Final Bachelor's Thesis

Consistency on the Vertices, Edges and Mask: A Semi-Supervised Learning Approach for Image Segmentation

Degree in Mathematics & Degree in Engineering Physics

Author:
Jordi Fortuny Profitós

Supervisor:
Sanja Fidler
Local Tutor:
Xavier Giró i Nieto

April 2019

Jordi Fortuny Profitós: *Consistency on the Vertices, Edges and Mask: A Semi-Supervised Learning Approach for Image Segmentation* © May 2019

A Bachelor's Final Thesis Submitted to the Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona (ETSETB), Facultat de Matemàtiques i Estadística (FME) and Centre de Formació Interdisciplinària Superior (CFIS) of the Universitat Politècnica de Catalunya (UPC) in partial fulfilment of the requirements for the degrees of DOBLE TITULACIÓ EN EL GRAU DE MATEMÀTIQUES I GRAU D'ENGINYERIA FÍSICA.

Thesis done at the Vector Institute, University of Toronto, under the supervision of Prof. Sanja Fidler.

SUPERVISOR:
Sanja Fidler

HOME SUPERVISOR:
Xavier Giró i Nieto

LOCATION:
Toronto, ON, Canada

Computer Science is no more about computers than astronomy is about telescopes.

— Edsger Dijkstra

Abstract

In this thesis, we present a novel method for performing image segmentation in a semi-supervised approach, which we consider to be particularly relevant because of the substantial cost of obtaining pixel-wise annotations required to train supervised. This method, that we will call Consistency on the Vertices, Edges and Mask, is one of the first methods that can be used for training deep neural networks to perform image segmentation in a semi-supervised setting where only a small portion of training data is labeled. In our setting, we train a network to predict masks, edges and vertices for a given input image, and then we penalize the network for not being consistent with its predictions obtaining the theoretical edges an vertices from the predicted mask using a derivable version of the Canny edge detector. We also present results on the Cityscapes Dataset where we obtain outstanding results achieving about 92% of the fully supervised performance labeling only 10% of the data and 98% labeling 25%.

This thesis also contains a brief introduction on the field of deep learning and semi-supervised learning, relevant previous work that has been published in the last year which inspired this work and finally, information on the architecture of our model and its performance as well as some experiments, results and ablation studies.

KeyWords: Computer Vision • Machine Learning • Deep Learning • Semi-Supervised Learning • Image Segmentation • Cityscapes Dataset • Consistency • Canny Edge Detector.

Acknowledgments

I would like to express my very great appreciation to Professor Sanja Fidler, my research supervisor, for all her valuable and constructive suggestions during the planning and development of this research work. I would also like to thank Dr. David Acuna and Kevin Shen, for their advice and assistance in keeping my progress on schedule. Their willingness to give their time so generously has been very much appreciated.

I would also like to thank the staff of the Vector Institute for enabling me to use their offices and their computing resources, allowing me to participate in talks, events and learning sessions as well as assisting me for any trouble inside or outside the office.

I also wish to thank various people for their indirect contribution to this project, Prof. Miguel Ángel Barja, Prof. Toni Pascual, Prof. Xavier Giró and Prof. Eduardo Alarcón, allowing me the opportunity to spend time conducting research at the Vector Institute under one of the excellence mobility CFIS grants.

I would like to thank all the colleagues that I've worked with during my research internship, specially Eric Guisado, Rafel Palliser and Dídac Surís, with whom I have shared moments of deep anxiety but also of big excitement. Their presence was very important in a process that sometimes can be felt as tremendously solitaire.

I would also like to extend my thanks to my life partner, Ariadna, without whom it would have been impossible to carry on this project. She has been the light of my life for the last two years and she has given me the strength and motivation to get things done. This paper is dedicated to her.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Inside the field of computer science, there is a research area that lacks of a precise definition but curiously, that is what has it one of the most promising fields of study in the recent years. Researchers and developers of Artificial Intelligence (AI) are guided by a rough sense of direction and an imperative to “get on with it.” However, on 2010 Nils J. Nilsson [20] provided a useful definition that is still used nowadays: ”Artificial intelligence is that activity devoted to making machines intelligent, and intelligence is that quality that enables an entity to function appropriately and with foresight in its environment.”

In fact, sometimes the field of AI is called machine intelligence since its goal can be understood as creating ”intelligent agents” that can perceive their environment and take actions in order to achieve specific goals and tasks through flexible adaptation. Pamela McCorduck [18] explains that AI suffers from a repeating pattern, the “AI effect” —AI brings a new technology into the common fold, people become accustomed to this technology, it stops being considered AI, and newer technology emerges. Nevertheless, all that technology is always focused on mimicking cognitive functions that are associated with human minds, such as pattern recognition, learning, image classification, problem solving...

One of the newest areas inside Artificial Intelligence is Deep Learning (DL) which at the same time is also part of Machine Learning. DL is the field that learns data representations through basically deep neural networks and recurrent neural networks. Deep learning can be used in a lot of different computer science environments, but it is typically associated with image recognition and computer vision (although one can find examples in the literature [3] where has been used for medical image analysis, natural language processing, speech recognition, machine translation, drug design and board game programs such as chess or go, where they have produced results superior to human experts and any other ”non-intelligent” existing machine).

I personally think that AI is where every computer scientist knows that they can find the solution they need or where they should research for developing that solution, and hence has awaken interest of mathematicians, engineers and even physicists or doctors. As a mathematician, I love the statistics of the models, how you can translate what you code to mathematical models and the ”good praxis” to first formulate how the maths of the model are gonna behave. On the other hand, as an engineer, I love developing the models and fight to solve all the problems that the coding part of the project proposes.

This thesis contains a summary of all the research and work conducted under the supervision of Prof. Sanja Fidler during my internship at the Vector Institute in Toronto. The main focus of the work has been Semi-Supervised Deep Learning and the following chapters contain a brief introduction of the field to the reader, some of the relevant related work that has been done in the previous years as well as my personal contribution to the field with results and my personal thoughts and conclusions about the internship.

1.2 Objectives

In the field of computer science, and more concretely in the field of deep learning, one of the objectives of the researchers worldwide is to extract as many information as possible from a given dataset. This objective can also be thought as understanding how big should the datasets be, or how many data points should be labeled in order to obtain good performance. In a field where bigger is not necessarily better due to the acquiring, storage and labelling costs, it is essential to solve this questions

A lot of research has been conducted in the previous years towards the goal of reducing the dataset size [21], choosing more accurately what images should be labeled [26] or even how a smaller network can distill all the information of a big dataset [31]. There is also a branch of research that focuses on how is it possible to complement a labeled dataset with non-labeled images that are from the same domain. This is Semi-Supervised Learning and the recent literature has proven it can be really useful on image classification [16, 22, 29].

The goal of this thesis and the objective of my internship were finding a way to use unlabeled data to improve the performance on image segmentation (i.e. presenting Semi-Supervised Learning for image segmentation). I consider this to be a very relevant topic nowadays since we are living in the big data era where large companies are hunting out there for billions and billions of data. It would be really nice for all those companies, but also for research purposes, to be able to achieve the same performance with only 10% of the available data. First step was being able to obtain impressive results in image classification and the next natural step would be learning how to segment semi-supervising.

In order to achieve this big objective, other intermediate goals have arisen such as understanding the current state-of-the art results, understanding the recent literature and learning the programming skills needed. In fact, at the beginning of this internship, my supervisor advised me to read a lot and she assigned me tasks such as coding in Pytorch [23] (which to my understanding is the best library in python for Deep Learning) or summarizing some of the most recent techniques from relevant papers which form a future point of view it is something that I am completely grateful to have done.

Another important goal that was set at the begging of this internship was being able to come back to Barcelona with more knowledge than when I left. Learning has always been one of my passions and I feel specially happy with the opportunities that this project has given me for doing such thing. For being a researcher at the Vector Institute, I've been able to assist to at least 10 research talks, 4 round-tables, 4 group meeting with the supervisor's team, a dozen student discussion meetings, a Fields Institute talk and Vector's insight meetings on a monthly basis, and hence improving as a scientist and as a person.

Finally, the last objective that I considered was growing as a human being. It is a goal that I have always had in mind since I have conscience of my decisions and I think it is really important to make an effort to understand people with different backgrounds, religions and cultures from the ones we are used in our country. I think that visiting Toronto, the most multicultural city in the world, and the Vector Institute were some of the best researchers worldwide spend 10 hours on a daily basis, has allowed me to learn a lot, understand different points of view and maintain a lot of interesting conversations.

2. Deep Learning and Semi-Supervised Learning

On May 2015, Nature published a short review titled "Deep Learning" [3]. On the abstract, LeCun, Bengio and Hinton summarized how deep learning methods improved at that time the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Moreover, they highlight the multiple levels of abstraction that such methods "learn" using the back-propagation algorithm to update its internal parameters.

A quick search into any search engine would lead one to some definitions explaining that DL is a function that imitates the workings of the human brain trying to process data and create patterns to make the right decisions, or that DL is a subset of machine learning and AI that consists in building networks capable of learning from data that is unstructured or unlabeled, or even one can find a magnification of the field as stated by Andrew Ng in its own web-page www.deeplearning.ai: "Deep Learning is a superpower. With it you can make a computer see, synthesize novel art, translate languages, render a medical diagnosis, or build pieces of a car that can drive itself. If that isn't a superpower, I don't know what is."

But, what are the pieces that build together that concept? What are the basics that one must know to understand DL? I would say that there are two main networks that are used in the 95% of Deep Learning Applications, Convolutional Neural Networks and Recurrent Neural Networks. Each one has its own particularities and ingredients, and while the first is mainly used for of those visual object recognition and object detection, the later is more effective for visual question answering or machine translation. On this paper the focus lies on the first one because recurrent neural networks are not used in this project.

Semi-supervised Learning is a class of Machine Learning and Deep Learning that uses both labeled data and unlabeled data. In that sense, it is able to train both using supervision of the tasks through the labeled data where ground truth is available and unsupervised on the unlabeled data (i.e. it is able to use the unlabeled data to train and improve although it doesn't has labels to understand it).

This section explains the basics of the methods used in the literature to provide context and knowledge to the reader about the different networks, structures and techniques used in the project. It first focus on Deep Learning and shows and example of one very important network used in this project and there is also presented a brief introduction to semi supervised learning.

2.1 Deep Learning

Deep Learning is such an extensive field that one could write an entire book about it, and I would like to address to the reader with some of my suggestions of really good deep learning books that they could find in the bibliography: [8, 11, 25]. Specially, I would recommend ”Deep Learning for Computer Vision with Python” [25], the book that Adrian Rosebrock, phD and entrepreneur published a couple of years ago. It has received lots of good reviews from other successful authors such as Francois Chollet, AI researcher at Google and creator of Keras, a python library, that said:

> This book is a great, in-depth dive into practical deep learning for computer vision. I found it to be an approachable and enjoyable read: explanations are clear and highly detailed. You'll find many practical tips and recommendations that are rarely included in other books or in university courses. I highly recommend it, both to practitioners and beginners. — Francois Chollet

I personally think that this book is useful to understand the basics of deep learning and also the techniques that are used in object detection and segmentation while at the same time it provides contrasted theory and Python implementation.

Having said that, in this section I would like to explain some of the basics of deep learning that are most used in this project. I refer the reader to the bibliography to extend their knowledge in Deep Learning.

2.1.1 Neural Networks

Artificial neural networks (ANN) are machine learning frameworks that were inspired by the biological neural networks that constitute animal brains [30]. The goal of this frameworks is being able to ”learn” to process complex data inputs and perform tasks without any given a priory rule, just only considering the examples given during training, as a human being would do.

An ANN [2] is formed by a collection of connected units or nodes called artificial neurons because they model model the neurons in a biological brain. The neurons are connected like the synapses in a biological brain and can transmit a signal from one to another. In common ANN implementations, all the input signals are real numbers and every neuron outputs another real number obtained by computing a non-linear function of the sum of all the inputs averaged by some weights and bias. Those weights and bias are the learnable parameters of the ANN and an optimization problem will be solved in order to obtain a local minima (or global ideally) according to some loss function that could generalize to any input.

Normally, the neurons are aggregated into layers and the synapse function may vary from one layer to another. In an ANN, the signals travel from the first (or input) layer until the last (or output) layer through all different neuron layers and at the end, a loss function is computed. Normally this loss function will be differentiable with respect to the network weights and then all those parameters are optimized via a backpropagation algorithm.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are Neural Networks that are made up of neuron layers that have convolutional kernels with learnable weights and biases. Each of the neurons receives one or some inputs and performs some operations that can be expressed as dot products

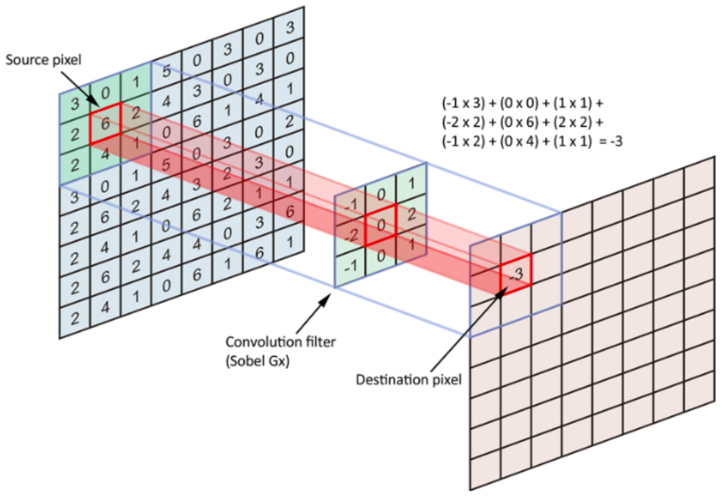

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 2.1: Example of a convolution with a convolutional filter of dimensions $(3, 3, 1)$ and a input tensor of size $(8, 8, 1)$.

and additions and then usually there is a non-linearity. This operations make the network differentiable with respect to its weights. In the following, there is presented a list of the 5 layers that are present in every architecture of a CNN, with the operations and details that are used.

- **Input** layer: The input layer is just the first input that recieves the model. It typically is an image which would have size $(h, w, 3)$ where h and w are the number of heigh and width pixels of the image and 3 represent the RGB channels. In Deep Learning, it is very typical to work with batches of data, so the input layer will be a tensor $(b, h, w, 3)$ where b just indicates how many images fit in the memory of the GPU and are processed at the same time by the computer.

- **Convolutional** layers: This layers are the ones that give the name to the entire network and it is not in vain because since their discovery, when the input is and image they are always present. Let's consider that the input of one of those layers is a tensor of size (h, w, c_{in}). Then this layers have c_{out} kernels of size (h_k, w_k, c_{in}) of learnable weights that will be convolving on top of the input tensor (i.e. computing a local dot product) to produce an output of size $(h_{out}, w_{out}, c_{out})$ where h_{out} and w_{out} are determined by the padding and the stride factors. Figure 2.1 shows how the dot product in a convolution layer are performed. Padding (or more typically zero-padding) is a factor that determines how many extra rows and columns one will add on the sides of the image. It is a common strategy to preserve size. On the other side, to downsample the spatial dimensions h and w one could use convolutional layers with stride. The stride factor indicates how many pixels the filter is moving at a time. For a 3x3 convolution as in figure 2.1, the standard stride factor is 1 and the padding factor is also 1 resulting an output with the same spatial dimension as the input.

- **Pooling** layers: The pooling layers perform a downsampling operation along the spatial dimensions: width and height. In order to do so, the pooling operation converts regions of the image of squared size (typically 2 x 2) into one value, hence reducing the spatial dimensions by 2 each (or the volueme by 4). There are basically two different types of pooling: Average pooling and max pooling. The first one returns the average of the neighbours while the later just returns the maximum of them.

- **Fully Connected** layers: This layers connect all the input neurons to all the output neurons. They are typically used as last layers in a classification problem because they can connect through weights and bias all the input features into output probabilities of each class. Figure 2.2 shows an example of two fully connected layers in a configuration with 3 neurons in the input layer, 4 in the hidden layer and 2 classes as the output.

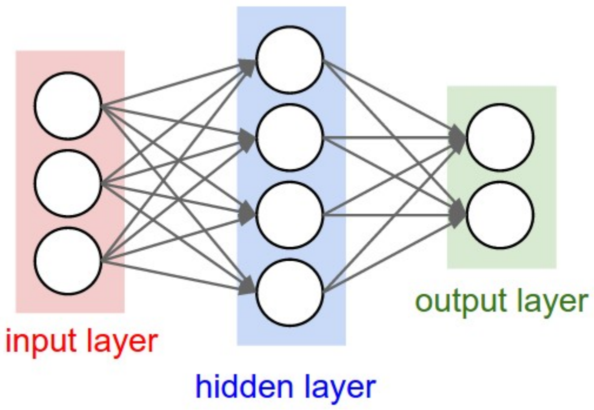


Figure 2.2: Example of a toy neural network with two fully connected layers.

- **Non-linear** layers: Non linear layers or also typically called Rectifier Linear Units (ReLU, Leaky ReLU) are layers that preserve the spatial dimensions and are defined as the positive part of its argument with maybe some alteration:

$$ReLU(x) = x^+ = max(0, x)$$

$$LeakyReLU(x) = \begin{cases} x^+ = max(0, x), & \text{if } x > 0 \\ -\alpha \cdot x, & \text{if } x < 0 \text{ where usually } \alpha = 0.01 \end{cases}$$

An example of how one could use a CCN to classify images according to their content would be the following [17]:

Consider images from the CIFAR-10 dataset [15], those images have shape $(32, 32, 3)$ pixels and contain images within 10 different classes: cat, dog, bird, horse, frog, deer, automobile, train, ship and airplane.

1. The **input** layer will be a batch of size $(b, 32, 32, 3)$
2. The first layer we could use is **64 3x3 convolutional** filters with one pixel of zero-padding and stride 1.
3. Then the resulting dimensions will be $(b, 32, 32, 64)$ and after applying a **ReLU** layer and a **2x2 MaxPooling** we will obtain a tensor of dimensions $(b, 16, 16, 64)$ with all positive values.
4. Next, use again **64 3x3 convolutional** filters with one pixel of zero-padding and stride 1 followed by a **ReLU** layer and a **2x2 MaxPooling**. This results into a tensor of dimensions $(b, 8, 8, 64)$
5. Follow up with **128 3x3 convolutional** filters with stride 2 resulting into a $(b, 4, 4, 128)$ tensor.
6. Finally, end with a **Leaky ReLU**, another **2x2 MaxPooling** with $\alpha = 0.01$ and a **fully connected layer** to compute the class scores, resulting in volume of size $(b, 1x1x10)$, where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10.

As with ordinary Neural Networks and as the name implies, each neuron in the fully connected layer will be connected to all the numbers in the previous volume, so in order to reduce dimensionality of the parameter space, it is important to use the convolutional filters combined with the poolings. As part of my deep learning training, I designed this network and it obtained around a 85% accuracy on the CIFAR-10 dataset [15]. Code can be found on Appendix 1.

2.2 ResNet

As deep learning architectures go deeper with convolutional layers, they are more difficult to train because the number of parameters is bigger and more importantly because backpropagation has to travel through more layers. On 2015, Microsoft's Deep Learning researchers He, Zhang, Ren and Sun [13] presented on the conference about Computer Vision and Pattern Recognition (CVPR) a model that has a deep architecture but was more easy to train than before: ResNet.

ResNet is the abbreviation of Residual Network and this name was given because the solution of the above mentioned problem lies in building a new architecture with residual blocks (see Figure 2.3). Each block consists on two weight convolutional layers as well as batch normalization and a non linearity (such as ReLU or leaky ReLU) that are applied to an input x and then added to the same input x before going into the next block. In that sense, the network is not predicting how to change the input to match the desired output but only the residual transformations that should be applied to the input to make a better version of it.

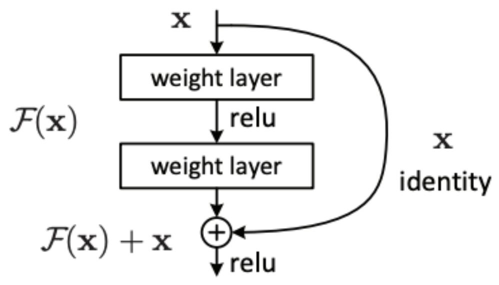

x
weight layer
relu
$\mathcal{F}(\mathbf{x})$
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.3: Representation of a building block of a residual learning architecture. Figure originally published in [13].

On figure 2.4 one can see schematically the difference between some of the architectures that were used as a baseline in [13] (left and middle) as well as a 34-layer ResNet (right). One can see that the main change is the fact that there are residual connections every two layers in the residual network (i.e. it is build with residual building blocks). Note that when a connection is represented with a dotted line means that there has been a down-sampling operation applied to the input data.

That trick, which is relatively easy to understand, broke all the state-of-the-art performances on image classification and object location and ResNet is still one of the most used architectures in deep learning. Particularly, in this project, the backbone architecture is a pretrained ResNet which is used to obtain reliable features of the input images.

2.3 Semi Supervised Learning

When working under a semi-supervised regime it is typical to assume at least one of the following [6]:

- **Continuity**: Data points which are close in the feature space, are more likely to share a label. This assumption is also assumed in almost all supervised vision problems when the objective is to find a good feature space representation through which the decision boundaries should be easy to find.

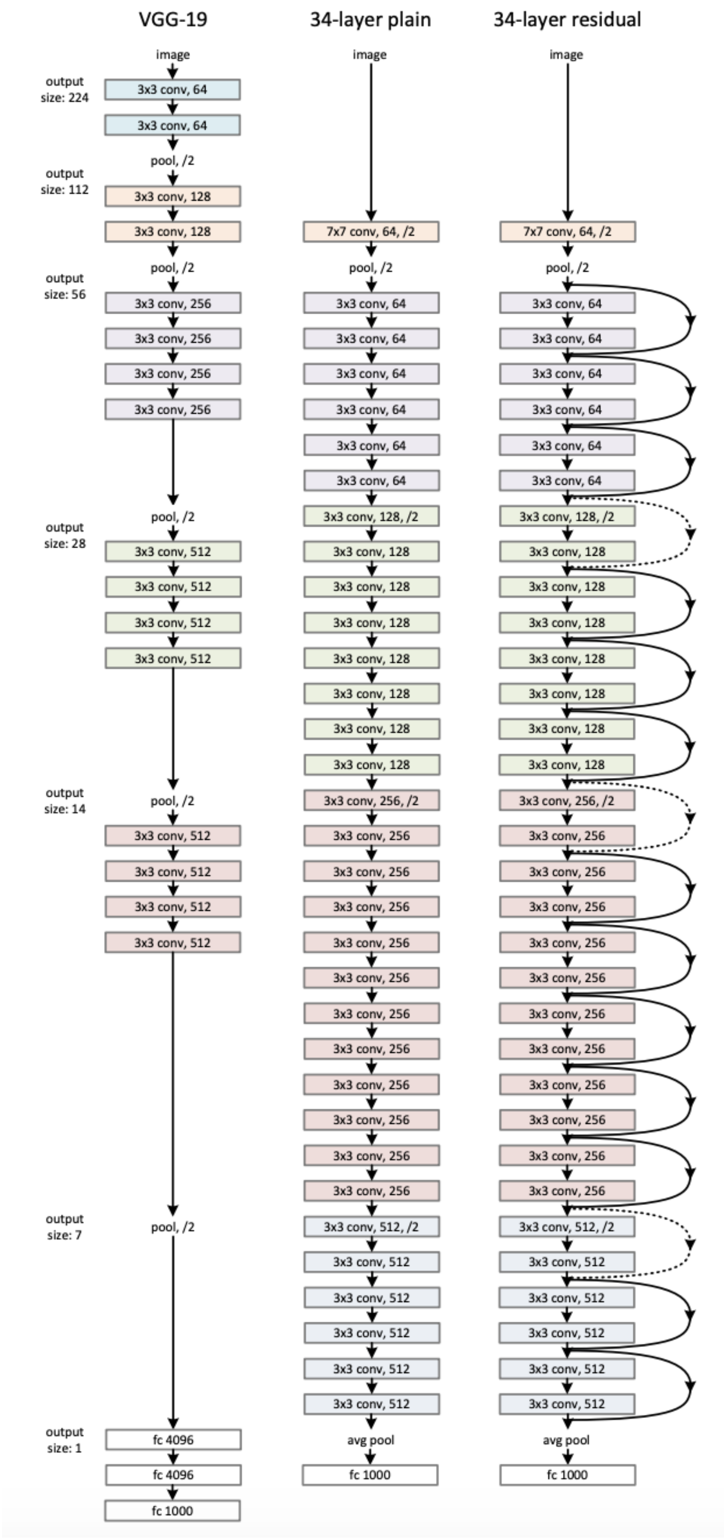


Figure 2.4: Comparison between deep architectures. **Left:** VGG [27], the deep model presented by Google, **Middle:** A deep architecture with 34 layers, **Left:** A deep residual architecture of also 34 layers. One can see how the residual one is completely build through residual building blocks. Figure originally published in [13].

- **Data Clustering**: The data points are grouped into discrete clusters, and points in the same cluster are more likely to share a label. This doesn't necessarily mean that there are as many clusters than labels, it can be more clusters than labels, meaning that more than one cluster will have the same label (distributed labelling) or even more labels than clusters, meaning that some clusters will have data from different labels (close labelling).
- **Manifold approximation**: The data points can be approximately found on a manifold of much lower dimension than the input space. Then the objective will be to learn the manifold and work on the manifold space where less parameters are needed.

As one can see, all those assumptions make it easy to exploit the use of unlabeled data. In the first scenario, one will penalize if the model predicts different labels for nearer points, on the second one, the penalization would be imposed within clusters and the last scenario allows to penalize the model if it represents the data in many dimensions.

In image classification, the continuity and clustering assumptions are the most used in the recent literature [16, 22, 29]. More emphasis on this point will be made in the section 3 where there are presented some papers that are state-of-the-art on Semi Supervised image classification and that have been used as an inspiration of the project.

3. Related Work

This chapter contains an introduction to some of the related work and papers that have inspired this project. The goal of this chapter is to familiarize the reader with the notation and concepts that are used in the semi-supervised learning framework as well as present the papers that improved my knowledge and motivated all my ideas.

3.1 Related Work in Semi Supervised Learning

In the following, there are presented some approaches of classical semi supervised learning proposed in [16] and [29] described in the context of traditional image classification networks: Let D be the training data consisting of N inputs, M of which are labeled. Let x_i where $i \in \{1...N\}$ be the inputs and let the set L contain the labels y_i where $i \in L$. For every i in L there is a correct label $y_i \in \{1...C\}$ where C is the number of classes. Let ω_t be weighting ramp factor that scales from 0 to w_{max}.

These two papers have served as inspiration for the consistency technique used in our semi-supervised approach. Personally, I find them surprisingly interesting and simple at the same time and I think that is precisely this simplicity that makes them relevant. A simple technique that thinking a bit out of the box can be used in another setting under different conditions.

The first paper [16] presents a simple and efficient method for training deep neural networks in a semi-supervised setting. They introduce self-ensembling, where they form a consensus prediction (i.e. averaging label predictions) of the unknown labels using the outputs of the network-in-training on different epochs and under different regularization and input augmentation conditions. This ensemble prediction can be expected to be a better predictor for the unknown labels than the output of the network at the most recent training epoch, and can thus be used as a target for training. Using our method, they set new records for some standard semi-supervised learning benchmarks and they demonstrate good tolerance to incorrect labels.

On the other hand, on the Mean Teacher model [29], the authors suggest that the Temporal Ensembling [16] method fails for large datsets and they propose a method that overcomes this situation. Mean Teacher is a method that averages model weights instead of label predictions. As an additional benefit, Mean Teacher improves test accuracy and enables training with fewer labels than Temporal Ensembling. Without changing the network architecture, Mean Teacher achieves better results than Temporal Ensembling on all the benchmarks.

3.1.1 Π-model

The Π-model relies on evaluating twice the inputs x_i through an stochastic network, (i.e. containing Dropout or applying data augmentation twice) resulting in two different predictions z_i and $\widetilde{z_i}$. In training mode there are two different loss terms. The first one is the typical cross-entropy loss which can only be evaluated for the labeled data. On the other hand, the second component is computed taking the mean square difference between the two predictions z_i and $\widetilde{z}_i$. Then the total loss (Equation 3.1) is the sum of the supervised and the unsupervised terms weighted by ω_t. Hence, in the beginning the total loss and the learning gradients are dominated by the supervised loss component, i.e., the labeled data only and due to a slow ramp-up weighting function the network learns slowly the consistency terms and does not fall into a degenerate solution with no meaning for classification.

$$L = L_{ce} + \omega_t L_{cons} = -\frac{1}{L}\sum_{i\in L}\log z_i[y_i] + \frac{\omega_t}{N}\sum_{i=1}^{N} z_i^2 - \widetilde{z}_i^{\,2} \tag{3.1}$$

3.1.2 Temporal Ensembling model

The Temporal ensembling model is also a consistency semi-supervised learning approach that instead of evaluating twice the inputs through the network just uses the network once and keeps in memory aggregated predictions of multiple previous network evaluations as an ensemble prediction. It maintains an exponential moving average of label predictions on each training example, and penalizes predictions that are inconsistent with this target. Hence, it speeds up the training and it alleviates the noisy targets that are obtained from the Π-model based on a single evaluation of the network. The formal structure of the model is the following:

Each input x_i is evaluated once per epoch, and a prediction z_i is obtained that it is used both for the typical cross entropy loss term and also a consistency loss term between z_i and the target vectors $\widetilde{z}_i$. In this model, the target vectors are obtained from the ensemble prediction Z_i. Equation 3.2 show the loss and the definitions for the ensemble prediction and target vectors used on this approach. Note that α is a momentum hiperparameter that controls then memory of the ensemble prediction. Figure 1 shows an original scheme published by the authors in [16].

$$\widetilde{z}_i^{\,t} = \frac{Z_i^t}{1-\alpha} \tag{3.2a}$$

$$L = L_{ce} + \omega_t L_{cons} = -\frac{1}{L}\sum_{i\in L}\log z_i[y_i] + \frac{\omega_t}{N}\sum_{i=1}^{i=N} z_i^2 - \widetilde{z}_i^{\,t\,2} \tag{3.2b}$$

$$Z_i^{t+1} = \alpha Z_i^t + (1-\alpha) z_i^t \tag{3.2c}$$

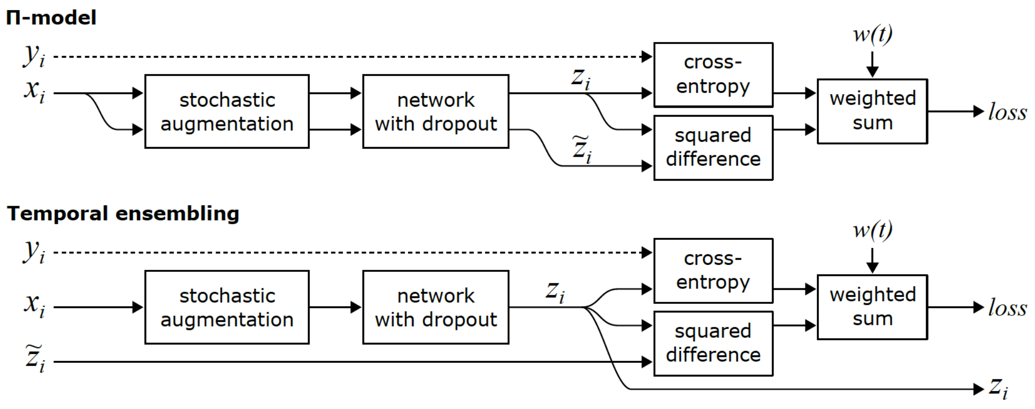
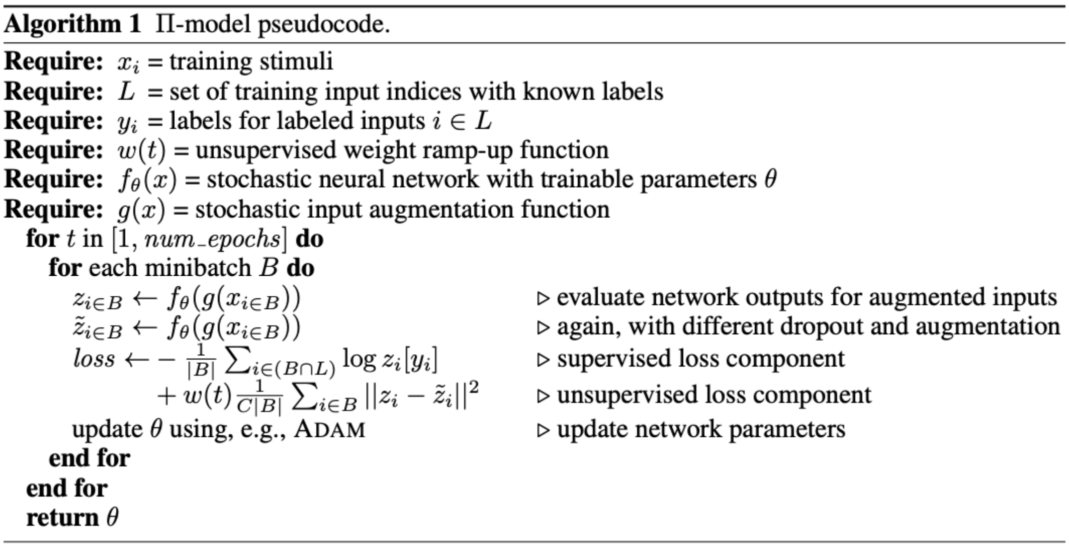
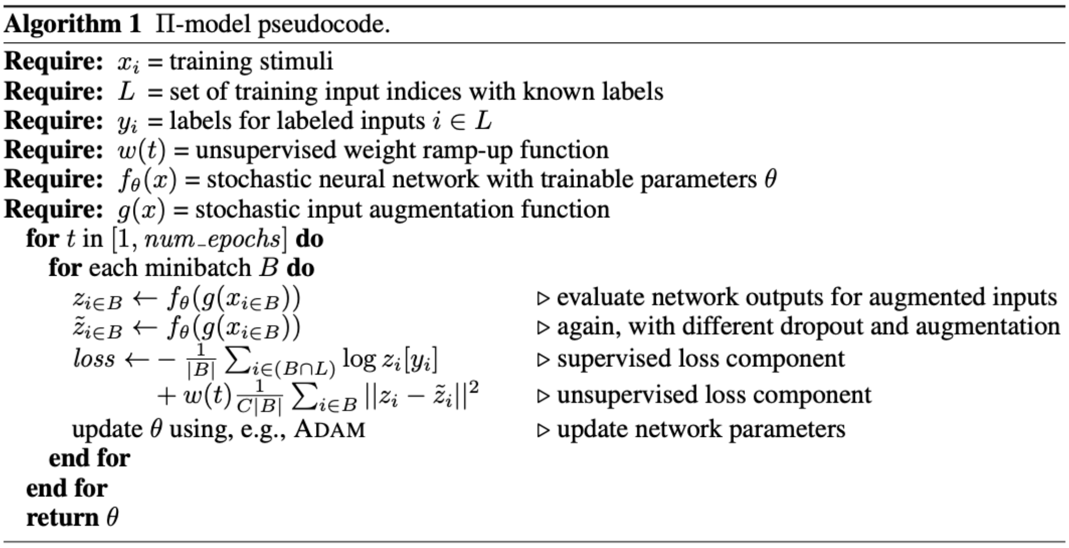


Figure 3.1: Scheme of the pi model and the temporal ensembling model. Figure originally published in [16].



```
Algorithm 1 Π-model pseudocode.
Require: $x_i$ = training stimuli
Require: $L$ = set of training input indices with known labels
Require: $y_i$ = labels for labeled inputs $i \in L$
Require: $w(t)$ = unsupervised weight ramp-up function
Require: $f_\theta(x)$ = stochastic neural network with trainable parameters $\theta$
Require: $g(x)$ = stochastic input augmentation function
  for $t$ in $[1, num\_epochs]$ do
    for each minibatch $B$ do
      $z_{i \in B} \leftarrow f_\theta(g(x_{i \in B}))$              ▷ evaluate network outputs for augmented inputs
      $\tilde{z}_{i \in B} \leftarrow f_\theta(g(x_{i \in B}))$      ▷ again, with different dropout and augmentation
      $loss \leftarrow -\frac{1}{|B|} \sum_{i \in (B \cap L)} \log z_i[y_i]$      ▷ supervised loss component
            $+ w(t) \frac{1}{C|B|} \sum_{i \in B} ||z_i - \tilde{z}_i||^2$      ▷ unsupervised loss component
      update $\theta$ using, e.g., ADAM                 ▷ update network parameters
    end for
  end for
  return $\theta$
```

Figure 3.2: Pseudocode describing the Pi Model. Figure originally published in [16].



```
Algorithm 1 Π-model pseudocode.
Require: $x_i$ = training stimuli
Require: $L$ = set of training input indices with known labels
Require: $y_i$ = labels for labeled inputs $i \in L$
Require: $w(t)$ = unsupervised weight ramp-up function
Require: $f_\theta(x)$ = stochastic neural network with trainable parameters $\theta$
Require: $g(x)$ = stochastic input augmentation function
  for $t$ in $[1, num\_epochs]$ do
    for each minibatch $B$ do
      $z_{i \in B} \leftarrow f_\theta(g(x_{i \in B}))$              ▷ evaluate network outputs for augmented inputs
      $\tilde{z}_{i \in B} \leftarrow f_\theta(g(x_{i \in B}))$      ▷ again, with different dropout and augmentation
      $loss \leftarrow -\frac{1}{|B|} \sum_{i \in (B \cap L)} \log z_i[y_i]$      ▷ supervised loss component
            $+ w(t) \frac{1}{C|B|} \sum_{i \in B} ||z_i - \tilde{z}_i||^2$      ▷ unsupervised loss component
      update $\theta$ using, e.g., ADAM                 ▷ update network parameters
    end for
  end for
  return $\theta$
```

Figure 3.3: Pseudocode describing the Temporal Ensembling Model. Figure originally published in [16].

3.1.3 Mean Teacher model

The Mean teacher model [29] proposes averaging model weights instead of averaging predictions as in the temporal ensembling model. Typically, averaging models or model weights produces a more accurate model than using the final weights of one of the trained model alone. The result is a teacher model which is an average of consecutive student models, and hence the name Mean Teacher. On training time, there will be also two loss terms. The softmax output of the student model is compared with the one-hot label using a classification cost such as cross entropy and with the teacher output using a consistency cost such as a squared-difference loss. Let θ be the parameters of the student model and θ' from the teacher model. Let f be the result of predicting targets with the network. Equation 3.3 presents this semi-supervised approach where α, λ and ω_t are defined as before. Also note that $f(x, \theta) = z_i$ in the previous models. At the end of training, both models can be used for prediction but the model teacher is more likely to be correct. As in the other models it is also possible to evaluate the same inputs applying different Gaussian noise or data augmentation.

$$\theta'_t = \alpha\theta'_{t-1} + (1-\alpha)\theta_t \tag{3.3a}$$

$$L = L_{ce} + \omega_t L_{cons} = -\sum_{i \in L} y_i \log f(x,\theta) + \frac{\lambda\omega_t}{N}\sum_{i=1}^{i=N} f(x,\theta)^2 - f(x,\theta')^2 \tag{3.3b}$$

3.2 Polygon-RNN

In this section there are presented two papers (”Annotating Object Instances with a Polygon-RNN” and ”Efficient Annotation of Segmentation Datasets with Polygon-RNN++” [1, 5]) that have inspired this work as well as having been the basis of which the code has been inherited. Both papers are fruit of the work of the Vector Institute and during my internship I've been able to learn and discuss topics with its authors.

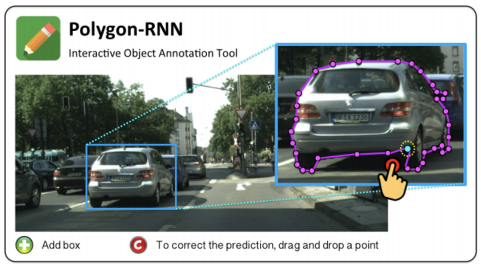

Polygon-RNN
Interactive Object Annotation Tool
Add box
To correct the prediction, drag and drop a point

Figure 3.4: Presentation of the Polygon-RNN annotation tool. Figure originally published in [5]

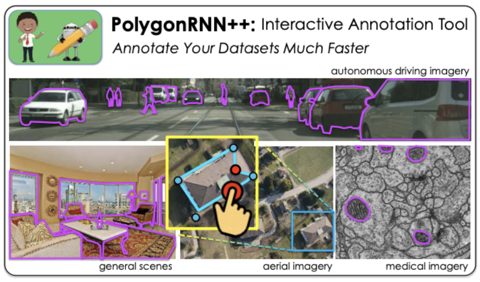

PolygonRNN++: Interactive Annotation Tool
Annotate Your Datasets Much Faster
autonomous driving imagery
general scenes
aerial imagery
medical imagery

Figure 3.5: Presentation of the Polygon-RNN annotation tool. Figure originally published in [1]l

Polygon-RNN and Polygon-RNN++ [1,5] are interactive annotation tools to produce polygonal annotations of objects interactively using humans-in-the-loop. The main benefit of using a polygon to annotate instances (instead of doing it pixel-wise) is that polygons are sparse and more easy to interact with resulting in less annotator corrections and faster performance. With their publications respectively in 2017 and 2018, both Polygon-RNN and Polygon-RNN++ were trained and evaluated on the Cityscapes Dataset [9] and achieved human-level state of the art performance on the semantic classes of car, truck and bus.

3.2.1 Polygon-RNN

Polygon-RNN [5] is an annotation tool for labeling object instances with polygons. The system acts assuming that the user provides the bounding box around the object and it predicts a polygon outlining the object using a Recurrent Neural Network. After the prediction it is possible to correct one or some predicted vertices of the polygon.

The architechture of the model strongly relies on a RNN that predicts a vertex at every time step until it closes the polygon. The inputs of the RNN are a CNN representation of the image crop, as well as the vertices predicted one and two time steps ago, plus the first point. By explicitly providing information of the past two points the RNN gets some help to follow a particular orientation of the polygon. On the other hand, the first vertex helps the RNN to decide when to close the polygon.

The architecture can be trained end-to-end which allows the CNN to be fine-tuned to object boundaries, while the RNN learns to follow these boundaries and exploits its recurrent nature to also encode priors on object shapes. Figure 3.6 shows an overview of the model.

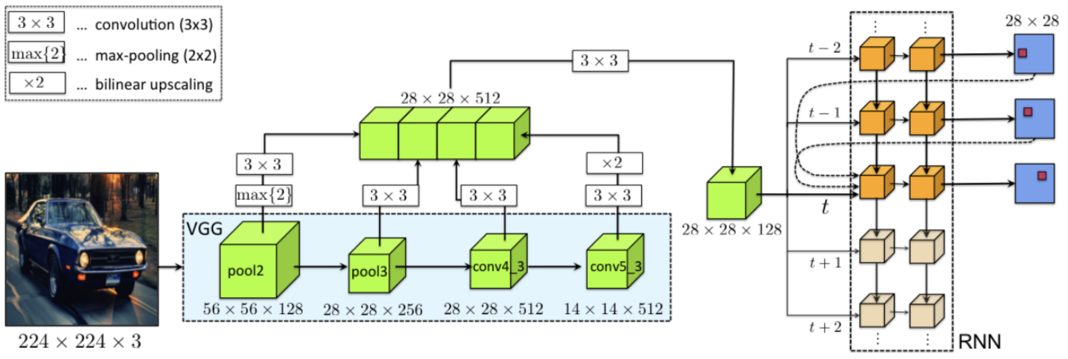


Figure 3.6: Scheme of the architecture of the Polygon-RNN model. At each time step of the RNN-decoder (right), an image representation is fed in using a modified VGG architecture. The RNN is a two-layer convolutional LSTM with skip-connection from one and two time steps ago. At the output at each time step, a new vertex of the polygon is predicted. Figure and description originally published in [5].

More concretely, the RNN is fed by an image representation which is obtained using a modified VGG-16. The modifications are the following: Removal of the fully connected layers as well as the last max-pooling layer, pool5. The output of this modified network has a downsampling factor of 16. On top of that, a couple additional convolutional layers with skipconnections are used to fuse information from the previous layers and upscale the output by factor of 2. This allows the CNN to extract features that contain both low-level information about the edges and corners, as well as semantic information about the object. The latter helps the model to “see” the object, while the former helps it to follow the object’s boundaries.

The Recurrent Neural Network that is used in Polygon-RNN is a Convolutional Long-Short-Term-Memory(LSTM) which operates in 2D and hence it preserves the spatial information received from the CNN. Furthermore, a Convolutional LSTM employs convolutions, thus greatly reducing the number of parameters to be learned compared to using a fully-connected RNN. To recall, in its simplest form, a single layer Convolutional LSTM computes the hidden state h_t given the input x_t according to the following equations:

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ g_t \end{bmatrix} = W_h * h_{t-1} + W_x + x_t + b \tag{3.4}$$

$$c_t = \sigma(f_t) \cdot c_{t1} + \sigma(i_t) \cdot tanh(g_t) \tag{3.5}$$

$$h_t = \sigma(o_t) \cdot tanh(c_t) \tag{3.6}$$

In this equations: i, f and o denote the input, forget, and output gate while h is the hidden state and c is the cell state. σ denotes the sigmoid function and $\cdot$ indicates an element-wise product and $*$ a convolution. The W_h and W_x matrices denote the hidden and input kernels which are 3x3 and have 16 channels. The vertex prediction is formulated as a classification task: at time step t, each vertex is encoded as a one-hot vector of a D×D+1 grid, where the D×D dimensions represent the possible 2D positions of the vertex, and the last dimension corresponds to the end-of-sequence token. The position of the vertices are thus quantized to the resolution of the output grid which was found to be optimum being 28 pixels. Finally the outputs are upsampled to original resolution of the image.

3.2.2 Polygon-RNN++

Polygon-RNN++ [1] is a following work of Polygon-RNN [5] that includes several important improvements to the existing model: In the first place, the encoder architechture changes from a VGG to a Resnet, also there is introduced a Reinforcement Learning training schedule and finally the output resolutions i s increased with a Graph Neural Network that allows the model to accurately annotate highresolution objects in images. With this improvements, the authors are able to significantly outperform Polygon-RNN (10% absolute and 16% relative improvement in mean IoU). They also show that Polygon-RNN++ exhibits powerful generalization capabilities, achieving significant improvements over existing pixel-wise methods. Figure 3.9 shows and scheme of the architecture of the model.

In more detail, the three factors that give this better performance are presented in the following. I encourage the reader to study the original paper [1] for further information:

- **The ResNet Architecture**. In Kaiming He's ResNet presentation he said: "lower time complexity than VGG-16/19 so this change in the architecture allows a faster training. " Apart from having less learnable parameters (VGG 16 has about 138 million parameters while ResNet has only 25.5 million) which is also better for training, the main reason that makes ResNet faster is that in VGG the first two layers apply convolution on top of the full 224x224 image (7x7 kernels) which is quite expensive while on Resnet only the first 3x3 kernel does. This simple computational case is enough for giving ResNet the victory in memory and time. Figure 3.7 shows an scheme of how the authors modified a ResNet with skip connections to obtain low-level and high-level combined features in two important tensors: the concat features of size $(112, 112, 256)$ and the skip features of size $(28, 28, 128)$. Those features are used aferwards to feed the First Vertex Network, the RNN, the Evaluation Network and the Graph Neural Network.

- **Reinforcement Learning**. On top of the CNN + RNN training that Polygon-RNN offered, in its improved version, the authors decided to train the outputs with the REINFORCE algorithm in order to make better decisions inside the 28x28 grid. In this training schedule, the reward function is the intersection over union (IoU) between the predicted polygon and the ground truth mask, so that in this schedule the goal is to grid-search over the 28x28 classes in order to improve vertex adjustments for larger IoU.

- **The Graph Neural Network**. Apart from the CNN + RNN and the reinforcement learning schedules, this model introduces a Graph Neural Network (GNN) that allows to scale from the 28x28 resolution back to 224x244 pixels from where it is easier to recover the original image in full resolution. The GNN interpolates the original polygon by creating a new vertex between every pair of existing vertices and then performs convolutions over the features of the vertices in order to decide which pixel is a better location for the vertex.

It is important to note that Polygon-RNN++ also includes attention and an evaluator network to predict better polygons. The evaluation network (Figure 3.8) is fed with the polygon mask prediction as well as the CNN features and the information of the last RNN hidden state. The evaluation network is trained to predict the Intersection over Union of the mask and can be used to maximize it. Its architecture is formed by 3x3 convolutions that convert from $(28, 28, 128 + 16 + 1)$ to $(28, 28, 16)$ and $(28, 28, 1)$ that is finally connected with

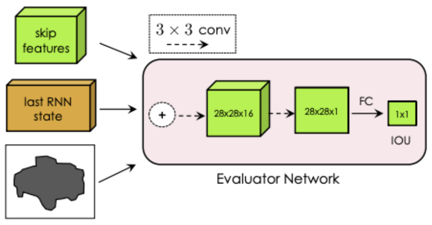


Figure 3.8: Scheme of evaluation network used in Polygon-RNN++. Figure originally published in [1].

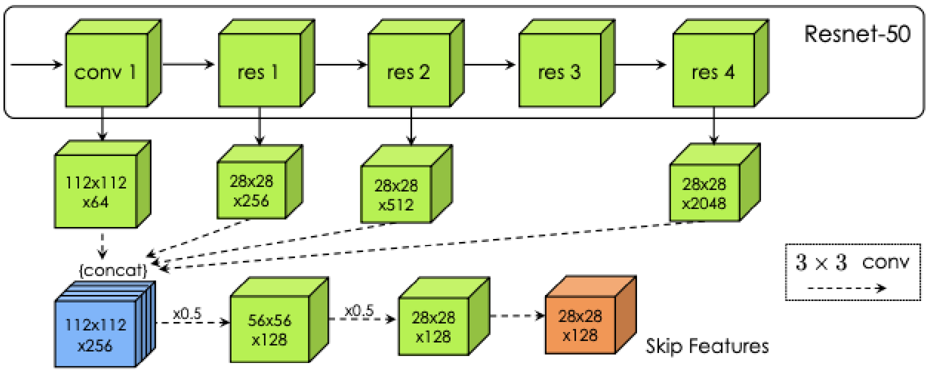


Figure 3.7: Scheme of the modified ResNet used in Polygon-RNN as a CNN encoder for obtaining features. Figure originally published in [1].

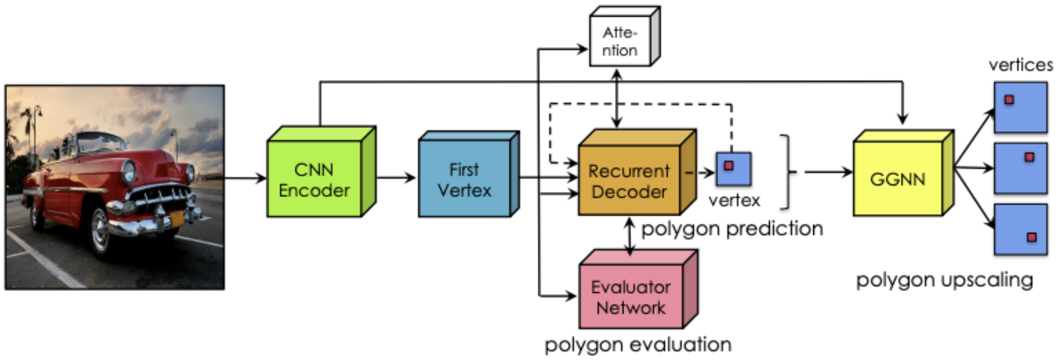


Figure 3.9: Scheme of the architecture of the Polygon-RNN++ model. Figure originally published in [1].

a fully connected layer to one unique value which will
be the IoU.

4. A semi-supervised method for image segmentation

This chapter can be considered the main chapter of this thesis. Under the different sections there will be presented the Cityscapes dataset, the Semi Supervised idea of the project along with the architecture and the models used, the supervised and unsupervised losses that have been implemented in order to train the model and finally some of the implementation details that have been chosen for obtaining the right performance.

4.1 The dataset: Cityscapes Dataset

In every Machine Learning or Deep Learning experiment, data is probably the most important thing. Not all methods can be used for any dataset and not all datasets allow the scientist to work with them in the same way. When facing a new dataset, a good data scientist must:

- **Understand** the dataset. Because it is very likely that there are some particularities of the dataset such as class imbalance, noisy inputs or wrong labels that one is not aware of a priory.
- **Decide** which method will be used with such dataset. Sometimes, the task is defined but some models in the literature can perform that task and it is crucial to determine what model will better fit the data. On the other hand, sometimes the predifined final task is new and one has to develop the method from scratch. In those cases, this step can be done at the same time than the following ones.
- **Preprocess** the data according to the understanding of the dataset. It is very important to preprocess the data before using the chosen analysis method. Every method will work better under certain conditions such as normalized data, black and white images, ... Some of the most important preprocessing techniques are:

  Checking for missing values

  Checking for categorical data

  Standardize the data

  White the data and/or do a PCA transformation

  Split the data into train, validation and test sets
- **Test and Visualize** if the preprocessed dataset is intelligible and well prepared for the method chosen. Sometimes, after the preprocessing, some numerical errors might cause some disfunction in the posterior method. For this reason, I would say it is "good praxis" to visualize the preprocessed data and ensure that no mistakes have been made (for instance, ensure that the values belong to the desired range, check that the images have been whitened...)

Table 4.1: Number of object instances per class in each one of the different splits. Note that this is the same splitting published in [1]

Split	Images	Bicycle	Bus	Person	Train	Truck	Motorcycle	Car	Rider	Total
Train	2711	3400	352	16452	136	455	657	24982	1575	48009
Val.	264	258	27	1462	32	27	78	1962	180	4026
Test	500	1129	98	3239	23	93	148	4517	537	9784

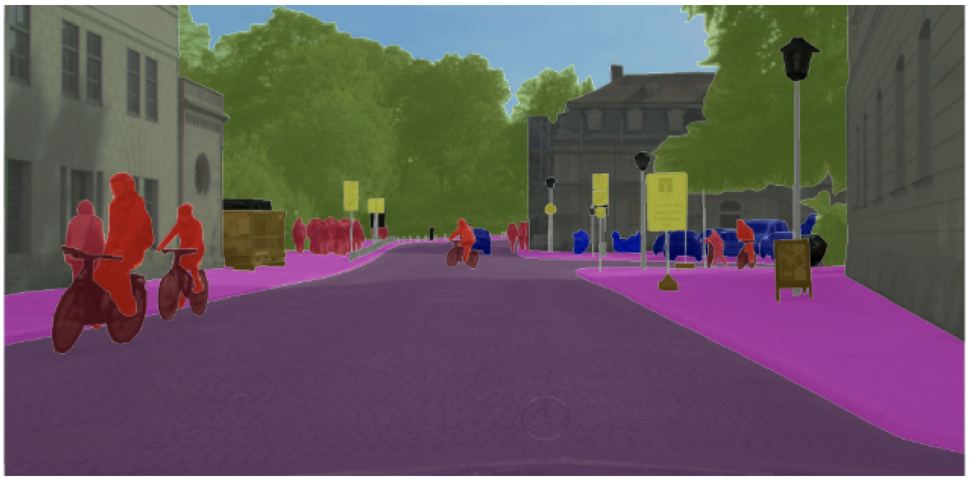


Figure 4.2: Example image of the Cityscapes Dataset. More concretely it was taken in the city of Weimar, federal state of Thuringia, Germany.

In this project, the dataset that has been used is the Cityscapes Dataset [9]. The Cityscapes Dataset contains 2975 training, 500 validation and 1525 test images with 8 semantic classes: bicycle, bus, person, train, truck, motorcycle, car and rider. All images contain annotations for all those classes and also for the following 11 categories: road, sidewalk, building, wall, fence, pole, traffic light, traffic sign, vegetation, terrain and sky. Figures 4.2 and 4.3 show a couple of examples of how are the annotated images of the dataset. All the images were taken in different cities of the center of Europe, more concretely Germany and Switzerland. Figure 4.1 shows a map with all the locations where pictures were taken. Note that for this project there has been only the possibility to work with the train and validation sets because the test images are private and cannot be downloaded. According to the available data, the splitting that we use is the following: we divide the 2975 training images into 2711 for training and 264 for validation while we keep the 500 for testing. The same split was used in [1]. Table 4.1 describes how many object instances are in every one of the splits used in the experiments.

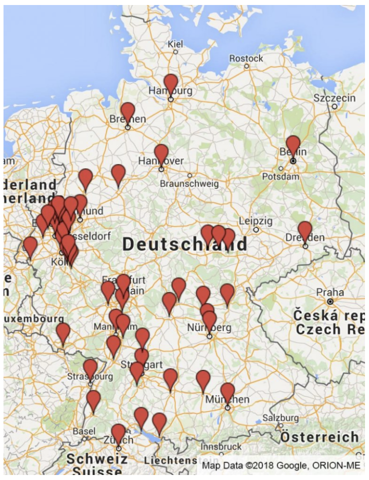


Figure 4.1: Map of the cities where have been taken the images of the Cityscapes dataset

4.1.1 Cityscapes preprocessing

The final goal of this project is predicting masks as accurate as possible of the 8 classes of the Cityscapes dataset with as less labeled data used as possible. Since the same dataset and splitting was recently used in [1], we also preprocess the data in the same way:

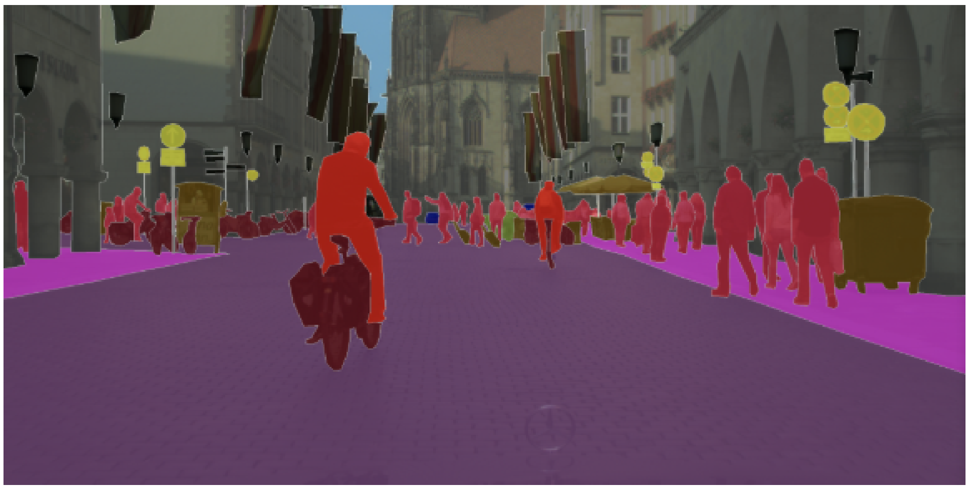


Figure 4.3: Example image of the Cityscapes Dataset. More concretely it was taken in the city of Münster, in North Rhine-Westphalia, Germany

1. Take every instance of one of the 8 classes and crop it from the image.
2. Remove some noisy annotated crops that could be present such as those that the area is less that 100 pixels.
3. Add some random context (i.e. margins) to the cropped image in order to obtain the instance centered but also to include background in the image.
4. Resize the cropped image to $224x224$ pixels.
5. Resize the annotated mask and to $224x224$ pixels.
6. Prepare pixel maps that contain information about where the vertices and edges of the mask could be.
7. Remove all the labeling from some of the images according to the desired experiment. In this step we remove every annotation that was present in the dataset and we only keep the $224x224$ raw image.
8. Feed the network with batches of some labeled and some unlabeled images.

4.1.2 Cityscapes benchmark

As complimentary imformation, Table 4.2 presents some of the current benchmarks of pixel segmentation among the 8 classes and the 11 categories. To assess performance, in Cityscpaes [9] use the standard Jaccard Index, also known as the PASCAL VOC intersection-over-union metric [10]:

$$IoU = \frac{TP}{TP + FP + FN} \tag{4.1}$$

where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively, determined over the whole test set. Note that pixels labeled as void do not contribute to the score.

In order to solve the problem that the global IoU measure is biased toward object instances that cover a large image area, what Cityscapes proposed is to additionally evaluate the semantic

Table 4.2: Cityscapes Pixel Segamentation Benchmark.

Model Name	IoU class	iIoU class	IoU category	iIoU category
iFLYTEK-CV	83.6	64.7	92.1	82.3
NV-ADLR	83.2	64.2	92.1	82.2
Tencent AI Lab	82.9	63.9	91.8	80.4
DRN_CRL_Coarse	82.8	61.1	91.8	80.7
DPC	82.7	63.3	92.0	82.5
SRC-B-MachineLearningLab	82.5	60.7	91.8	81.5
RelationNet_Coarse	82.4	61.9	91.8	81.4
DeepLabv3+	82.1	62.4	92.0	81.9
Auto-DeepLab-L	82.1	61.0	91.9	82.0

labeling using an instance-level intersection-over-union metric. For each type of instance i,

$$iIoU = \frac{iTP}{iTP + FP + iFN} \tag{4.2}$$

where again iTP, FP, and iFN denote the numbers of true positive, false positive, and false negative pixels, respectively but in contrast to the standard IoU measure, iTP and iFN are computed by weighting the contribution of each pixel by the ratio of the class' average instance size to the size of the respective ground truth instance. Note that since the methods only yield a standard per-pixel semantic class labeling as output, the false positive FP pixels are not associated with any instance and thus do not require normalization. The final scores, iIoUcategory and iIoUclass, are obtained as the means for the two semantic granularities.

I would like to point out in particular one method that is present in 4.2: DeepLabv3+ [7]. I would like to address the reader to that bibliography to further improve knowledge of pixel segmentation since both methods improved state-of-the-art performance in diverse tasks and datasets and allow the reader to understand how pixel segmentation can be trained not only on 8 classes of objects but on hundreds using tricks as encoding and decoding, atrous convolutions and conditional random fields and obtain such as good results as shown in Figure 4.4.

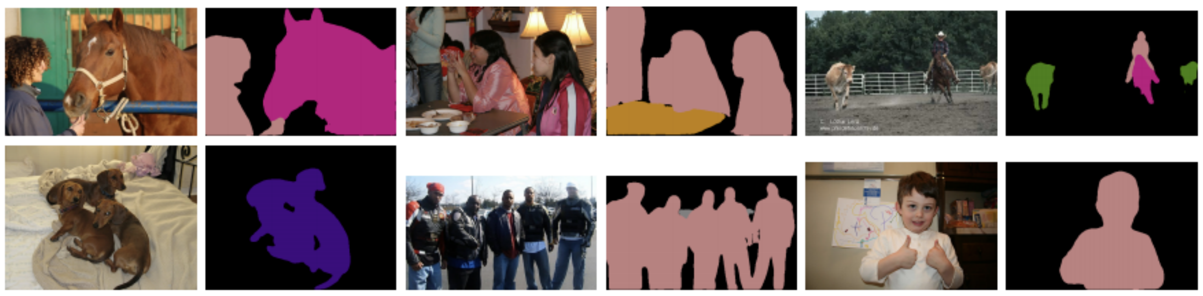


Figure 4.4: Example output of some masks using Deeplabv3+ in the ImageNet dataset

4.2 The Semi-Supervised Approach

This section presents the problem that I faced at the beginning of this project as well as the solution that I propose to solve it. At the beginning of my research internship, I was told that some previous work in the Vector Institute laboratory was on the Cityscapes Dataset [9] (namely

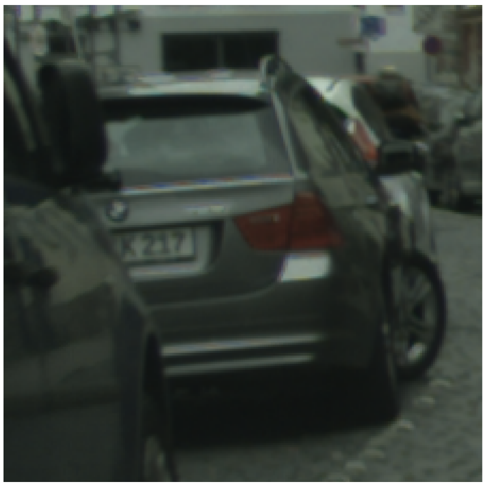


Figure 4.5: Example of an input image from the Cityscapes Dataset. This input has been cropped from a bigger image that contains a lot of entities and has been resized to 224x224 pixels.

both Polygon-RNN and Polygon-RNN++ [1, 5]) and that also in the field of Deep Learning, some relevant papers in Semi-Supervised Learning had been published. I started questioning if it was possible to combine those ideas and try to use a semi-supervised approach for image segmentation, and more concretely in the Cityscapes Dataset.

On the next pages, I would like to present the idea of how those ideas can be combined, the losses that we have designed for training a model to understand the unlabeled data, the architecture of the model that we used for training and finally the schedule and hyper-parameters that we used for training.

4.2.1 The idea

To train in a Semi-Supervised way, the most important problem to solve is how one will use the unsupervised data. This is a non-trivial question to answer in a segmentation framework where the labeled data are pixelwise masks of the objects given an image but for the unlabeled data there's only the raw image without any annotation. More concretely, in the processed Cityscapes dataset [9], which was already used in [1, 5] the input for any model is always a crop from a region containing one entire entity (car, person, bicycle, ...) and maybe partially regions of other images. Figure 4.5 shows an example of an input image which has a resolution of 224x224 pixels. In the Cityscapes dataset, it is possible to have pixelwise ground truth for vertices, edges and mask of the input image. Figure 4.6 represents the ground truth that is available for the input image of Figure 4.5.

With those ideas in mind, in a Supervised Framework, it is possible to use a pixelwise binary cross-entropy to supervise the edges, vertices and mask prediction. Assume that the ground truth maps are threee pixel maps of 224x224 resolution containing a value of 1 when the pixel is white (i.e. it is a vertex or forms part of an edge or mask) and 0 otherwise (represented by black in Figure 4.6). Also assume that the network produces three pixel map outputs also at a 224x224 resolution with values between 0 and 1 expressing the confidence of that pixel to be triggered in that map (this is typically done using a sigmoid function per pixel as a last layer of the network). Consider that $q(i,j)_k$ is the ground truth value for the pixel of the ith row and the jth column of the map k, where for instance $k = 1$ will denote verices, $k = 2$ edges and $k = 3$ mask. Let's define $p(i,j)_k$ as the predicted probability for the same pixel, then equations 4.3 and 4.4 show the binary cross-entropy loss that can be used for training. This loss can only

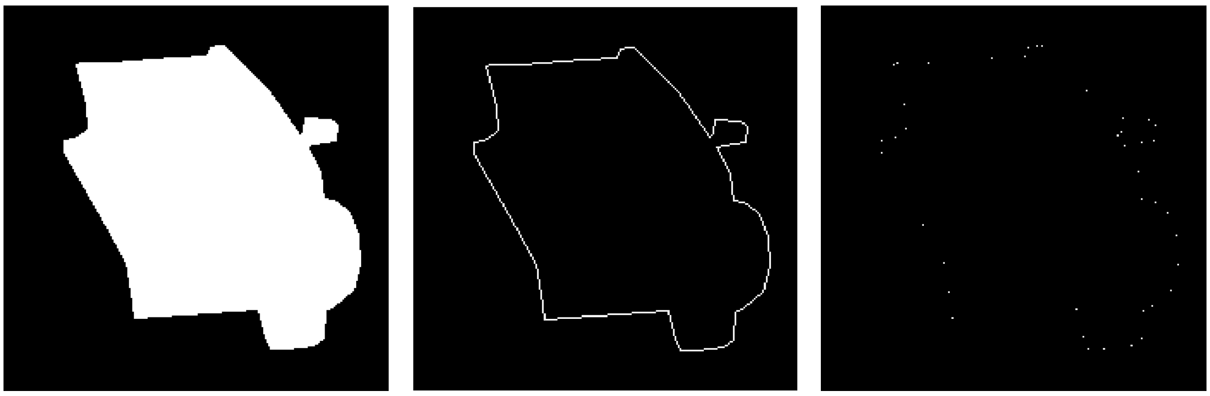


Figure 4.6: Examples of the Ground Truth Data of the Cityscapes Dataset. Those three images can be used as ground truth for the input image in Figure 4.5

be used when a ground truth map is available, and hence in a semi-supervised approach a new loss will have to deal with the unlabeled data.

$$\mathcal{L}_{bce}(q,p) = -q \cdot log(p) - (1-q) \cdot log(1-p) \tag{4.3}$$

$$\mathcal{L}_{BCE} = \sum_{k=1}^{3} \frac{1}{224^2} \sum_{i=1,j=1}^{224,224} \mathcal{L}_{bce}(q(i,j)_k, p(i,j)_k) \tag{4.4}$$

Inspired by the papers on Semi-Supervised Learning for Image Classification [16,19,22,29], the main idea of a semi supervised approach is forcing the network to be consistent with its predictions of the unlabeled data. In a regime such as image segmentation, I personally think that this consistency can be applied in two different ways. On the one hand, it is possible to evaluate twice the same input and penalize the network if it's not confident on giving the same result twice (note that if the network contains stochasticity, either in the input or in some layers, this is not trivial), this can be thought as being consistent within every pixel map. On the other hand, since we are producing output maps of vertices, edges and masks and there is a strong relation between those maps (the edge map should be the external pixels of the mask map and the vertices map should contain vertices that are extremes of the edge map), one could take leverage by penalizing a non-consistent output.

A lot of methods exist in the literature to extract edges or vertices form a mask but I would like to focus on the Canny edge detector [4] and the Harris corner detector [12].

The Canny Edge Detector.

Traditionally, the Canny edge detection algorithm can be divided into 5 different steps:

- Apply a **Gaussian filter** to smooth the image in order to remove the noise. Matrices in 4.5 show the typically used 3x3 and 5x5 Gaussian filters. Those filters take into account the intensities of the neighbours of a pixel in order to smooth the whole image and hence easy the next steps.

$$\frac{1}{6}\begin{bmatrix}0 & 1 & 0\\ 1 & 2 & 1\\ 0 & 1 & 0\end{bmatrix}, \quad \frac{1}{159}\begin{bmatrix}2 & 4 & 5 & 4 & 2\\ 4 & 9 & 12 & 9 & 4\\ 5 & 12 & 15 & 12 & 5\\ 4 & 9 & 12 & 9 & 4\\ 2 & 4 & 5 & 4 & 2\end{bmatrix} \tag{4.5}$$

- Find the intensity of the **gradients** of the image. This can be done using convolutional kernels that approximate the limit $f'(x) = \lim_{h\to 0}(f(x+h) - f(x-h))/h$. In 4.6 are the typical 3x3 gradient filters.

$$G_x = \begin{bmatrix}0 & 0 & 0\\ 1 & 0 & -1\\ 0 & 0 & 0\end{bmatrix}, \quad G_y = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 0\\ 0 & -1 & 0\end{bmatrix} \tag{4.6}$$

- Apply **non-maximum suppression** to get rid of false positive responses that are fired during edge detection. This is done in order to obtain thinner edges.
- Apply **double threshold** to determine the potential edges. The regions of the image that are more likely to be an edge, will always have a response bigger than a certain threshold.
- Finally it is also possible to **suppress all the remaining edges that are weak** and not connected to strong edges.

<u>The Harris Corner Detector.</u>

As the Canny edge detection algorithm, the Harris corner detector also can be thought as a 5-step process:

- Color to **grayscale**. This algorithm works using only one channel with values between 0 and 255.
- Compute the **spatial derivatives**. This step is analogus to the first two steps of the Canny edge detector method. One should apply convolutional filters to first obatain an smoothed image and then obtain the gradient intensity maps I_x and I_y.
- Build a 2x2 **tensor for every pixel** using the formula in equation 4.7.

$$M(i,j) = \begin{bmatrix} I_x^2(i,j) & I_x(i,j)I_y(i,j)\\ I_x(i,j)I_{y(i,j)} & I_y^2(i,j)\end{bmatrix} \tag{4.7}$$

- Calculate the **Harris response**. This is calculating the minimum eigenvalue of the matrix for every pixel. We know that a point is a corner if there are high perturbations in both directions and hence if and only if both eigenvalues are high and positive. After calculating the minimum eigenvalue per pixel using the approximation in equation 4.8, there will be corners for all the values that have a value of λ_{min} higher than certain threshold.

$$\lambda_{min} \approx \frac{\lambda_1\lambda_2}{(\lambda_1+\lambda_2)} = \frac{det(M)}{trace(M)} \tag{4.8}$$

- End the process performing a **Non-maximum suppression** as in the Canny edge detector.

Using the Canny Edge Detector [4] and the Harris Corner Detector [12], it is possible to obtain a map of the edges and a map of the vertices using a mask. Hence, the idea of this project lies in applying a consistency loss between the edge map predicted by the network and the obtained by performing one of those detection methods on top of the predicted mask map. This is a fully unsupervised approach, like a regularization method that just takes into account the predictions and penalizes the model whenever they are not consistent.

When the input data is supervised, it is also possible to use this consistency between the ground truth maps and the predictions (i.e. Applying the detectors on the ground truth mask and comparing against the predicted edges/vertices and/or applying the detectors on the predicted mask and comparing against the ground truth edges/vertices.

On top of that idea, it is also interesting the fact that the number of vertices/edges of the prediction cannot be extremely big. For this reason, it is also possible to include another loss that penalizes the fact that too many or too less vertices are predicted. This is typically done in the supervised data using the weighted binary cross entropy which is very similar to the binary cross entropy (Equation 4.3) but weighting by an alpha parameter that typically is inversely proportional to the quantity of ground truth positive predictions (Equation 4.9). In our case, we propose to penalize a big number of predicted edges/vertices by the difference against the allowed maximum or a low number by the difference against the allowed minimum.

$$\mathcal{L}_{wbce}(q,p) = -\alpha\, q\, \log(p) - (1-\alpha)(1-q)\, \log(1-p) \tag{4.9}$$

4.2.2 The losses

Assume that x is the Cityscapes input data and let's define X_L and X_UL as the labeled and unlabeled sets respectively. Assume that $p(x,y)_m, p(x,y)_e, p(x,y)_v$ are the the mask, edges and vertices predicted pixel-wise probability maps for one instance, $(x,y) \in \{224\}x\{224\}$. Assume also that $\hat{y}$ are predicted probabilities for the instance being in each one of the 8 classes of Cityscapes. Finally, assume that $q(x,y)_m, q(x,y)_e, q(x,y)_v$ and y are the corresponding ground truth labels. In the following, there are presented all the losses that are used during the semi-supervised training where the data is processed with mini-batches B.

1. **Binary Cross Entropy (BCE).** The BCE is used on the labeled fraction of the mask predictions. Equation 4.10 shows the loss for each mini-batch:

$$\mathcal{L}_{BCE} = \begin{cases} \frac{1}{|B \cap X_L| \cdot 224^2} \sum\limits_{\substack{i=1,j=1 \\ x \in X_L \cap B}}^{224,224} \mathcal{L}_{bce}(q(i,j)_{m,x}, p(i,j)_{m,x})) & \text{if } B \cap X_L \neq \phi \\ 0 & \text{if } B \cap X_L = \phi \end{cases} \tag{4.10}$$

2. **Weighted Binary Cross Entropy (WBCE).** The WBCE is used on the labeled fraction of the edges and vertices predictions where there is a huge class imbalance between positives and negatives. Equation 4.11 shows the loss for each mini-batch where k can be both

edges e or vertices v:

$$\mathcal{L}_{WBCE} = \begin{cases} \frac{1}{|B \cap X_L| \cdot 224^2} \sum\limits_{\substack{i=1,j=1 \\ x \in X_L \cap B}}^{224,224} \mathcal{L}_{wbce}(q(i,j)_{k,x}, p(i,j)_{k,x})) & \text{if } B \cap X_L \neq \phi \\ 0 & \text{if } B \cap X_L = \phi \end{cases} \tag{4.11}$$

3. **Classification Loss (CL).** The classification loss is used to help the network identify what kind of instance it is trying to segment. It is a loss only used on the labeled data and it is a cross entropy between the predicted classes $\hat{y}$ and the ground truth labels as one-hot encoding vectors y (Equation 4.12).

$$\mathcal{L}_{CL} = \begin{cases} \sum\limits_{x \in B \cap X_L} \sum\limits_{i=1}^{8} -y_i^x \log(\hat{y}_i^x) & \text{if } B \cap X_L \neq \phi \\ 0 & \text{if } B \cap X_L = \phi \end{cases} \tag{4.12}$$

4. **Consistecny Loss (C).** The consistency loss is directly a squared difference between two predictions of the same input. This is analogous as the procedure done in the Pi Model [16] and the loss is explicitly shown in equations 4.13 and 4.14 where q_1 and q_2 are two predictions and k can be masks m, edges e or vertices v:

$$\mathcal{L}_c(q_1, q_2) = \frac{1}{224^2} \sum_{i=1,j=1}^{224,224} (q_1 - q_2)^2 \tag{4.13}$$

$$\mathcal{L}_C = \frac{1}{|B|} \sum_{x \in B} \mathcal{L}_c(q_{1,k,x}, q_{2,k,x}) \tag{4.14}$$

5. **Consistency between Mask and Edges (CME).** The CME loss takes into account how consistent is the predicted edge map with the predicted mask. In order to do so, for the unlabeled data the penalization is a consistency loss between the predicted edges and the Canny detector map over the predicted mask. Nevertheless, on the labeled data it is possible to use this consistency between one of the predicted maps and one ground truth map. This loss is shown in equation 4.15 as a sum of three consistency losses, where Can is understood as the operation that using Canny edge detection results as the map of the detected edges over an image.

$$\mathcal{L}_{CME} = \sum_{x \in B \cap X_{UL}} \mathcal{L}_c(Can(q_m), q_e) + \sum_{x \in B \cap X_L} (\mathcal{L}_c(Can(p_m), q_e) + \mathcal{L}_c(Can(q_m), p_e)) \tag{4.15}$$

6. **Consistency between Mask and Vertices (CMV).** The CMV loss works exactly as the CME loss but taking into account the consistency between mask and vertices. Equation 4.16 shows explicitly the loss. Note that Har denotes the operation that using Harris corner detection results as the map of the detected vertices over an image.

$$\mathcal{L}_{CMV} = \sum_{x \in B \cap X_{UL}} \mathcal{L}_c(Har(q_m), q_e) + \sum_{x \in B \cap X_L} (\mathcal{L}_c(Har(p_m), q_e) + \mathcal{L}_c(Har(q_m), p_e)) \tag{4.16}$$

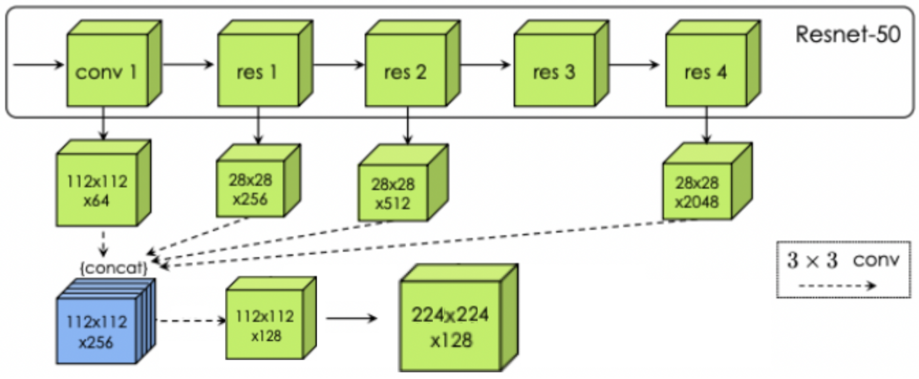


Figure 4.7: Scheme of the modified ResNet-50 used as a backbone model for this project.

7. **Some Vertices (SV).** The SV loss encourages the network to predict a number of vertices and edges over a certain range. Let's define v_{min} and v_{max} as the minimum and maximum values of vertices that we want a prediction to have and e_{min} and e_{max} the same for edges. Equations 4.17 and 4.18 show the SV loss where k can be used for either vertices v or edges e.

$$Q_k = \sum_{i=1,j=1}^{224,224} q(i,j)_k \mathbb{1}_{q(i,j)>0.5} \tag{4.17}$$

$$\mathcal{L}_{SV} = \begin{cases} k_{min} - Q_k & \text{if } Q_k < k_{min} \\ 0 & \text{if } Q_k \in [k_{min}, k_{max}] \\ Q_k - k_{max} & \text{if } Q_k > k_{max} \end{cases} \tag{4.18}$$

A loss composed by the sum of all those losses is used during training time. Equation 4.19 shows this total loss. The weight hyper-parameters that are used during training are the following: $\lambda_1 = 1000$, $\lambda_2 = 100$, $\lambda_3 = 10$ and $\lambda_4 = 0.1$.

$$\mathcal{L}_{TOTAL} = \lambda_1(\mathcal{L}_{BCE} + \mathcal{L}_{WCE}) + \lambda_2(\mathcal{L}_C + \mathcal{L}_{CL}) + \lambda_3(\mathcal{L}_{CMV} + \mathcal{L}_{CME}) + \lambda_4\mathcal{L}_{SV} \tag{4.19}$$

4.2.3 Our backbone model

Since the code of this project was build on top of the inherited Polygon-RNN++ code [1], the main experiments and all the testing research was done with a ResNet-50 modified model very similar to the one they used (See Figure 4.7 for an orientating scheme). We keep the same structure with skip conections but we change the last layers:

- Keep exactly as in PolygonRNN++ the conv1, res1, res2, res3 and res4 layers of the ResNet.
- Concatenate the features obtained at the biggest scale (112x112 pixels) to obtain a tensor of size $(112, 112, 256)$.
- Use two 3x3 convolutional layers of 128 layers mantaining the spatial dimension.
- Upsample the final features to obtain a $(224, 224, 128)$ tensor that will be used to predict the mask, edges, vertices (through a 3x3 kernel) and the classes of the instance (with a fully connected).

Note that all the losses should be independent of the backbone model since they only require the prediction pixel-wise maps. Nevertheless, it is future work test that this theoretical behaviour will allow to obtain good results with all kind of different backbones.

4.3 Implementation Details and Training Schedule

During training time, the first step we perform is preparing a Dataloader with the number of labels proposed for the experiment. In the next section, we will present results with 2000, 4000, 6000, 80000 and 10000 labels. We chose 16 to be the mini-batch size and at every forward pass we compute all 7 losses and then decide which combination will be backpropagated following the experiment instructions.

We have chosen the learning rate to start at 0.0001 and decrease by a factor of 0.3 every 3 epochs. We also include the same ramp schedule than in [16] multiplying the unsupervised losses by 0.01 on the first epoch and increasing gradually until 1 on the sixth.

We have chosen to preserve the same hyper-parameters for all experiments except for the period of decreasing the learing rate which we scale inversely proportionally to the amount of labeled data in the supervised experiments.

The code and the training schedule will be released at `http://github.com/jordifortuny/VertexEdgeMask`.

5. Experiments and Results

In order to test the improvement of the losses, we evaluate some experiments on the Cityscapes Dataset [9]. More concretely, this section presents ablation studies that show how the accuracy (Intersection over Union or F-score) increases when adding different combinations of our losses. There are also included some comparisons between the best Semi-Supervised models versus models that have been only supervised on different regimes. Moreover, we also show some visual results of the semi-supervised models that prove that gain better understanding over the dataset classes.

Table 5.1 and Images 5.1, 5.2 and 5.3 show a comparison between the best trained models on different data regimes on the Cityscapes Dataset. The models have been trained and tested using 5%, 10%, 15%, 20% and 25% of the data. We considered that testing the model on both low data regimes (5%) and high data regimes (25%) was a fair study to show how the model performs. As one can see, with only 5% of the data labeled, our semi-supervised model achieves an outstanding 93% of the fully supervised Intersection over Union. Note that when one adds more data, the results are better and reach about 98% of the fully supervised approach using only 25% of the data. It is also important to note that in all experiments the semi-supervised models improve the only supervised models by at least 2 points.

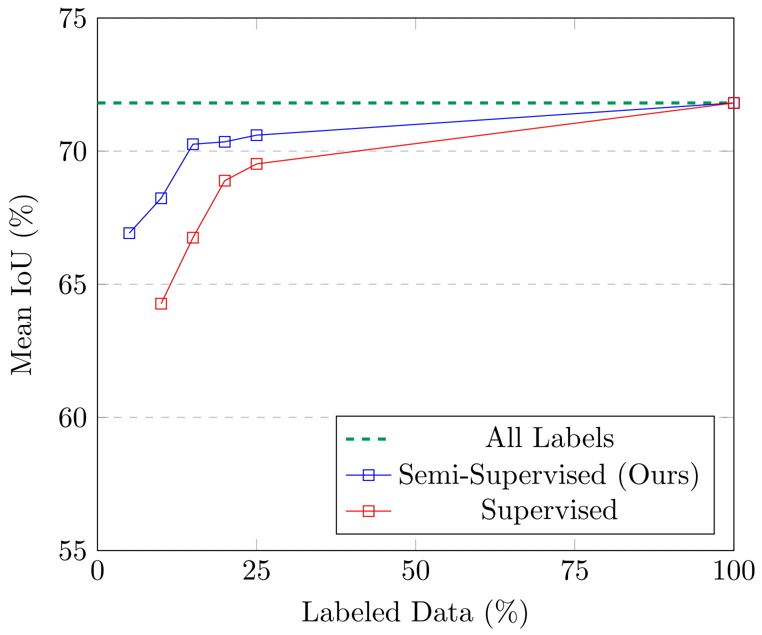


Figure 5.1: IoU when different data regimes are labeled. The Semi-Supervised Model outperforms the only supervised model by more than 4 points on low labeled regimes and by approximately 2 points on high labeled regimes

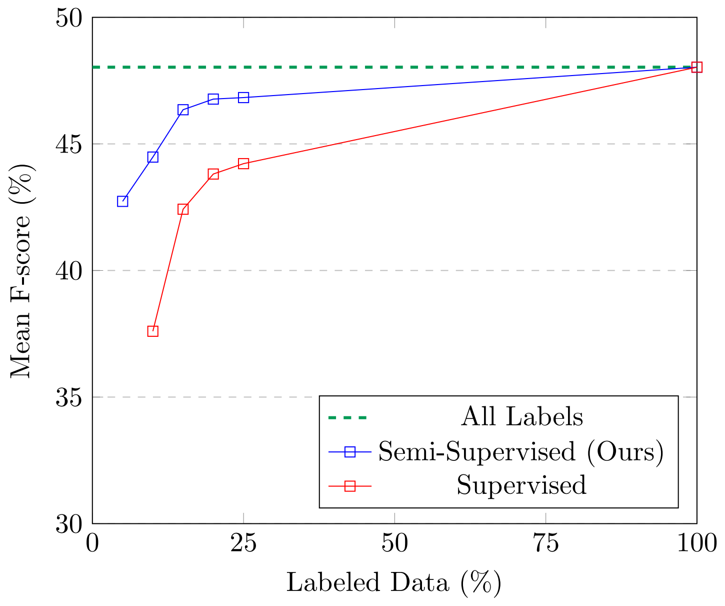


Figure 5.2: F-score with threshold of 1 pixel when different data regimes are labeled. The Semi-Supervised Model outperforms the only supervised model by more than 6 points on low labeled regimes and by approximately 4 points on high labeled regimes

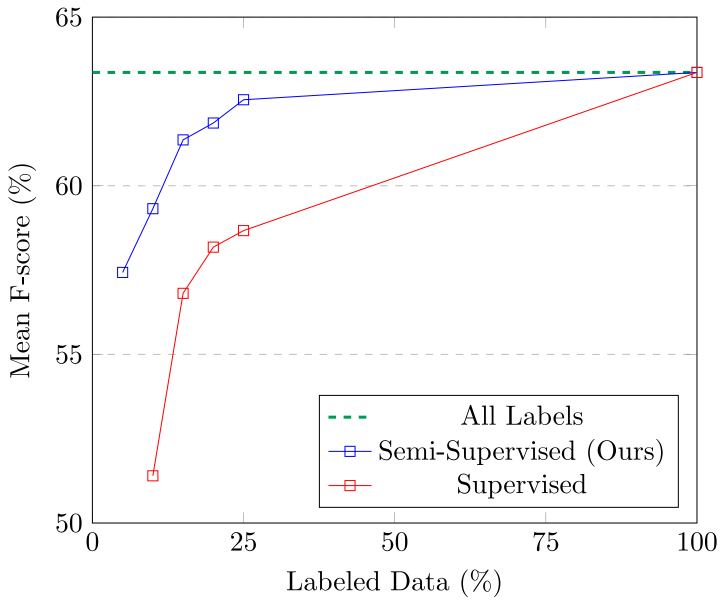


Figure 5.3: F-score with threshold of 2 pixels when different data regimes are labeled. The Semi-Supervised Model outperforms the only supervised model by more than 7 points on low labeled regimes and by approximately 5 points on high labeled regimes

Tables 5.2 and 5.3 present ablation studies on the Intersection over Union and the F-scores obtained using different losses. As one can see, the classification, the consistency loss and the consistency between edges, vertices and mask losses all increase the accuracy in almost all classes. The mean of all classes definitely proves that our method is able to obtain information from the unlabeled images and that the consistency between the vertices and the mask as well as the edges and the mask. It is interesting that on low labeled data regime (Table 5.3), the semi-supervised model with all the losses outperforms all the other models on all classes, increasing the accuracy specially on the Train and Rider classes.

Table 5.1: Comparison between the best trained models on different data regimes on the Cityscapes Dataset. As more data is labeled, mean Intersection over Union (mIoU), and F-score (F) with thresholds of both 1 and 2 pixels increase. %FS denotes percentage of the fully supervised experiment on the whole dataset. As one can see, only labelling 10% of the data our model performs over the 92% in all three categories and over 97% when 25% of the data is labeled.

Model	mIOU	%FS	F, th=2	%FS	F, th=1	%FS
Ours 5%	0,6692	93,19%	0,5743	90,64%	0,4273	88,97%
Ours 10%	0,6823	95,01%	0,5932	93,62%	0,4448	92,61%
Ours 15%	0,7026	97,84%	0,6136	96,84%	0,4635	96,50%
Ours 20%	0,7035	97,97%	0,6186	97,63%	0,4677	97,38%
Ours 25%	0,7060	98,31%	0,6255	98,71%	0,4683	97,51%
Fully Supervised	0,7181	100,00%	0,6336	100,00%	0,4803	100,00%

Table 5.2: Ablation Study on the Intersection over Union (IoU) and the F-score with threshold of both 1 and 2 pixels using 10000 labels which is 25% of the Cityscaspes dataset. Note that: BCE = Binary Cross Entropy Loss, +CL = With Classification Loss, +C = With Consistency Loss, +CME & CMV = With Consistency between Mask and Edges and Mask and Vertices.

IoU with 10000 Labels (25%)

Model	Person	Bicycle	Train	Truck	Motorcycle	Bus	Car	Rider	Mean
Baseline (BCE)	0,7049	0,6175	0,6126	0,7755	0,6153	0,7868	0,7755	0,6738	0,6952
+ CL	0,7081	0,6200	0,6125	0,7734	0,6203	0,7869	0,7764	0,6796	0,6972
+ C	0,7108	0,6254	0,5892	0,7759	0,5925	0,7821	0,7770	0,6944	0,6934
+ CME & CMV	**0,7135**	**0,6287**	**0,6368**	**0,7792**	**0,6065**	**0,7973**	**0,7873**	**0,6986**	**0,7060**

F-score with threshold=1px with 10000 Labels (25%)

Model	Person	Bicycle	Train	Truck	Motorcycle	Bus	Car	Rider	Mean
Baseline (BCE)	0,5434	0,4119	0,2356	0,4552	0,4091	0,3970	0,5812	0,5043	0,4422
+ CL	0,5446	0,4117	0,2332	0,4557	**0,4091**	0,4000	0,5817	0,5055	0,4427
+ C	0,5604	**0,4265**	0,2342	0,4635	0,4049	0,4218	**0,5944**	0,5169	0,4528
+ CME & CMV	**0,5677**	0,4181	**0,3059**	**0,4905**	0,4062	**0,4493**	0,5887	**0,5202**	**0,4683**

F-score with threshold=2px with 10000 Labels (25%)

Model	Person	Bicycle	Train	Truck	Motorcycle	Bus	Car	Rider	Mean
Baseline (BCE)	0,7192	0,5602	0,3296	0,6050	0,5403	0,5352	0,7360	0,6680	0,5867
+ CL	0,7208	0,5543	0,3309	0,6078	0,5408	0,5400	0,7398	0,6740	0,5886
+ C	0,7270	**0,5768**	0,3354	0,6155	0,5426	0,5635	0,7541	0,6832	0,5998
+ CME & CMV	**0,7414**	0,5760	**0,4162**	**0,6467**	**0,5589**	**0,6109**	**0,7608**	**0,6927**	**0,6255**

Table 5.3: Ablation Study on the Intersection over Union (IoU) and the F-score with threshold of both 1 and 2 pixels using 4000 labels which is 10% of the Cityscaspes dataset. Note that: BCE = Binary Cross Entropy Loss, +CL = With Classification Loss, +C = With Consistency Loss, +CME & CMV = With Consistency between Mask and Edges and Mask and Vertices.

IoU with 4000 Labels (10%)

Model	Person	Bicycle	Train	Truck	Motorcycle	Bus	Car	Rider	Mean
Baseline (BCE)	0,6649	0,5661	0,5390	0,7173	0,5188	0,7474	0,7461	0,6419	0,6427
+ CL	0,6683	0,5678	0,5716	0,7198	0,5496	0,7476	0,7523	0,6462	0,6529
+ C	0,6689	0,5789	0,5765	0,7234	0,5487	0,7512	0,7564	0,6458	0,6562
+ CME & CMV	**0,7036**	**0,5999**	**0,6391**	**0,7397**	**0,5611**	**0,7544**	**0,7696**	**0,6906**	**0,6823**

F-score with threshold=1px with 4000 Labels (10%)

Model	Person	Bicycle	Train	Truck	Motorcycle	Bus	Car	Rider	Mean
Baseline (BCE)	0,4831	0,3505	0,1978	0,3580	0,3369	0,3376	0,5127	0,4313	0,3760
+ CL	0,4861	0,3535	0,1932	0,3571	0,3541	0,3194	0,5182	0,4340	0,3770
+ C	0,4943	0,3640	0,1997	0,3675	0,3548	0,3237	0,5245	0,4460	0,3843
+ CME & CMV	**0,5547**	**0,3900**	**0,3053**	**0,4379**	**0,3668**	**0,4065**	**0,5817**	**0,5155**	**0,4448**

F-score with threshold=2px with 4000 Labels (10%)

Model	Person	Bicycle	Train	Truck	Motorcycle	Bus	Car	Rider	Mean
Baseline (BCE)	0,6480	0,4934	0,2720	0,4933	0,4691	0,4708	0,6753	0,5900	0,5140
+ CL	0,6509	0,4957	0,2725	0,4959	0,4860	0,4585	0,6787	0,5932	0,5164
+ C	0,6570	0,5008	0,2698	0,5008	0,4870	0,4716	0,6897	0,6043	0,5226
+ CME & CMV	**0,7238**	**0,5426**	**0,4049**	**0,5856**	**0,5038**	**0,5542**	**0,7495**	**0,6811**	**0,5932**

Images 5.4, 5.5 present visual results on the Cityscapes Dataset. One can see how the semi-supervised model learns better how to model the masks. I would like to highlight the following corrections that can be generalized for the entire dataset:

1. Correction of motorbikes mask, adjusting tires, seat and handlebar.
2. Better understanding of trains, both semantic and segmentation.
3. Distinction of multiple motorbike instances, as well as better understanding of the segmentation.
4. Overall understanding of crowded scenes correcting bicycles and people.
5. Completion of broken or missing parts of masks such as legs or motorbike's tires.
6. Better understanding of parked cars, with more accurate and fine boundaries between them.
7. Better segmentation of occluded people.
8. Detection of street poles.
9. Better segmentation of distant people with clear distinctions and boundaries.

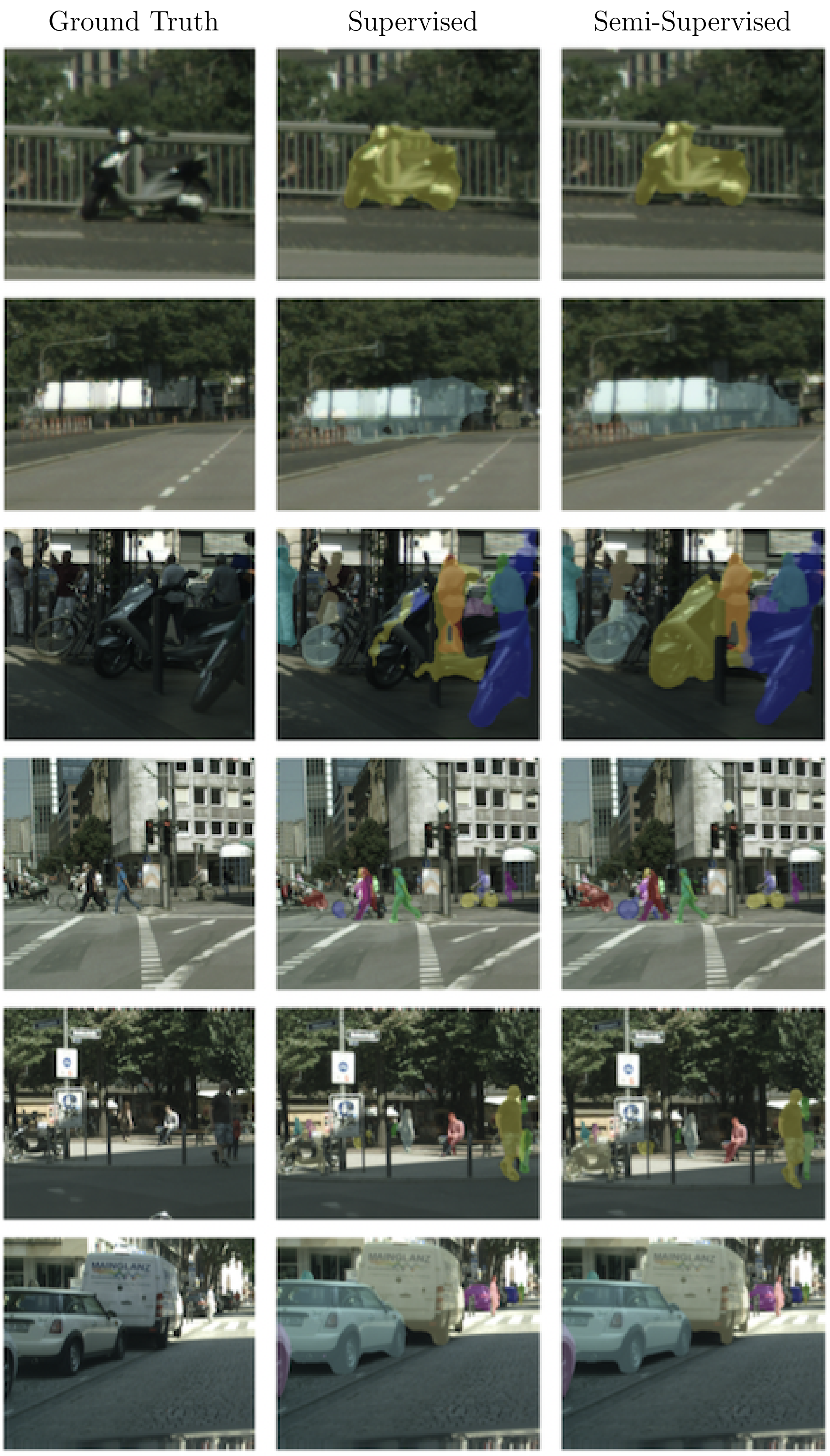


Figure 5.4: Some examples of the results obtained on the Cityscapes Dataset. **Left:** Ground Truth, **Center:** Supervised Network on 10% of the data. **Right:** Predictions with our semi-supervised method.

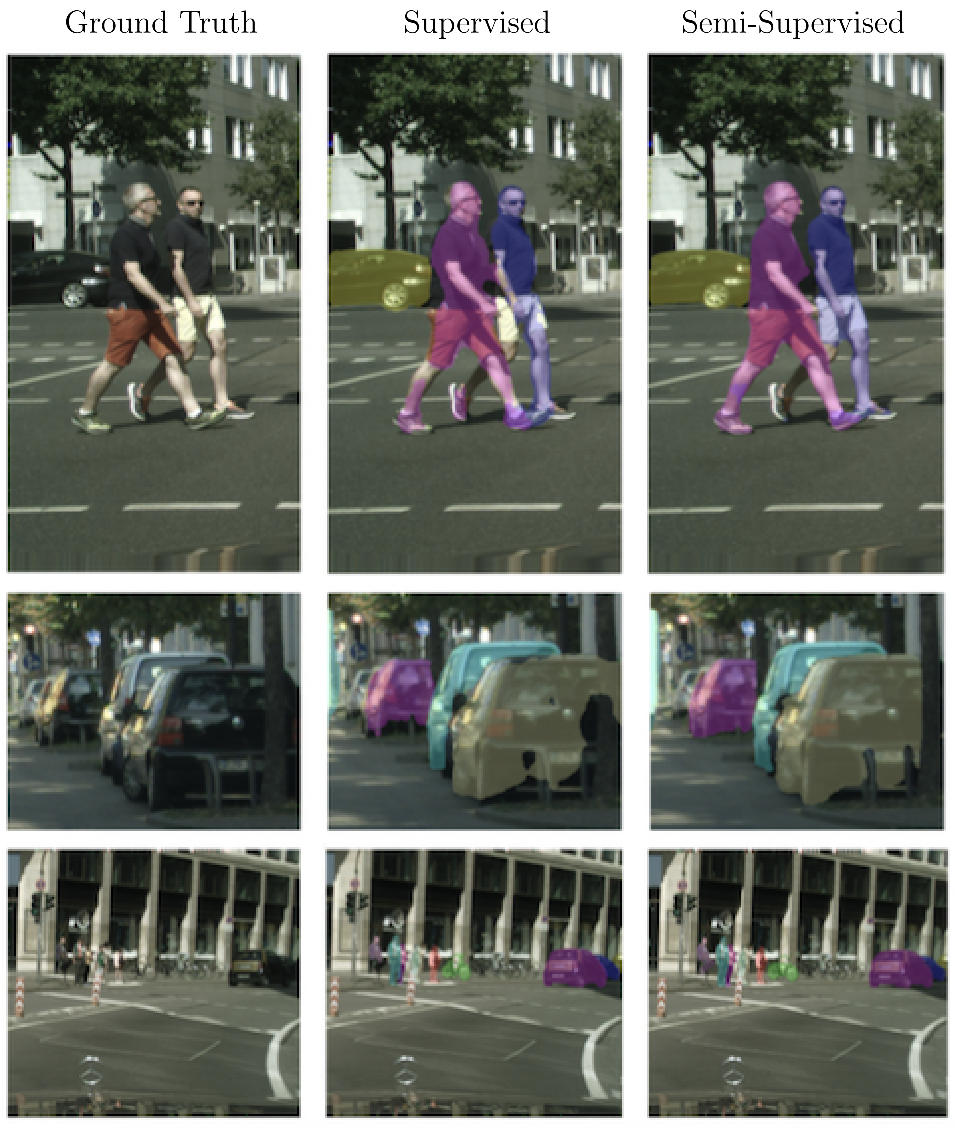


Figure 5.5: Some examples of the results obtained on the Cityscapes Dataset. **Left:** Ground Truth, **Center:** Supervised Network on 10% of the data. **Right:** Predictions with our semi-supervised method.

6. Conclusions and Future Work

This is the last chapter of this thesis and I would like to finish with some conclusions that I've obtained during this internship and future work that I feel could be useful in order to improve the methodology and results explained in this thesis.

6.1 Conclusions

In order to give conclusions of the project, I would like to refer the reader to chapter 1 where the objectives are stated. I will mention each one of those objectives and explain how it was accomplished and what personal impressions I've had about it.

- **Understanding the current state-of-the art results and the recent literature.**

  I consider that one may never be able to catch up with all the relevant literature of the field of deep learning and much less the field of computer science. Those are tremendously broad and thousands of papers are published per week. On the other hand, once one chooses a clear direction of research, it is easier to start a review of the most important literature, familiarize with the author names and institutions and get to know the evolution of the field as well as the current situation.

  I feel that Semi-Supervised Deep Learning is a field that is starting to gain popularity but that was not particularly explored during the last couple decades. This allowed me to catch up quickly with the literature and understand how important models such as [16, 19, 29] were improvements one to another and state-of-the art during the past two years.

  Having read at least 50 papers in detail and having skimmed about 100 more, I feel that I've become a literature expert of the field as I proposed at the beginning of the internship.

- **Learning the programming skills needed.**

  During an IT internship, it is almost mandatory to have good coding skills. My case wasn't different and during the previous months of travelling to Toronto I completed the CS231n course from the Stanford University [17]. In that course, that I absolutely recommend to anybody who wants to learn deep learning, I was taught the basis of deep learning and how to code them in Pytorch [23].

  During my degree I was able to understand the basics of programming and the most useful algorithms. Now, after 6 months of learning python tips every day, I've finished both degrees building and entire project that will be released at `https://github.com/`

`jordifortuny/VertexEdgeMask`. I think that in this sense, this internship has been extremely successful and has allowed me to improve a lot my coding skills.

- **Semi-Supervised Learning for image segmentation.**
  The main goal of this internship was creating a method that could be used as a semi-supervised framework to train a network to perform image segmentation. I believe that the goal has been accomplished since our method precisely performed image segmentation on a semi-supervised framework. On top of that, I think that we have obtained outstanding results where labelling only a few percentage of the data images we have been able to extract knowledge from the other images in order to improve the overall performance up to a 98% of the fully supervised approach.

  Observing the visual results, it is impressive how such technique is able to localize the overlapping objects and adjust the edges in order to produce more accurate masks. When I obtained the numerical results, the first thing that came into my mind was the wonder of how it could boost the performance, and the visual results confirmed that it was taking into account all the details and using all the information that was hidden on the edges.

  During the internship it came to our attention that there was recent literature published related to this field but none of them was tested under the same conditions that we were using. My feeling is that we will be performing as well as the other methods and it is future work to assert that. Moreover, it will be possible to combine all the ideas and maybe boost a few more point the accuracy.

- **Gain knowledge.**
  This has been the easiest objective to fulfill. Every day that I've been at the Vector Institute has been a new source of knowledge and I've learned a lot of mathematics and statistics as well as deep learning. Through meetings and talks, I've been able to hear from some of the best scientists in the field and get to know their techniques and understand some of the papers they recently published.

- **Growing as a person.**
  I would like to conclude by thanking this internship for bringing me many good things such as the mixture of knowledge and culture from people from very different backgrounds at the Vector Institute. It has helped me to understand a lot of cultural traditions and grow as a human being for sure.

6.2 Future work

Although this project has achieved results that fulfill the expectations and the objectives that I proposed at the beginning of the internship, I think that it could be still improved in some ways:

- **Performing experiments on different datasets:** It would be nice to continue this work analyzing how the model would train and perform on other datasets such as VOC2012 or PASCAL [10]. It has recently come to my attention that recent papers [14, 24, 28] have attempted the semi-supervised segmentation task with different settings an approaches than our method and have been only tested on Medical Datasets or on VOC2012. Future work would be reading carefully the settings and experiments they perform and try to reproduce them and compare against them.

- **Using different backbone architectures:** Another way to continue this work is testing that the semi-supervised approach will work with different backbone networks. Since the network was not specially build for this particular task, we consider that the method will work for any deep enough backbone network but it would be nice to test it performing different ablation studies.

- **Improving the Canny ideas:** During the whole internship, an important idea that we have always had in mind was how could we use an already existing computer vision technique such as the Canny Edge detector to help deep learning algorithms. After this internship we feel that with the help of the losses that use Canny techniques, we can leverage having unlabeled data with semi-supervision. However, the Some Vertices Loss can still be improved and used to obtain better results.

Bibliography

[1] David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient annotation of segmentation datasets with polygon-rnn++. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[2] Deep AI. Build with ai. 2017. `https://deepai.org`.

[3] Yoshua Bengio, Yann LeCun, and Geoffrey Hinton. Deep learning. In *Nature*, 2015.

[4] John Canny. A computational approach to edge detection. In *Transactions on pattern Analysis and machin intelligence, vol. PAMI-8*, 1986.

[5] Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Annotating object instances with a polygon-rnn. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. Semi-supervised learning. Cambridge, Mass.: MIT Press, 2006.

[7] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proc. IEEE European Conference on Computer Vision (ECCV)*, 2018.

[8] Francois Chollet. *Deep Learning with Python*. Manning, 2017.

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M.Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] M. Everingham, A. S. M. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. IJCV, vol. 111, iss. 1, 2014.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Plessey Research Roke Manor*, 1988.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Supervision (CVPR)*, 2015.

[14] Seunghoon Hong, Hyeonwoo Noh, and Bohyung Han. Decoupled deep neural network for semi-supervised semantic segmentation. In *The Neural Information Processing Systems (NIPS)*, 2018.

[15] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[16] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *The International Conference on Learning Representations (ICLR)*, 2017.

[17] Fei-Fei Li, Justin Johnson, and Serena Yeung et al. *Convolutional Neural Networks for Visual Recognition.* Stanford University, 2018. `http://cs231n.stanford.edu/`.

[18] Pamela McCorduck. Machines who think: A personal inquiry into the history and prospects of artificial intelligence. Natick, MA: A. K. Peters, Ltd.,, 2004.

[19] Takeru Miyato, Shin ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. In *Proc. International Conference on Learning Representations (ICLR)*, 2016.

[20] Nils J. Nilsson. The quest for artificial intelligence: A history of ideas and achievements. Cambridge, UK: Cambridge University Press, 2010.

[21] Alexandros Ntoulas, Omar Alonso, and Vasileios Kandylas. A data management approach for dataset selection using human computation. In *Proc. International Conference on Human-centric Computing*, 2013.

[22] Avital Olivera, Augustus Odena, Colin Raffel, Ekin D. Cubuk, and Ian J. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In *The Neural Information Processing Systems (NIPS)*, 2018.

[23] 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. In *NIPS-W*, 2017.

[24] Christian S. Perone and Julien Cohen-Adad. Deep semi-supervised segmentation with weight-averaged consistency targets. In *arXiv:1807.04657*, 2018.

[25] Adrian Rosebrock. *Deep Learning for Computer Vision with Python.* pyimaginsearch, 2017.

[26] Burr Settles. Active learning literature survey. In *Computer Sciences Technical Report 1648, University of Wisconsin–Madison*, 2010.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale recognition. In *The International Conference on Learning Representations (ICLR)*, 2015.

[28] Nasim Souly, Concetto Spampinato, and Mubarak Shah. Semi supervised semantic segmentation using generative adversarial network. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[29] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *The Neural Information Processing Systems (NIPS)*, 2017.

[30] Marcel van Gerven and Sander Bohte. Artificial neural networks as models of neural information processing. *Front. Computer Neuroscience*, 2017.

[31] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Appendix 1

This appendix contains a python code used for training a very simple deep neural network on the CIFAR-10 dataset. It was designed in order to learn Pythorch [23] and the basics of computer vision and deep learning and trained during 500 epochs on a Titan-XP GPU achieves an 85% of accuracy.

I particularly like the object oriented style that Pytorch provides the user. It makes everything clear to understand and follow. We would train the network running in a terminal "python train.py".

net.py

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(64, 64, 3)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.conv3 = nn.Conv2d(64, 128, 3, stride=2)
        self.fc = nn.Linear(128 * 4 * 4, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = self.pool2(F.leaky_relu(self.conv2(x)))
        x = x.view(-1, 128 * 4 * 4)
        x = self.fc(x)
        return x
```

train.py

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
from net import Net

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=32,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=100,
                                         shuffle=False, num_workers=2)

net = Net().to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(500):

    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

torch.save(net.state_dict(), 'trained_net.pth')
```