it is visible that for test:seen and training subsets the UNET model always places itself above the OP with respect to the  $F_1$  score. In case of test:unseen the results are however intertwined by all considered architectures. We match the performance of Psota et al.<sup>160</sup> as visible when comparing rows 18 and 34 of Table 4.21. It is worth to note that our UNET model is different than the

one by Psota et al. The main differences include: 1) lack of max-unpooling layers, 2) composite output containing foreground mask and embeddings in addition to the keypoint heatmaps and part affinity fields.

#	Model	Set	Matching	TP	FP	FN	Precision	Recall	$F_1$
1	UNET <sub>margin</sub>	train	EMB, HM	19352	105	990	0.995	0.951	0.972
2	UNET <sub>margin</sub>	train	PAF, HM	19352	105	990	0.995	0.951	0.972
3	UNET <sub>ssmax</sub>	train	EMB, HM	19264	175	1078	0.991	0.947	0.969
4	UNET <sub>ssmax</sub>	train	PAF, HM	19264	175	1078	0.991	0.947	0.969
5	UNET <sub>margin</sub>	train	EMB, CCM	18839	618	1503	0.968	0.926	0.947
6	UNET <sub>margin</sub>	train	PAF, CCM	18761	696	1581	0.964	0.922	0.943
7	UNET <sub>ssmax</sub>	train	EMB, CCM	18689	750	1653	0.961	0.919	0.940
8	UNET <sub>ssmax</sub>	train	PAF, CCM	18589	850	1753	0.956	0.914	0.935
9	OP <sub>margin</sub>	train	EMB, HM	16231	150	4111	0.991	0.798	0.884
10	OP <sub>margin</sub>	train	PAF, HM	16231	150	4111	0.991	0.798	0.884
11	OP <sub>ssmax</sub>	train	EMB, HM	15568	141	4774	0.991	0.765	0.864
12	OP <sub>ssmax</sub>	train	PAF, HM	15568	141	4774	0.991	0.765	0.864
13	OP <sub>margin</sub>	train	PAF, CCM	15226	1155	5116	0.929	0.749	0.829
14	OP <sub>ssmax</sub>	train	PAF, CCM	14499	1210	5843	0.923	0.713	0.804
15	OP <sub>ssmax</sub>	train	EMB, CCM	14110	1599	6232	0.898	0.694	0.783
16	OP <sub>margin</sub>	train	EMB, CCM	14070	2311	6272	0.859	0.692	0.766
17	UNET <sub>Psota et al.</sub>	train	PAF, CCM	19999	13	743	0.964	0.999	0.981
18	UNET <sub>margin</sub>	test:seen	EMB, HM	2205	18	98	0.992	0.957	0.974
19	UNET <sub>margin</sub>	test:seen	PAF, HM	2205	18	98	0.992	0.957	0.974
20	UNET <sub>ssmax</sub>	test:seen	EMB, HM	2192	40	111	0.982	0.952	0.967
21	UNET <sub>ssmax</sub>	test:seen	PAF, HM	2192	40	111	0.982	0.952	0.967
22	UNET <sub>margin</sub>	test:seen	EMB, CCM	2137	86	166	0.961	0.928	0.944
23	UNET <sub>margin</sub>	test:seen	PAF, CCM	2118	105	185	0.953	0.920	0.936
24	UNET <sub>ssmax</sub>	test:seen	EMB, CCM	2110	122	193	0.945	0.916	0.931
25	UNET <sub>ssmax</sub>	test:seen	PAF, CCM	2105	127	198	0.943	0.914	0.928
26	OP <sub>margin</sub>	test:seen	EMB, HM	1779	28	524	0.985	0.772	0.866
27	OP <sub>margin</sub>	test:seen	PAF, HM	1779	28	524	0.985	0.772	0.866
28	OP <sub>ssmax</sub>	test:seen	EMB, HM	1703	26	600	0.985	0.739	0.845
29	OP <sub>ssmax</sub>	test:seen	PAF, HM	1703	26	600	0.985	0.739	0.845
30	OP <sub>margin</sub>	test:seen	PAF, CCM	1647	160	656	0.911	0.715	0.801
31	OP <sub>ssmax</sub>	test:seen	PAF, CCM	1553	176	750	0.898	0.674	0.770
32	OP <sub>ssmax</sub>	test:seen	EMB, CCM	1521	208	782	0.880	0.660	0.754
33	OP <sub>margin</sub>	test:seen	EMB, CCM	1542	265	761	0.853	0.670	0.750
34	UNET <sub>Psota et al.</sub>	test:seen	PAF, CCM	2273	1	94	0.960	1.000	0.980
35	UNET <sub>margin</sub>	test:unseen	EMB, HM	656	84	32	0.886	0.953	0.919
36	UNET <sub>margin</sub>	test:unseen	PAF, HM	656	84	32	0.886	0.953	0.919
37	UNET <sub>ssmax</sub>	test:unseen	EMB, HM	671	107	17	0.862	0.975	0.915
38	UNET <sub>ssmax</sub>	test:unseen	PAF, HM	671	107	17	0.862	0.975	0.915
39	UNET <sub>margin</sub>	test:unseen	EMB, CCM	615	125	73	0.831	0.894	0.861
40	UNET <sub>ssmax</sub>	test:unseen	EMB, CCM	627	151	61	0.806	0.911	0.855
41	UNET <sub>margin</sub>	test:unseen	PAF, CCM	606	134	82	0.819	0.881	0.849
42	UNET <sub>ssmax</sub>	test:unseen	PAF, CCM	622	156	66	0.799	0.904	0.849
43	OP <sub>margin</sub>	test:unseen	EMB, HM	459	0	229	1.000	0.667	0.800
44	OP <sub>margin</sub>	test:unseen	PAF, HM	459	0	229	1.000	0.667	0.800
45	OP <sub>ssmax</sub>	test:unseen	EMB, HM	454	0	234	1.000	0.660	0.795
46	OP <sub>ssmax</sub>	test:unseen	PAF, HM	454	0	234	1.000	0.660	0.795
47	OP <sub>margin</sub>	test:unseen	PAF, CCM	424	35	264	0.924	0.616	0.739
48	OP <sub>ssmax</sub>	test:unseen	PAF, CCM	403	51	285	0.888	0.586	0.706
49	OP <sub>margin</sub>	test:unseen	EMB, CCM	402	57	286	0.876	0.584	0.701
50	OP <sub>ssmax</sub>	test:unseen	EMB, CCM	396	58	292	0.872	0.576	0.694
51	UNET <sub>Psota et al.</sub>	test:unseen	PAF, CCM	1150	112	573	0.667	0.911	0.771

Table 4.21: Performance evaluation results for the OP and UNET models, for the OpenPose (PAF) and Hybrid (EMB) method, with HM and CCM matching for all three datasets.

## 4.7 Semantic Instance Segmentation Evaluation

This section describes the Semantic Instance Segmentation method as a purely clustering-based method relying only on foreground mask, embedding vectors, and known number of instances  $k$ . It was determined that attempts on guessing the number of instances purely on the embedding vectors themselves does not yield satisfying results as shown in Section 4.5.3. Thus, a ground-truth number of instances is used instead. Here, due to the fact that method produces labels spanning the spatial extent of the image, no keypoint locations are estimated. Thus, matching has to be done differently. Our measure of accuracy for this task is based on the TPR (recall, Equation. 4.2). We use the CCM matching (Equation 4.11) method with respect to the Intersection Over Union (IoU) threshold  $IoU_{th} \in (0, 1)$ . Numerical results are presented in Table 4.22 and in Figures 4.16, 4.16, and 4.16.

When investigating the values in Table 4.22 it is visible that models trained using the  $L_{margin}$  achieved marginally higher score for the training and test:seen subsets. When comparing the models, UNET architecture significantly outperformed the OP model for every subset. It is partially attributed to the fact that the OP model produces *thinner* looking shapes in the foreground masks. Investigation of Figures 4.16 through 4.18 indicates that even the peak recall for the OP model does not exceed the values for UNET at  $IoU_{th} = 0.5$ . This result can be attributed to fact that UNET is more suited for segmentation tasks like indicated in the literature.

When comparing the results of the  $UNET_{margin}$  model for the training subset with the results for the test:seen subset in Table 4.22, the values indicate that the model generalized well for the task of semantic instance segmentation. Comparison with the test:unseen dataset has to be made while keeping in mind the substantial difference between the subsets.

Figure 4.19 depicts multiple selected cases of the output of our method when applied

Model	Set	TP	TOTAL	Recall
UNET <sub>margin</sub>	train	16866	20342	0.829
UNET <sub>ssmax</sub>	train	16795	20342	0.826
OP <sub>margin</sub>	train	7340	20342	0.361
OP <sub>ssmax</sub>	train	6651	20342	0.327
UNET <sub>margin</sub>	test:seen	1842	2303	0.800
UNET <sub>ssmax</sub>	test:seen	1841	2303	0.799
OP <sub>margin</sub>	test:seen	790	2303	0.343
OP <sub>ssmax</sub>	test:seen	707	2303	0.307
UNET <sub>ssmax</sub>	test:unseen	399	688	0.580
UNET <sub>margin</sub>	test:unseen	383	688	0.557
OP <sub>margin</sub>	test:unseen	230	688	0.334
OP <sub>ssmax</sub>	test:unseen	187	688	0.272

Table 4.22: Results of the semantic instance segmentation using clustering of the deep multi-dimensional embeddings at  $\text{IoU}_{\text{th}} = 0.5$  with known number of instances  $k$ .

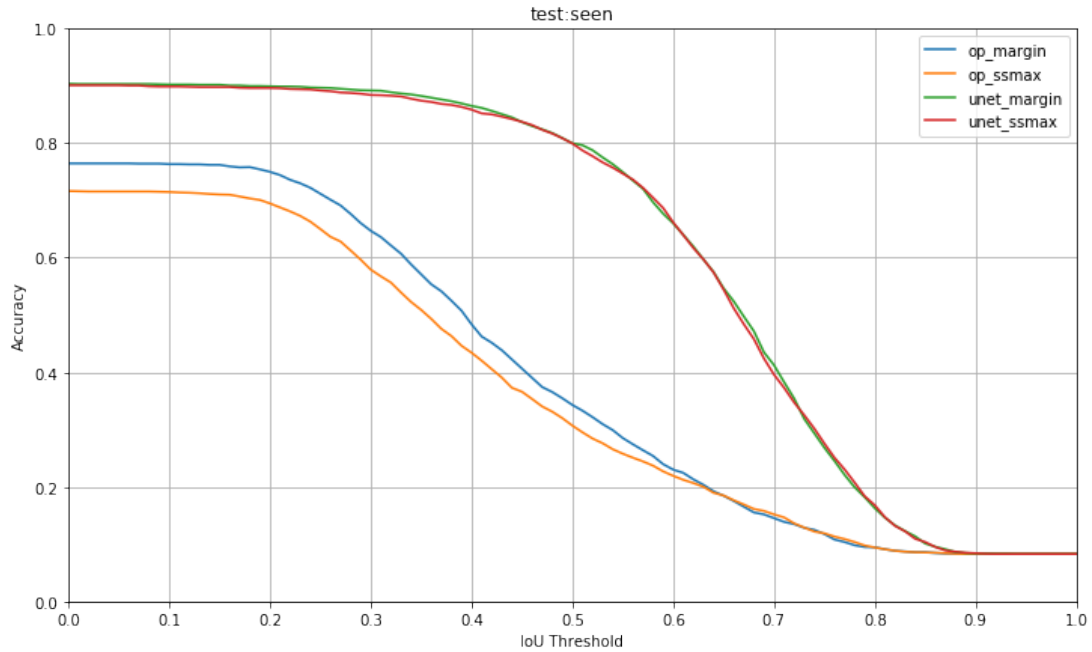


Figure 4.16: Recall curve of Semantic Instance Segmentation method with respect to the IoU threshold for the test:seen subset of PDD2019.

to the group-housed pigs. Figure 4.19 (a) shows the case of a rare texture on a big animal which caused the foreground estimator to fail to produce the appropriate semantic segmentation mask. Figure 4.19 (b) shows a successful case for the test:unseen with a minor mask misalignment for the purple pig in the bottom-right corner. Figure 4.19 (c)

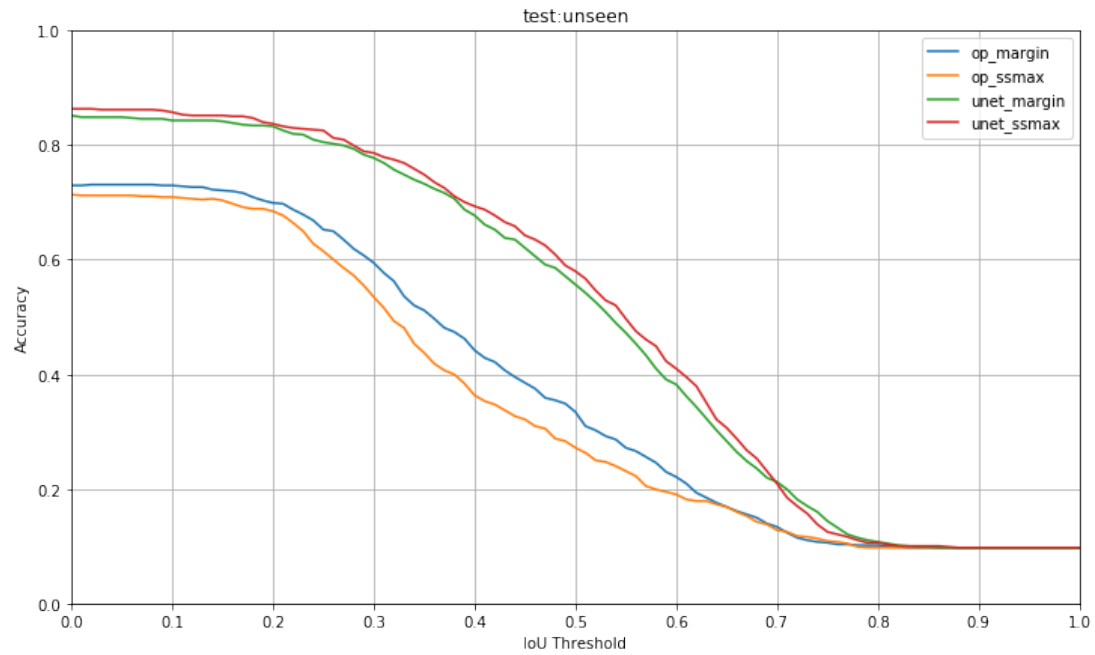


Figure 4.17: Recall curve of Semantic Instance Segmentation method with respect to the IoU threshold for the test:unseen subset of PDD2019.

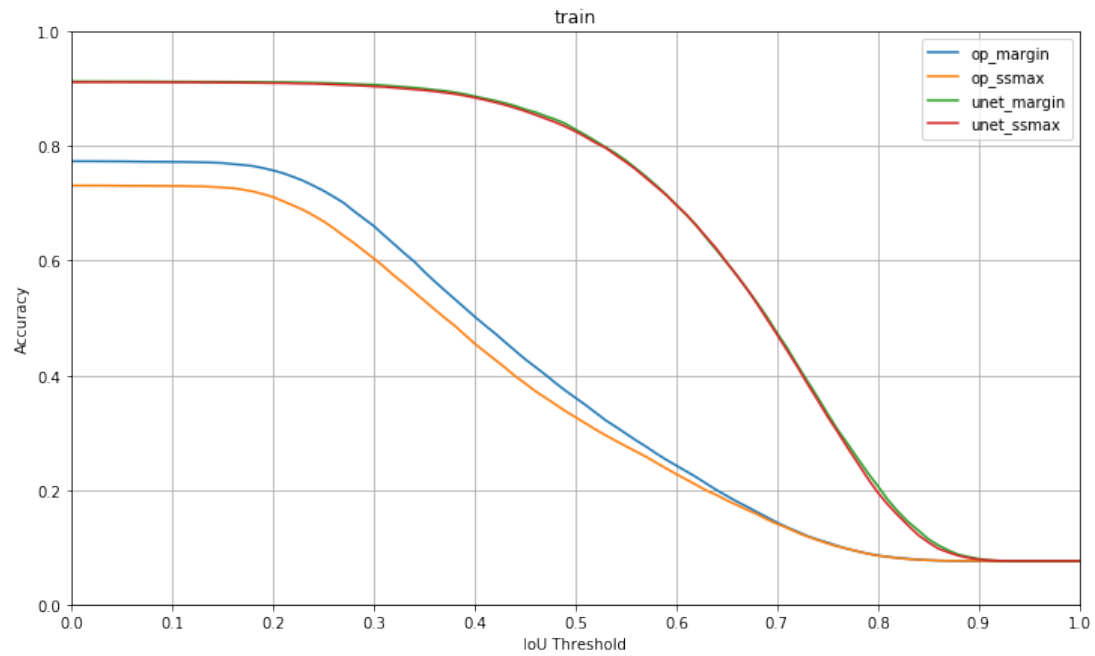
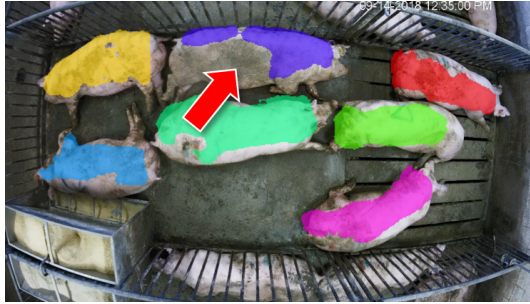


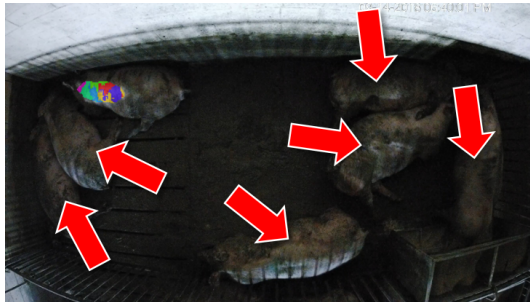
Figure 4.18: Recall curve of Semantic Instance Segmentation method with respect to the IoU threshold for the training subset of PDD2019.



(a) test:unseen #51



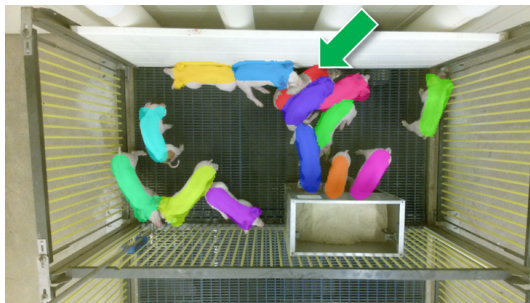
(b) test:unseen # 54



(c) test:unseen # 12



(d) test:seen # 56



(e) test:seen # 28



(f) test:seen # 23

Figure 4.19: Selected cases of failure and success of our semantic instance segmentation of group-housed pigs. Examples picked for visual inspection.

shows the utter failure of our method as the animals are barely visible due to the lighting conditions. Figure 4.19 (d) shows another successful output, this time for the test:seen subset. Figures 4.19 (e) and (f) indicate successful handling of partial occlusions thanks to the embeddings as the instances indicated by the green arrow are properly identified.

## CHAPTER 5

---

### Conclusion

In this work we have presented a novel method of processing large scale, sparsely annotated datasets using machine learning for the tasks of semantic segmentation, object detection, pose estimation and semantic instance segmentation. We have shown that the outputs of those tasks are essential in the context of multiple object tracking. The key factor contributing to our success was the use of deep convolutional neural network to produce multi-dimensional embeddings which after minor post-processing can be converted to unique per-pixel instance membership labels. The use of embeddings was also show successful in the task of pose estimation.

#### 5.1 Challenges

One of the common challenges in the process of designing an application-specific tracking method is the selection of proper equipment and data collection strategy. It was witnessed in the literature that researchers struggled with processing images from cameras with significant distortion introduced by wide angle lenses.<sup>42</sup>

Tracking multiple objects usually depends heavily on either of the two clues: appearance or motion. In our application we were faced with the task of tracking homogeneous objects that move randomly. Thus, a classic MOT approach using weak appearance affinity measures was bound to fail.

Occlusions are the main challenge in MOT. In methods that heavily depend on object detection, lack of thereof puts the entire responsibility of resolving trajectories on the tracking algorithm and the affinity measure between the target before and after the occlusion has happened. To overcome the problem of occlusions one can resort to building a temporally global object descriptor that will retain the value specific to the object and not its position or orientation in the image.

Novel methods of object detection including impressive Mask r-cnn<sup>64</sup> achieve remarkable performance due to availability of high quality datasets like MPII<sup>177</sup> or COCO.<sup>8</sup> Those datasets target the general purpose methods and contain natural images. In the context of our application however, existing methods are not well suited to handle the specific properties of homogeneous object tracking. The lack of domain-specific datasets was another major challenge that this work overcame.

## 5.2 Approaches and contributions

We tackled the data collection problem by deploying a custom platform based around Microsoft Kinect v2 camera allowing for simultaneous capture of color and depth image pairs with relatively minimal distortion. We describe this process in Section 3.1.

We used deep-multi-dimensional embeddings as easy-to-cluster features to produce per-pixel membership labels for the task of semantic instance segmentation of group-housed animals that look very similar. Thus, we produced a reliable object detector with a mask descriptor suitable for multiple object tracking.

Due to the lack of z-ordering or depth information in our annotated dataset<sup>160</sup> we resorted to producing ground-truth per-pixel labels in random order. We chose this approach as an attempt to force the neural network to (stochastically) produce per-pixel embeddings representing the true instance membership. We identified a few examples of robustness against partial object occlusions. We attribute those successful examples to

the chosen method of ground-truth generation.

We based our method around pose estimation with additional features: embeddings and foreground mask. We used those features to be able to process sequences of video using a tracker of arbitrary complexity to build datasets for tracking of group housed pigs.

We confirm the findings from our previous contributions in<sup>160</sup> regarding the model selection. Our UNET model trained using the  $L_{\text{margin}}$  loss function achieved 99% precision and 96% recall in the task of bottom-up object detection using pose estimation when exposed to examples matching the environments and lighting conditions of the training set. It also achieved 87% precision and 95% recall on the subset containing images with drastically different properties. Our method matches the performance of the model by Psota et al.<sup>160</sup> on the *test:seen* subset but is more likely to produce false negative results on the *test:unseen*.

We extend on the results presented in Psota et al.<sup>160</sup> by reporting 82% accuracy on matching instances detected using our semantic instance segmentation technique at intersection-over-union threshold of 0.5. Our method provides semantic instance segmentation masks which (after manual inspection and adjustment) can be included in the next revision of the Pig Part Detection dataset.<sup>160</sup>

### 5.3 Recommendations

The use of vast dataset of the color and depth image pairs was focused to the foreground mask extraction. The description of the method of obtaining those masks from color-depth image pairs is a contribution that is believed to be a good starting point for more challenging tasks, and introduces the reader to aspects of multi-view geometry. Initial intuition which motivated the capture of depth images however was a correct path. Use of Microsoft Kinect v2 allowed for capturing color images in  $1920 \times 1080$

resolution which ensured *longevity* of the collected datasets over the span of multiple years. Author would like to recommend the use high-resolution, high quality image sensors to the practitioners in the stage of initial data collection for product development. Early attempts to instance-level segmentation on depth images did not yield the sought impact at the time of completing this work when compared to working with manually annotated sets. Author however believes that such approach should be revisited in the future. Author would like to recommend exploring mesh processing and instance extraction from depth images to the practitioners.

The choice of TensorFlow<sup>5</sup> as the main computational framework was motivated by its superior at the time abilities of symbolic operations definition, *automatic differentiation*, ease of feeding the data, and very compelling *Python* interface with the ability to export the graph for deployment. Support of Nvidia GPUs and quick response of the developers to the CUDA framework made it an excellent choice. However, it lacks flexibility in the sense of selective gradient application when using implemented solvers. Namely, in the sense of Stochastic Gradient Descent, one would like to *accumulate* the gradient-based corrections over the course of multiple training examples in a memory-efficient fashion, and then apply it after multiple forward passes. TensorFlow implicitly allows for that operation through the use of batches - as the inputs are conventionally formatted in a  $B \times H \times W \times C$ . Modern frameworks like PyTorch<sup>178</sup> allow (and unfortunately require) specification of gradient calculation and application explicitly.

## 5.4 Future work

To tackle the problem of occlusions directly using machine learning, the author would like to proceed with another attempt based on the synthetic dataset generation. Given the input images and semantic instance segmentation masks produced by the method presented in this work one could produce a set of plausible composition of instances with

known occlusion properties by injecting a foreign instance *on top* of the existing image. Having known instance segmentation masks for the original image and the introduced foreign object one can train a neural network to estimate the features indicating the presence of occlusions. To produce convincing images author explored the concept of blending images using Laplacian pyramids presented in Section A.1.

---

## References

- [1] Wenhao Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Xiaowei Zhao, and Tae-Kyun Kim. Multiple object tracking: A literature review. *arXiv preprint arXiv:1409.7618*, 2014.
- [2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [3] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- [5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul

- Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
  - [7] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.
  - [8] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
  - [9] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: <http://www.cs.toronto.edu/kriz/cifar.html>*, 2014.
  - [10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.
  - [11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
  - [12] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7263–7271, 2017.

- [13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [14] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [15] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [16] CY Ren, VA Prisacariu, O Kähler, ID Reid, and DW Murray. Real-time tracking of single and multiple objects from depth-colour imagery using 3d signed distance functions. *International Journal of Computer Vision*, 124(1):80–95, 2017.
- [17] FAM Tuytens, Sophie de Graaf, Jasper LT Heerkens, Leonie Jacobs, Elena Nalon, Sanne Ott, Lisanne Stadig, Eva Van Laer, and Bart Ampe. Observer bias in animal behaviour research: can we believe what we score, if we score what we believe? *Animal Behaviour*, 90:273–280, 2014.
- [18] PIC North America. Standard animal care: Daily routines. In *Wean to finish manual*, pages 23–24, 2014.
- [19] David Mellor. Updating animal welfare thinking: Moving beyond the “five freedoms” towards “a life worth living”. *Animals*, 6(3):21, 2016.
- [20] Christian Boessen, Georgeanne Artz, and Lee Schulz. A baseline study of labor issues and trends in us pork production, 2018.
- [21] S. G. Matthews, A. L. Miller, J. Clapp, T. Plötz, and I. Kyriazakis. Early detection of health and welfare compromises through automated detection of behavioural changes in pigs. *The Veterinary Journal*, 217:43–51, November 2016.

- [22] Abozar Nasirahmadi, Sandra A Edwards, and Barbara Sturm. Implementation of machine vision for detecting behaviour of cattle and pigs. *Livestock Science*, 202:25–38, 2017.
- [23] CM Wathes, Helle Halkjær Kristensen, J-M Aerts, and Daniel Berckmans. Is precision livestock farming an engineer's daydream or nightmare, an animal's friend or foe, and a farmer's panacea or pitfall? *Computers and electronics in agriculture*, 64(1):2–10, 2008.
- [24] Thomas M Banhazi, H Lehr, JL Black, H Crabtree, P Schofield, M Tschärke, and D Berckmans. Precision livestock farming: an international review of scientific and commercial aspects. *International Journal of Agricultural and Biological Engineering*, 5(3):1–9, 2012.
- [25] E Tullo, I Fontana, and M Guarino. Precision livestock farming: an overview of image and sound labelling. In *European Conference on Precision Livestock Farming 2013:(PLF) EC-PLF*, pages 30–38. KU Leuven, 2013.
- [26] So-Hyeon Kim, Do-Hyeun Kim, and Hee-Dong Park. Animal situation tracking service using rfid, gps, and sensors. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 153–156. IEEE, 2010.
- [27] A. Stukenborg, I. Traulsen, B. Puppe, U. Presuhn, and J. Krieter. Agonistic behaviour after mixing in pigs under commercial farm conditions. *Applied Animal Behaviour Science*, 129(1):28–35, 2011.
- [28] SMC Porto, C Arcidiacono, A Giummarra, U Anguzza, and G Cascone. Localisation and identification performances of a real-time location system based on ultra wide band technology for monitoring and tracking dairy cow behaviour in a semi-open free-stall barn. *Computers and electronics in agriculture*, 108:221–229, 2014.

- [29] Guerino Giancola, Ljubica Blazevic, Isabelle Bucaille, Luca De Nardis, M-G Di Benedetto, Yves Durand, Gwillerm Froc, Begoña Molinete Cuezva, J-B Pierrot, Pekka Pirinen, et al. Uwb mac and network solutions for low data rate with location and tracking applications. In *2005 IEEE International Conference on Ultra-Wideband*, pages 758–763. IEEE, 2005.
- [30] Patrick E Clark, Douglas E Johnson, Mark A Kniep, Phillip Jermann, Brad Huttash, Andrew Wood, Michael Johnson, Craig McGillivan, and Kevin Titus. An advanced, low-cost, gps-based animal tracking system. *Rangeland Ecology & Management*, 59(3):334–340, 2006.
- [31] Mac Schwager, Dean M Anderson, Zack Butler, and Daniela Rus. Robust classification of animal tracking data. *Computers and Electronics in Agriculture*, 56(1):46–59, 2007.
- [32] K. Taylor. Cattle health monitoring using wireless sensor networks. In *Proceedings of the Communication and Computer Networks Conference*, 2004.
- [33] L. Ruiz-Garcia, L. Lunadei, P. Barreiro, and I. Robla. A Review of Wireless Sensor Technologies and Applications in Agriculture and Food Industry: State of the Art and Current Trends. *Sensors*, 9(6):4728–4750, June 2009.
- [34] H. J. Escalante, S. V. Rodriguez, J. Cordero, A. R. Kristensen, and C. Cornou. Sow-activity classification from acceleration patterns: a machine learning approach. *Computers and electronics in agriculture*, 93:17–26, 2013.
- [35] F. A. P. Alvarenga, I. Borges, L. Palkovič, J. Rodina, V. H. Oddy, and R. C. Dobos. Using a three-axis accelerometer to identify and classify sheep behaviour at pasture. *Applied Animal Behaviour Science*, 181:91–99, August 2016.
- [36] Athanasios S Voulodimos, Charalampos Z Patrikakis, Alexander B Sideridis, Vasileios A Ntafis, and Eftychia M Xylouri. A complete farm management system

- based on animal identification using rfid technology. *Computers and electronics in agriculture*, 70(2):380–388, 2010.
- [37] Jianying Feng, Zetian Fu, Zaiqiong Wang, Mark Xu, and Xiaoshuan Zhang. Development and evaluation on a rfid-based traceability system for cattle/beef quality safety in china. *Food control*, 31(2):314–325, 2013.
  - [38] Raymond E Floyd. Rfid in animal-tracking applications. *IEEE Potentials*, 34(5):32–33, 2015.
  - [39] Mateusz Mittek, Eric T Psota, Lance C Pérez, Ty Schmidt, and Benny Mote. Health monitoring of group-housed pigs using depth-enabled multi-object tracking. In *Proceedings of Int Conf Pattern Recognit, Workshop on Visual observation and analysis of Vertebrate And Insect Behavior*, 2016.
  - [40] Mateusz Mittek, Eric T Psota, Jay D Carlson, Lance C Pérez, Ty Schmidt, and Benny Mote. Tracking of group-housed pigs using multi-ellipsoid expectation maximisation. *IET Computer Vision*, 12(2):121–128, 2017.
  - [41] Suresh Neethirajan. Recent advances in wearable sensors for animal health management. *Sensing and Bio-Sensing Research*, 12:15–29, 2017.
  - [42] P. Ahrendt, T. Gregersen, and H. Karstoft. Development of a real-time computer vision system for tracking loose-housed pigs. *Computers and Electronics in Agriculture*, 76(2):169 – 174, 2011.
  - [43] Salah Sukkarieh, Eduardo Mario Nebot, and Hugh F Durrant-Whyte. A high integrity imu/gps navigation loop for autonomous land vehicle applications. *IEEE Transactions on Robotics and Automation*, 15(3):572–578, 1999.
  - [44] Hakan Ardö, Oleksiy Guzhva, Mikael Nilsson, and Anders H Herlin. Convolutional neural network-based cow interaction watchdog. *IET Computer Vision*, 12(2):171–177, 2017.

- [45] Miso Ju, Yunchang Choi, Jihyun Seo, Jaewon Sa, Sungju Lee, Yongwha Chung, and Daihee Park. A kinect-based segmentation of touching-pigs for real-time monitoring. *Sensors*, 18(6):1746, 2018.
- [46] Massimo Piccardi. Background subtraction techniques: a review. In *Systems, man and cybernetics, 2004 IEEE international conference on*, volume 4, pages 3099–3104. IEEE, 2004.
- [47] Eric T Psota, Lance C Perez, Mateusz Mittek, and Ty Schmidt. Systems for tracking individual animals in a group-housed environment, May 9 2019. US Patent App. 16/114,565.
- [48] A. Nasirahmadi, U. Richter, O. Hensel, S. Edwards, and B. Sturm. Using machine vision for investigation of changes in pig group lying patterns. *Computers and Electronics in Agriculture*, 119:184–190, 2015.
- [49] M. A. Kashiha, C. Bahr, S. Ott, C. PH Moons, T. A. Niewold, F. Tuytens, and D. Berckmans. Automatic monitoring of pig locomotion using image analysis. *Livestock Science*, 159:141–148, 2014.
- [50] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.
- [51] Stephen G Matthews, Amy L Miller, Thomas Plötz, and Ilias Kyriazakis. Automated tracking to measure behavioural changes in pigs for health and welfare monitoring. *Scientific reports*, 7(1):17582, 2017.
- [52] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. Tracking without bells and whistles. *arXiv preprint arXiv:1903.05625*, 2019.
- [53] Wongun Choi. Near-online multi-target tracking with aggregated local flow descriptor. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3029–3037, 2015.

- [54] Afshin Dehghan. Global data association for multiple pedestrian tracking. 2016.
- [55] Joao F Henriques, Joao Carreira, Rui Caseiro, and Jorge Batista. Beyond hard negative mining: Efficient detector learning via block-circulant decomposition. In *proceedings of the IEEE International Conference on Computer Vision*, pages 2760–2767, 2013.
- [56] Hedi Harzallah, Frédéric Jurie, and Cordelia Schmid. Combining efficient object localization and image classification. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 237–244. IEEE, 2009.
- [57] Andrea Vedaldi, Varun Gulshan, Manik Varma, and Andrew Zisserman. Multiple kernels for object detection. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 606–613. IEEE, 2009.
- [58] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2544–2550. IEEE, 2010.
- [59] David S Bolme, Bruce A Draper, and J Ross Beveridge. Average of synthetic exact filters. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2105–2112. IEEE, 2009.
- [60] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.
- [61] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

- [62] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.
- [63] Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Multi-person tracking by multicut and deep matching. In *European Conference on Computer Vision*, pages 100–111. Springer, 2016.
- [64] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.
- [65] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.
- [66] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.
- [67] Kaiqiang Wei and Xu Zhao. Multiple-Branches Faster RCNN for Human Parts Detection and Pose Estimation. In *Asian Conference on Computer Vision*, pages 453–462. Springer, 2016.
- [68] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [69] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2017.
- [70] Gary R Bradski. Computer vision face tracking for use in a perceptual user interface. 1998.

- [71] Li Zhang, Yuan Li, and Ramakant Nevatia. Global data association for multi-object tracking using network flows. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.
- [72] Y Bar-Shalom, T Fortmann, M Scheffe, and others. Joint probabilistic data association for multiple targets in clutter. In *Proc. Conf. on Information Sciences and Systems*, pages 404–409, 1980.
- [73] Afshin Dehghan, Shayan Modiri Assari, and Mubarak Shah. Gmmcp tracker: Globally optimal generalized maximum multi clique problem for multiple object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4091–4099, 2015.
- [74] Amir Roshan Zamir, Afshin Dehghan, and Mubarak Shah. Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs. In *Computer Vision–ECCV 2012*, pages 343–356. Springer, 2012.
- [75] Hamed Pirsiavash, Deva Ramanan, and Charless C Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1201–1208. IEEE, 2011.
- [76] Jerome Berclaz, Francois Fleuret, Engin Turetken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. *IEEE transactions on pattern analysis and machine intelligence*, 33(9):1806–1819, 2011.
- [77] Laura Leal-Taixé, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, and Stefan Roth. Tracking the Trackers: An Analysis of the State of the Art in Multiple Object Tracking. *arXiv preprint arXiv:1704.02781*, 2017.
- [78] Chanh Kim, Fuxin Li, Arridhana Ciptadi, and James M Rehg. Multiple hypothesis tracking revisited. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4696–4704, 2015.

- [79] Donald Reid. An algorithm for tracking multiple targets. *IEEE transactions on Automatic Control*, 24(6):843–854, 1979.
- [80] Ingemar J. Cox and Sunita L. Hingorani. An efficient implementation of Reid’s multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Transactions on pattern analysis and machine intelligence*, 18(2):138–150, 1996.
- [81] Dimitri J Papageorgiou and Michael R Salpukas. The maximum weight independent set problem for data association in multiple hypothesis tracking. In *Optimization and Cooperative Control Strategies*, pages 235–255. Springer, 2009.
- [82] Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.
- [83] Justin Domke and Yiannis Aloimonos. Deformation and Viewpoint Invariant Color Histograms. In *BMVC*, pages 509–518, 2006.
- [84] Siyu Tang, Bjoern Andres, Miykhaylo Andriluka, and Bernt Schiele. Subgraph decomposition for multi-target tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5033–5041, 2015.
- [85] Margret Keuper, Evgeny Levinkov, Nicolas Bonneel, Guillaume Lavoué, Thomas Brox, and Bjorn Andres. Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1751–1759, 2015.
- [86] Sunil Chopra and Mendu R Rao. The partition problem. *Mathematical Programming*, 59(1-3):87–115, 1993.
- [87] Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. *arXiv preprint arXiv:1701.01909*, 2017.

- [88] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-RNN: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.
- [89] Anton Milan, Stefan Roth, and Konrad Schindler. Continuous energy minimization for multitarget tracking. *IEEE transactions on pattern analysis and machine intelligence*, 36(1):58–72, 2014.
- [90] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.
- [91] Stefan Walk, Nikodem Majer, Konrad Schindler, and Bernt Schiele. New features and insights for pedestrian detection. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 1030–1037. IEEE, 2010.
- [92] Min Hu, Saad Ali, and Mubarak Shah. Detecting global motion patterns in complex videos. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–5. IEEE, 2008.
- [93] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.
- [94] Andreas Ess, Bastian Leibe, Konrad Schindler, and Luc Van Gool. Robust multiperson tracking from a mobile platform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10):1831–1846, 2009.
- [95] Bo Yang and Ram Nevatia. An online learned CRF model for multi-target tracking. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2034–2041. IEEE, 2012.
- [96] Yaowen Guan, Xiaoou Chen, Deshun Yang, and Yuqian Wu. Multi-person tracking-by-detection with local particle filtering and global occlusion handling. In

- Multimedia and Expo (ICME), 2014 IEEE International Conference on*, pages 1–6. IEEE, 2014.
- [97] Caglayan Dicle, Octavia I Camps, and Mario Sznaiier. The way they move: Tracking multiple targets with similar appearance. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2304–2311, 2013.
  - [98] Bogdan Alexe, Viviana Petrescu, and Vittorio Ferrari. Exploiting spatial overlap to efficiently compute appearance distances between image windows. In *Advances in Neural Information Processing Systems*, pages 2735–2743, 2011.
  - [99] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer vision and image understanding*, 110(3):346–359, 2008.
  - [100] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.
  - [101] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. Learning by tracking: Siamese CNN for robust target association. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 33–40, 2016.
  - [102] Anton Andriyenko, Konrad Schindler, and Stefan Roth. Discrete-continuous optimization for multi-target tracking. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1926–1933. IEEE, 2012.
  - [103] Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32):2, 2016.
  - [104] M. Ren and R. S. Zemel. End-to-end instance segmentation with recurrent attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6656–6664, 2017.

- [105] Alireza Fathi, Zbigniew Wojna, Vivek Rathod, Peng Wang, Hyun Oh Song, Sergio Guadarrama, and Kevin P Murphy. Semantic instance segmentation via deep metric learning. *arXiv preprint arXiv:1703.10277*, 2017.
- [106] B. De Brabandere, D. Neven, and L. Van Gool. Semantic instance segmentation with a discriminative loss function. In *Deep Learning for Robotic Vision, workshop at CVPR 2017*, pages 1–2. CVPR, 2017.
- [107] S. Kong and C. Fowlkes. Recurrent pixel embedding for instance grouping. *arXiv preprint arXiv:1712.08273*, 2017.
- [108] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother. Augmented reality meets deep learning for car instance segmentation in urban scenes. In *British Machine Vision Conference (BMVC)*, 2017.
- [109] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [110] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [111] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei. Fully convolutional instance-aware semantic segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2359–2367, 2017.
- [112] B. Romera-Paredes and P. H. S. Torr. Recurrent instance segmentation. In *European Conference on Computer Vision*, pages 312–329. Springer, 2016.
- [113] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *CVPR*, 2016.

- [114] T. Simon, H. Joo, I. Matthews, and Y. Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017.
- [115] Jonathan Taylor, Lucas Bordeaux, Thomas Cashman, Bob Corish, Cem Keskin, Toby Sharp, Eduardo Soto, David Sweeney, Julien Valentin, Benjamin Luff, et al. Efficient and precise interactive hand tracking through joint, continuous optimization of pose and correspondences. *ACM Transactions on Graphics (TOG)*, 35(4):143, 2016.
- [116] C Loop. Smooth subdivision surfaces based on triangles, master’s thesis. *University of Utah, Department of Mathematics*, 1987.
- [117] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, and Kevin Murphy. Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. *arXiv preprint arXiv:1803.08225*, 2018.
- [118] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) - Volume 1 - Volume 01*, CVPR ’05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society.
- [119] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.
- [120] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.
- [121] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional

- nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.
- [122] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [123] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *Advances in Neural Information Processing Systems*, pages 658–666, 2016.
- [124] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *IEEE Int. Conf. Comput. Vision (ICCV)*, pages 5907–5915, 2017.
- [125] Xi Chen, Yan Duan, Rein Houthoofd, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.
- [126] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.
- [127] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.
- [128] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (5):603–619, 2002.

- [129] Ke Li and Jitendra Malik. Amodal instance segmentation. In *European Conference on Computer Vision*, pages 677–693. Springer, 2016.
- [130] Qin Huang, Chunyang Xia, Siyang Li, Ye Wang, Yuhang Song, and C-C Jay Kuo. Unsupervised clustering guided semantic segmentation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1489–1498. IEEE, 2018.
- [131] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.
- [132] Yan Zhu, Yuandong Tian, Dimitris Metaxas, and Piotr Dollár. Semantic amodal segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1464–1472, 2017.
- [133] Anton Milan, Stefan Roth, and Konrad Schindler. Continuous energy minimization for multitarget tracking. *IEEE transactions on pattern analysis and machine intelligence*, 36(1):58–72, 2014.
- [134] Y. Yang, S. Hallman, D. Ramanan, and C. Fowlkes. Layered object detection for multi-class segmentation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3113–3120, June 2010.
- [135] Y. Yang, S. Hallman, D. Ramanan, and C. C. Fowlkes. Layered object models for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1731–1743, Sept 2012.
- [136] Y. T. Chen, X. Liu, and M. H. Yang. Multi-instance object segmentation with occlusion handling. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3470–3478, June 2015.
- [137] J. Uhrig, M. Cordts, U. Franke, and T. Brox. Pixel-level encoding and depth layering for instance-level semantic labeling. In *German Conference on Pattern Recognition*, pages 14–25. Springer, 2016.

- [138] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.
- [139] Siyang Li, Bryan Seybold, Alexey Vorobyov, Alireza Fathi, Qin Huang, and C-C Jay Kuo. Instance embedding transfer to unsupervised video object segmentation. *arXiv preprint arXiv:1801.00908*, 2018.
- [140] NVIDIA Corporation. Nvidia cuda c programming guide, 2012. [Online; accessed 9-November-2018].
- [141] Mizell, E. Gpus: The key to cognitive computing, 2017. [Online; accessed 9-November-2018].
- [142] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [143] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [144] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
- [145] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
- [146] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

- [147] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.
- [148] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [149] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [150] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.
- [151] MathWorks. Pretrained deep neural networks, 2019. [Online; accessed 16-June-2019].
- [152] DP Kingma and J Ba. Dp kingma and j. ba, adam: A method for stochastic optimization, arxiv: 1412.6980. *Adam: A Method for Stochastic Optimization*.
- [153] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.
- [154] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 1538–1546. IEEE, 2015.

- [155] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
- [156] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.
- [157] Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Pablo Arbeláez, and Luc Van Gool. Convolutional oriented boundaries: From image segmentation to high-level tasks. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):819–833, 2018.
- [158] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.
- [159] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [160] Eric T Psota, Mateusz Mittek, Lance C Pérez, Ty Schmidt, and Benny Mote. Multi-pig part detection and association with a fully-convolutional network. *Sensors*, 19(4):852, 2019.
- [161] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1812.08008*, 2018.
- [162] Shuran Song and Jianxiong Xiao. Tracking revisited using rgbd camera: Unified benchmark and baselines. In *Proceedings of the IEEE international conference on computer vision*, pages 233–240, 2013.
- [163] Phillip A Laplante. *Electrical engineering dictionary*. CRC Press LLC, 2000.

- [164] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.
- [165] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.
- [166] Wikipedia contributors. Hsl and hsv — Wikipedia, the free encyclopedia, 2018. [Online; accessed 18-July-2018].
- [167] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.
- [168] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M Pérez, and Iñigo Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256, 2013.
- [169] Renato Cordeiro de Amorim and Christian Hennig. Recovering the number of clusters in data sets with noise features using feature rescaling factors. *Information Sciences*, 324:126–145, 2015.
- [170] sklearn.metrics.silhouette\_score. [Online, accessed 11-August-2019].
- [171] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [172] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- [173] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

- [174] sklearn.cluster.kmeans. [Online, accessed 5-December-2019].
- [175] Kuntal Chowdhury, Debasis Chaudhuri, Arup Kumar Pal, and Ashok Samal. Seed selection algorithm through k-means on optimal number of clusters. *Multimedia Tools and Applications*, pages 1–35, 2019.
- [176] Peter J Rousseeuw and L Kaufman. Finding groups in data. *Hoboken: Wiley Online Library*, 1990.
- [177] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pages 3686–3693, 2014.
- [178] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

## A.1 Image blending using selective reconstruction of Laplacian Pyramids

Given an image with dimensions  $w, h$  (being the width and height of the image respectively), represented by a  $h \times w$  matrix  $I$ , the Laplacian pyramid of this image at level  $n$   $\Delta_I^n$  is a collection of:

$$\Delta_I^n = [L_I^n, H_I^n, H_I^{n-1}, \dots, H_I^0], \quad (1)$$

where  $L_I^n, H_I^n$  are matrices representing respectively the low ( $\frac{h}{2^n} \times \frac{w}{2^n}$ ) and high ( $\frac{h}{2^{n-1}} \times \frac{w}{2^{n-1}}$ ) frequency components obtained on  $n - th$  level of decomposition such that:

$$L(I) = conv(I, K), \quad (2)$$

$conv(I, K)$  is the convolution operation over image  $I$  using Gaussian kernel  $K$ . In other words,  $L$

$$H(I) = I - conv^{-1}(L_I(I), K * 4), \quad (3)$$

where  $conv^{-1}$  is the deconvolution operation (also referred to as a convolution with fractional stride), and  $K$  is the Gaussian kernel. Construction of the pyramid can be described like in algorithm 4.

Once the Laplacian pyramid  $\Delta_I^n$  is obtained for an image  $I$ , the image can be fully reconstructed with the following operation:

Now, given two Laplacian representations  $\Delta_A^n, \Delta_B^n$ , one of image  $A$  and another of image  $B$  and a binary mask  $M$  indicating which parts of image  $A$  should be replaced by content of image  $B$ , one could formulate a selective recomposition algorithm using a modified version of algorithm 5 with the goal of obtaining smooth transition between the

**Algorithm 4** Construct Laplacian pyramid  $\Delta_I^n$  for image  $I$  given  $n$ . Note operations:  $L.append(e)$  adds element  $e$  to the list  $L$  in-place,  $L.reverse()$  reverses the order of elements in the list in-place.

---

```

1: procedure LAPCONSTRUCT( $I, n$ )
2:   pyramid = []
3:    $T = I$ 
4:   for  $i = [0, \dots, n - 1]$  do
5:      $L_I^i = conv(T, K)$ 
6:      $H_I^i = I - conv^{-1}(L_I^i, K)$ 
7:      $T = L_I^n$ 
8:     pyramid.append( $H_I^i$ )
9:   pyramid.append( $L_I^{n-1}$ )
10:  return pyramid.reverse()

```

---

**Algorithm 5** Reconstruct original image  $I$  given its Laplacian pyramid  $\Delta_I^n$

---

```

1: procedure LAPRECONSTRUCT( $\Delta_I^n, n$ )
2:    $I = \Delta_I^n[0] = L_I^n$ 
3:   for  $i = [1, 2, \dots, n - 1]$  do
4:      $I = conv^{-1}(I, K) + H_I^{n-i} = conv^{-1}(I, K) + \Delta_I^n[i]$ 
5:   return  $I$ 

```

---

images yet preserving the high frequency content of both. Such method is presented in algorithm 6.

**Algorithm 6** Create a composition of two images using their Laplacian pyramids  $\Delta_A^n, \Delta_B^n$  and a binary mask  $M$

---

```

1: procedure LAPCOMPOSE( $\Delta_A^n, \Delta_B^n, n, M$ )
2:    $f = 2^{n-1}$ 
3:    $I_A = \Delta_A^n[0] = L_A^n$ 
4:    $I_B = \Delta_B^n[0] = L_B^n$ 
5:    $m = downSampleByFactor(M, f)$ 
6:    $I = (1 - m) \odot I_A + m \odot I_B$ 
7:   for  $i = [1, 2, \dots, n - 1]$  do
8:      $f = 2^{n-i-1}$ 
9:      $m = downSampleByFactor(M, f)$ 
10:     $I_A = \Delta_A^n[i] = H_A^{n-i}$ 
11:     $I_B = \Delta_B^n[i] = H_B^{n-i}$ 
12:     $H = (1 - m) \odot I_A + m \odot I_B$ 
13:     $I = conv^{-1}(I, K) + H$ 
14:  return  $I$ 

```

---

Observing the results presented in figure A.1 one could ask if this method is any

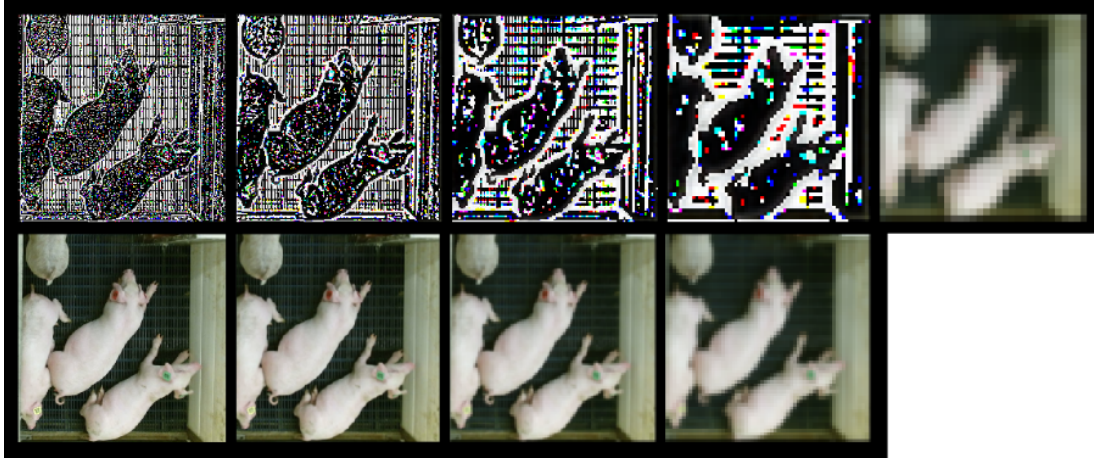


Figure A.1: Example of Laplacian decomposition (top part from left to right) and recombination (bottom from right to left) using 4-level pyramid. Sample image taken from the 2017-04-19-21-30-25 subset of image-depth datasets listed in table 3.2.

more convincing than simply normalizing the smoothed mask  $M$  and then composing using the copy&paste method. Please note that the The mask is purposefully containing floor areas to illustrate difficulties associated with the blending task. A comparison of three basic schemes is presented in figure A.1. It is visible, that simple copy and paste operation is very sensitive to the *tightness* of the mask around the object and thus, most likely creates artifacts. Blurring the mask does provide some level of smooth transitioning but it is not dependent on the image content and does not favor detail preservation. Laplacian-based blending yields most visually convincing results as there is no *ghosting* present in the picture.

It is worth noting that when looking at the *problematic areas* (indicated by red circles) of the composition presented in figure A.1, some artifacts still exist when using the Laplacian method (c) - especially the features of the floor as they do contain high-frequency content. The mask-dependency was however resolved in most visually pleasing fashion among all three presented techniques.

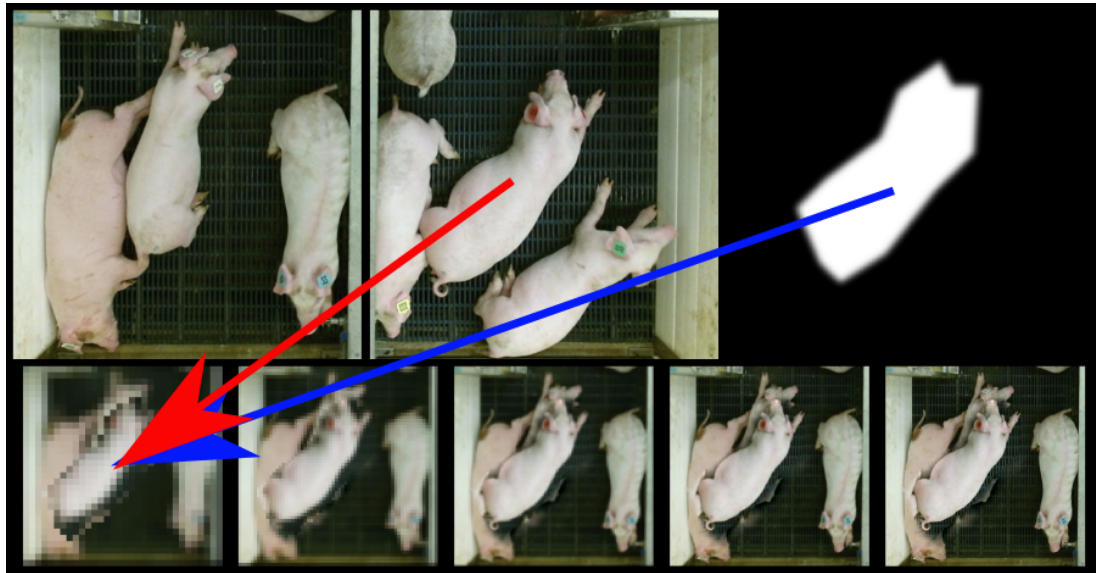


Figure A.2: Example of Laplacian composition using algorithm 6 with pyramids represented like in algorithm 4. Instance indicated by red arrow is introduced to the original image (top-left) using a mask indicated by blue arrow. Arguments are presented on the top (left to right):  $A$ ,  $B$ ,  $M$ , and  $n = 4$ , and results at each level  $i = [4, 3, 2, 1, 0]$  are presented in the bottom part of figure.

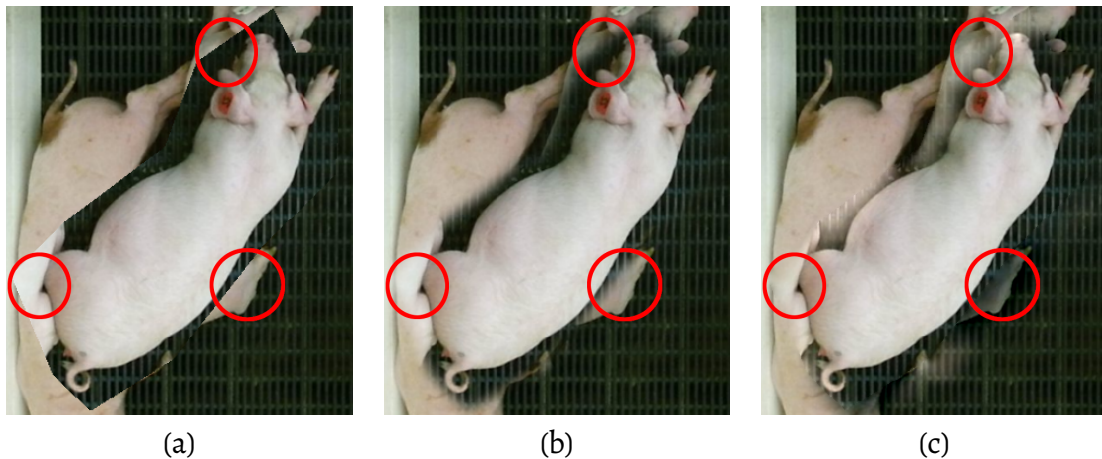


Figure A.3: Focused visual comparison of the effect of three considered composition operations: naive copy and paste (a), mask blurring (b), and using Laplacian pyramid (c).