CentroidNetV2

Dijkstra, Klaas; van de Loosdrecht, Jaap; Atsma, Waatze A.; Schomaker, Lambert R. B.; Wiering, Marco A.

Published in:
Neurocomputing

DOI:
10.1016/j.neucom.2020.10.075

Publication date:
2021

Neurocomputing 423 (2021) 490–505

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

CentroidNetV2: A hybrid deep neural network for small-object segmentation and counting

Klaas Dijkstra [a,c,*], Jaap van de Loosdrecht [a], Waatze A. Atsma [b], Lambert R.B. Schomaker [c], Marco A. Wiering [c]

[a] Professorship Computer Vision and Data Science, NHL Stenden University of Applied Sciences, P.O. Box 1080, 8900 CB Leeuwarden, The Netherlands
[b] Vitens N.V., Snekertrekweg 61, 8912 AA Leeuwarden, The Netherlands
[c] Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 22 December 2019
Revised 26 July 2020
Accepted 11 October 2020
Available online 5 November 2020
Communicated by Lei Zhang

2000 MSC:
68T10
68T45

Keywords:
Deep Learning
Computer Vision
Convolutional Neural Networks
Object Detection
Instance Segmentation

ABSTRACT

This paper presents CentroidNetV2, a novel hybrid Convolutional Neural Network (CNN) that has been specifically designed to segment and count many small and connected object instances. This complete redesign of the original CentroidNet uses a CNN backbone to regress a field of centroid-voting vectors and border-voting vectors. The segmentation masks of the individual object instances are produced by decoding centroid votes and border votes. A loss function that combines cross-entropy loss and Euclidean-distance loss achieves high quality centroids and borders of object instances. Several backbones and loss functions are tested on three different datasets ranging from precision agriculture to microbiology and pathology. CentroidNetV2 is compared to the state-of-the art networks You Only Look Once Version 3 (YOLOv3) and Mask Recurrent Convolutional Neural Network (MRCNN). On two out of three datasets CentroidNetV2 achieves the highest F1 score and on all three datasets CentroidNetV2 achieves the highest recall. CentroidNetV2 demonstrates the best ability to detect small objects although the best segmentation masks for larger objects are produced by MRCNN.

1. Introduction

Counting many small and connected objects is an important and challenging image analysis task [1,2]. Many applications for counting objects exist ranging from microbiology [3] to precision agriculture [4]. For example, to test the quality of drinking water a sample of water is inoculated on a Petri-dish containing an agar. This dish is then placed inside an incubator to promote bacterial growth. The number of bacterial-colony clusters growing inside the dish is an important indicator for the quality of the water. This type of microbiological procedure usually involves counting many small and connected circular colonies with a specific morphology [5]. Automated approaches use either traditional computer vision [6] or are based on deep learning [3]. Another example is in the field of precision agriculture. The state of crops needs to be monitored continuously and is an important indicator for predicting crop yield is the number of crops and the size of the crops. In previous research the use of deep learning was investigated for counting and localizing crops in images produced by a camera mounted on an Unmanned Aerial Vehicle (UAV). This paper shows several improvements over the original CentroidNet algorithm [4] and discusses additional results on other datasets as well as ablation studies on the backbones, the loss functions and on pretraining.

Deep neural networks have consistently been shown to produce state-of-the-art results for many complex image analysis tasks for which enough data is available. Due to the large variety in counting tasks, this data-driven approach is promising for getting good results. In deep learning a large set of annotated data is used to train a specific model. Nowadays mainly Convolutional Neural Networks (CNNs) are used for a multitude of image analysis tasks like classification, segmentation, object detection, instance segmentation, image data synthesis and resolution enhancement in hyperspectral images [7–12].

A typical method to count objects with a CNN is to train an object detection model and subsequently count the number of detected objects [13–15]. Because most object-detection neural

* Corresponding author.
E-mail address: klaas.dijkstra@nhlstenden.com (K. Dijkstra).

https://doi.org/10.1016/j.neucom.2020.10.075
0925-2312/

networks are designed to detect typical everyday objects, they might provide inferior results on counting tasks where small and connected objects are involved. An alternative method to count objects is to regard counting as a regression task. In this case the number of counted objects is directly estimated from images or crops of images [16–19]. This is mostly used in congested scenes when it is difficult to individually detect objects. An example of this approach is estimating the number of people in crowds [20–22]. Recent approaches combine object localization and object detection with counting [4,23].

When counting objects in an image without regarding their location there is a risk of unwanted count compensation. When this happens an underestimate of the count in one part of the image compensates the overestimation in another part of the image. To correctly validate counting results the location of the objects should also be taken into account. A suitable metric for detection and counting is the F1 score which is the harmonic mean between precision and recall that represents the optimal equilibrium between overestimating and underestimating the number of objects. This paper will focus on models for object detection and instance segmentation because these models can estimate the location and dimensions of the counted objects simultaneously. In this paper a new hybrid deep learning architecture called CentroidNetV2 is introduced.

1.1. Contributions and research questions

The original version of CentroidNet [4] is a Fully Convolutional Network (FCN) that is trained using a standard Mean Squared Error (MSE) loss function. A U-net backbone is used to regress a field of 2-d vectors. These vectors are trained to predict the location of the centroid of the nearest object. CentroidNet is independent of image size during training and during inference, because vectors only encode relative positions and are not scaled by the image size. A voting algorithm is used to produce the actual centroids of the objects. Although demonstrating state-of-the-art results, the original CentroidNet has some limitations: the size of the objects are not estimated and the standard MSE loss does not specifically penalize the segmentation and the voting mechanism incorporated in the algorithm. Finally CentroidNet was only evaluated using the U-net backbone.

In CentroidNetV2 several improvements are proposed, while still maintaining the deep-learning and computer-vision hybrid design and the majority voting mechanisms. Firstly, for each pixel, an additional 2-d vector is predicted which represents the relative location of the nearest border of the object with the nearest centroid. This border information is used to predict the delineation of objects. Therefore, in a computer vision context, CentroidNetV2 is regarded as a form of contour fitting [24] with properties similar to the generalized Hough transform [25]. CentroidNetV2 produces instance-segmentation masks by fitting a predefined geometric shape through the border points. In a deep learning context CentroidNetV2 is considered an instance segmentation model.

We compare CentroidNetV2 to other well-known state-of-the-art networks that have comparable complexity and goals. The ResNet backbones for Mask Recurrent Convolutional Neural Network (MRCNN) and CentroidNetV2 give them comparable complexity. A specific shared goal of CentroidNetV2 and You Only Look Once Version 3 (YOLOv3) is small-object detection. In this paper we aim to focus on the general applicability of the CentroidNetV2 architecture for detecting and segmenting small objects. Therefore we have chosen three datasets to cover a broader range of applications. This also means that we do not focus on solving any one specific application (for example, colony counting or crop detection). Furthermore, and because of this broader scope, we compare to well-known and general methods for object detection and segmentation. Furthermore, we provide a comparison between the properties of the original CentroidNet and CentroidNetV2.

In addition to the architectural changes several ablation studies are performed. The loss function is redesigned to contain two Euclidean loss terms and a cross entropy term. The loss terms are compared to the original MSE loss function. We aim to investigate the effect of several architectural choices. In principle any semantic segmentation network can serve as a backbone for CentroidNetV2. In the experiments U-net and DeepLabV3 with three backbones, ResNet50, ResNet101 and Xception, are evaluated as representative backbones. Finally, we also investigate if transfer learning improves the performance of CentroidNetV2.

This leads to the following research questions:

1. What is the performance of CentroidNetV2 for detecting and counting many small objects?
2. How does the performance of CentroidNetV2 compare to well known state-of-the-art models for object detection and instance segmentation?
3. What backbones and loss functions are most suitable for CentroidNetV2?
4. What is the effect of transfer learning on the performance of CentroidNetV2?

In this paper we generally refer to a 1-d structure as a vector, a 2-d structure as a matrix and a 3-d structure as a tensor. A matrix that contains vectors is referred to as a tensor where the name indicates the type of vectors. For example: the target-centroid-vectors tensor is a matrix containing target-centroid vectors.

The remainder of this paper is structured as follows. In Section 3 the formal design of CentroidNetV2 is discussed. Section 4 explains the contents of the aerial-crops, cell-nuclei and bacterial-colonies datasets that are used for this research. The method of training and validation is discussed in Section 5 and the results are presented in Section 6. Finally, in Section 7 the conclusion and future work are discussed.

2. Related work

CNNs [26–28] are applied to an increasing number of complex image analysis tasks. One of the first break-through applications of CNNs was the classification of images from the ImageNet challenge [7]. Classification models take an image as an input and produce a single prediction for the whole image. Image segmentation using a CNN is performed by classifying each pixel into a one-hot vector representing the class of that pixel. This results in a dense segmentation mask of the entire image. Impressive performance was achieved by U-net on biomedical image data [8] and by DeepLabV3+ on segmenting everyday scenes [29]. A sparser detection is achieved by object detection CNNs like YOLOv3 [30] and RetinaNet [31]. These architectures directly estimate the bounding box and class of individual object instances in images with everyday objects. YOLOv3 focuses specifically on small-object detection.

Instance segmentation can be regarded as a combination of object detection and segmentation. MRCNN is a widely used state-of-the-art instance segmentation architecture that uses the detected boxes, called region proposals, to predict a dense segmentation mask of individual object instances [10] and this requires a two-stage training process. A Recurrent Neural Network (RNN) for instance segmentation is proposed [32], where recurrent attention is used to alternate between producing bounding boxes and producing segmentation masks for the objects within these boxes.

A proposal-free instance segmentation network is proposed [33], where segments and boxes are directly regressed and a

traditional off-the-shelf clustering method is used to create individual instances. This approach, where deep learning and traditional deterministic algorithms are combined, belongs to an emerging class of hybrid algorithms. In DCAN individual dense-object instances are produced by post processing the segmentation result using the probability map to estimate segment boundaries [34]. In a similar fashion a deterministic temporal consistency algorithm is combined with a CNN to segment RGB + depth videos [35]. InstanceCut produces object instances by deterministically combining two output modalities of the CNN: a semantic segmentation mask and an instance boundary. An alternative method to estimate boundaries is proposed by the deep watershed transform, which is a deep-learning based instance segmentation method inspired by a traditional watershed transform [36].

Other instance segmentation methods directly estimate decodable shape representations. In the straight-to-shapes approach the embeddings produced by a CNN are decoded into shapes with various methods to produce delineations of instances [37]. The star-convex polygon method uses radial distances to encode object instances with a CNN [38].

Related to this are methods that predict the centroids of individual object instances. These are proposed in [39] and in CentroidNet [4]. Both of these methods use a traditional Hough-transform inspired algorithm for determining centroids after model inference. The former method uses the bounding boxes to predict dense segments and the latter uses a fixed-size bounding box and uses binning to produce sharper centroids. CentroidNet has shown to produced state-of-the-art performance on a dataset for counting potato crops. In that approach dense spatial-voting vectors are predicted using a CNN and a majority voting algorithm combined with a non-max-suppression is subsequently used to produce centroid locations.

Conceptually, the integration of machine vision and deep learning can be viewed as embedding and exploiting prior knowledge in the algorithm. For example, in CentroidNet, partially occluded and connected objects still produce votes because patches of the objects are assumed, by the algorithm, to have information about the location of the centroid. For example, the leaves of a plant and the grain of these leaves naturally point outward. This means that implicit information about the location of the centroid of a plant is contained in a small patch of the image. This way of prior-knowledge embedding has been demonstrated to outperform non-hybrid approaches [4].

3. The CentroidNetV2 architecture

The main architecture of CentroidNetV2 is shown in Fig. 1. The top part of the graph shows the inference pipeline to predict instances and their corresponding class from input images. The bottom part shows the pipeline for converting the annotations to a suitable CentroidNet format for training. An image tensor **X** serves as an input to the model indicated by $\mathtt{f}(\cdot)$, which in turn predicts an output tensor **Y** containing the centroid vectors, border vectors and class logits (the score for each class). This tensor is then decoded into instance ids, class ids and class probabilities by the decoding function $\mathtt{g}(\cdot)$. The ground-truth tensor **Z** contains class ids and instance ids and is encoded into centroid vectors, border vectors and class logits. This is done by the inverse transform of $\mathtt{g}(\cdot)$, indicated by $\mathtt{g}'(\cdot)$.[1] Additionally the loss function $\mathtt{l}(\cdot,\cdot)$ calculates a loss between the output tensor **Y** and the target tensor **T**.

[1] The function $\mathtt{g}'(\cdot)$ is the inverse transform of $\mathtt{g}(\cdot)$ if the class probability is disregarded.

For convenience and without loss of generality the functions in this section are defined using 3-d image-like tensors. However the actual implementation uses mini batches of 3-d tensors. The three main functions $\mathtt{f}(\cdot), \mathtt{g}(\cdot)$ and $\mathtt{l}(\cdot,\cdot)$ are explained in sub-Section 3.1, 3.2 and 3.3 respectively.

3.1. Backbones

Function $\mathtt{f}(\cdot)$ in Fig. 1 is the backbone of CentroidNetV2 and represents the trainable part. A multi-channel image serves as an input. In our experiments this is an Red Green Blue (RGB) image. The output tensor contains three separate types of predictions: each spatial position of the first two planes contains the y and x components of a relative vector that points to the nearest centroid of an object. Each spatial position of the next two planes contains the y and x components of the vectors pointing to the nearest border of the object with the nearest centroid. The final planes of the output tensor contain the logits for the semantic segmentation of all pixels. In this paper we only test binary classification which means that this logits output consists of two planes (foreground/background). The spatial dimensions of **X** and **Y** should be identical and any semantic segmentation network can serve as backbone $\mathtt{f}(\cdot)$. In our experiments the depth of the input **X** is 3 (RGB) and the depth of the output **Y** is 6 (a 2-d centroid vector, a 2-d border vector and 2 logits). This is mathematically expressed by

$$\mathbf{Y} = f(\mathbf{X}) \tag{1}$$

$$\mathbf{Y} = [\mathbf{Yc}|\mathbf{Yb}|\mathbf{Yl}], \tag{2}$$

where **X** is the input tensor of the model, **Y** is the output tensor with stacked tensors containing the centroid-vectors tensor **Yc**, border-vectors tensor **Yb** and the logits tensor **Yl**.

Additionally the probabilities per logit are determined by dividing each logit by the sum of all logits for that pixel:

$$\mathbf{Yp}_{z,y,x} = \frac{\mathbf{Yl}_{z,y,x}}{\sum_{z\in[c]} \mathbf{Yl}_{z,y,x}}, \tag{3}$$

where **Yp** contains the class probabilities and c is the number of classes.

Some example centroid vectors and border vectors in **Yc** and **Yb** are geometrically shown in Fig. 2, where p_i, c_i and b_i represent the pixel coordinate, the vector of the nearest centroid and its nearest border, with three example pixels: $i \in \{1,2,3\}$. An important detail about border vectors is that for some coordinates, like p_1, the nearest border coordinate of the object with the nearest centroid is different from the nearest border coordinate. The nearest centroid to p_1 is of object B, but the nearest border coordinate of p_1 is of object A. In this case b_1 is the correct border vector (which is not equal to b_1').

3.2. Loss functions

Function $\mathtt{l}(\cdot,\cdot)$ in Fig. 1 calculates the loss between the output tensor **Y** and the target tensor **T**. Depending on the loss function we use the logits output **Yl** or the probability output **Yp**. The target tensors are defined in a similar way as Eq. (2):

$$\mathbf{T} = [\mathbf{Tc}|\mathbf{Tb}|\mathbf{Tp}], \tag{4}$$

where **T** consists of the stacked tensors with target-centroid-vectors tensor **Tc**, target-border-vectors tensor **Tb** and the target-probability tensor **Tp** containing n planes. Note that the target probability for a certain class is always 0 or 1.

3.2.1. MSE loss

The original CentroidNet used the mean squared error (MSE) loss defined as:

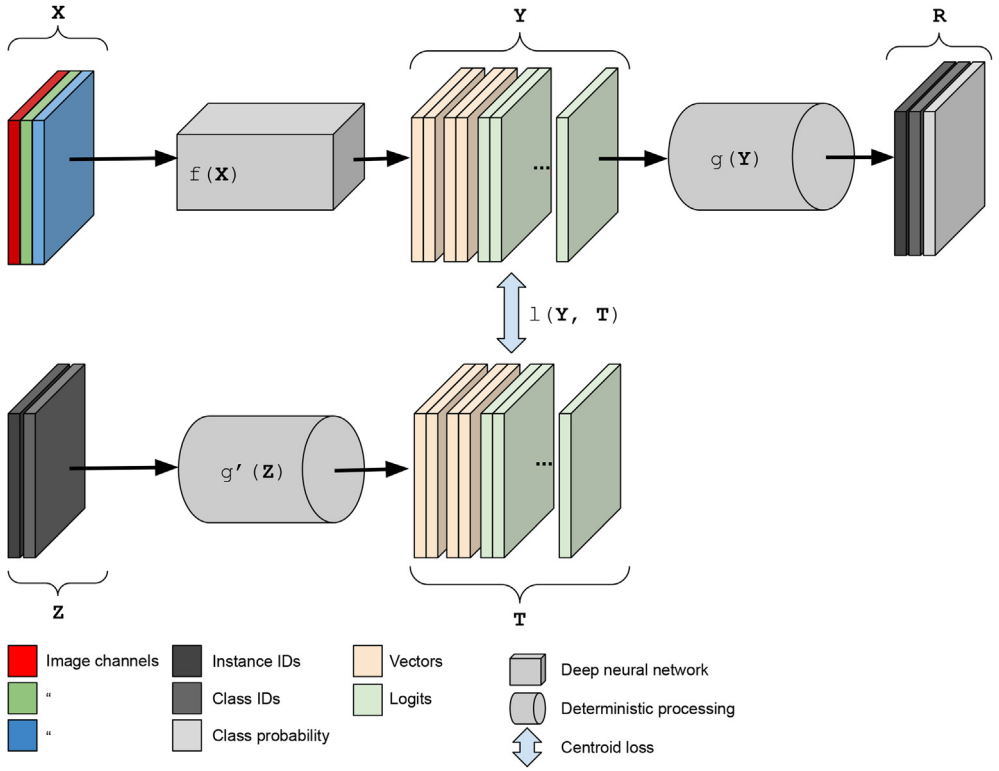

Fig. 1. The CentroidNetV2 architecture. The top part shows the inference pipeline and the bottom part shows the pipeline for encoding the ground-truth annotations. The encoder function $g'(\cdot)$ is the inverse transform of decoder function $g(\cdot)$.

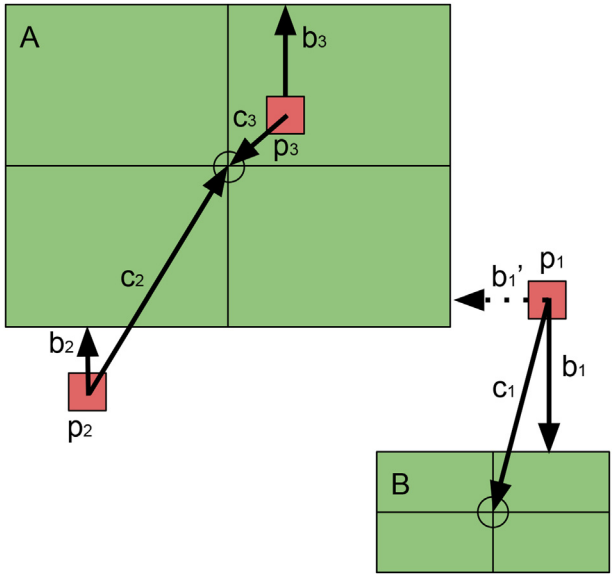

Fig. 2. Examples of centroid vectors (c1, c2, and c3) pointing from the pixel coordinates (p1, p2 and p3) to the nearest centroids of object A and B. The border vectors (b1, b2 and b3) point to the nearest border of the objects with the nearest centroid.

$$l_{\text{mse}}(\mathbf{Y},\mathbf{T}) = \frac{1}{c \cdot h \cdot w} \sum_{z\in[c]} \sum_{y\in[h]} \sum_{x\in[w]} \left(\mathbf{Y}_{z,y,x} - \mathbf{T}_{z,y,x}\right)^2 \tag{5}$$

where **Y** and **T** are the output and target tensor with a size of $c \times h \times w$. In our experiments the output tensor consists of five planes and consequently z runs over 1 through 5.

A limitation of using the MSE loss is the fact that it ignores the meaning of the specific planes in the output tensor **Y** and target tensor **T**. For example, the first two planes contain the y and x component of the centroid-voting vectors. For these two planes it makes more sense to use an Euclidean distance as the loss function, while the cross-entropy loss is more useful for the planes that contain the classification logits per pixel. Therefore, in CentroidNetV2, the loss function is decomposed into two different terms: vector loss and segmentation loss. These are discussed in the remaining part of this sub-section.

3.2.2. Vector loss

The Euclidean distance loss between the output-centroid vectors and target-centroid vectors or the output-border vectors and target-border vectors can be calculated by:

$$\begin{aligned} \mathbf{D}^2_{y,x} &= \sum_{z\in[c]} \left(\mathbf{Yv}_{z,y,x} - \mathbf{Tv}_{z,y,x}\right)^2 \\ l_{\text{d}}(\mathbf{Yv},\mathbf{Tv}) &= \tfrac{1}{h\cdot w} \sum_{y\in[h]} \sum_{x\in[w]} \mathbf{D}_{y,x}, \end{aligned} \tag{6}$$

where **Yv** and **Tv** have size $c \times h \times w$ and represent the output- and target-vectors tensors. Because both the centroid vectors and border vectors are two dimensional, each vector has two components ($c = 2$). The size of the spatial dimensions h and w are the same as the dimensions of input image.

The vector loss is calculated separately for the centroid vectors and the border vectors using Eq. (6) and then the sum is calculated.

$$l_{\text{vl}}(\mathbf{Yc},\mathbf{Yb},\mathbf{Tc},\mathbf{Tb}) = l_{\text{d}}(\mathbf{Yc},\mathbf{Tc}) + l_{\text{d}}(\mathbf{Yb},\mathbf{Tb}), \tag{7}$$

where **Yc**, **Yb**, **Tc**, **Tb** contain the output-centroid vectors, output-border vectors, target-centroid vectors and target-border vectors respectively.

3.2.3. Segmentation loss

The second term, the per-pixel classification loss or segmentation loss, can be calculated in two ways. The cross entropy loss is defined as:

$$\begin{aligned} \mathbf{CE}_{y,x} &= -\sum_{z\in[c]}(\mathbf{Tp}_{z,y,x}\log(\mathbf{Yp}_{z,y,x})) \\ \mathtt{l}_{\mathtt{ce}}(\mathbf{Yp},\mathbf{Tp}) &= \tfrac{1}{h\cdot w}\sum_{y\in[h]}\sum_{x\in[w]}\mathbf{CE}_{y,x}, \end{aligned} \tag{8}$$

where **Yp**, **Tp** are the output-probability tensor and the target-probability tensor (with values of either one or zero), c is the number of classes and h and w are the spatial dimensions of the respective tensors.

The Intersection over Union (IoU) loss is defined as 1 minus the intersection divided by the union of the class probabilities. IoU loss has been shown to outperform the cross-entropy loss in [40,41] and is defined by:

$$\begin{aligned} \mathbf{I}_z &= \sum_{y\in[h]}\sum_{x\in[w]}\mathbf{Yp}_{z,y,x}\times\mathbf{Tp}_{z,y,x} \\ \mathbf{U}_z &= \sum_{y\in[h]}\sum_{x\in[w]}(\mathbf{Yp}_{z,y,x}+\mathbf{Tp}_{z,y,x})-(\mathbf{Yp}_{z,y,x}\times\mathbf{Tp}_{z,y,x}) \\ \mathtt{l}_{\mathtt{iou}}(\mathbf{Yp},\mathbf{Tp}) &= 1-\tfrac{1}{c}\sum_{z\in[c]}\tfrac{\mathbf{I}_z}{\mathbf{U}_z} \end{aligned} \tag{9}$$

3.2.4. CentroidNetV2 loss

The individual terms of the loss functions are tested and their performance is reported in the results section of this paper. Eq. (10) combines vector loss and the cross entropy loss and Eq. (11) combines the vector loss and the IoU loss.

$$\mathtt{l}_{\mathtt{vl.ce}}(\mathbf{Y},\mathbf{T}) = \mathtt{l}_{\mathtt{vl}}(\mathbf{Yc},\mathbf{Yb},\mathbf{Tc},\mathbf{Tb}) + \mathtt{l}_{\mathtt{ce}}(\mathbf{Yp},\mathbf{Tp}) \tag{10}$$

$$\mathtt{l}_{\mathtt{vl.iou}}(\mathbf{Y},\mathbf{T}) = \mathtt{l}_{\mathtt{vl}}(\mathbf{Yc},\mathbf{Yb},\mathbf{Tc},\mathbf{Tb}) + \mathtt{l}_{\mathtt{iou}}(\mathbf{Yp},\mathbf{Tp}), \tag{11}$$

where **Y** is the output tensor containing output-centroid-vectors tensor **Yc**, output-border-vectors tensor **Yb** and output-probabilities tensor **Yp**, similarly **T** is the target tensor containing target-centroid-vectors tensor **Tc**, target-border-vectors tensor **Tb** and target-probabilities tensor **Tp**.

3.3. Coders

This sub-section discusses the decoder function $\mathtt{g}(\cdot)$ and the encoder function $\mathtt{g}'(\cdot)$ of Fig. 1. These functions represent the deterministic parts of CentroidNetV2. During inference the decoder calculates the output tensor **R** from the output **Y** of the model. The decoder is responsible for decoding centroid vectors, border vectors and logits into instance ids, class ids and their probabilities. The encoder generates the target tensor **T** given the annotations. This can be regarded as preprocessing the ground truth. The encoder is responsible for encoding instance ids and class ids into centroid vectors, border vectors and class logits.

3.3.1. Decoder

Individual object instances are calculated from the output of the model using the centroid-vectors tensor **Yc**, border-vectors tensor **Yb** and class-probabilities tensor **Yp**, defined in Eqs. (1)–(3).

Algorithm 1. Calculate the voting matrix given the output-voting-vectors tensor

```
function vote Yv
    h, w ← height, width of Yv
    V ← zero-filled matrix of size (h, w)
    for y ← 1 to h do
        for x ← 1 to w do
            y' ← y + Yv_{1,y,x}        ▷ Get the absolute y component
            x' ← x + Yv_{2,y,x}        ▷ Get the absolute x component
            V_{y',x'} ← V_{y',x'} + 1  ▷ Aggregate votes
        end for
    end for
    return V
end function
```

Algorithm 2. Calculate the border coordinates of an instance with respect to a given centroid.

```
function border(y_c, x_c), Yb, Yc
    B = {}
    h, w ← height, width of Yc          ▷ Get spatial dimensions of the input
    for y ← 1 to h do
        for x ← 1 to w do
            y'_c ← y + Yc_{y,x,1}       ▷ Get absolute centroid vector y
            x'_c ← x + Yc_{y,x,2}       ▷ Get absolute centroid vector x
            if (y'_c, x'_c) == (y_c, x_c)   ▷ Contributed to centroid (y_c, x_c) then
                y'_b ← y + Yb_{y,x,1}   ▷ Get absolute border vector y
                x'_b ← x + Yb_{y,x,2}   ▷ Get absolute border vector x
                B ← B ∪ {(y'_b, x'_b)}  ▷ Add border coordinate
            end if
        end for
    end for
    return B                            ▷ Border coordinates of object with centroid (y_c, x_c)
end function
```

Initially the $\mathtt{vote}(\cdot)$ function in Algorithm 1 calculates the voting matrix. An output-voting-vectors tensor Yv serves as an input (this can either be Yc or Yb). This tensor contains the relative 2-d centroid vectors for every spatial location of the corresponding input image. The absolute vectors y' and x' are calculated by adding the image coordinate y and x to each vector component. In the voting map the votes, represented by these absolute[2] voting vectors, are summed.

The decoder then selects centroid locations which received a high number of votes. The key idea of CentroidNetV2 is that the image locations which provided the vectors for these selected cen-

[2] In this context the term 'absolute' refers to the fact that all vectors are recalculated so that they have a common origin at the top-left of the image. It does not refer to the absolute value of the vector elements.

troids might be in high-information areas in the image. The hypothesis is that these high-information locations also provide a good estimate for the border location.

In Algorithm 2 these border coordinates are calculated. A centroid coordinate (y_c, x_c), the border-vectors tensor and centroid-vectors tensor serve as inputs to the algorithm. Using nested for-loops the image locations which contributed to centroid y_c, x_c are calculated (Line 8) and subsequently the absolute border coordinate is added to a set of border coordinates $\mathcal{B}$ for that centroid (Line 11).

These border coordinates can be quite noisy, therefore a geometric shape is fitted through this set of border coordinates. This allows additional prior knowledge about the shape of the objects to be embedded in the algorithm. For example: if the goal is to look for elliptical objects, an ellipse is fitted through the border coordinates. By fitting a convex-hull, arbitrary convex shapes can be accommodated by CentroidNetV2.

Finally, the class ids and probabilities of each spatial coordinate are calculated from the logits layers of the model output. The class of an object instance is determined by determining the highest class probability at the location of the centroid of that object.

The decoder is more formally defined in the following steps. The intermediate images that support the explanation of the decoder are shown in Fig. 3.

1. Calculate the centroid-vote matrix $V = \texttt{vote}(Yc)$, where Yc is a tensor containing the centroid vectors predicted by the model and $\texttt{vote}(\cdot)$ is the voting function defined in Algorithm 1.
2. Calculate the suppressed-voting matrix $\hat{V} = \psi(V)$. Function $\psi(\cdot)$ only keeps maximum values in a local window and is given by:

$$\psi(V_{y,x}) = V_{y,x} \times \begin{cases} 1, & \text{if} \quad V_{y,x} == \max\limits_{(v,u)\in[0..n)\times[0..m)} V_{y-v,x-u} \\ 0, & \text{otherwise}, \end{cases}$$

   where y and x are spatial coordinates, and v and u are coordinates inside an $n \times m$ window of the non-max suppression. In our case plateaus of equal maxima are reduced to single points.
3. Select voting peaks by applying a threshold θ to the suppressed-voting matrix $\hat{V}$ to generate the set of selected votes $\mathcal{C}$ given by:

$$\mathcal{C} = \left\{(y_c, x_c) \in [h] \times [w] | \hat{V}_{y_c,x_c} \geqslant \theta\right\}, \tag{12}$$

   where y_c, x_c are the peak coordinates and h and w are the dimensions of matrix $\hat{V}$.
4. Select the set of border coordinates corresponding to a centroid. The set of border coordinates for a centroid at coordinate $(y_c, x_c) \in \mathcal{C}$ is given by:

$$\mathcal{B} = \texttt{border}((y_c, x_c) \in \mathcal{C}, Yb, Yc),$$

   where the function $\texttt{border}(\cdot,\cdot,\cdot)$ calculates border coordinates for a given centroid at (y_c, x_c) and is given by Algorithm 2, Yb and Yc are tensors containing the border and centroid vectors respectively.
5. Fit a geometric shape (e.g. circle, ellipse, convex hull, etc.) through the set of border coordinates $\mathcal{B}$ for a given centroid and draw the geometric shape with a unique value in the instance-ids matrix I.
6. Calculate the class-ids matrix and probabilities matrix C and P respectively by taking the $\texttt{argmax}(\cdot)$ and $\texttt{max}(\cdot)$ over class probabilities:

$$C_{y,x} = \texttt{argmax}_{z\in[c]}\left(\texttt{Yp}_{z,y,x}\right)$$
$$P_{y,x} = \texttt{max}_{z\in[c]}\left(\texttt{Yp}_{z,y,x}\right),$$

   where c is the number of logits in the output-probabilities tensor Yp. When the probability of an element in matrix P is above ϕ, it is accepted in the corresponding class-id matrix C, otherwise the element is assigned to the background. In our experiments a probability threshold of 0.2 gave the best results. The class of an instance with centroid y, x is defined by the value of $C_{y,x}$.
7. Guarantee that for each element in instance-ids matrix I and class-ids matrix C, both the instance id and the class id are known. This means that if either the instance id or the class id is background for a certain element, both the instance id and class id for that element are set to background. Because masking is performed per pixel, the final shape of object instances can be different from the fitted shape.

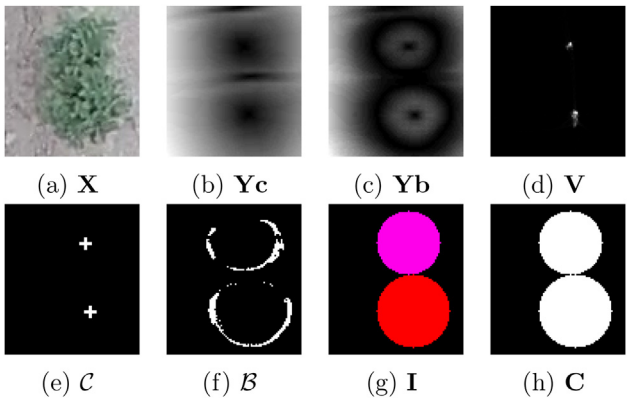

Fig. 3. An example of the data of the decoder represented by images of potato plants annotated with circles. From left to right and top to bottom: input image X, magnitudes of the centroid vectors Yc, magnitudes of border vectors Yb, accumulated centroid votes V, set of centroid coordinates $\mathcal{C}$, set of border coordinates $\mathcal{B}$, color-coded instances I and per-pixel class ids C.

The instance-ids matrix I, class-ids matrix C and class-probabilities matrix P are the final outputs of CentroidNetV2.

3.3.2. Encoder

Encoding is a preprocessing step needed to convert the ground-truth annotations to a format that can be used to train the model. An annotation of an input image X consists of the target-class-ids matrix C' in which each element represents the class of a pixel in the input image, and the target-instance-ids matrix I' in which each element represents the id of an individual object instance in the input image. The encoder can be regarded as the inverse of the decoder and therefore the input matrices are named the same as the output matrices of the decoder, but are denoted by an additional apostrophe ($\prime$). The output of the encoder is the target-centroid-vectors tensor Tc, the target-border-vectors tensor Tb and the target-probabilities matrix Tp defined in Eq. (4).

The encoding process is defined in the remaining part of this section and the intermediate images to support the explanation are shown in Fig. 4. The black border around the objects in Tc and Tb is caused by the clipping of voting vectors. This is set to roughly twice the average radius of the target objects.

All unique instance ids in matrix I' are represented by the set $\mathcal{I}$. A set of coordinates of an instance with id i is and given by:

$$\mathcal{O}'_i = \left\{(y_o, x_o) | I'_{y_o,x_o} == i \in \mathcal{I}\right\},$$

where y_o and x_o represent the coordinates within the instance-ids matrix I'.

The set of centroids for all objects are calculated by taking the average y and x coordinate of each set of coordinates:

$$\mathcal{C}' = \left\{\overline{\mathcal{O}'_1}, \overline{\mathcal{O}'_2}, \ldots, \overline{\mathcal{O}'_n}\right\},$$

(a) $\mathbf{X}$ (b) $\mathbf{I}'$ (c) $\mathbf{C}'$ (d) $\mathbf{Tc}$ (e) $\mathbf{Tb}$

Fig. 4. Data of the encoder represented by images. From left to right: input image *X*, color-coded target-instance ids *I′*, target-class ids *C′*, magnitudes of the target-centroid-vectors *Tc* and target-border-vectors *Tb*. Voting vectors with high magnitudes are bright white and voting vectors with low magnitudes appear darker.

where $\mathcal{C}'$ is the set of target centroids of the object instances, $\overline{\mathcal{O}_i}$ is the centroid of the spatial coordinates that belong to instance with id *i*.

Subsequently the tensor with target-centroid vectors *Tc* is calculated by taking the difference between a spatial coordinate of *Tc* and the coordinate of the nearest centroid:

$$Tc_{y,x} = \mathtt{argmin}_{(\mathtt{y_c},\mathtt{x_c})\in\mathcal{C}'}||(\mathtt{y_c},\mathtt{x_c}) - (\mathtt{y},\mathtt{x})|| - (\mathtt{y},\mathtt{x}),$$

where (y,x) is a spatial coordinate of the target-centroid-vectors tensor Tc, (y_c,x_c) are the centroid coordinates from the set of centroids $\mathcal{C}'$. Note that *Tc* is a 3-d tensor where the third dimension has size two and contains the relative vectors $((y_c,x_c)\in\mathcal{C}') - (y,x)$ pointing to the nearest centroid. Also note that the `argmin` function returns a vector $(y_c,x_c)\in\mathcal{C}'$.

The set of border coordinates for a certain instance *i* is given by $\mathcal{B}'_i$. The target-border-vectors tensor is then calculated as follows:

$$Tb_{y,x} = \mathtt{argmin}_{(\mathtt{y_b},\mathtt{x_b})\in\mathcal{B}'_i}||(\mathtt{y_b},\mathtt{x_b}) - (\mathtt{y},\mathtt{x})|| - (\mathtt{y},\mathtt{x}),$$

where y,x are the spatial coordinates of the target-border-vectors tensor *Tb* and (y_b,x_b) are border coordinates. *Tb* contains the relative vectors $((y_b,x_b)\in\mathcal{B}'_i) - (y,x)$ pointing to the nearest border of the object instance with the nearest centroid.

Finally, the target-probabilities matrix is given by:

$$Tp_{c,y,x} = \mathbb{1}\left(C'_{y,x} == c\right),$$

where *Tp* contains target logits, C' is the target-class-ids matrix, y and x are the spatial coordinates and c is the target-class id. The indicator function $\mathbb{1}(\cdot)$ returns one if the condition is true and zero otherwise.

The target-centroid-vectors tensor *Tc*, target-border-vectors tensor *Tb* and target-probabilities matrix *Tp* are the outputs of the decoder and are used as a target to train the model.

4. Datasets

In this research three datasets are used to test CentroidNetV2 and compare it to the other well-known models. These datasets are discussed in this section.

4.1. Aerial crops

The aerial-crops dataset contains images of potato crops taken with a low-cost drone which navigated over a potato field [4]. It consists of 10 frames randomly sampled from a 24 fps video shot at 10 meters altitude. The dataset contains a mix of small, connected and distinct potato plants as well as background soil. The resolution of each image is 1500 × 1800 pixels. The set contains over 3000 annotated plants using circles to indicate the location of the plants. See Fig. 5 for some examples. A 50%/50% training/validation split of the dataset is used for validation.

This set is used to compare the individual models on a relatively small amount of images, but a large amount of small objects per image. This has proven to be a good dataset for investigating how well the various networks handle a mix of small and large objects as well as high connectedness between objects. For CentroidNetV2 circles are fitted through the border coordinates to produce instances. For YOLOv3 and MRCNN the circles are calculated from the predicted bounding boxes.

4.2. Cell nuclei

The cell-nuclei dataset was used for the Kaggle data science bowl 2018.[3] It contains annotated samples of cell-nuclei images taken with a microscope. This dataset consists of 673 images and has a total of 29,461 annotated nuclei. The images vary in resolution, cell type, magnification and imaging modality. The annotations are per-pixel masks indicating the individual instances of each cell nucleus. See Fig. 6 for some examples. A 80%/20% training/validation split of the dataset is used for validation.

This dataset is used to investigate how the models perform on complex data with much variation. Also the dataset is ideal for investigating how varying image resolutions are handled. For CentroidNetV2 rotated ellipses are fitted through the predicted border coordinates to produce instances. MRCNN predicts instances directly as masks. YOLOv3 has not been tested on this set because it is not able to produce instances of arbitrary shapes or rotated ellipses.

4.3. Bacterial colonies

The bacterial-colonies dataset contains images of Petri dishes with bacterial growth from water samples. In this study Legionella colonies which were cultivated on Buffered Charcoal Yeast Agar were used [4]. The dataset has been created by a water company in the Netherlands. A domain expert annotated colonies which have a typical morphology for Legionella. Additional tests were used to confirm that the colonies are Legionella species. The dataset consists of 79 images with a total of 2541 annotated bacterial colonies. The images have a resolution of 1024 × 1024 pixels. See Fig. 7 for some examples. A 80%/20% training/validation split of the dataset is used for validation.

This set is used to test the ability of the models to detect multiple connected objects with various sizes and to not detect colonies which are not Legionella suspected (the yellow colonies). An image of a dish typically contains many colonies which makes this a good dataset for testing approaches to count many-small objects. The most important reason to test various approaches on this set is because colony counting is a real practical example of a counting task which has not been sufficiently solved and, to date, still requires manual labor.

4.4. Tiling

All of the datasets described in this section contain images which are either too large or contain images of various sizes. This means they cannot be used directly for training because a mini batch should consist of multiple equally sized images. A common approach to handle this problem is to resize all images to some predefined size. However, this would not achieve the desired result because small objects could be removed by this action. We employ two strategies to handle this problem. For CentroidNetV2 and MRCNN we randomly crop the image during training with 256 × 256 image crops.

[3] https://www.kaggle.com/c/data-science-bowl-2018.
[4] ISO 11731:2017, Water quality – Enumeration of Legionella

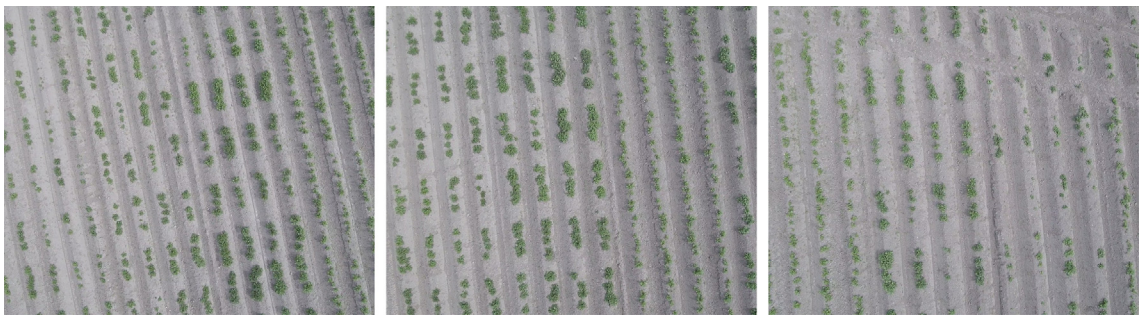

Fig. 5. Example images from the aerial-crops dataset. The images show variations in the size of the crops and high connectedness between individual crops.

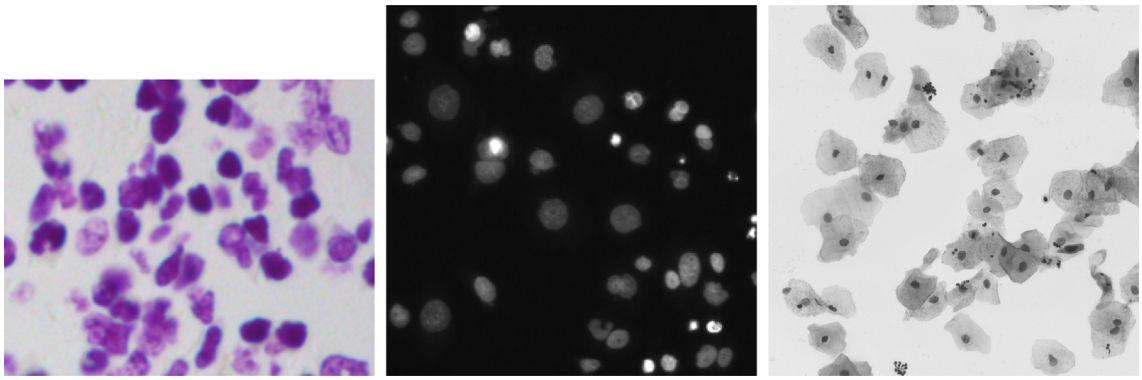

Fig. 6. Example images from the cell-nuclei dataset. The images show variations in resolution, cell type, magnification and imaging modality.

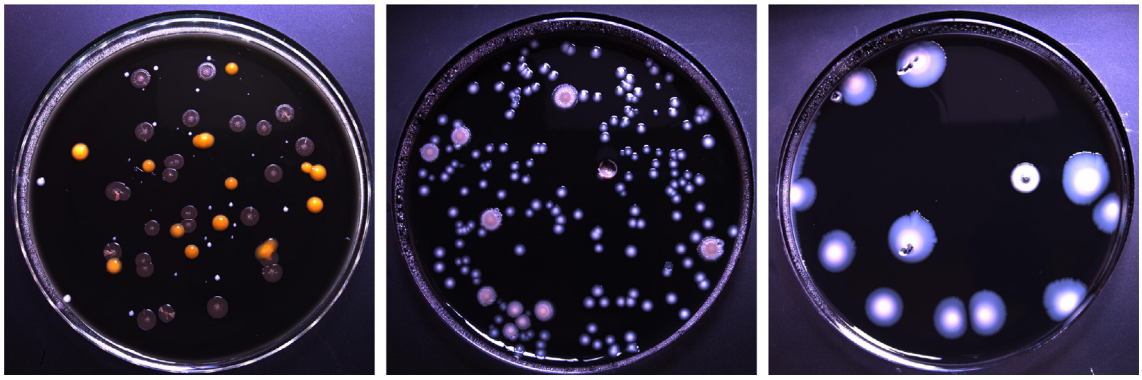

Fig. 7. Example images from the bacterial-colonies dataset. The images show variations in size, color and number of colonies per Petri dish.

The best performing YOLOv3 should be trained with images of 608 $\times$ 608 pixels as described in the original paper [30]. To be able to generate a dataset that can be used to train the original YOLOv3 in DarkNet, images are tiled with 50% overlap in both directions. This overlap is used to prevent clipped objects at the edges of the tiles. When recombining the results to get object locations in the original images, only objects at the center of each tile are kept. We observed that this approach works remarkably well for YOLOv3 because the tiles are much larger than the objects in the images. In Fig. 8 this tiling process is explained.

5. Training and validation

In this section the methods for training and validation of the various models are discussed. Each model is trained using a training set and validated using a disjoint validation set. The split is randomly determined.

5.1. Training

For CentroidNetV2 the input data is normalized using the theoretical range of the image data: subtracting 128 and dividing each pixel by 255. Typically the data is normalized using the statistics of the dataset or the statistics of the dataset that was used to pretrain the model. However, in practice we did not observe any significant loss in performance when using fixed normalization coefficients for all datasets. Furthermore, Adam is used to train CentroidNetV2 with a learning rate of 0.001 and a momentum of 0.9. To avoid overfitting and to select the best model during training, early stopping was applied [28]. In each experiment it was observed that the trained model did not improve significantly after 500 epochs.

MRCNN and YOLOv3 apply various methods to optimize performance (augmentation, optimizers, normalization, etc.) The maximum amount of instances that MRCNN can produce has been

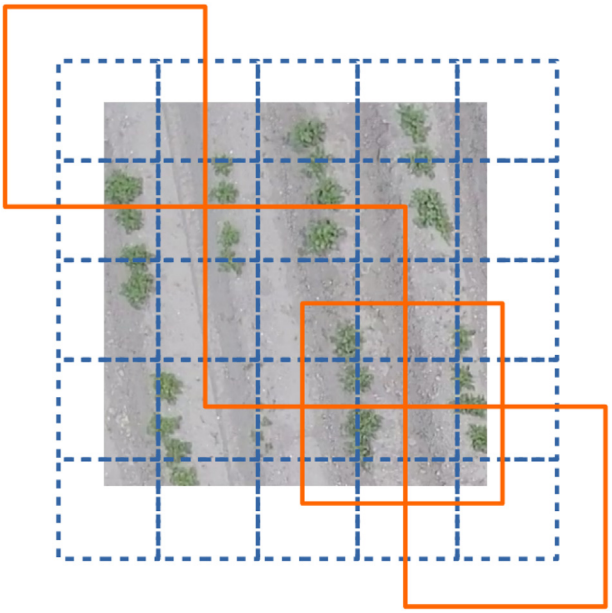

Fig. 8. Tiling process. The large rectangles (orange) represent four examples of the actual tiles used for training. The smaller dashed tiles (blue) at the center of each large tile represent the areas in which objects are kept during the recombination of the instances that have been predicted within the tiles.

increased to 2048 to accommodate the many objects found in the aerial-crops dataset. Random resizing of input images has been disabled for all networks because it does not seem appropriate for counting many-small objects as it might result in the removal of object details or remove small objects altogether.

5.2. Validation

Validation is done using a number of metrics for instance segmentation and counting. Most important is the F1 score which gives the equilibrium between overestimating and underestimating instance counts. For further analysis the precision and recall are used. The true positives, false positives, false negatives and counting results give an indication of the number of objects that have been either correctly or incorrectly detected.

The validation of each method is based on the ability of the model to provide instances at the correct locations. The output-instances matrix I of a model and the target-instances matrix I' are compared. Each element of these matrices contains a value indicating the instance id that pixel belongs to. The apostrophe ($\prime$) indicates that the symbol contains data from the ground truth. If a model gives a perfect output the symbols with and without an apostrophe are identical. The two sets of image coordinates representing the object instances are defined by:

$$\mathcal{O}_a = \{(y,x) \in [h,w] | I_{y,x} == a\}$$
$$\mathcal{O}'_b = \left\{(y',x') \in [h,w] | I'_{y',x'} == b\right\},$$

where $\mathcal{O}_a$ is the set of output-object coordinates for object instance a, $\mathcal{O}'_b$ is the set of target-object coordinates for instance b, the height and width are indicated by h and w, the spatial coordinates are indicated by y, x, y' and x'.

Result instances are matched against target instances based on their respective overlap. The overlap between two objects is defined by the IoU which is used to calculate a normalized output between zero and one, where one means a perfect match and zero means no match. IoU is defined by:

$$\texttt{IoU}(\mathcal{O}, \mathcal{O}') = \frac{\mathcal{O} \cap \mathcal{O}'}{\mathcal{O} \cup \mathcal{O}'},$$

Matching of object instances is based on a certain minimum IoU threshold. The set of output-instance ids is given by $\mathcal{A} = [m]$ and the set of target-instance ids is given by $\mathcal{B} = [n]$, where m and n are the number of output instances and target instances respectively. A match between output-instance id b and all target-instance ids in $\mathcal{A}$ is given by:

$$\texttt{is\_match}(b \in \mathcal{B}) = \max_{a \in \mathcal{A}} \left(\texttt{IoU}(\mathcal{O}_a, \mathcal{O}'_b)\right) > \tau$$

where τ is the minimum IoU threshold, $\texttt{is\_match}(\cdot)$ returns true when a match is found. For counting tasks the IoU threshold can be set to a low value because the goal is to know if an object is roughly found in the correct location, therefore in our experiments we set $\tau = 0.1$.

If there is a match between an output-instance id a and a target-instance id b, the matching ids are removed from both the set of output ids $\mathcal{A}$ and from the set of target ids $\mathcal{B}$. The matching ids are then added to the set of matches by $\mathcal{M} = \mathcal{M} \cup \{(a,b)\}$. This process of matching and removing is repeated for all output-instance ids in $\mathcal{B}$. If all objects have a match, both $\mathcal{A}$ and $\mathcal{B}$ will be empty and $\mathcal{M}$ will contain all matching instance-id pairs, but in practice this is almost never the case. From the number of items in these sets the performance metrics are calculated:

$$TP = |\mathcal{M}| \tag{13}$$
$$FP = |\mathcal{A}| \tag{14}$$
$$FN = |\mathcal{B}| \tag{15}$$
$$P = \frac{TP}{TP + FP} \tag{16}$$
$$R = \frac{TP}{TP + FN} \tag{17}$$
$$F1 = 2 \times \frac{P \times R}{P + R} \tag{18}$$
$$Count = TP + FP, \tag{19}$$

where $TP, FP, TN, P, R, F1$ and $Count$ are the true positives, false positives, true negatives, precision, recall, F1 score and object count respectively. Theoretically these metrics can be calculated per individual object class and, in that case, the metrics usually have prefix mA, for 'mean Average', indicating the mean over classes and the average over all images. In the experiments discussed in this paper only two classes are used (background and foreground).

6. Experiments and results

In this section the results of the experiments are discussed. Each sub-section shows the performance of the various models, loss functions and backbones on each of the three datasets. The final part of this section discusses common failure cases of all approaches and also an analysis about the difference in performance between CentroidNetV1 and CentroidNetV2 is given.

In summary, CentroidNetV2 achieves the best experimental performance based on F1-score on the aerial crops dataset (94.7%) and on the bacterial colonies dataset (92.6%). MRCNN achieves the best F1 score on the cell nuclei dataset (92.3%). In general, better, or on-par results for the various metrics are obtained by our proposed algorithm. The remainder of this section gives a more thorough analysis of the experimental results using the proposed metrics.

Each table with results has the same basic structure. The model name, backbone name and loss function used is shown in the first three columns of the tables. The metrics given by Eqs. 13,19 are reported in the remaining columns. The cursive text in the rows

of each table indicate the category of the experiment and is used to group experiments in a logical manner.

Because the highest F1 score represents the best equilibrium between overestimating and underestimating the number of objects, the network threshold hyperparameter that determines the trade off between precision and recall is optimized on the training set by an exhaustive search. For CentroidNetV2 the integer voting threshold θ discussed in Eq. (12) is optimized, for MRCNN and YOLOv3 the confidence threshold is optimized. After the thresholds have been optimized the metrics are calculated on the validation set and reported in the respective tables.

The naming of the loss functions in this section follows the naming scheme introduced in Section 3. MSE loss is the standard loss defined by Eq. (5). The Vector Loss (VL) is computed by the Euclidean distance between the target-voting vectors and the output-voting vectors and is defined by Eq. (7). The Cross Entropy (CE) loss and IoU loss, defined in Eqs. 8 and 9, are calculated using the output logits and the target logits. Finally the combined losses used for the analysis in this section are MSE, VL-CE and VL-IoU, defined in Eqs. 5, 10, 11 respectively.

An open-source reference implementation of OpenCentroidNet written in Python, using PyTorch 1.0 [42] is published with this paper. A fully annotated dataset containing images of potato crops and a dataset containing annotated Legionella bacterial colonies are also published together with this paper.

6.1. Results on aerial crops

The results of the performance on the aerial-crops dataset for the different models are shown in Table 1.

The first part of Table 1 (*Comparing to the state-of-the-art*) shows the comparison between CentroidNetV2 and the other models. The overall best F1 score is achieved by CentroidNetV2 (94.7%). YOLOv3 achieves an F1 score of 94.3%. This shows that the tiling scheme used for YOLOv3 is quite optimal. MRCNN achieves an F1 score of 92.4%. Further analysis shows that MRCNN fails to detect small crops. This automatically results in the highest precision for MRCNN (97.7%) caused by the low amount of false positives (34 crops). When using MSE loss and a U-net backbone, a configuration similar to the original CentroidNet, a lower F1 score of 93.5% is achieved.

The visual differences between the individual models are shown in Fig. 9. CentroidNetV2 shows the most correctly detected crops in Fig. 9a. YOLOv3 seems to not find the right balance between the false positives and false negatives indicated by the false positive crop found in the left-bottom of Fig. 9b and the two missed small crops. Fig. 9c shows that MRCNN failed to detect two small potato-plant crop and also a misses a crop closely connected to a bigger crop (shown in the left bottom of Fig. 9c).

The second part of Table 1 (*Comparing loss functions*) shows that the MSE loss achieves the lowest F1 score (93.5%) compared to the other loss functions and using the same backbone.

The third part of Table 1 (*Comparing backbones*) shows the performance of the alternative backbones for CentroidNetV2. The extra 51 layers of the ResNet101 backbone only achieve a 0.1% higher F1 score compared to the ResNet50 backbone for CentroidNetV2. The Xception backbone achieves a 4.4% lower F1 score. Also the U-net backbone shows a lower F1 score (0.7% lower). From this can be concluded that the overall best backbone for CentroidNetV2 on the aerial-crops dataset is DeepLabV3 + _ResNet101.

6.2. Results on cell nuclei

The results of the performance on the cell-nuclei dataset for the different models are shown in Table 2. The first part of the table (*Usage of pretraining with ResNet101 backbone*) shows the performance when using the ResNet101 backbone with and without pretraining (indicated by the PT column). Also an experiment with the alternative VL-IoU loss function has been included here. The MRCNN model with a ResNet101 backbone pretrained on ImageNet achieves the highest F1 score (92.3%). The runner up is a pretrained CentroidNetV2 with a DeepLabV3 + _ResNet101 backbone (91.9% F1 score). Furthermore CentroidNetV2 shows the highest recall (89.9%) which indicates that CentroidNetV2 tends to detect more objects and achieves the lowest amount of false negatives (583 nuclei) at its highest F1 score.

In Fig. 10 an example of the instances produced by MRCNN and CentroidNetV2 is shown on a challenging image. It can be seen that MRCNN gives more accurate instance segmentation masks which explains the higher F1 score. The higher recall of CentroidNetV2 is explained by the fact that more small and low-contrast cell nuclei are predicted.

From the literature is it well known that pretraining improves the performance of models [43] and this is confirmed by the measured increase in F1 score for MRCNN. An interesting observation is that this also holds for CentroidNetV2 which achieves a 1.3% higher F1 score when using pretrained weights. This confirms that the regression of centroid- and border-voting vectors also benefits from a ResNet101 backbone pretrained on ImageNet and that pretrained convolutional filter weights are quite general in that they can be repurposed for predicting voting vectors. The only case where the pretrained backbone has a lower F1 score compared to the non-pretrained model is when a ResNet50 backbone is used with MRCNN. However, the pretrained version still achieves the highest precision (96.3%) at its highest F1 score. Interestingly the use of the VL-IoU with pretraining achieves the lowest F1 score (90.3%).

The third part of Table 2 (*Comparison to U-net backbone*) shows the performance of CentroidNetV2 using the original U-net backbone on the cell-nuclei dataset. That configuration is similar to the original version of CentroidNet, which used MSE loss and a U-net backbone, and has among the lowest F1 scores (90.6%). Using the VL-CE loss function in conjunction with the U-net backbone yields better results (91.1%). But still the conclusion holds that the best CentroidNetV2 configuration uses a ResNet101 backbone and the VL-CE loss function. CentroidNetV2 seems to have no obvious advantage when using the U-net backbone because the precision for CentroidNetV2 (93.3%) is lower compared to the original CentroidNet (94.3%). This means that the improvements of both the loss function and the backbone together yield a higher performance on all metrics (91.7% F1 score, 94.3% precision and 89.3% recall).

6.3. Results on bacterial colonies

On the bacterial-colonies dataset CentroidNetV2 achieves the overall highest F1 score of 92.6% shown in Table 3. YOLOv3 is the runner op with an F1 score of 92.3%. The U-net backbone of CentroidNetV2 struggles to get good results and achieves only an F1 score of 87.1%. This confirms the added value of the ResNet101 backbone on this dataset. Also in this case CentroidNetV2 achieves the highest recall (91.0%). MRCNN seems to miss objects and achieves the highest precision of 95.4% at the cost of lower recall (89.1%).

In Table 3 it is shown that the number of predicted objects in the image (indicated by the 'Count' column) is not representative for the actual number of correctly detected colonies. It seems that YOLOv3 only counts one less colony compared to CentroidNetV2 (885 and 886). However, when looking at the difference in the number of true positives (indicating colonies found at the right location) it can be seen that YOLOv3 actually misses three colonies (832 and 835). The two extra colonies in the 'Count' column are

Table 1
Results for counting crops with 1660 annotated validation samples. Performance of several configurations of CentroidNetV2 and comparison to YOLOv3 and MRCNN (in percentages). The best precision, recall and F1 score are boldface.

Model	Backbone	Loss	F1	P	R	TP	FP	FN	Count
Comparing to the state-of-the-art									
CentroidNetV2	DLV3-RN101	VL-CE	**94.7**	94.4	95.1	1578	94	82	1672
CentroidNet	U-net	MSE	93.5	92.2	94.8	1573	133	87	1706
YOLOv3	Default	Default	94.3	93.7	94.9	1575	106	85	1681
MRCNN	RN101	Default	92.4	**97.7**	87.7	1456	34	204	1490
Comparing loss functions									
CentroidNetV2	DLV3-RN101	MSE	93.5	92.5	94.6	1570	127	90	1697
CentroidNetV2	DLV3-RN101	VL-IoU	94.3	93.9	94.6	1571	102	89	1673
Comparing backbones									
CentroidNetV2	U-net	VL-CE	94.0	92.3	**95.7**	1588	132	72	1720
CentroidNetV2	DLV3-XC	VL-CE	90.3	86.6	94.3	1566	242	94	1808
CentroidNetV2	DLV3-RN50	VL-CE	94.6	94.7	94.5	1569	87	91	1656
MRCNN	RN50	Default	93.4	97.3	89.8	1491	41	169	1532

(a) CN V2 (b) YOLOv3 (c) MRCNN

Fig. 9. Red circles show the prediction of the three models and the annotations are shown in green. CentroidNetV2 detected most crops (one false negative), MRCNN has three false negatives and YOLOv3 produced a false positive and two false negatives.

Table 2
Results for counting nuclei with 5755 annotated validation samples. Performance of several configurations of CentroidNetV2 and MRCNN (in percentages). PT indicates if a model is pretrained. The best precision, recall and F1 score are boldface.

Model	Backbone	Loss	PT	F1	P	R	TP	FP	FN	Count
Usage of pretraining with ResNet101 backbone										
CentroidNetV2	DLV3-RN101	VL-CE	Yes	91.9	94.1	**89.9**	5172	323	583	5495
CentroidNetV2	DLV3-RN101	VL-CE	No	90.6	93.8	87.7	5048	335	707	5383
CentroidNetV2	DLV3-RN101	VL-IoU	Yes	90.3	94.0	86.8	4993	314	762	5307
MRCNN	RN101	Default	Yes	**92.3**	96.1	88.9	5116	210	639	5326
MRCNN	RN101	Default	No	91.5	95.3	87.9	5061	248	694	5309
Usage of pretraining with ResNet50 backbone										
CentroidNetV2	DLV3-RN50	VL-CE	Yes	91.7	94.3	89.3	5138	309	617	5447
CentroidNetV2	DLV3-RN50	VL-CE	No	91.4	94.1	88.8	5112	318	643	5430
MRCNN	RN50	Default	Yes	91.0	**96.3**	86.3	4966	193	789	5159
MRCNN	RN50	Default	No	91.5	95.1	88.1	5072	260	683	5332
Comparison to U-net backbone										
CentroidNetV2	U-net	VL-CE	No	91.1	93.3	88.9	5116	365	639	5481
CentroidNet	U-net	MSE	No	90.6	94.3	87.2	5021	304	734	5325

caused by the two extra false positives found elsewhere in the image. This is why we argue that for counting tasks the validation should be based on F1 score rather than raw object-detection count because it takes the location of the object into account.

The visual differences in performance between the models are shown in Fig. 11. The thick red circles indicate the predictions and the thin green circles indicate the annotations. In the top row a cropped part of an image with bacterial colonies is shown. Each model correctly ignores the yellow colony which is not Legionella. In Fig. 11b YOLOv3 incorrectly detects the large colony that has not been annotated as Legionella suspected. MRCNN fails to detect the small colony near the right bottom of Fig. 11c. The bottom row of Fig. 11 gives another interesting insight in the differences between the models. The large black-ish structure at the left of each image is an air bubble adjacent to a colony. Air bubble formation is a common problem for certain types of culturing media. However, this exact visual appearance is rare in the training set. In Fig. 11e it is shown that YOLOv3 fails to detect the colony, probably because it has not seen something similar before. Both CentroidNetV2 and MRCNN detect this colony correctly. For CentroidNetV2 this is probably because the partial bacterial colony still produces part of the votes (similar to when two colonies are overlapping).

In this section our focus has been to compare the F1 score, Precision, Recall and Count metrics of the various approaches and therefore did not include inference-time metrics. For an analysis of the inference time of MRCNN and YOLOv3 we would like to refer the reader to [44]. In that paper the authors provide an extensive

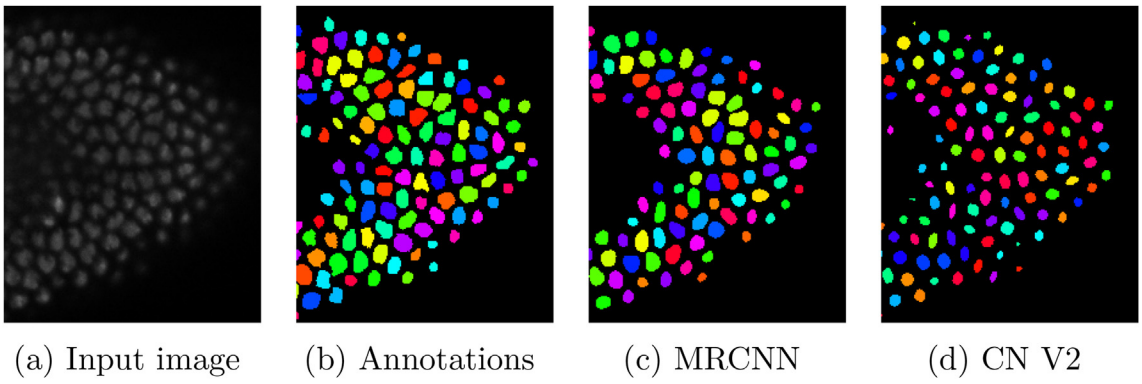

Fig. 10. Instance segmentation results on an image of the cell-nuclei dataset. The input image and ground truth are shown on the left and the predicted output of the models is shown on the right. MRCNN predicts more accurate segments. CentroidNetV2 detects small and low contrast objects that MRCNN fails to detect.

Table 3
Results for counting bacterial colonies with 918 annotated validation samples. Performance of CentroidNetV2 compared to YOLOv3 and MRCNN (in percentages). The best precision, recall and F1 score are boldface.

Model	Backbone	Loss	F1	P	R	TP	FP	FN	Count
CentroidNetV2	DLV3-RN101	VL-CE	**92.6**	94.2	**91.0**	835	51	83	886
YOLOv3	Default	Default	92.3	94.0	90.6	832	53	86	885
MRCNN	RN101	Default	92.2	**95.4**	89.1	818	39	100	857
CentroidNetV2	U-net	VL-CE	87.1	90.5	84.0	771	81	147	852

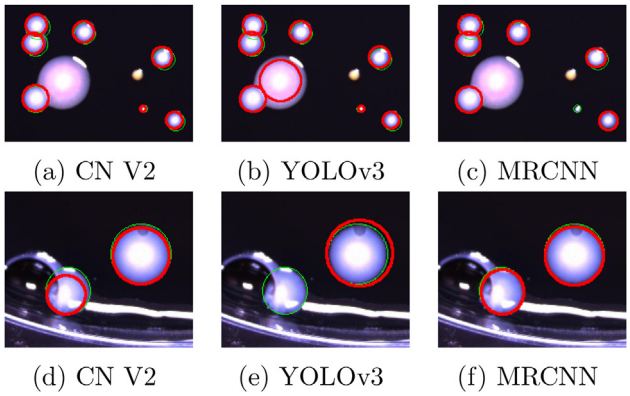

Fig. 11. Object detection results on an image of the bacteria-colonies dataset. The thick red circles indicate the predicted colonies and thin green circles represent the annotations. In this example CentroidNetV2 detects all colonies correctly, MRCNN fails to detect a small colony and YOLOv3 produces a false positive in the top image and a false negative in the bottom image.

comparison between the approaches (including the ResNet backbones that have been used by CentroidNetV2). The authors report an inference time of 27 ms, 100 ms and 130 ms for YOLOv3, MRCNN RN50 and MRCNN RN101 on the PASCAL VOC dataset.

Because of the additional decoding algorithm on top of the backbone the run-time performance of CentroidNetV2 will most likely be worse compared to the other methods. We did not focus on optimizing the run-time efficiency of the decoding algorithm. The current version that is implemented in Python is not representative for the potential inference time (the decoding process currently takes multiple seconds.).

6.4. Common failure cases

In previous subsections the quantitative performance differences between the models have been discussed. This subsection will provide a more elaborate qualitative analysis of the common failure cases of the three approaches, MRCNN, YOLOv3 and CentroidNetV2. The failure cases are divided into three categories: Detection of small objects, detection of low-contrast objects and detection of connected objects. By analyzing details of the results on individual images, interesting insights can be gained into the properties of the algorithms, details that are not always apparent from the reported quantitative metrics in the previous sections.

In Fig. 12, detailed parts of images are shown where the first and the third row contain images from the bacterial-colony dataset and the images in the middle row are from the aerial-crops dataset. The red circles denote detections and the green circles show the ground-truth. In Figs. 12c, f and i can be seen that MRCNN fails to detect the smallest objects. Furthermore, Figs. 12b and e show that YOLOv3 detects all objects but misses one colony in Fig. 12h. CentroidNetV2 detects all objects in these images but the position is slightly misaligned with the ground-truth.

In Fig. 13 the first two rows contain parts from the cell nuclei and the bacterial-colonies datasets. In those images almost no objects are visible due to the very low contrast in parts of the original images. These images have deliberately not been enhanced to show the real contrast. The red circles, which indicate detections, show that all approaches have difficulty detecting all objects, but in Fig. 13a and c CentroidNetV2 is able to detect more of the objects. Fig. 13g shows that MRCNN did not detect the faint purple nucleus and a small nucleus at the bottom edge, however, these are detected by CentroidNetV2. But because these specific cases are relatively rare in the dataset their effect in the F1 score is minimal. As explained earlier, for the cell-nuclei dataset YOLOv3 was not a suitable approach.

In Fig. 14, parts of some challenging images from the cell-nuclei dataset are shown that contain densely connected objects. When comparing Fig. 14a and b it can be seen that CentroidNetV2 is able to distinguish more of the individual objects where MRCNN wrongly detects the cluster of multiple objects as one (indicated by the largest red circle in the Fig. 14b). Furthermore, CentroidNetV2 detects the closely connected object in Fig. 14c, but has difficulty determining the correct size and shape.

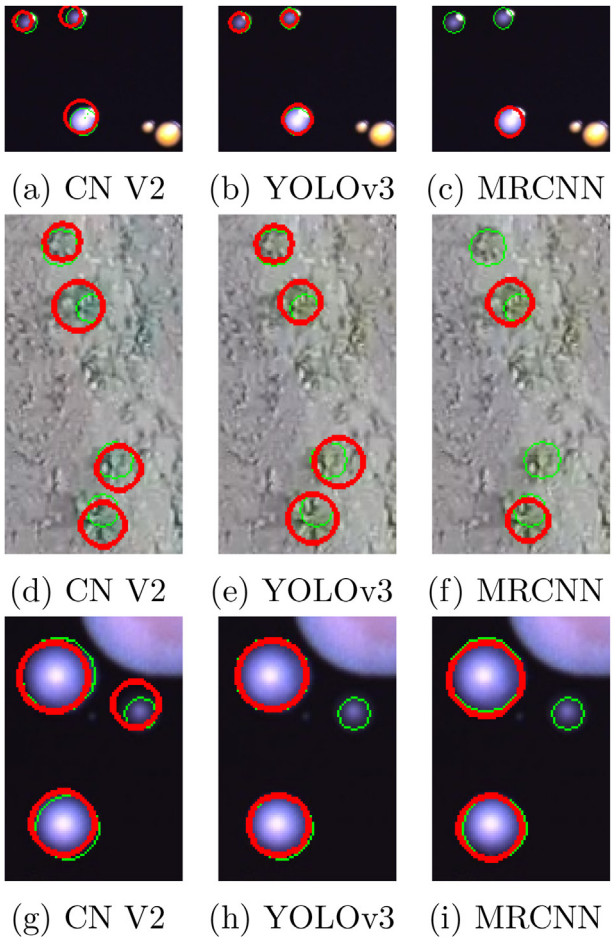

Fig. 12. Detection of small objects. The red circles show detections and the green circles represent the ground truth. This shows that MRCNN detect fewer of the small bacterial colonies and potato-plant crops.

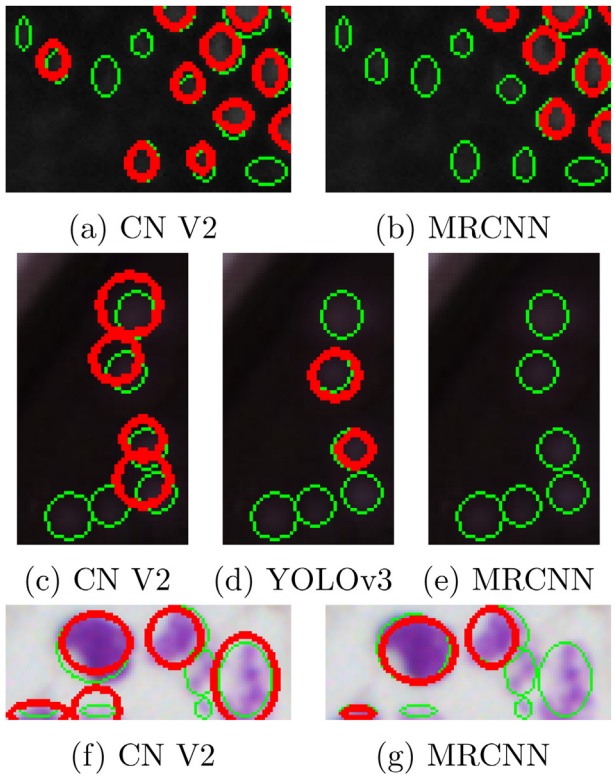

Fig. 13. Detection of low-contrast objects. Images (c), (d) and (e) contain bacterial colonies, the other images contain cell nuclei. The red circles show detections and the green circles represent the ground truth. Images (a) through (e) seem to contain no image information, however this is the true contrast in the image. CentroidNetV2 is able to detect more low contrast objects.

6.5. Comparison of CentroidNetV1 and CentroidNetV2

In this final subsection we reflect on the differences in performance between the original CentroidNet and CentroidNetV2. The orignal CentroidNet is designed as an object localization algorithm that only detects centroids of objects. CenroidNetV2 is an object detection or instance segmentation approach that is designed to also detect borders of objects. Therefore, it is difficult to make a direct comparison. However, both approached have similarities that can be used to compare them. Both approaches utilize a segmentation backbone and an accompanying loss function. The original CentroidNet utilizes a U-net backbone and an MSE loss function. By choosing a comparable configuration for CentroidNetV2 both approaches are compared.

In Tables 1 and 2 the model denoted CentroidNet with a U-net backbone and an MSE loss function is as close as possible to the original CentroidNet model that is still comparable to CentroidNetV2. Therefore, that model will be referred to as CentroidNetV1. The results on the potato-crops dataset in Table 1 show that CentroidNetV1 achieves an F1 score of 93.5% and that CentroidNetV2 achieves a better F1 score of 94.7%. In Table 2 a similar observation is made that CentroidNetV1 achieves an F1 score of 90.6% and CentroidNetV2 shows a better F1 score of 91.9%.

Some voting images of the CentroidNets are shown in Fig. 15. Overall, the votes appear brighter for CentroidNetV2 which indicates that more votes appear on the same locations which, in turn, results in more robust detections. Furthermore the two voting maxima in the top image of Fig. 15c are farther apart. Generally it is better for a counting model to detect an actual object at a slightly wrong location than to not detect it at all.

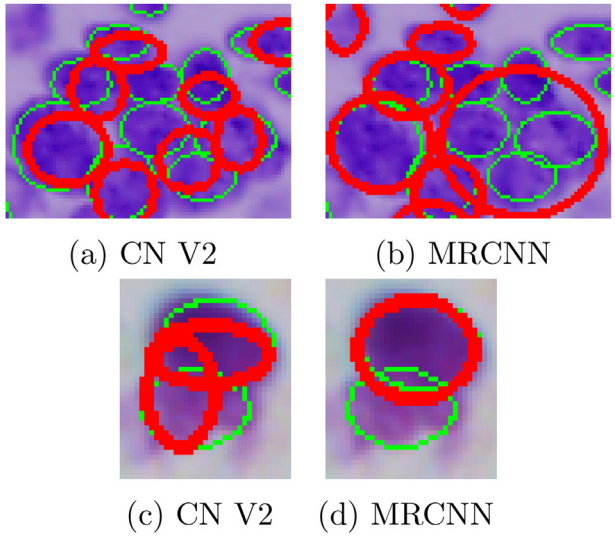

Fig. 14. Detection of connected objects. The red circles show detections and the green circles represent the ground truth. Images (a) through (d) show that CentroidNetV2 detects more of the densely connected nuclei as individuals. In image (f) YOLOv3 is the only approach that detects all bacterial colonies.

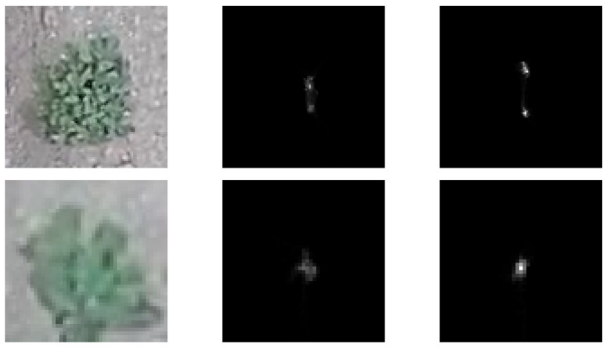

Fig. 15. Voting matrices for CentroidNetV1 and CentroidNetV2. In this example the ground-truth centroids are detected with both approaches. The improvements made to CentroidNetV2 are shown to produce sharper votes.

7. Discussion and conclusion

Experiments have been performed on three datasets with three different models. The datasets and models can be divided in two categories: object detection and instance segmentation. The models for instance segmentation: CentroidNetV2 and MRCNN have been tested on all datasets. The object-detection model YOLOv3 has only been tested on the object-detection datasets: aerial-crops and bacterial-colonies. This is because an instance-segmentation model can be used for object detection but not vice versa. The F1 score has been the main metric by which to evaluate performance, because it indicates the best trade off between over-estimation and underestimation of the number of counted object instances. Precision and recall have been calculated at the point of the highest F1 score determined by an exhaustive search on the training set. All reported metrics are calculated using a disjoint validation set.

CentroidNetV2 shows the best F1 score for the aerial-crops dataset (94.7%) and the bacterial-colonies dataset (92.6%). The best F1 score on the cell-nuclei dataset is achieved by MRCNN (92.3%). For all datasets CentroidNetV2 consistently shows the highest recall: 95.7%, 89.9% and 91.0% on the aerial-crops, cell-nuclei and bacterial-colonies datasets respectively. MRCNN shows the highest precision: 97.7%, 96.3% and 95.4% on the aerial-crops, cell-nuclei and bacterial-colonies datasets respectively. MRCNN has the tendency to miss small objects which results in a high precision at the cost of recall. YOLOv3 generally achieves a high precision, recall and F1 score but is always outperformed by either CentroidNetV2 or MRCNN.

The measured differences among the best-performing models are mostly small, but these differences are consistent over the various datasets. Each model has its own unique properties and the choice ultimately depends on the application. If accurate counting of objects is needed for a large number of small and connected objects, CentroidNetV2 is preferable. When accurate masks of objects should be determined with high recall then MRCNN is preferable. YOLOv3 does a good job at detecting small objects but it is only able to detect bounding boxes whereas CentroidNetV2 produces a complete circumference of objects.

For CentroidNetV2 and MRCNN, images of various sizes are handled in a similar fashion and has thus been made completely transparent by using random image crops during training. However, CentroidNetV2 truly does not take into account image dimensions because all voting vectors are relative. The original YOLOv3 implementation is defined for fixed-size images and therefore requires tiling of the images prior to training and recombination of tiles after inference to avoid scaling. The overlapped tiling method did not seem to adversely affect the performance of YOLOv3.

MRCNN needs to be trained in two stages while CentroidNetV2 and YOLOv3 can be trained in only one stage. YOLOv3 has the benefit of being fully end-to-end trainable, but the decoding of voting vectors and the choice of geometric output shape gives the ability to configure CentroidNetV2 for a specific application. In this hybrid approach, where deep learning is integrated with traditional computer vision, the black-box nature of CNNs is mitigated and, at the same time, performance is improved on certain tasks like counting many small and connected objects.

The remainder of this section will reflect specifically on the research questions.

1. *What is the performance of CentroidNetV2 for detecting and counting many small objects?*
   CentroidNetV2 is considered to be the preferable approach for counting many small objects because the results show that it either achieves the highest F1 score or achieves the best recall and tends to detect more small objects.
2. *How does the performance of CentroidNetV2 compare to well-known state-of-the-art neural networks for object detection and instance segmentation?*
   On two datasets CentroidNetV2 outperforms the well-known state-of-the-art networks on object detection and instance segmentation. Only on the cell-nuclei dataset does MRCNN produce a higher F1 score.
3. *What backbone and loss function is best suitable for CentroidNetV2?*
   The loss function combining vector loss and cross-entropy loss gives sharper voting peaks and consistently achieves the best F1 score compared to the original MSE loss function. The DeepLabV3 + _ResNet101 backbone generally obtains the best performance.
4. *What is the effect of transfer learning on the performance of CentroidNetV2?*
   The results show that the vector-voting method of CentroidNetV2 also benefits from a pretrained backbone of the model. This means that pretrained feature maps of the CNNs are general enough to have a beneficial impact on the F1 score.

7.1. Future work

CentroidNetV2 was compared to the popular and general architectures MRCNN and YOLOv3. Newer and more advanced CNN architectures are introduced regularly. In the future CentroidNetV2 can be compared to recent advances in object detection and segmentation.

The run-time performance of the decoding algorithm of CentroidNetV2 can probably be further optimized by making use of the GPU or by implementing the algorithm in a language that allows for lower-level access to the CPU (for example C++).

Many applications exist for counting that are closely related to the research discussed in this paper. Many different types of vegetation exist that need to be counted. This does not necessarily have to be crops, but can also be trees or other types of large vegetation. Also in the field of microbiology, many applications for colony counting exist. CentroidNetV2 can be tested on other types of bacterial colonies and research into colony counting can be extended to other microbiological fields like medical pathology. Other fields unrelated to counting and more related to object detection and instance segmentation can be investigated. For example, segmentation of everyday objects like persons, cars, etc. CentroidNetV2 might be able to detect smaller everyday objects.

This paper has shown that the results of CentroidNetV2 improved by changing the backbone and the loss function. In the future new segmentation backbones can be integrated with CentroidNetV2. Further investigation of other loss functions might also improve the results.

In this research only classification between background and foreground has been investigated. Future work might focus on counting objects of multiple classes separately. Furthermore multichannel images can serve as an input to CentroidNetV2. Therefore future work might include using hyperspectral imaging to count objects. For this, additional image channels like fluorescent images can be recorded. Even data outside of the visible spectrum can be used like thermal or short-wave infrared images.

Architectural changes could be made to CentroidNetV2 to reduce the number of hyperparameters and this should make the decoding process more straightforward. Such new architectures could be compared to other novel architectures of promising object-detection and instance-segmentation networks.

CRediT authorship contribution statement

Klaas Dijkstra: Conceptualization, Methodology, Software, Investigation, Writing - original draft. **Jaap de Loosdrecht:** Conceptualization, Writing - review & editing, Supervision. **Waatze A. Atsma:** Writing - review & editing, Resources, Data curation. **Lambert R.B. Schomaker:** Conceptualization, Writing - review & editing, Supervision. **Marco A. Wiering:** Conceptualization, Writing - review & editing, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

References

[1] J. Paul Cohen, G. Boucher, C.A. Glastonbury, H.Z. Lo, Y. Bengio, Count-ception: counting by fully convolutional redundant counting, International Conference on Computer Vision (2017) 18–26.
[2] M. Baygin, M. Karakose, A. Sarimaden, E. Akin, An image processing based object counting approach for machine vision application, in: Conference on Advances and Innovations in Engineering, 2018, pp. 966–970.
[3] A. Ferrari, S. Lombardi, A. Signoroni, Bacterial colony counting with Convolutional Neural Networks, Conference of the IEEE Engineering in Medicine and Biology Society (2015) 7458–7461.
[4] K. Dijkstra, J. van de Loosdrecht, L.R. Schomaker, M.A. Wiering, Centroidnet: a deep neural network for joint object localization and counting, in: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, 2018, pp. 585–601.
[5] A. Croxatto, K. Dijkstra, G. Prod'hom, G. Greub, Comparison of inoculation with the InoqulA and WASP automated systems with manual inoculation, J. Clin. Microbiol. 53 (7) (2015) 2298–2307.
[6] A.U.M. Khan, A. Torelli, I. Wolf, N. Gretz, AutoCellSeg: Robust automatic colony forming unit (CFU)/cell analysis using adaptive image segmentation and easy-to-use post-editing techniques, Nat. Sci. Rep. 8 (1) (2018) 2045–2322.
[7] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. (2012) 1097–1105.
[8] O. Ronneberger, P. Fischer, T. Brox, U-net: convolutional networks for biomedical image segmentation, Conference on Medical Image Computing and Computer-Assisted Intervention (2015) 234–241.
[9] A.R. Pathak, M. Pandey, S. Rautaray, Application of deep learning for object detection, Procedia Comput. Sci. 132 (2018) 1706–1717.
[10] K. He, G. Gkioxari, P. Dollar, R. Girshick, R.-C.N.N. Mask, Conference on Computer Vision and Pattern Recognition (2017) 2980–2988.
[11] T. Karras, S. Laine, T. Aila, A style-based generator architecture for generative adversarial networks, Conference on Computer Vision and Pattern Recognition (2019) 4401–4410.
[12] K. Dijkstra, J. van de Loosdrecht, L.R. Schomaker, M.A. Wiering, Hyperspectral demosaicking and crosstalk correction using deep learning, Mach. Vision Appl. 30 (1) (2018) 1–21.
[13] P. Ren, W. Fang, S. Djahel, A novel yolo-based real-time people counting approach, in: 2017 International Smart Cities Conference (ISC2), IEEE, 2017, pp. 1–2.
[14] A. Özlü, TensorFlow Object Counting API (2018), https://github.com/ahmetozlu/tensorflow_object_counting_api.
[15] B. Chen, X. Miao, Distribution line pole detection and counting based on yolo using uav inspection line video, J. Electr. Eng. Technol. (2019) 1–8.
[16] W. Xie, J.A. Noble, A. Zisserman, Microscopy cell counting and detection with fully convolutional regression networks, Comput. Methods Biomech. Biomed. Eng. Imag. Visualiz. 6 (3) (2018) 283–292.
[17] T. Stahl, S.L. Pintea, J.C. Van Gemert, Divide and count: generic object counting by image divisions, IEEE Trans. Image Process. 28 (2019) 1035–1044.
[18] J. Wan, W. Luo, B. Wu, A.B. Chan, W. Liu, Residual regression with semantic prior for crowd counting, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4036–4045.
[19] F. Dai, H. Liu, Y. Ma, J. Cao, Q. Zhao, Y. Zhang, Dense scale network for crowd counting. arXiv preprint arXiv:1906.09707.
[20] Y. Li, X. Zhang, D. Chen, CSRNet: dilated convolutional neural networks for understanding the highly congested scenes, Conference on Computer Vision and Pattern Recognition (2018) 1092–1100.
[21] J. Gao, Q. Wang, X. Li, PCC net: perspective crowd counting via spatial convolutional network, IEEE Trans. Circ. Syst. Video Technol.
[22] Q. Wang, M. Chen, F. Nie, X. Li, Detecting coherent groups in crowd scenes by multiview clustering, IEEE Trans. Pattern Anal. Mach. Intell. 42 (1) (2020) 46–58.
[23] M.R. Hsieh, Y.L. Lin, W.H. Hsu, Drone-based object counting by spatially regularized regional proposal network, Conference on Computer Vision and Pattern Recognition (2017) 4165–4173.
[24] M. Kass, A. Witkin, D. Terzopoulos, Snakes: active contour models, Int. J. Comput. Vision 1 (4) (1988) 321–331.
[25] D.H. Ballard, Generalizing the hough transform to detect arbitrary shapes, Pattern Recogn. 13 (2) (1981) 111–122.
[26] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.
[27] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Networks 61 (2015) 85–117.
[28] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[29] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, Encoder-decoder with atrous separable convolution for semantic image segmentation, European Conference on Computer Vision (2018) 801–818.
[30] J. Redmon, A. Farhadi, Yolov3: an incremental improvement, arXiv preprint arXiv:1804.02767 (2018).
[31] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár, Focal loss for dense object detection, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2980–2988.
[32] M. Ren, R.S. Zemel, End-to-end instance segmentation with recurrent attention, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6656–6664.
[33] X. Liang, L. Lin, Y. Wei, X. Shen, J. Yang, S. Yan, Proposal-free network for instance-level object segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 40 (12) (2017) 2978–2991.
[34] H. Chen, X. Qi, L. Yu, P.-A. Heng, Dcan: deep contour-aware networks for accurate gland segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2487–2496.
[35] C. Couprie, C. Farabet, L. Najman, Y. Lecun, Convolutional nets and watershed cuts for real-time semantic labeling of RGBD videos, J. Mach. Learn. Res. 15 (1) (2014) 3489–3511.
[36] M. Bai, R. Urtasun, Deep watershed transform for instance segmentation, Conference on Computer Vision and Pattern Recognition (2017) 2858–2866.
[37] S. Jetley, M. Sapienza, S. Golodetz, P.H. Torr, Straight to shapes: real-time detection of encoded shapes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6550–6559.
[38] U. Schmidt, M. Weigert, C. Broaddus, G. Myers, Cell detection with star-convex polygons, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2018, pp. 265–273.
[39] Z. Wu, C. Shen, A. v. d. Hengel, Bridging category-level and instance-level semantic image segmentation. arXiv preprint arXiv:1605.06885.
[40] M.A. Rahman, Y. Wang, Optimizing intersection-over-union in deep neural networks for image segmentation, in: International Symposium on Visual Computing, 2016, pp. 234–244.
[41] F. van Beers, A. Lindstrom, E. Okafor, M.A. Wiering, Deep neural networks with intersection over union loss for binary image segmentation, Conference on Pattern Recognition Applications and Methods (2019) 438–445.
[42] A. Paszke, G. Chanan, Z. Lin, S. Gross, E. Yang, L. Antiga, Z. Devito, Automatic differentiation in PyTorch, Neural Inf. Process. Syst.
[43] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning?, J. Mach. Learn. Res. 11 (2010) 625–660.
[44] N.-D. Nguyen, T. Do, T.D. Ngo, D.-D. Le, An evaluation of deep learning methods for small object detection, J. Electr. Comput. Eng. (2020).

Dr. Klaas Dijkstra is an associate professor in computer vision and data science at NHL Stenden University of Applied Sciences. His main research interests are in computer vision, machine learning and hyperspectral imaging. After completing his B.Eng. degree in technical information science in 2005, he has been active in the field of computer vision by doing applied research projects in several domains. In 2013 he obtained his M. Sc. degree from the Limerick Institute of Technology in Ireland, on the application of evolutionary algorithms and computer vision to the domain of microbiological analysis. He obtained his Ph.D. degree in 2020 from the University of Groningen on the topic of deep learning and hyperspectral imaging for unmanned aerial vehicles.

Jaap van de Loosdrecht is a professor in computer vision and data scienceat the NHL Stenden University of Applied Sciences. His main research interests are in computer vision, deep learning and hyperspectral imaging. In 1996 he has founded the professorship Computer Vision and & Data Science. His staff, researchers, teachers and students have carried out more than 350 research projects for the business community in the field of Computer Vision & Data Science, including the Raak-Award 2016 project 'Smart Vision for UAV's'. He is Comenius Senior Fellow at KNAW (Royal Dutch Academy of Sciences).

Waatze A. Atsma, received his engineering degree in Biotechnology at the Noordelijke Hogeschool in Leeuwarden in 2002 and has been working at the drinking water laboratory Vitens N.V. as a principle analyst and project leader since 2003. His specialty is mainly in the field of drinking water diagnostics, in particular the development and implementation of new (molecular based) microbiological methods within the Dutch drinking water laboratories. In addition, Atsma is a member and chairman of various national working groups in the field of implementation of rapid microbiological methods in the drinking water laboratories and is closely involved in drawing up guidelines for drinking water-related, microbiological methods, including for Legionella diagnostics. On behalf of the Netherlands, Atsma is one of the delegation members for the ISO/TC147/SC4 microbiological parameters with the aim of drawing up or changing international standards for water microbiology tests.

Prof. dr. Lambert Schomaker is professor in artificial intelligence at the university of Groningen since 2001. He is known for research in simulation and recognition of handwriting, writer identification, style-based document dating and other studies in pattern recognition, machine learning and robotics. He has (co)authored over 200 publications and was involved in the organization of many conferences in handwriting recognition and document analysis. In recent years he and his team have worked on the Monk system: an interactively trainable search engine and e-Science service for historical manuscripts. The availability of up to thousands of training images for single classes of complex patterns has brought pattern recognition and machine learning into the ballpark of big data. Other recent work is in the area of robotics and industrial maintenance, in the EU ECSEL project Mantis. In 2015, he became co-chair of the Data Science and Systems Complexity center at the Faculty of Science and Engineering at the University of Groningen. In 2017, he joined the CogniGron center for cognitive systems and materials in a largescale seven-year project in neuromorphic computing. He is a member of the IAPR and senior member of IEEE.

Dr. Marco Wiering is an assistant professor in the department of artificial intelligence from the University of Groningen, the Netherlands. He performed his PhD research in the research institute IDSIA in Switzerland and graduated in 1999 on the topic of reinforcement learning. Before going to the University of Groningen, he worked as an assistant professor at Utrecht University. Dr. Wiering has co-authored more than 170 conference or journal papers and has supervised or is supervising 12 PhD students and more than 110 Master student graduation projects. His main research topics are reinforcement learning, deep learning, neural networks, robotics, computer vision, game playing, timeseries prediction and optimization.