Chapter 5

CentroidNetV2: A Hybrid Deep Neural Network for Small-Object Segmentation and Counting

This chapter 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.
This chapter is submitted to:

This chapter shows several improvements over the original CentroidNet algorithm (Dijkstra et al., 2018a) 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 been 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 (Krizhevsky et al., 2012; Ronneberger et al., 2015; Pathak et al., 2018; He et al., 2017; Karras et al., 2019; Dijkstra et al., 2018b).

A typical method to count objects with a CNN is to train an object detection model and subsequently count the number of detected objects (Ren et al., 2017; Özlü, 2018; Chen and Miao, 2019). Most object-detection neural networks are designed to detect typical everyday objects and therefore, 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 (Xie et al., 2018; Stahl et al., 2019; Wan et al., 2019; Dai et al., 2019). 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 a crowd (Li et al., 2018). Recent approaches combine object localization and object detection with counting (Dijkstra et al., 2018a; Hsieh et al., 2017).

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 chapter 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 chapter a new hybrid deep learning architecture called CentroidNetV2 is introduced.

5.1 Related work

CNNs (LeCun et al., 2015; Schmidhuber, 2015; Goodfellow et al., 2016) 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 (Krizhevsky et al., 2012). 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 (Ronneberger et al., 2015) and by DeepLabV3+ on segmenting everyday scenes (Chen et al., 2018). A sparser detection is achieved by object detection CNNs like You Only Look Once Version 3 (YOLOv3) (Redmon and Farhadi, 2018) and RetinaNet (Lin et al., 2017c). 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. Mask Recurrent Convolutional Neural Network (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 (He et al., 2017) and requires a two-stage training process. A Recurrent Neural Network (RNN) for instance segmentation is proposed (Ren and Zemel, 2017), 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 (Liang et al., 2017), where segments and boxes are directly regressed and a traditional off-the-shelf clustering method is used to create individual
5.1. Related work

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 (Chen et al., 2016a). In a similar fashion a deterministic temporal consistency algorithm is combined with a CNN to segment RGB + depth videos (Couprie et al., 2014). 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 (Bai and Urtasun, 2017).

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 (Jetley et al., 2017). The star-convex polygon method uses radial distances to encode object instances with a CNN (Schmidt et al., 2018).

Related to this are methods that predict the centroids of individual object instances. These are proposed by (Wu et al., 2016) and in CentroidNet (Dijkstra et al., 2018a). 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-supression 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.

5.1.1 Deep design patterns

Most of the recently published CNN architectures are not built from scratch. Several building blocks are combined to create new neural network architectures or are generated automatically (Elsken et al., 2019). In this chapter we use the term ‘deep design patterns’ to discuss some general topological approaches encountered in deep learning. In Figure 5.1 several deep design patterns used for the CNNs are shown. This is not a comprehensive overview but merely contains deep design patterns encountered in the models discussed in this chapter. The blue blocks represent the input and output tensors. Each tensor can have an arbitrary depth and the size of a block is an indication for the spatial dimensions of the tensor. Tensor operations are denoted by black lines and can be defined by any combination of deep-learning layers. Examples are convolution, transposed convolution, max-pooling, concatenation, element-wise arithmetic, etc.

Encoder/Decoder is a pattern where the input tensors are spatially reduced and depth-wise increased. The encoder pathway transforms spatial information into semantic information (Badrinarayanan et al., 2017). Subsequently this intermediate tensor is upscaled to its original size by the decoder pathway. This pattern is often used in semantic
5.1. Related work

segmentation tasks to predict the semantics of each spatial location and is used extensively by U-net (Ronneberger et al., 2015).

Skip connection (or residual connection) is a pattern commonly used to relay information over larger distances in the network graph. This sharpens segmentation results or reduces the risk of vanishing gradients in very deep neural networks. The ResNet (Szegedy et al., 2017) backbones of DeepLabV3+ follow this pattern.

The multi-filter pattern applies multiple convolution filters with different properties simultaneously to the input. The network implicitly chooses the appropriate filter configuration by adjusting the weights of each filter (Chollet, 2017). This allows a more flexible and faster design of the network and is used by the Xception backbone of DeepLabV3+.

The feature-pyramid design follows a classic pattern in image processing where multiple scales of the image are processed separately to introduce scale invariance. The down scaling is often performed using convolution and max-pooling operators. Several scales of the input serve as a new input to other parts, or heads, of the neural network graph (Lin et al., 2017b). This essentially makes feature pyramids a combination of the encoder/decoder and the skip-connection pattern as can be seen in Figure 5.1. Among other patterns, this is the main pattern used by YOLOv3 to detect small objects (Redmon and Farhadi, 2018).

Fully Convolutional Networks (FCNs) are used in most modern object detection and segmentation models and have no fully-connected layers. They are only comprised of size-invariant layers, such as convolution and max pooling. FCNs can be trained using varying input sizes without the need to resize the input images to a predefined fixed size (Long et al., 2015). This reduces the risk of down scaling an image to a point where small objects cannot be detected because there are not enough pixels left.

Many modern network architectures combine several deep design patterns to achieve state-of-the-art results. Earlier, CNNs were designed from scratch but nowadays existing models, called backbones, are embedded into larger CNN structures to form new network designs. The backbone is mostly used to produce (pretrained) features and the remaining sub-networks are often referred to as heads.

The introduction of these patterns allows us to reason about CNNs and provide a clear description of the benefits and properties of the networks used throughout this chapter.
5.2 Contributions and research questions

The original version of CentroidNet (Dijkstra et al., 2018a) is a FCN that is trained using a standard Mean Squared Error (MSE) loss function. A U-net backbone is used to regress a field of 2D 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 2D 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 (Kass et al., 1988) with properties similar to the generalized Hough transform (Ballard, 1981). 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 MRCNN and CentroidNetV2 give them comparable complexity. A specific shared goal of CentroidNetV2 and YOLOv3 is small-object detection.

In addition to the architectural changes several ablation studies are performed. The loss function is redesigned to contain two MSE 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 (Ronneberger
et al., 2015) and DeepLabV3 (Chen et al., 2016b) with three backbones, ResNet50, ResNet101 and Xception (Szegedy et al., 2017; Chollet, 2017), 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 chapter we generally refer to a 1D structure as a vector, a 2D structure as a matrix and a 3D 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 chapter is structured as follows. In Section 5.3 the formal design of CentroidNetV2 is discussed. Section 5.4 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 5.5 and the results are presented in Section 5.6. Finally, in Section 5.7 the conclusion and future work are discussed.

5.3 The CentroidNetV2 architecture

The main architecture of CentroidNetV2 is shown in Figure 5.2. 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 $f(\cdot)$, 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 $g(\cdot)$. 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 $g(\cdot)$, indicated by $g'(\cdot)$. Additionally the loss function $l(\cdot, \cdot)$ calculates a loss between the output tensor $Y$ and the target tensor $T$.

For convenience and without loss of generality the functions in this section are defined using 3D image-like tensors. However the actual implementation uses mini batches of 3D tensors. The three main functions $f(\cdot)$, $g(\cdot)$ and $l(\cdot, \cdot)$ are explained in sub-section 5.3.1, 5.3.2 and 5.3.3 respectively.

---

$^1$The function $g'(\cdot)$ is the inverse transform of $g(\cdot)$ if the class probability is disregarded.
5.3. The CentroidNetV2 architecture

Figure 5.2: 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 $g(\cdot)$.

5.3.1 Backbones

Function $f(\cdot)$ in Figure 5.2 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 chapter 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 $f(\cdot)$. In our experiments the depth of the input $X$ is 3 (RGB) and the depth of the output $Y$ is 6 (a 2D centroid vector, a 2D border vector and 2 logits).

This is mathematically expressed by

\[
Y = f(X) \quad (5.1)
\]
\[
Y = [Y_c | Y_b | Y_l], \quad (5.2)
\]

where $X$ is the input tensor of the model, $Y$ is the output tensor with stacked tensors containing the centroid-vectors tensor $Y_c$, border-vectors tensor $Y_b$ and the logits tensor $Y_l$.

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

\[
Y_{p_{z,y,x}} = \frac{Y_{l_{z,y,x}}}{\sum_{z \in [c]} Y_{l_{z,y,x}}}, \quad (5.3)
\]

where $Y_l$ contains the class logits, $Y_p$ contains the class probabilities and $c$ is the number of classes.

Some example centroid vectors and border vectors in $Y_c$ and $Y_b$ are geometrically shown in Figure 5.3, 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 $b'_1$).
5.3. The CentroidNetV2 architecture

Figure 5.3: 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.

5.3.2 Loss functions

Function $l(\cdot, \cdot)$ in Figure 5.2 calculates the loss between the output tensor $Y$ and the target tensor $T$. Depending on the loss function we use the logits output $Y_l$ or the probability output $Y_p$. The target tensors are defined in a similar way as Equation 5.2:

$$T = [T_c|T_b|T_p], \quad (5.4)$$

where $T$ consists of the stacked tensors with target-centroid-vectors tensor $T_c$, target-border-vectors tensor $T_b$ and the target-probability tensor $T_p$ containing $n$ planes. Note that the target probability for a certain class is always 0 or 1.
MSE loss

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

\[
\ell_{\text{mse}}(Y, T) = \frac{1}{c \cdot h \cdot w} \sum_{z \in [c]} \sum_{y \in [h]} \sum_{x \in [w]} (Y_{z,y,x} - T_{z,y,x})^2
\]  

(5.5)

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 a distance-based 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.

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:

\[
D_{y,x}^2 = \sum_{z \in [c]} (Yv_{z,y,x} - Tv_{z,y,x})^2
\]

\[
\ell_d(Yv, Tv) = \frac{1}{h \cdot w} \sum_{y \in [h]} \sum_{x \in [w]} D_{y,x},
\]  

(5.6)

where \( Yv \) and \( Tv \) have size \( c \times h \times w \) and represent the output- and target-vectors tensors. Both the centroid vectors and border vectors are two dimensional therefore, 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.
5.3. The CentroidNetV2 architecture

The vector loss is calculated separately for the centroid vectors and the border vectors using Equation 5.6 and then the sum is calculated. The Euclidean distance is chosen because then the loss can be interpreted as the average distance deficit in pixels between the target-voting vectors and the output-voting vectors.

\[ 1_{v1}(Y_c, Y_b, T_c, T_b) = 1_d(Y_c, T_c) + 1_d(Y_b, T_b), \quad (5.7) \]

where \( Y_c, Y_b, T_c, T_b \) contain the output-centroid vectors, output-border vectors, target-centroid vectors and target-border vectors respectively.

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:

\[ \text{CE}_{y,x} = - \sum_{z \in [c]} (T_p_{z,y,x} \log(Y_p_{z,y,x})) \]

\[ 1_{ce}(Y_p, T_p) = \frac{1}{h \cdot w} \sum_{y \in [h]} \sum_{x \in [w]} \text{CE}_{y,x}, \quad (5.8) \]

where \( Y_p, T_p \) 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 Rahman and Wang (2016) and van Beers et al. (2019) and is defined by:

\[ I_z = \sum_{y \in [h]} \sum_{x \in [w]} Y_p_{z,y,x} \times T_p_{z,y,x} \]

\[ U_z = \sum_{y \in [h]} \sum_{x \in [w]} (Y_p_{z,y,x} + T_p_{z,y,x}) - (Y_p_{z,y,x} \times T_p_{z,y,x}) \]

\[ 1_{iou}(Y_p, T_p) = 1 - \frac{1}{c} \sum_{z \in [c]} \frac{I_z}{U_z}, \quad (5.9) \]
CentroidNetV2 loss

The individual terms of the loss functions are tested and their performance is reported in the results section of this chapter. Equation 5.10 combines vector loss and the cross entropy loss and Equation 5.11 combines the vector loss and the IoU loss.

\[ l_{v1,ce}(Y, T) = l_{v1}(Y_c, Y_b, T_c, T_b) + l_{ce}(Y_p, T_p) \]  \hspace{1cm} (5.10)
\[ l_{v1,iou}(Y, T) = l_{v1}(Y_c, Y_b, T_c, T_b) + l_{iou}(Y_p, T_p), \]  \hspace{1cm} (5.11)

where \( Y \) is the output tensor containing output-centroid-vectors tensor \( Y_c \), output-border-vectors tensor \( Y_b \) and output-probabilities tensor \( Y_p \), similarly \( T \) is the target tensor containing target-centroid-vectors tensor \( T_c \), target-border-vectors tensor \( T_b \) and target-probabilities tensor \( T_p \). No weighing of the individual terms of the loss function was used.

5.3.3 Coders

This sub-section discusses the decoder function \( g(\cdot) \) and the encoder function \( g'(\cdot) \) of Figure 5.2. 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.

Decoder

Individual object instances are calculated from the output of the model using the centroid-vectors tensor \( Y_c \), border-vectors tensor \( Y_b \) and class-probabilities tensor \( Y_p \), defined in Equations 5.1, 5.2 and 5.3.

Initially the \( \text{vote}(\cdot) \) function in Algorithm 2 calculates the voting matrix. An output-voting-vectors tensor \( Y_v \) serves as an input (this can either be \( Y_c \) or \( Y_b \)). This tensor contains the relative 2D centroid vectors...
Algorithm 2 Calculate the voting matrix given the output-voting-vectors tensor

1: function vote(Yv)
2: \( h, w \leftarrow \text{height, width of } Yv \)
3: \( V \leftarrow \text{zero-filled matrix of size } (h, w) \)
4: for \( y \leftarrow 1 \) to \( h \) do
5:     for \( x \leftarrow 1 \) to \( w \) do
6:         \( y' \leftarrow y + Yv_{1,y,x} \) \hfill \text{Get the absolute y component}
7:         \( x' \leftarrow x + Yv_{2,y,x} \) \hfill \text{Get the absolute x component}
8:         \( V_{y',x'} \leftarrow V_{y',x'} + 1 \) \hfill \text{Aggregate votes}
9:     end for
10: end for
11: return \( V \)
12: end function

Algorithm 3 Calculate the border coordinates of an instance with respect to a given centroid.

1: function border\((y_c, x_c), Yb, Yc\)
2: \( \mathcal{B} = \{ \} \)
3: \( h, w \leftarrow \text{height, width of } Yc \) \hfill \text{Get spatial dimensions of the input}
4: for \( y \leftarrow 1 \) to \( h \) do
5:     for \( x \leftarrow 1 \) to \( w \) do
6:         \( y'_c \leftarrow y + Yc_{y,x,1} \) \hfill \text{Get absolute centroid vector } y\)
7:         \( x'_c \leftarrow x + Yc_{y,x,2} \) \hfill \text{Get absolute centroid vector } x\)
8:         if \( (y'_c, x'_c) == (y_c, x_c) \) then \hfill \text{Contributed to centroid } (y_c, x_c)
9:             \( y'_b \leftarrow y + Yb_{y,x,1} \) \hfill \text{Get absolute border vector } y\)
10:            \( x'_b \leftarrow x + Yb_{y,x,2} \) \hfill \text{Get absolute border vector } x\)
11:            \( \mathcal{B} \leftarrow \mathcal{B} \cup \{ (y'_b, x'_b) \} \) \hfill \text{Add border coordinate}
12:        end if
13:     end for
14: end for
15: return \( \mathcal{B} \) \hfill \text{Border coordinates of object with centroid } (y_c, x_c)
16: end function
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 centroids 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 3 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 $B$ 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 Figure 5.4.

$^2$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.
5.3. The CentroidNetV2 architecture

![Figure 5.4: An example of the data of the decoder represented by images. From left to right: input image X, magnitudes of the centroid vectors Yc, magnitudes of border vectors Yb, accumulated centroid votes V, set of centroid coordinates C, set of border coordinates B, color-coded instances I and per-pixel class ids C.](image)

1. Calculate the centroid-vote matrix \( V = \text{vote}(Yc) \), where \( Yc \) is a tensor containing the centroid vectors predicted by the model and \( \text{vote}(\cdot) \) is the voting function defined in Algorithm 2.

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:

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

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 \( \theta \) to the suppressed-voting matrix \( \hat{V} \) to generate the set of selected votes \( C \) given by:

\[
C = \{(y_c, x_c) \in [h] \times [w] \mid \hat{V}_{y_c,x_c} \geq \theta\}, \quad (5.12)
\]

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 \((y_c, x_c) \in C\) is given by:

\[
B = \text{border}((y_c, x_c) \in C, Yb, Yc),
\]
where the function $\text{border}(\cdot, \cdot, \cdot)$ calculates border coordinates for a given centroid at $(y_c, x_c)$ and is given by Algorithm 3, $\mathbf{Y}_b$ and $\mathbf{Y}_c$ are tensors containing the border and centroid vectors respectively.

5. Fit a geometric shape (e.g. circle, ellipse, 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 $\mathbf{I}$.

6. Calculate the class-ids matrix and probabilities matrix $\mathbf{C}$ and $\mathbf{P}$ respectively by taking the $\arg\max(\cdot)$ and $\max(\cdot)$ over class probabilities:

$$
\mathbf{C}_{y,x} = \arg\max_{z \in [c]} (\mathbf{Y}_p_{z,y,x})
$$

$$
\mathbf{P}_{y,x} = \max_{z \in [c]} (\mathbf{Y}_p_{z,y,x}),
$$

where $c$ is the number of logits in the output-probabilities tensor $\mathbf{Y}_p$. When the probability of an element in matrix $\mathbf{P}$ is above $\phi$, it is accepted in the corresponding class-id matrix $\mathbf{C}$, 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 $\mathbf{C}_{y,x}$.

7. Guarantee that for each element in instance-ids matrix $\mathbf{I}$ and class-ids matrix $\mathbf{C}$, 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. Masking is performed per pixel and the final shape of object instances can be different from the fitted shape.

The instance-ids matrix $\mathbf{I}$, class-ids matrix $\mathbf{C}$ and class-probabilities matrix $\mathbf{P}$ are the final outputs of CentroidNetV2.

**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 $\mathbf{X}$ consists of the target-class-ids matrix $\mathbf{C}'$ 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 ('). The output of the encoder is the target-centroid-vectors tensor $T_c$, the target-border-vectors tensor $T_b$ and the target-probabilities matrix $T_p$ defined in Equation 5.4.

The encoding process is defined in the remaining part of this section and the intermediate images to support the explanation are shown in Figure 5.5. The black border around the objects in $T_c$ and $T_b$ is caused by the clipping of voting vectors. This is set to roughly twice the average radius of the target objects.

![Figure 5.5: Data of the encoder represented by images of potato plants annotated with circles. From left to right: input image $X$, color-coded target-instance ids $I'$, target-class ids $C'$, magnitudes of the target-centroid-vectors $T_c$ and target-border-vectors $T_b$. Voting vectors with high magnitudes are bright white and voting vectors with low magnitudes appear darker.](image)

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:

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

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:

$$C' = \{\mathcal{O}'_1, \mathcal{O}'_2, ..., \mathcal{O}'_n\},$$
where $C'$ is the set of target centroids of the object instances, $\overline{O}_i$ 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:

$$
Tc_{y, x} = \arg \min \{ (y_c, x_c) \in C' \} ||(y_c, x_c) - (y, x)|| - (y, x),
$$

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 $C'$. Note that $Tc$ is a 3D tensor where the third dimension has size two and contains the relative vectors $((y_c, x_c) \in C') - (y, x)$ pointing to the nearest centroid. Also note that the $\arg \min$ function returns a vector $(y_c, x_c) \in C'$.

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

$$
Tb_{y, x} = \arg \min \{ (y_b, x_b) \in B'_i \} ||(y_b, x_b) - (y, x)|| - (y, x),
$$

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 $((y_b, x_b) \in B'_i) - (y, x)$ pointing to the nearest border of the object instance with the nearest centroid.

Finally, the target-probabilities matrix is given by:

$$
Tp_{c, y, x} = 1(C'_{y, x} == c),
$$

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 $1(\cdot)$ 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.
5.4 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.

5.4.1 Aerial crops

The aerial-crops dataset contains images of potato crops taken with a low-cost drone which navigated over a potato field (Dijkstra et al., 2018a). 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 $1500 \times 1800$ pixels. The set contains over 3000 annotated plants and has been annotated by two domain experts using circles to indicate the location of the plants. See Figure 5.6 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. Each network is trained twice using annotations from each expert individually. This is done to investigate the effect of the annotator on the results. 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. The reason for using circles is because the general shape of these crops is circular and because the camera was mounted perpendicular to the image plane the perspective distortion is minimal.
Figure 5.6: Example images from the aerial-crops dataset. The images show variations in the size of the crops and high connectedness between individual crops.

5.4.2 Cell nuclei

The cell-nuclei dataset was used for the Kaggle data science bowl 2018. 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 Figure 5.7 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.

3https://www.kaggle.com/c/data-science-bowl-2018
5.4. Datasets

5.4.3 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. 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 Figure 5.8 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.

\[^{4}\text{ISO 11731:2017, Water quality – Enumeration of Legionella}\]
Chapter 5. CentroidNetV2

5.4.4 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 $\times$ 256 image crops.

The best performing YOLOv3 should be trained with images of 608 $\times$ 608 pixels as described in the original paper (Redmon and Farhadi, 2018). 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 Figure 5.9 this tiling process is explained.
5.5 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.

5.5.1 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 (Goodfellow et al., 2016). 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 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.

5.5.2 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 $\mathbf{I}$ of a model and the target-instances matrix $\mathbf{I}'$ are compared. Each element of these matrices contains a value indicating the instance id that pixel belongs to. The apostrophe (‘) 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:

$$\mathcal{O}_a = \{(y, x) \in [h, w] \mid \mathbf{I}_{y,x} = a\}$$

$$\mathcal{O}'_b = \{(y', x') \in [h, w] \mid \mathbf{I}'_{y',x'} = b\},$$

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:

\[
\text{IoU}(O, O') = \frac{O \cap O'}{O \cup O'},
\]

Matching of object instances is based on a certain minimum IoU threshold. The set of output-instance ids is given by \( A = [m] \) and the set of target-instance ids is given by \( B = [n] \), 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 \( A \) is given by:

\[
\text{is\_match}(b \in B) = \max_{a \in A} (\text{IoU}(O_a, O'_b)) > \tau
\]

where \( \tau \) is the minimum IoU threshold, \( \text{is\_match}(\cdot) \) 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 \( A \) and from the set of target ids \( B \). The matching ids are then added to the set of matches by \( M = M \cup \{(a, b)\} \). This process of matching and removing is repeated for all output-instance ids in \( B \). If all objects have a match, both \( A \) and \( B \) will be empty and \( M \) 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:
\[ TP = |M| \quad (5.13) \]
\[ FP = |A| \quad (5.14) \]
\[ FN = |B| \quad (5.15) \]
\[ P = \frac{TP}{TP + FP} \quad (5.16) \]
\[ R = \frac{TP}{TP + FN} \quad (5.17) \]
\[ F1 = 2 \times \frac{P \times R}{P + R} \quad (5.18) \]
\[ Count = TP + FP, \quad (5.19) \]

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 the next section only two classes are used (background and foreground).

### 5.6 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. 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 Equations 5.13 through 5.19 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.

The highest F1 score represents the best equilibrium between overestimating and underestimating the number of objects and 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 \( \theta \) discussed in
Equation 5.12 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 5.3. MSE loss is the standard loss defined by Equation 5.5. The Vector Loss (VL) is computed by the Euclidean distance between the target-voting vectors and the output-voting vectors and is defined by Equation 5.7. The Cross Entropy (CE) loss and IoU loss, defined in Equations 5.8 and 5.9, 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 Equations 5.5, 5.10, 5.11 respectively.

5.6.1 Results on aerial crops

The results of the performance on the aerial-crops dataset for the different models are shown in Table 5.1 and Table 5.2.

![Figure 5.10: Red circles show the prediction of the three models and the annotations are shown in green. CentroidNetV2 detected most crops, MRCNN missed small objects and YOLOv3 produced a false positive and a false negative.](image)

When comparing the overall F1 scores of annotations created by Expert A and Expert B it is clear that using the annotations of Expert A gives the best performance. After investigation, it was found that the annotations of Expert A were more precise compared to the annotations of expert B. This shows that the effect caused by the annotator is quite large: the best performing model, when using annotations from Expert A, achieves a 2.7% higher F1 score compared to the best performing model
when using annotations of Expert B. Therefore the analysis will primarily be focused on the annotations created by expert A.

The first part of Table 5.1 (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 many 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. When using annotations of expert B (shown in Table 5.2) CentroidNetV2 still achieves the highest F1 score, only this time the U-net backbone with standard MSE loss achieves the highest F1 score (92%).

The visual differences between the individual models are shown in Figure 5.10. CentroidNetV2 shows the most correctly detected crops in Figure 5.10a. 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 Figure 5.10b and the two missed small crops. Figure 5.10c shows that MRCNN failed to detect the small objects in this dataset.

The second part of Table 5.1 (Comparing loss functions) shows that the MSE loss achieves the lowest F1 score (93.5%) compared to the other loss functions. Some voting images of CentroidNetV2 are shown in Figure 5.11 for a model trained with MSE loss and a model trained with VL-CE loss. Overall, the votes appear brighter for the VL-CE loss which indicates that more votes appear on the same locations which, in turn, results in more robust detections. Furthermore the two voting maxima for the VL-CE loss in the top image of Figure 5.11c 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.
### 5.6. Experiments and results

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Loss</th>
<th>F1</th>
<th>P</th>
<th>R</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN101</td>
<td>VL-CE</td>
<td>94.7</td>
<td>94.4</td>
<td>95.1</td>
<td>1578</td>
<td>94</td>
<td>82</td>
<td>1672</td>
</tr>
<tr>
<td>CentroidNetV2</td>
<td>U-net</td>
<td>MSE</td>
<td>93.5</td>
<td>92.2</td>
<td>94.8</td>
<td>1573</td>
<td>133</td>
<td>87</td>
<td>1706</td>
</tr>
<tr>
<td>YOLOv3</td>
<td>Default</td>
<td>Default</td>
<td>94.3</td>
<td>93.7</td>
<td>94.9</td>
<td>1575</td>
<td>106</td>
<td>85</td>
<td>1681</td>
</tr>
<tr>
<td>MRCNN</td>
<td>RN101</td>
<td>Default</td>
<td>92.4</td>
<td>97.7</td>
<td>87.7</td>
<td>1456</td>
<td>34</td>
<td>204</td>
<td>1490</td>
</tr>
</tbody>
</table>

**Comparing loss functions**

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Loss</th>
<th>F1</th>
<th>P</th>
<th>R</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN101</td>
<td>MSE</td>
<td>93.5</td>
<td>92.5</td>
<td>94.6</td>
<td>1570</td>
<td>127</td>
<td>90</td>
<td>1697</td>
</tr>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN101</td>
<td>VL-IoU</td>
<td>94.3</td>
<td>93.9</td>
<td>94.6</td>
<td>1571</td>
<td>102</td>
<td>89</td>
<td>1673</td>
</tr>
</tbody>
</table>

**Comparing backbones**

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Loss</th>
<th>F1</th>
<th>P</th>
<th>R</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>U-net</td>
<td>VL-CE</td>
<td>94.0</td>
<td>92.3</td>
<td>95.7</td>
<td>1588</td>
<td>132</td>
<td>72</td>
<td>1720</td>
</tr>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-XC</td>
<td>VL-CE</td>
<td>90.3</td>
<td>86.6</td>
<td>94.3</td>
<td>1566</td>
<td>242</td>
<td>94</td>
<td>1808</td>
</tr>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN50</td>
<td>VL-CE</td>
<td>94.6</td>
<td>94.7</td>
<td>94.5</td>
<td>1569</td>
<td>87</td>
<td>91</td>
<td>1656</td>
</tr>
</tbody>
</table>

Table 5.1: Results for counting crops with 1660 annotated validation samples. Performance of several configurations of CentroidNetV2 and comparison to YOLOv3 and MRCNN on the annotations provided by expert A (in percentages).
TABLE 5.2: Results for counting crops with 1587 annotated validation samples. Performance of several configurations of CentroidNetV2 and comparison to YOLOv3 and MRCNN on the annotations provided by expert B (in percentages).

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Loss Function</th>
<th>P</th>
<th>R</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN50</td>
<td>VL-CE</td>
<td>91.4</td>
<td>90.7</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLV3-XC</td>
<td>VL-CE</td>
<td>92.4</td>
<td>91.4</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td></td>
<td>U-net</td>
<td>VL-CE</td>
<td>91.3</td>
<td>91.3</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
</tbody>
</table>

Comparing backbones

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Loss Function</th>
<th>P</th>
<th>R</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN50</td>
<td>VL-CE</td>
<td>91.4</td>
<td>90.7</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLV3-XC</td>
<td>VL-CE</td>
<td>92.4</td>
<td>91.4</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td></td>
<td>U-net</td>
<td>VL-CE</td>
<td>91.3</td>
<td>91.3</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
</tbody>
</table>

Comparing loss functions

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Loss Function</th>
<th>P</th>
<th>R</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRCNN</td>
<td>Default</td>
<td>0.4</td>
<td>91.4</td>
<td>91.4</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td>YOLOv3</td>
<td>Default</td>
<td>0.4</td>
<td>91.4</td>
<td>91.4</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSM</td>
<td>0.4</td>
<td>91.4</td>
<td>91.4</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
</tbody>
</table>

Comparing to the state-of-the-art

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Loss Function</th>
<th>P</th>
<th>R</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN50</td>
<td>VL-CE</td>
<td>91.4</td>
<td>90.7</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLV3-XC</td>
<td>VL-CE</td>
<td>92.4</td>
<td>91.4</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td></td>
<td>U-net</td>
<td>VL-CE</td>
<td>91.3</td>
<td>91.3</td>
<td>152</td>
<td>120</td>
<td>1467</td>
<td></td>
</tr>
</tbody>
</table>
5.6. Experiments and results

![Images](image1.png)

**Figure 5.11:** Voting matrices when using different loss functions with CentroidNetV2. In this example the ground-truth centroids are detected with both loss functions. The loss function VL-CE produces sharper votes compared to MSE.

The third part of Table 5.1 *(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.

### 5.6.2 Results on cell nuclei

The results of the performance on the cell-nuclei dataset for the different models are shown in Table 5.3. 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). 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.
### Table 5.3: 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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Usage of pretraining</th>
<th>Loss</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN101</td>
<td>Yes</td>
<td>VL-CE</td>
<td>91.9</td>
</tr>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN101</td>
<td>No</td>
<td>VL-CE</td>
<td>90.6</td>
</tr>
<tr>
<td>MRCNN RN101</td>
<td>Default</td>
<td>Yes</td>
<td>VL-CE</td>
<td>92.3</td>
</tr>
<tr>
<td>MRCNN RN101</td>
<td>Default</td>
<td>No</td>
<td>VL-CE</td>
<td>91.5</td>
</tr>
</tbody>
</table>

Usage of pretraining with ResNet50 backbone:

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Usage of pretraining</th>
<th>Loss</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN50</td>
<td>Yes</td>
<td>VL-CE</td>
<td>91.7</td>
</tr>
<tr>
<td>CentroidNetV2</td>
<td>DLV3-RN50</td>
<td>No</td>
<td>VL-CE</td>
<td>91.4</td>
</tr>
<tr>
<td>MRCNN RN50</td>
<td>Default</td>
<td>Yes</td>
<td>VL-CE</td>
<td>91.0</td>
</tr>
<tr>
<td>MRCNN RN50</td>
<td>Default</td>
<td>No</td>
<td>VL-CE</td>
<td>91.5</td>
</tr>
</tbody>
</table>

Comparison to U-net backbone:

<table>
<thead>
<tr>
<th>Model</th>
<th>Backbone</th>
<th>Usage of pretraining</th>
<th>Loss</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>U-net</td>
<td>No</td>
<td>VL-CE</td>
<td>91.1</td>
</tr>
<tr>
<td>CentroidNetV2</td>
<td>U-net</td>
<td>No</td>
<td>MSE</td>
<td>90.6</td>
</tr>
<tr>
<td>MRCNN</td>
<td>Default</td>
<td>No</td>
<td>VL-CE</td>
<td>92.3</td>
</tr>
<tr>
<td>MRCNN</td>
<td>Default</td>
<td>No</td>
<td>MSE</td>
<td>91.5</td>
</tr>
</tbody>
</table>

In Figure 5.12 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.

![Figure 5.12: 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.](image)

From the literature is it well known that pretraining improves the performance of models (Erhan et al., 2010) 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.

The third part of Table 5.3 (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.

5.6.3 Results on bacterial colonies

On the bacterial-colonies dataset CentroidNetV2 achieves the overall highest F1 score of 92.6% shown in Table 5.4. YOLOv3 is the runner up 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 5.4 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 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 Figure 5.13. 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 Figure 5.13b 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 Figure 5.13c. The bottom row of Figure 5.13 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 Figure 5.13e 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).

\[ \text{(A) CentroidNetV2} \] \[ \text{(B) YOLOv3} \] \[ \text{(C) MRCNN} \]

\[ \text{(D) CentroidNetV2} \] \[ \text{(E) YOLOv3} \] \[ \text{(F) MRCNN} \]

**Figure 5.13**: 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.

## 5.7 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
TABLE 5.4: Results for counting bacterial colonies with 918 annotated validation samples.

<table>
<thead>
<tr>
<th>Model Backbone</th>
<th>Loss</th>
<th>F1</th>
<th>PR</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentroidNetV2</td>
<td>VL-CE</td>
<td>87</td>
<td>90</td>
<td>77</td>
<td>81</td>
<td>147</td>
<td>852</td>
</tr>
<tr>
<td>DLV3-RN101</td>
<td>VL-CE</td>
<td>89</td>
<td>91</td>
<td>81</td>
<td>89</td>
<td>100</td>
<td>852</td>
</tr>
<tr>
<td>VL-CE</td>
<td></td>
<td>88</td>
<td>90</td>
<td>81</td>
<td>82</td>
<td>53</td>
<td>885</td>
</tr>
<tr>
<td>U-net</td>
<td></td>
<td>83</td>
<td>83</td>
<td>51</td>
<td>83</td>
<td>886</td>
<td></td>
</tr>
</tbody>
</table>

Performance of CentroidNetV2 compared to YOLOv3 and MRCNN (in percentages).
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 overestimation 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 (92.4%). 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: 95.2%, 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 only case where YOLOv3 achieves the highest precision is on the aerial-crops datasets when annotations are made by a different expert, but in this case YOLOv3 is outperformed by CentroidNetV2 in terms of F1 score.

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 seems preferable. When accurate masks of objects should be determined with high recall then MRCNN seems 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. In the future research can be done to collect more experimental results and determine the statistical significance of the differences between the various algorithms.

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.

This chapter has shown that CentroidNetV2, a model for instance segmentation, achieves excellent results on three different datasets. These results are produced by a hybrid deep neural network that has been specifically designed for counting and segmentation of many small and connected objects.

5.7.1 Future work

Many applications exist for counting that are closely related to the research discussed in this chapter. 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 chapter 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.