Improving Object Detection with MatrixNets

by

Rishav Raj Agarwal

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Object detection is a popular task in computer vision with various applications, from pedestrian detection to face detection. Following the success of Convolutional Neural Networks (CNNs), many CNN based object detectors have been proposed to solve the object detection task. Early CNN based detectors suggested using deeper networks to detect objects in images. However, deeper networks cannot capture objects of varied sizes and aspect ratios with high accuracy. Thus, CNN-based detectors have two main challenges — scale invariance (detecting objects at multiple scales) and aspect-ratio invariance (detecting objects at various aspect ratios).

Modern CNN-based object detectors have two main components — a backbone network that learns features from an image and an output network that leverages these features to make predictions. Scale and aspect-ratio invariance are typically added by either making changes to the backbone or to the output network. Adding scale awareness to the output network is often computationally expensive. Thus, a popular method to add scale invariance by changing the backbone is Feature Pyramid Networks (FPNs) [31]. FPNs create a hierarchy of features at different scales and implicitly capture objects at various resolutions. Thus, FPNs are able to identify objects at different scales without the need to resize the input image.

However, FPNs have a square-bias and favour square objects over asymmetric ones. One solution to alleviate the square biasedness of FPNs is to add template anchor boxes of various sizes to add more bias towards non-square objects. However, anchor boxes are set as hyperparameters and add a computational overhead to the network. Newer architectures have thus moved towards anchor-free techniques; however, they still rely on FPNs, which are square-biased. Recently, MatrixNets [41] has been proposed as a general-purpose aspect-ratio aware extension of FPNs that can explicitly model aspect-ratios better than anchor boxes while keeping the model anchor-free. Matrixnets expands the FPN backbone by applies asymmetrically strided convolutions to create skewed receptive fields, making rectangular objects appear more square to the network While MatrixNets has been shown to improve keypoint based object detectors significantly, the implementation makes significant changes to the architecture, making it difficult to isolate the solo impact of MatrixNets.

In this thesis, we explore MatrixNets as a viable method to add aspect-ratio awareness. Specifically, we study MatrixNets along three axes — 1) Does MatrixNets make anchor-based detectors anchor-free. 2) Does MatrixNets add aspect-ratio awareness to object detectors, and 3) can MatrixNets be used for other, more complicated computer vision tasks

like instance segmentation .We explore these questions via three case studies. We demonstrate the effectiveness of MatrixNets by replacing anchor boxes in RetinaNet [32] with our MatrixNets module and showing better performance on skewed boxes while making the detector anchor-free. Then, we extend the anchor-free CornerNets [27] to x-CornerNet to support multiple output heads and smaller backbones. We then apply MatrixNets to x-CornerNet and demonstrate a similar improvement in skewed boxes leading to an overall 5.6% mAP improvement on MS COCO, achieving competitive results. Finally, we add MatrixNets to Mask RCNN [17] to tackle the instance segmentation tasks. While object detection draws bounding boxes delineating an object, instance segmentation goes one step further and draws pixel-wise masks delineating objects. This level of detail makes instance segmentation a more difficult problem to solve than object detection. We propose a new loss function, Mask Edge Loss (MEL), that leverages mask contours to reduce coarseness in predicted masks, thereby achieving higher accuracy. Together these three case studies demonstrate the effectiveness of MatrixNets for adding aspect-ratio awareness to object detectors. The code-base for our implementation will be made public.

Acknowledgments

I want to thank my supervisor, Dr. Lukasz Golab, for allowing me to work with him and continually guiding me throughout my master's degree. His Yoda-like approach of supervision has taught me to ask the right questions, then methodically search for the answer. I thank Dr. Abdullah Rashwan for coming up with MatrixNets and giving me free rein over it. I would also like to thank Dr. Robin Cohen and Dr. Srinivasan Keshav, for their excellent advice. In general, University of Waterloo has helped me foster great collaborations and grow as a researcher.

I would like to thank my roommates. Shubhankar Mohapatra, for always sharing coffee, Aseem Baranwal, for sharing my frustration when things didn't work out, and Manav Mehra, for sharing my happiness when something did. I would also like to thank Soham Mukherjee, Agastya Kalra and Suvi Agrawal for reading the first draft of this thesis and coming up with the lamest jokes.

Finally, I would like to thank my parents. They let me travel 10,000 miles and pursue my dreams even though they did not fully understand them.

Dedication

This thesis is dedicated to Ansel Adams and the photography club at IIT Kanpur, who kindled my love for photography and helped me "see".

P.S. No sheep were harmed in the making of this thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Object detection is the task of detecting instances of a class (or classes) of objects in images and videos and localizing each instance by drawing a bounding box around it. Object detection has a wide variety of applications, from tracking objects and video surveillance to pedestrian detection and face detection. Instance segmentation goes one step further and aims to delineate each object by computing a pixel-wise mask over the object. It may also be thought of as a pixel-wise classification of an image. Instance segmentation has various applications where individual objects need to be separated. For example, instance segmentation is useful in biomedical image analysis where individual organisms in cell cultures need to be separated and identified. Figure 1.1 shows examples of object detection and instance segmentation tasks. The object detection task shown on the left draws boxes bounding each object. In contrast, instance segmentation draws a precise pixel-wise mask.

With the success of AlexNet [26] in 2012, deep learning architectures have become a popular solution to the object detection task. The most widely used deep learning architectures rely on Convolutional Neural Networks (CNNs). CNNs are a class of Neural Networks, which draw inspiration from the biological visual cortex. Before we discuss CNN-based object detectors, we fist give some background necessary to understand deep learning architectures. Advanced readers may skip directly to Section 1.3.

1.1 Feedforward Neural Networks

Feedforward neural networks are a class of machine learning algorithms that approximate a function through a series of intermediate computational layers. Thus, given an input-output pair of $x, y : x \in X, y \in Y$, the network aims to approximate $\hat{y} = f(x, \Theta)$ where

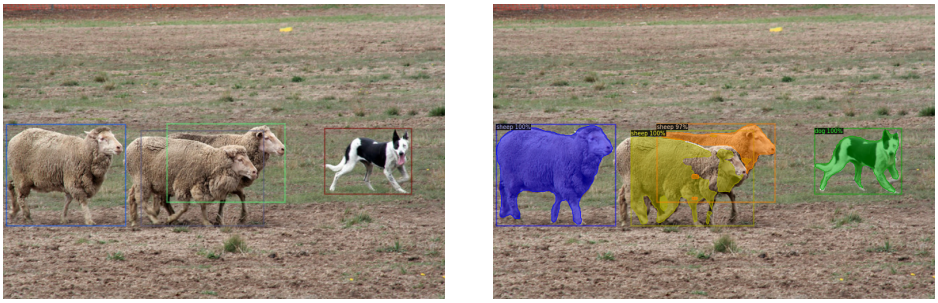


Figure 1.1: The object detection task is shown on the left and the instance segmentation task is shown on the right. The image is taken from the MS COCO dataset [33]

Θ parameterizes the layers of the network. The input is passed between the intermediate layers without any cyclical connections among them. The intermediary layers are simple functions that, when combined, approximate the function well. These intermediate layers are called hidden layers. These hidden layers are composed of interconnecting nodes or neurons, which are activation functions that take in a linear combination of input data via a layer weight vector W, add a bias b, and output values through a non-linear transformation (f_{act}). Thus, each neuron has the form $n = f_{act}(W^T X + b)$. b is called bias as it influences the output scores, but without interacting with the input data(X). Non-linear activations are used as they can capture complex relationships in the input data. Several f_{act} have been proposed based on the characteristics required by the network. A popular f_{act} used is Rectified Linear Unit or ReLU, given as $\text{ReLU}(k) = \max(0, k)$ where k is a real number. An example of a feed-forward neural network called a Fully Connected Network (FCN) is given in Figure 1.2. In the diagram, all neurons of a layer are connected to all the neurons in the previous layer.

The parameters of the neural networks are then optimized. We first define a loss function, which is a measure of how well the model is doing. In supervised learning tasks, typically used loss functions are the distances between the predicted $\hat{y}$ and actual output y. The choice of loss function depends on the type of input and the task at hand. For instance, in the case of multi-class classification, the input has to be assigned to one of the K output classes. The neural network outputs a probability distribution of each input belonging to the K classes. The most popular loss function used in such scenarios is the cross-entropy function which is defined as $CE(\hat{y}, y) = -\sum_{k=0}^{K} y_k log(\hat{y}_k)$. A popular optimization method is gradient descent, which minimizes the loss function by iteratively moving in the direction of steepest descent. In stochastic gradient descent, a random subset of the training dataset (called mini-batch) is used for calculating the gradients.

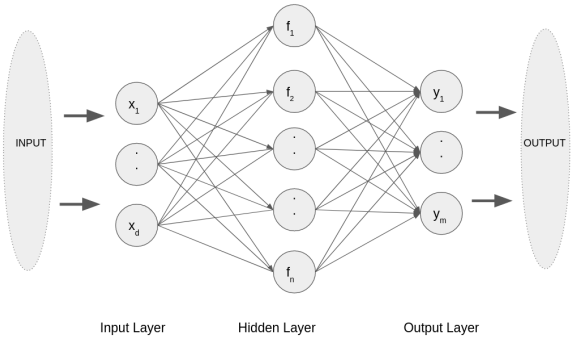
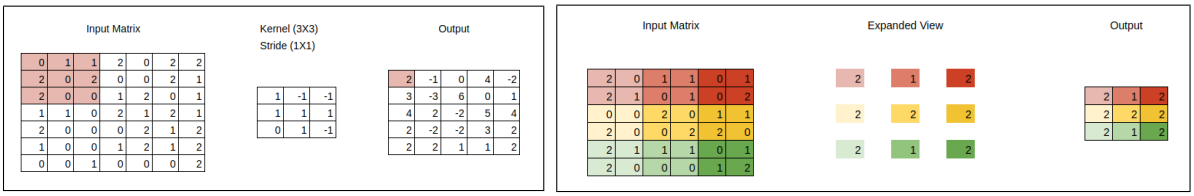


Figure 1.2: A simple Feedforward Neural Network with fully connected layers



(a) (b)

Figure 1.3: (a) Shows a Conv operation (3×3 filter 1×1 stride and 0 padding). Each pink 3×3 matrix in the input is multiplied element-wise with the filter and added together to compute one pink cell in the output matrix. (b) Shows a max pool operation (2×2 Maxpool). Each coloured region in the output is the max of a 2×2 region in the input of the same colour.

Modern Neural Network methods also add *batch normalization* (batchnorm) [20]. The weights of a neural network are usually initialized randomly. Moreover, the distribution of the input data may have some randomness as well. These sources of randomnesses affect the distribution of the inputs to each layer, and this effect is called internal covariance shift. Batchnorm adds a mini-batch-wise normalization step that fixes the means and variances of each layer's inputs. This normalization makes each hidden layer stable to internal covariance shift.

1.2 Convolutional Neural Networks

Images are represented as 3D matrices. Each dimension is called a channel and represents the three primary colours' intensities — red, blue and green. In order to learn information from images, we would require vast Fully Connected Networks. For instance, an image of size $64 \times 64 \times 3$ would need $12,288$ neurons to be present. Such a network would take a long time to train and would also overfit. In order to learn from images, we turn to Convolutional neural networks (CNNs). CNNs are a class of Neural Networks, which draw inspiration from the biological visual cortex. Neurons in a CNN are constrained, such that each neuron "looks" at only a small part of the input image. The region in the input which a neuron looks at is called its receptive field. CNNs are designed such that all the neurons in a layer do not have to be connected to all the neurons in the previous layer, i.e., they do not have to be fully connected. This property allows CNNs to have fewer parameters and be less prone to overfitting. Moreover, CNNs take advantage of hierarchical patterns in the images (e.g., pixels representing a line need to be close by) to learn complex patterns by assembling simpler ones. Thus, CNNs are well suited for learning features from images. CNNs are typically made up of convolutional layers, pooling layers and fully connected layers.

- The convolutional layers (*ConvLayer*) are the core building block of a CNN. A *ConvLayer*'s parameters consist of a set of learnable kernels or filters. Every filter is small spatially (along the width and height), but extends through the full depth of the input volume. The filters transform the image 3D volume to output 3D volume of activations, also called a feature map. This transformation is done by sliding the filter over the input volume's height and width and computing the dot product between each element of the filter and the input. Intuitively, the network will learn filters that activate when the network "sees" some visual features. Each point on the feature map is referred to as a *location* and is semantically equivalent to a pixel in an image. The "depth" of a Convlayer represents the number of filters used to learn various aspects of the input. To increase robustness, often CNNs have more channels than the input.

  ConvLayers generally share weights for each application of the filter. Hence, the number of parameters depends only on the filter size. An example of a 1-D *conv* operation is shown in Figure 1.3. Note, neurons in a ConvLayer are locally connected to a region in the input to reduce computational costs. Thus, the spatial extent of each neuron is constrained by the size of the filter used. For instance, if we use a filter of size 3×3, each neuron in the output looks at 3×3 area of the input. This

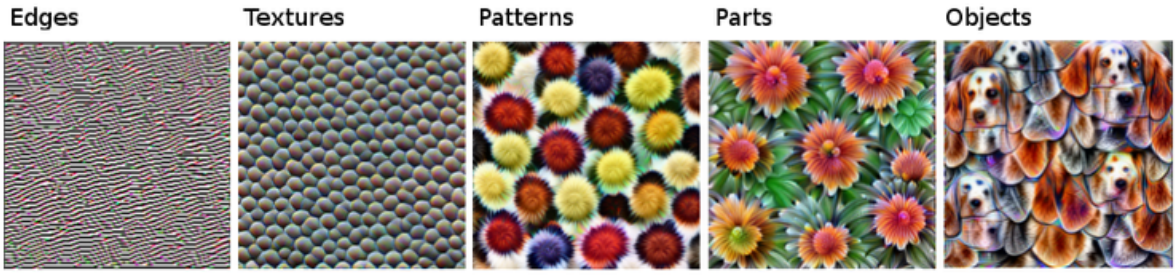


Figure 1.4: Examples of features learned by the various layers of a CNN trained on the ImageNet data. The features range from simple features in the lower layers of a CNN (left) to more abstract features in the higher layers of the CNN (right). Image taken from [37]

area is also called the *receptive field* of the neuron. Each filter has a predefined stride. The stride determines how many pixels are skipped when sliding the filter over the input. A stride of 1×1 goes over each pixel in the height and width, whereas a stride of 2×2 jumps 2 pixels when the filter is slid. Downsampling is whenever the output size reduces either by pooling or a ConvLayer. Similarly, upsampling pertains to increasing the size of output by using deconvolutional layers. Often the input is padded with zeros to keep the size of the input and the output the same.

- Pooling layers are added between ConvLayers to reduce the spatial extent of the representation and reduce the number of parameters to be learned. The pooling layer slides over the input volume and outputs the pooled value of the window. The pooling operation typically used is max pooling, so a Maxpool layer of size 2×2 will output the max over all the elements of the window of size 2×2 and reduce the input width and height by half. An example of max-pooling is shown in Figure 1.3.

- Finally, an FCN may be added towards the end to add more non-linearity to the network for better learning.

Some examples of features learned from images by the various layers of a CNN are given in Figure 1.4. As the receptive field is limited, different layers in a CNN learn different complexity of features. The first few layers learn basic features like lines and textures. In contrast, the later layers lean more complicated and often abstract features. Thus, the last layers of a CNN are said to have stronger semantics i.e. the features learned are more meaningful .

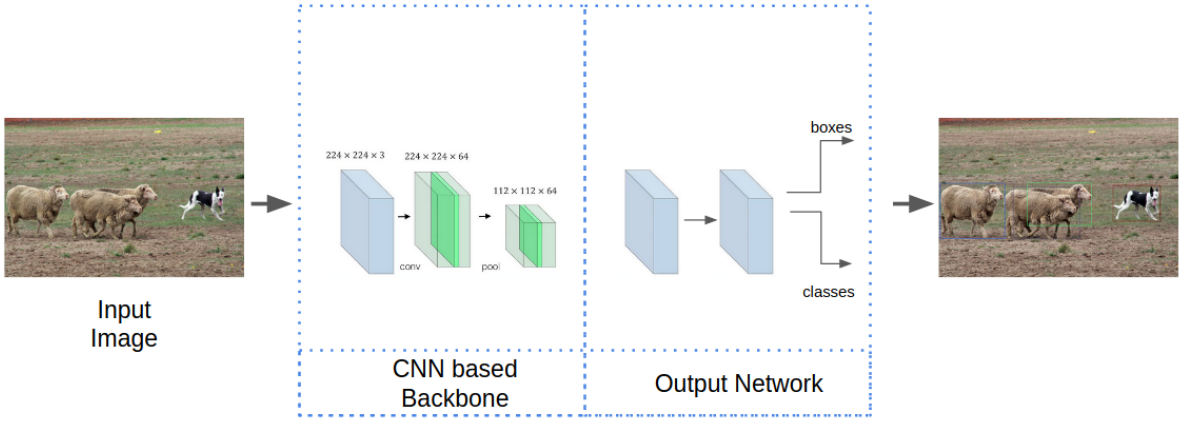


Figure 1.5: A high-level abstraction of a CNN based object detection architecture is shown. It has a CNN based backbone to extract features and an output network that makes predictions.

1.3 CNNs and Object Detection

A typical CNN based object detector uses a backbone network (also called *encoder*) to extract *features* from the input and create a *feature map*. An output network (also called *decoder*) processes the feature map and predicts classes and bounding boxes for the detected objects. Most of the time, Non-Maximum Suppression (NMS) is used as a post-processing step to remove overlapping boxes (NMS algorithm is available in Appendix A). A high-level representation of a CNN-based object detection architecture is given in Figure 1.5. With the availability of faster GPUs, features computed by CNNs have almost completely replaced the 'hand-engineered' features used previously for recognition tasks. An example of a hand-engineered feature would be to use the Canny Edge Detector [5] to explicitly find edges and then provide this information to the model rather than have it learn the feature. Initial CNN based deep learning models or architectures focused on deeper networks, i.e., adding more layers [18, 26]. However, as objects in an image can be of different scales (or sizes) and aspect ratios (as seen in Figure 1.6), deeper networks are not enough to achieve the best accuracy. This is because the feature maps are calculated at one resolution of the image. Hence, they often miss out on objects that may be very small. Thus, CNN based architectures have started to move towards architectures that are 1) scale-invariant and 2) aspect-ratio invariant.

Figure 1.6: Objects in the image on the left have different scales and have the same class (person). Objects in the right image are the same class but different aspect ratios. Images were taken from the COCO dataset [33]

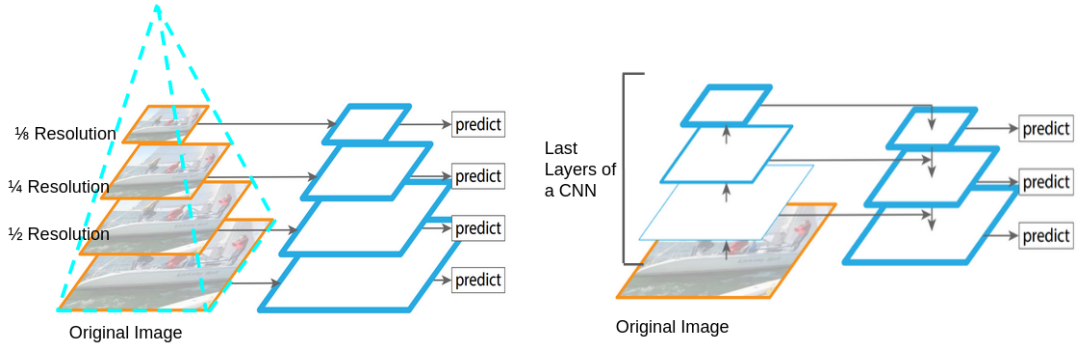


Figure 1.7: (Left) A featurized image pyramid where the image is scaled to various sizes, then features are computed independently at each scale and predictions are made. (Right) An FPN which uses the layers of a CNN to create a feature pyramid. Blue boxes show the last three layers of a CNN. In FPN, the last layer is upsampled twice, and weights from the preceding layer are added to each upsampling. This allows each layer of the FPN to access the weights from the last layer of the CNN, and thus having stronger semantics. The bold boxes indicate that the layer has the most robust semantics. The image is taken from [31].

1.3.1 Challenge 1: Scale Invariance

CNNs have two main components: the backbone and the output network. Scale invariance can be added by either changing the backbone or changing the output network.

Changing the Output Network

Early object detectors used featurized image pyramids (shown in Figure 1.7 (left) to effectively detect objects at various scales. Featurized image pyramids re-scale the images to smaller sizes, like levels in a pyramid, compute feature maps independently at each level and predict objects. This design choice makes featurized image pyramids scale-invariant, but the method is slow and computationally expensive. Thus, featurized image pyramids were only used at inference time and not for training deep learning models.

Changing the Backbone

With the advent of CNNs, researchers discovered that the various layers of a CNN naturally create a hierarchical feature map. The first few layers of a CNN hold simple features like lines and textures, and the later layers hold more complex patterns (as seen in Figure 1.4). The last layer holds the most meaningful or semantically strongest features (shown with bold blue in Figure 1.7 (left)). The next generation of deep learning architectures then aimed to achieve scale invariance by leveraging this inherent hierarchy and creating semantically strong feature maps from the layers of a CNN. Feature Pyramid Networks (FPNs) [31] (shown in Figure 1.7 (right)) proposed using the CNN hierarchy to create a feature pyramid such that semantically strong feature maps are available at multiple levels. Thus, FPNs add scale invariance by modifying the backbone. Now FPNs are the most popular backbone used in many new deep learning architectures (e.g., [46, 14, 4]) to learn objects of different scales better. While FPNs can detect objects at various scales well, they do not do so well with non-square objects as they have an inherent "square bias" (discussed in more detail in Section 2.1.1) in their receptive field.

1.3.2 Challenge 2: Aspect-ratio Invariance

Objects can not only be of different scales but also different aspect ratios. For instance, in the MS COCO dataset [33], 58% of boxes have a dimension ratio (ratio of maximum side/minimum side) greater than 1.5. More than 7% have a ratio higher than 3.5. This

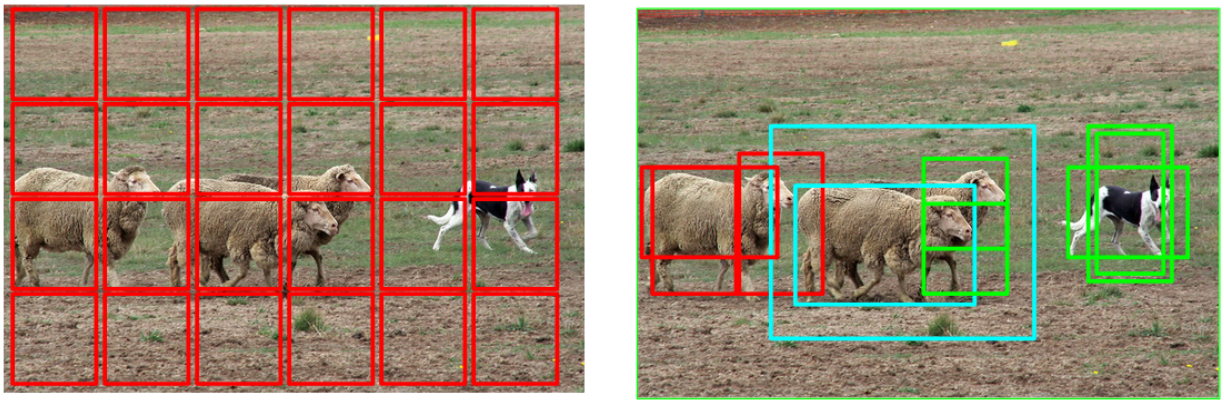
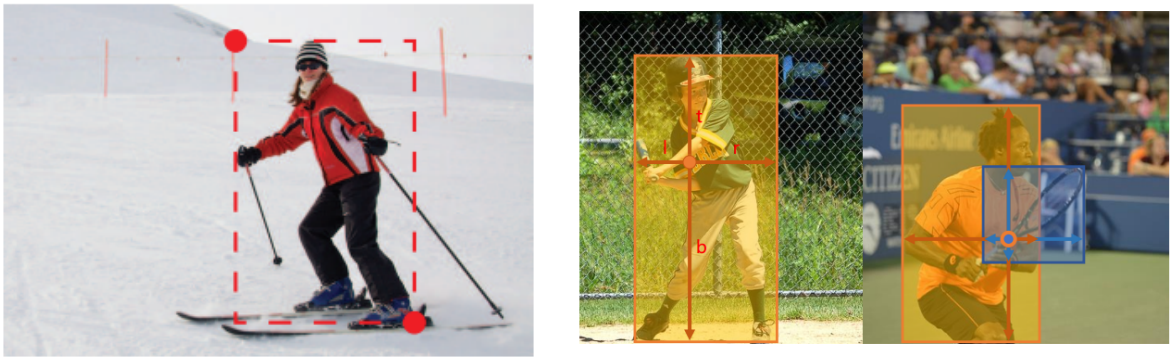


Figure 1.8: The image on the left shows some template anchor boxes, which are then refined by the network to obtain predictions as seen on the right. Note that the image on the left shows very few boxes for simplicity. In reality, there could be over 100K red boxes on the images.

issue is prominent in tasks like wildlife detection, where animals like giraffes have extremely skewed aspect ratios. In order to model objects of various aspect ratios, many FPN based methods use anchor boxes or templates in the output network. Like scale invariance, aspect-ratio invariance is also tackled by making changes to the backbone or the output network.

Changing the Output Network

Anchor boxes are preset boxes of various sizes and aspect ratios that are initially tiled over an image during training. Then the boxes are classified and refined to obtain the output detections. Figure 1.8 shows an example of anchor boxes. The boxes in red in the image on the left are examples of predefined boxes of a given size. Many such boxes of different heights and aspect ratios are drawn to cover the image densely. These boxes are then refined to obtain potential predictions (seen on the right). Anchor boxes can also be thought of as a way of providing training samples to the network. Thus, more anchor boxes of various sizes and aspect ratios mean more robust training samples. Different architectures may refine anchor boxes in different ways. For example, in Retinanet [32], the network predicts four offsets, one for each coordinate of the top left and bottom right corner of the anchor box, and a confidence score. The low confidence boxes are discarded. The offsets are applied to the surviving anchor boxes to obtain bounding box predictions. Anchor boxes have two main issues. (1) Many anchor boxes are needed to capture objects



(a) (b)

Figure 1.9: (a) A keypoint based object detector, Cornernet [27] is shown. It predicts corners, then groups them to bind the object. (b) A centre-based anchor-free method, FCOS [48] is shown. It directly predicts a 4D vector of distances from each pixel, treating it as the center. (l, t, r, b) are the (left, top, right, bottom) distances from each centre.

of varied shapes and sizes. For instance, Retinanet [32] requires $>$ 100K anchors per image. (2) They are set as hyperparameters, so researches have to make many parameter choices that are usually made via ad-hoc heuristics. Thus, there has been a paradigm shift among researchers towards anchor-free methods. Anchor-free methods work primarily in two ways:

1. They predict some "keypoints" and group them to localize the objects and thus predict bounding boxes. For example, Figure 1.9 shows predictions from Cornernet [27], which predicts corner points and groups them together.

2. They use the centre point of the object as an anchor point and then predict the four coordinates representing the top left and bottom right corners of the bounding box. This method is similar to having one anchor box. For example, FCOS [48] predicts a 4D vector encoding the location of a bounding box at each pixel (Figure 1.9).

Both centre-based and corner-based anchor-free methods rely on FPNs as their backbone. Even though they can model skewed boxes better, they still suffer from the "square-bias" of FPNs.

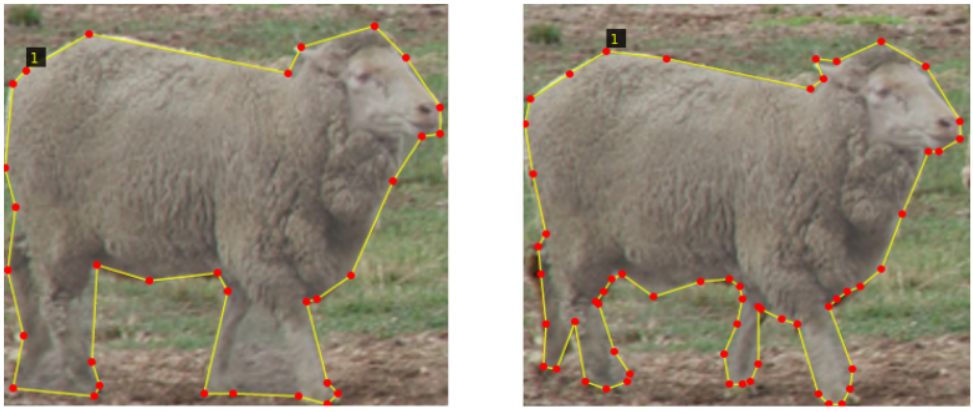


Figure 1.10: (a) The mask around the sheep is "coarse" as it misses fine details like the legs. (b) A finer mask that contours around the legs.

Changing the Backbone

Other aspect-aware architectures (e.g., [29, 10, 40]) aim to incorporate aspect-ratio awareness in the backbone itself. A recent framework, MatrixNets [40], generalizes FPNs by proposing unique "matrix layers" (more details in section 3.1) that capture asymmetric objects well. Moreover, as they inherently capture asymmetric aspect ratios, they can turn anchor-based methods anchor-free[1].

The rise of anchor-free methods highlights the need for implicit methods for modelling aspect-ratio awareness. A recent work, ATSS [58], explores differences between anchor-based and anchor-free methods and suggests algorithms to reduce the number of anchors required to achieve state of the art performance. However, ATSS still requires some hyperparameter to be set by the end-user. MatrixNets removes the need for any such hyperparameter by implicitly modelling aspect-ratios.

1.4 Thesis Goals and Contributions

FPNs have been studied in detail [14, 32, 14, 46] and applied to important computer vision tasks like instance segmentation and object detection. On the other hand, MatrixNets is a relatively new and understudied concept. Matrixnets has been shown to improve keypoint

[1]They still require one anchor per location, which is equivalent to centre-based anchor-free methods

based object detectors [41] significantly. However, the implementation makes significant changes to the architecture, making it difficult to isolate the solo impact of Matrixnets. Moreover, the claims of improvement in aspect-ratio awareness by eliminating the need for anchor boxes are not substantiated. Finally, the modular attribute of Matrixnets needs to be asserted, as well. The primary goal of this thesis is to study the impact of MatrixNets in more detail. We re-implement MatrixNets with several state-of-the-art architectures to understand its stand-alone impact. Specifically, we study MatrixNets along three axes:

- **Making anchor-based architectures anchor-free:** We first apply MatrixNets to an anchor-based architecture Retinanet [32]. We replace the FPN backbone in Retinanet with MatrixNets, and show that MatrixNets removes the need for anchor boxes of various sizes and aspect ratios, and leads to better performance.
- **Modelling aspect-ratio invariance:** We apply MatrixNets to an anchor-free architecture, Cornernet [27]. Cornernet is not compatible with FPNs, so we first propose x-Cornernet that allows FPNs to be used and then show the benefits of using MatrixNets, especially in modelling skewed aspect-ratio objects better.
- **Application to other Computer Vision tasks:** Instance segmentation goes one step further than object detection and predicts individual objects at the pixel level. As size and shape of the object become more important for instance segmentation, We explore MatrixNets for the instance segmentation task. We follow the Mask RCNN [17] architecture and improve it by proposing *Mask Edge Loss (MEL)* . MEL penalizes coarseness in masks (e.g., in Figure 1.10) and leverages the asymmetric receptive field of MatrixNets to improve mask predictions. We show that adding MatrixNets can again lead to improved performance without the need for additional anchor boxes. Moreover, MEL can also be combined with FPN based Mask RCNN for 0.3 mAP improvement in accuracy.

We choose Retinanet, Cornernet and Mask RCNN as they are the most popular methods for object detection and instance segmentation tasks. Several newer and more sophisticated methods have been built over these architectures. However, these remain the supreme example for each of the axes defined above. We believe that by showing the impact of MatrixNets with these simpler architectures, we can successfully isolate the impact of MatrixNets without any bells and whistles.

Neural Network architectures may have different implementations, so comparing them on the same baseline is difficult. Some of the performance gains in terms of inference time may depend on the implementation and software used. To level the playing field,

many software implementations (or frameworks) like MMdet [8], AdelaiDet [47], and Detectron2 [51] have come up which serve as general-purpose libraries for architecture implementation. They already have the popular architectures built-in, and newer architectures can easily be built on top of them. Using a framework allows for fair comparisons. It does not count gains, that may come from better software engineering practices. In this thesis, we choose Detectron2 as it is the most widely used open-source framework. We update the FPN already implemented in Detectron2 to MatrixNets and make minimal changes to the rest of the network to ensure fair comparison with previous work.

Finally, all experiments are done on the COCO dataset [33], which is the standard dataset used for benchmarking various object recognition tasks. The code for Detectron2 compatible MatrixNets will be made publicly available.

1.5 Thesis Organization

The subsequent chapters are organized as follows. We discuss the background and related work in Chapter 2. Then we discuss each architecture implementation in detail in Chapter 3, and we give the experiment results in Chapter 4. Finally, we conclude the thesis in Chapter 5.

Chapter 2

Related Work

In this chapter, we first discuss the recent work done to improve CNN based object detection. As a CNN-based object detector has two main components — the backbone, and the output network, the improvements in object detection come from improvements in either of these two components. We discuss work done to improve each component in more detail. Then, we discuss the recent work in the instance segmentation domain.

2.1 CNN based Object Detection

2.1.1 Backbones

CNNs use a backbone network to extract features from an image. Earlier methods like VGG [45] propose deeper architectures to improve performance. On the other hand, methods like Resnet [18] propose adding skip connections. Skip connections connect non-adjacent layers in a neural network, thus alleviate the problem of vanishing gradients [1] and aid in better learning. HourglassNet [53] proposes first downsampling convolutional layers then upsampling to learn global and local features in a unified structure. However, HourglassNets are computationally heavier than Resnets. These methods used only the last layer of the network to process outputs and thus do not work well with objects of various scales. One solution is to use features computed at multiple scales independently, but such a method is computationally expensive. Feature Pyramid Nets (FPNs) [31] offer a scale-invariant solution by calculating a feature pyramid directly from an underlying CNN. They use the layers from a Resnet [18] owing to its performance, lower complexity and flexibility. FPNs are also compatible with ResNext [52], ResNest [56] and Resnet-DCNs [10],

which are improvements over the basic Resnet model. To reduce training time, FPNs use a Resnet that has been pretrained on the Imagenet dataset.

FPN and Improvements

FPNs [31] introduced the idea of a top-down multi-scale fusion of features. They use the final layers $\{R_2, R_3, R_4, R_5\}$ (R_2 being the highest resolution) of a Restnet model and fuse them to obtain $\{P_2, P_3, P_3, P_4, P_5\}$ as outputs. The fusion of layers typically means that they are either added or averaged together. Top-down fusion happens in the following manner: We upsample R_5 and add R_4 to obtain P_5. Next, we upsample P_5 and add R_3 to obtain P_4. Similarly, we obtain P_3 and P_2. As the fusion happens from the smallest to the largest layers, we call it top-down fusion. Each output is a $2\times$ downsample along the width and height of the previous output layer. This step essentially doubles the receptive field of the layer, but the receptive field remains square. Since the original FPN paper, there have been many improvements in the fusion methodology of FPNs. Retinanet [32] modified the FPN by further downsampling P_5 to create smaller P_6 and P_7. These layers are better at predicting smaller objects owing to the larger receptive fields. PA-Net [34] added a bottom-up pathway to the original fusion method. NAS-FPN [14] performed a neural architecture search to produce a complex fusion graph. EfficientDet [46] proposed BiFPN, which learns the importance of each input feature map, and then fuses features based on importance. SpineNet [21] permutes the connections in the backbone instead of following a top-down or bottom-up fusion paradigm. Finally, DetectroRS [39] adds recursive calls to the bottom-up FPN backbone. All of the above methods still follow symmetric downsampling when computing the output layers. They thus suffer from square biasedness in the receptive field. A few examples of various fusion techniques are shown in Figure 2.1. MatrixNets [40] is the first to generalize FPNs by allowing asymmetric strides. MatrixNets adds asymmetric downsampling (more details in Section 3.1) and follows a top-down fusion method like the original FPN paper. As the diagonal of MatrixNets acts as an FPN, the different fusion methods can be applied to MatrixNets as well.

2.1.2 Output Methods

The output method is perhaps the most critical part of modern CNN based methods. The output method is where the core idea of the algorithm works — the features are processed, and loss functions are calculated. Two main types of output methods exist, anchor-based and anchor-free.

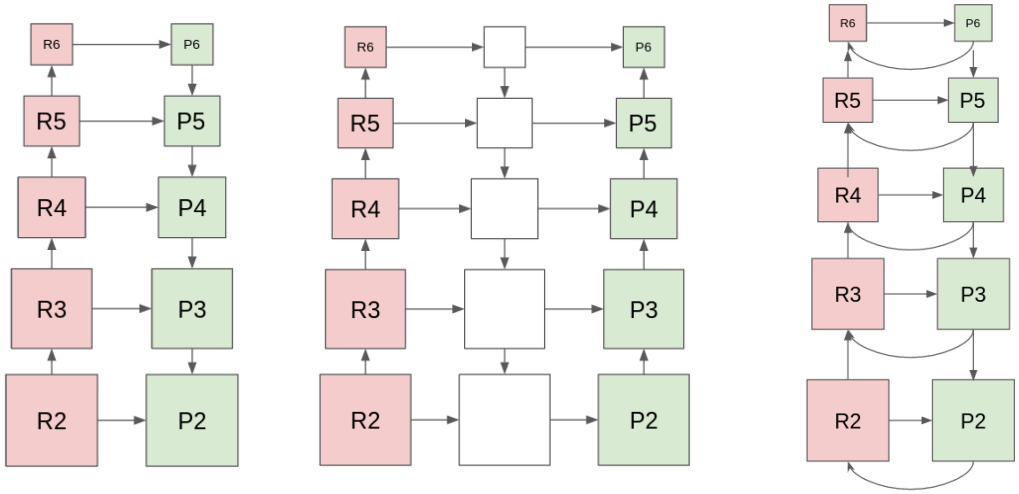


Figure 2.1: (a) FPN [31] introduces a top-down pathway to fuse multi-scale features; (b) PANet [34] adds a bottom-up pathway on top of FPN; (c) DetectoRS [39] adds a recursive connection to the top-down FPN

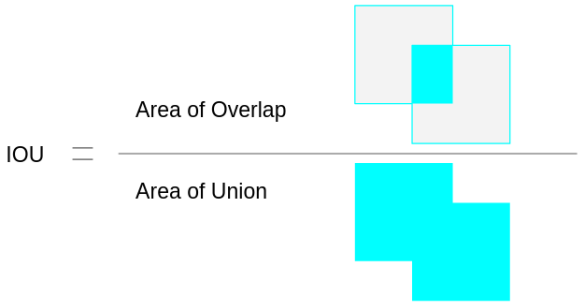


Figure 2.2: The formula for IoU

Anchor-Based

Anchor-based methods use predefined anchors at each position of the feature map and then refine them to obtain bounding box predictions. All anchor-based methods are either one-stage or two-stage. Most two-stage architectures are based on Faster-RCNN [43], which consists of two sub-networks: the first sub-network, called Region Proposal Network (RPN), proposes Regions of Interest (ROIs). The second sub-network classifies and regresses the RoIs to get final predictions. One-stage methods directly classify and regress bounding boxes without waiting for proposals from an RPN. The most popular single-stage method is Retinanet, after which many improvements have been proposed. FreeAnchor [60] improves Retinanet by improving anchor box to ground truth box matching during train time. MAL [22] improves anchor selection during train time. YOLO [3] is another popular one-stage method, but they make large speed-accuracy tradeoffs. ATSS [58] tries to reduce the number of anchors required for prediction by using a specialized training sample selection mechanism based on the standard deviation of the IoU (Figure 2.2) of the anchors. PPA [23] first learns the distribution of samples and then draws potential anchor locations from this distribution.

Anchor-Free

Anchor-free methods either predict keypoints and group them to localize each object or calculate the "centreness" of each pixel on the image (guided by the availability of ground truth) and then regress a bounding box around it. Cornernet [27] was the first method to use top-left and bottom-right key points for box localization rather than anchors. Centernet-HG [61] and Centernet [12] both extend this by predicting a centre key point as well. Centripetalnet [11] further improves Cornernet by predicting the offset of the centres from the corners and proposes a better corner matching algorithm. All of these methods use the heavy Hourglass [53] backbone, making them difficult to train and deploy. Other anchor-free methods such as FCOS [48] and FSAF [64] predict a centre and regress box coordinates per output. More recently, a fast, anchor-free two-stage architecture, RepPoints [54], has shown the best performance. All of these methods again rely on FPNs and suffer from a square biased receptive field, which MatrixNets alleviates. Our implementation of Cornernet with MatrixNets as the backbone further highlights this point in Section 3.1.

Loss Functions

Some researchers have proposed newer loss functions to improve performance. Retinanet [32] proposes a new algorithm "focal loss" to balance negative ROI samples. GIoULoss [44] incorporates the IoU metric into the loss function for better improvement. These loss functions are typically implemented in the output layer often do not interfere with the backbone. Thus, MatrixNets should be compatible with them, as well.

Aspect-Ratio Awareness

Anchor-based methods [43, 32, 60] generally use skewed anchor boxes to represent objects with different aspect ratios. However, they are still limited to anchor boxes for explicit aspect-ratio modelling. ME-RCNN [29] creates multiple second stage networks for predicting different aspect ratios. However, this only applies to two-stage anchor-based models, does not modify the receptive field, and has fallen out of favour. Anchors are hyperparameters, so the community is moving towards hyperparameter free methods that implicitly model aspect ratios.

Deformable Convolutions [10] create backbones that have adaptive receptive fields in their convolution kernels, which implicitly improves aspect-ratio awareness. MatrixNets [40] explicitly encodes aspect-ratio awareness by adding asymmetrically downsampled layers to the FPN backbone. However, more experimentation is needed to isolate the impact of MatrixNets on accuracy.

2.2 CNN based Instance Segmentation

Instance segmentation architectures can be divided into one-stage and two-stage methods. Two-stage detectors split the task into detection and segmentation. The detection task offers some template boxes over which masks are segmented. The first breakthrough two-stage detector that achieved high accuracy on the instance segmentation task was Mask-RCNN [17]. Mask RCNN is built over Faster RCNN [43] and adds a mask branch parallel to the bounding boxes and classification branches. Several architectures like Mask Scoring RCNN [19] and Hybrid Task Cascade (HTC) [7] improve over the Mask-RCNN architecture by either improving the underlying backbone or creating pathways between the mask and detection branch. However, these methods suffer from the main drawback of Mask-RCNN, which depends on anchor boxes. In anchor-free methods, Centermask [30] is one new architecture built upon FCOS [48], which is an anchor-free one-stage object

detector. Centermask achieves this by adding a Spatial Attention Guided (SAG) mask branch over FCOS. Incorporating MatrixNets in Mask RCNN can benefit all the methods mentioned above by adding aspect-ratio awareness and improving accuracy. Moreover, MatrixNets can reduce the number of anchor boxes required, as well.

One the other hand, one-stage methods directly predict masks without the need for detection first. Architectures like BlendMask [6], EmbedMask [55], SOLO [50], YOLO [42] and RDSNET [49] are one-stage but they are unable to compete with MaskRCNN in term of accuracy. A recent method, MEinst [57], achieves comparable performance to Mask RCNN by encoding a one-dimensional mask vector instead of 2D representations predicted by Mask RCNN. However, MEInst (and many other one-stage methods) uses an FPN based backbone. We posit that MatrixNets can replace FPNs in all these FPN based one-stage detectors and lead to better accuracy.

Chapter 3

Methodology

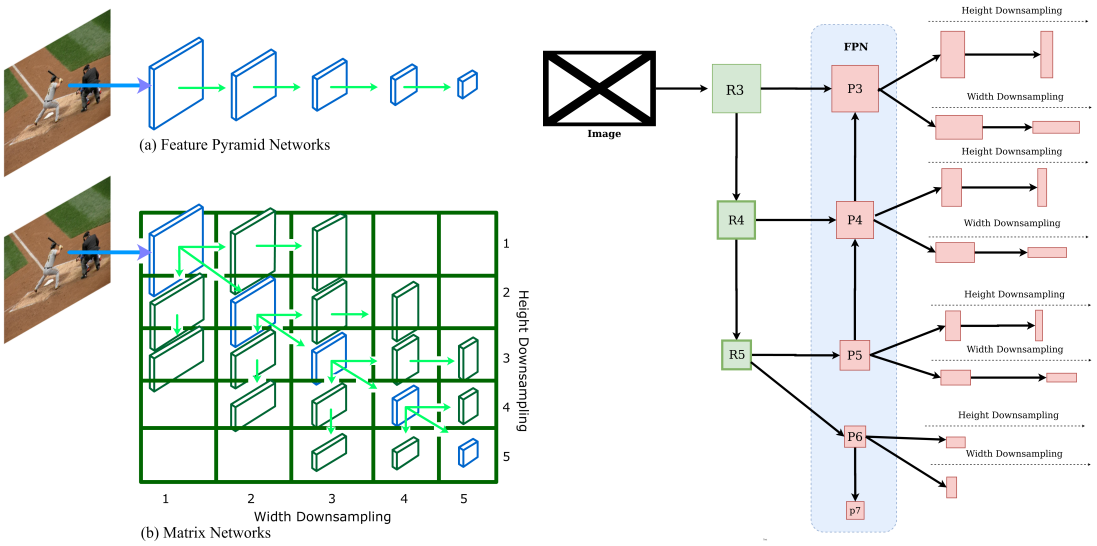
In this chapter, we first discuss the architectural details of MatrixNets. We then present three case studies. The first case study, x-Retinanet, is created by replacing the FPN in Retinatnet by MatrixNets. x-Retinanet shows that MatrixNets can make anchor-based detectors anchor-free with no changes to the output network. We then present x-Cornernet, an adaptation of Cornernets that works better with skewed boxes. Finally, we present x-Mask RCNN, an anchor-free version of Mask-RCNN that uses a MatrixNets backbone. x-Mask RCNN has a unique component, Mask Edge Loss (MEL), that reduces coarseness in predicted masks. Together, these three implementations demonstrate that MatrixNets is a general-purpose module that adds implicit aspect-ratio awareness to object detectors. This awareness can also make anchor-based detectors anchor-free without adding more parameters to the network. Some text in this section is taken from our previous work [40]. We provide a table of notations in Table 3.1 that we will follow throughout this chapter.

3.1 MatrixNets

FPNs, as described in [32], use the final layers $\{R_3, R_4, R_5\}$ of a Restnet model and fuse them in a top-down fashion to create five outputs $\{P_3, P_4, P_5, P_6, P_7\}$. Layer P_i has resolution 2^i lower than the input. Let us imagine a 5×5 matrix of *ConvLayers* with each layer represented by $l_{i,j}$ (as shown in Figure 3.1a). Then the FPN can be thought of like the diagonal of this matrix such that $\{P_3, P_4, P_5, P_6, P_7\}$ of the FPN correspond to $\{l_{1,1}, l_{2,2}, l_{3,3}, l_{4,4}, l_{5,5}\}$ of the matrix. At each level of the FPN, we add a width downsampling of 2^{i-1} and height downsampling of 2^{j-1} to obtain MatrixNets. Thus, the diagonal

Notation	Definition
(H_{inp}, W_{inp})	Height and width of the input image
(H, W)	Height and width of the output layer of the backbone
(h_r, w_r)	Height and width of a Region of Interest
K	Number of categories used for classification
m	Size of the downsampled mask for Mask RCNN
P_x	Layer number for FPN
R_x	Layer number for Resnet
$l_{i,j}$	Layer number for MatrixNets
A	Number of Anchors used in Retinanet and MaskRCNN

Table 3.1: Table of notations used in Chapter 3, unless stated otherwise.



(a) (b)

Figure 3.1: (a) The MatrixNet architecture with the matrix analogy taken from [40] (b) The MatrixNet architecture re-drawn to show all connections.

layers are square layers of different sizes, while the off-diagonal layers are rectangular layers, unique to MatrixNets. Matrix-layer $l_{1,1}$ is the largest layer in size and is called the base layer. Every step to the right reduces the width of the layer by half, while every step-down reduces the height by half. For example, $\text{Width}(l_{3,4}) = 0.5\text{Width}(l_{3,3})$. Diagonal layers model objects with square-like aspect-ratios because they have square receptive fields. Off-diagonal layers model objects with non-square aspect-ratios since they have skewed receptive fields. Layers close to the top right or bottom left corners of the matrix $\{l_{1,4}, l_{1,5}, l_{4,1}, l_{5,1}, l_{5,2}, l_{2,5}\}$ model objects with very skewed aspect-ratios ($\sim 1:8$). Such objects are scarce so that these layers can be pruned for efficiency. In Figure 3.1b, we redraw MatrixNets to show the various hierarchical connections more clearly. The pale blue rectangle shows an FPN. Asymmetric layers are added to each layer in the FPN to obtain MatrixNets. Note that we can use different FPN designs (e.g., from [43, 35, 32]) to create MatrixNets as described above.

3.1.1 Parameters

MatrixNets introduces as few new parameters as possible when following the Retinanet FPN. The upper triangular layers, i.e layers $l_{i,j}$ where, $i > j$, are obtained by applying a series of shared 3×3 convolutions with stride 1×2 on the diagonal layers. Bottom layers $l_{i,j}$ where $i < j$ are obtained using shared 3×3 convolutions with stride 2×1. This sharing helps reduce the number of additional parameters introduced by the matrix layers. Each layer has a standard batchnorm and ReLU. The addition of the off-diagonal layers usually adds as many parameters as multiple anchor boxes. For example, both anchor boxes and MatrixNets add $\sim$ 1M parameters to Retinanet.

3.1.2 Advantages of MatrixNets

The key advantage of MatrixNets is that they allow a square convolutional filter to accurately gather information about different aspect-ratios. In anchor-based object detection models, such as Retinanet, a square convolutional filter is required to output boxes of different aspect-ratios and scales. Using a square convolutional filter is counter-intuitive since boxes of different aspect-ratios and scales require different contexts. In MatrixNets, the same square convolutional filter can be used for detecting boxes of different scales and aspect-ratios since the context changes in each matrix layer. $l_{1,2}$ can be viewed as an alternative to a 1:2 anchor box on $l_{1,1}$, where the receptive field matches the targeted aspect-ratio. Note that custom anchor boxes like a 1:3 anchor box can be represented by using fractionally strid convolutions, so in this case, with a 1:1.5 stride applied to $l_{1,2}$.

We illustrate how matrix layers imitate anchor boxes. In anchor-based object detectors, different ground-truth boxes are assigned to the closest anchor boxes. For the purpose of this demonstration, we fix the range of widths and heights of objects assigned to each layer in the matrix. We set the range of width (W) and height(H) for the base layer to be $H = [24px, 48px], W = [24px, 48px]$. Based on this range, we use the MatrixNets definition to generate the range of heights and widths for all the other layers. A step to the right doubles the width range, and a step-down double the height range. We plot the distribution of boxes assigned to layers grouped by the receptive field in Figure 3.2. We compute these by going through a training subset of MS COCO, assigning each box to its appropriate layer based on its dimensions, creating a histogram for each layer, and grouping each histogram by receptive field. We group layers with receptive fields of 2:1 and 1:2 into the same group since we plot a histogram of dimension ratios (ratio of the maximum dimension by the minimum dimension). Our assignment mechanism assigns boxes with large dimension ratios (skewed aspect-ratios) to layers with similarly skewed receptive fields, allowing for explicit targeting of aspect-ratios. Since object sizes are nearly uniform within their assigned layers, the dynamic range of the widths and heights is smaller compared to other architecture such as FPNs. Hence, regressing the heights and widths of objects becomes an easier optimization problem. Finally, MatrixNets are general and can be used with any output method.

3.1.3 Dimension Ratio AP

The COCO dataset challenge[1] defines 12 metrics given by the first 3 blocks of Table 3.2. Precision and recall are calculated at a specific IoU threshold for the top 100 scoring detections from the model. AP is the precision averaged over all the categories in the dataset. mAP is the AP averaged over 10 IoU thresholds selected uniformly from 0.5 to 0.95. APs, APm, APl are mAP calculated for small, medium and large objects. AR is the recall averaged over all the categories in the dataset. AR1, AR10, and AR100 are ARs calculated by taking the top 1, 10 and 100 detections from the model and averaged over 10 IoU thresholds selected uniformly from 0.5 to 0.95. These metrics capture precision and recall at various scales, but such metrics are unavailable for aspect-ratios. We add four new metrics to capture the precision of predicting skewed objects. We define the *dimension ratio* as the ratio of maximum dimension to the minimum dimension of the object. We then bin dimension ratio in 4 categories as given in the last block of the table and calculate the AP.

[1] cocodataset.org/.

Average Precision (AP)	
mAP	AP over IoU range [.50, .95]
AP50	AP at IoU=.50
AP75	AP at IoU=.75
AP Across Scales	
APs	AP for small objects: area $< 32^2$
APm	AP for medium objects: $32^2 <$ area $< 96^2$
APl	AP for large objects: area $> 96^2$
Average Recall (AR)	
AR1	AR given 1 detection per image
AR10	AR given 10 detections per image
AR100	AR given 100 detections per image
AR Across Scales	
ARs	AR for small objects: area $< 32^2$
ARm	AR for medium objects: $32^2 <$ area $< 96^2$
ARl	AR for large objects: area $> 96^2$
AP Across Dimension ratios (Ours)	
$1 - 1.5$	AP for objects with dimension ratio $1 - 1.5$
$1.5 - 2.5$	AP for objects with dimension ratio $1.5 - 2.5$
$2.5 - 3.5$	AP for objects with dimension ratio $2.5 - 3.5$
3.5+	AP for objects with dimension ratio 3.5+

Table 3.2: Metrics used for evaluation. AP is the precision averaged over all categories (80 in MSCOCO) present in the data. Dimension ratio is computed as $max(h, w)/min(h, w)$ where h, w are the height and width of the object.

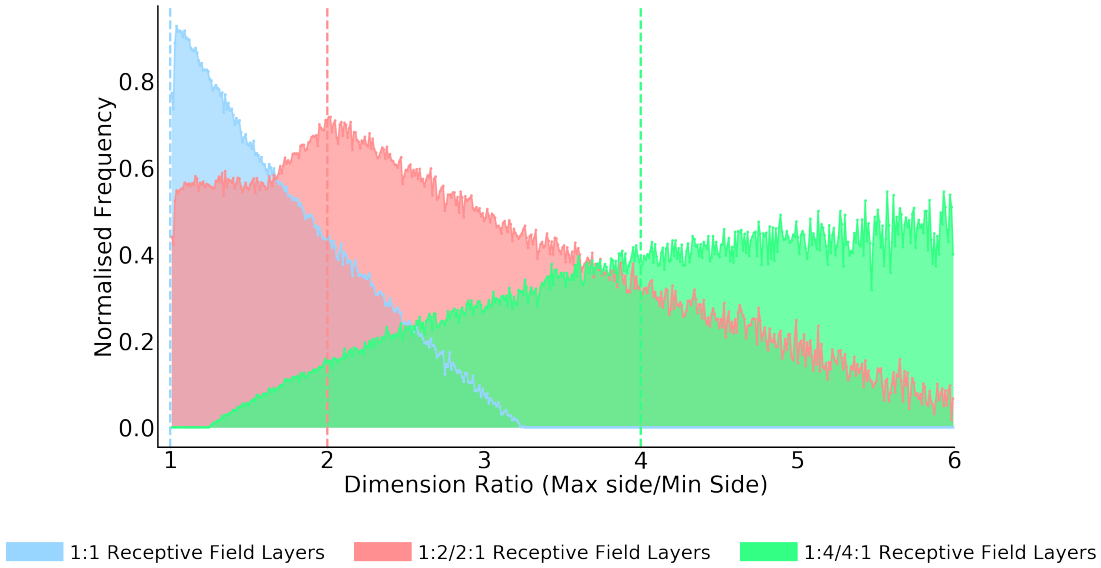


Figure 3.2: Normalized frequency of boxes assigned to matrix layers with different receptive fields. We see that matrix layers with a square (1:1) receptive field have mostly square boxes assigned. Matrix layers with 1:2 receptive fields have mostly boxes with a dimension ratio from 1.5-3. Finally, the most skewed matrix layers with a receptive field of 1:4 have mostly boxes with a dimension ratio 3.5+.

3.2 Retinanet

Retinanet [32] is a one-stage architecture that takes in an input image, uses anchor boxes as templates for objects, and outputs bounding box offsets and class prediction for each template. Predictions are obtained by applying the offsets to the template boxes and are then ranked by the model confidence score. Non-Maximum Suppression (NMS) is applied to eliminate duplicate predictions. The top-scoring bounding boxes that survive NMS are used as the final predictions from the model.

3.2.1 Architecture details

Retinanet consists of a backbone network and two task-specific sub-networks. The backbone is an FPN based on R_2 to R_5 layers of Resnet. The backbone computes a convolutional feature map over the input image. The first sub-network performs object classifi-

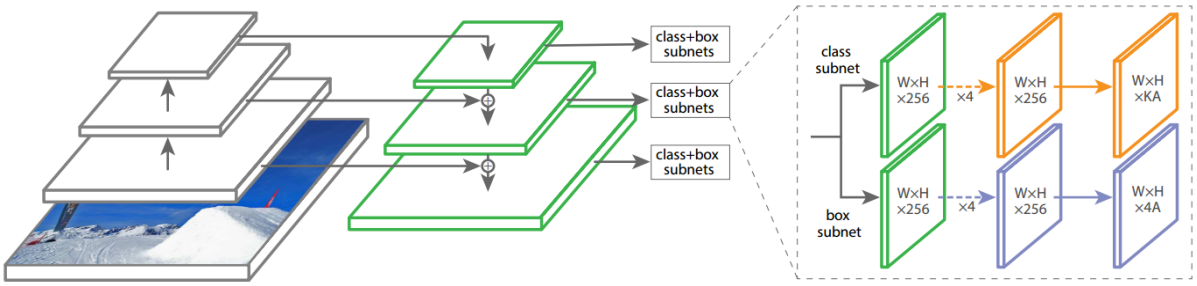


Figure 3.3: The Retinanet architecture taken from [32].

cation, and the second one performs bounding box regression and outputs bounding box offsets. The architecture is shown in Figure 3.3.

Backbone

Retinanet uses an FPN backbone (previously discussed in Section 2.1.1). Retinanet's FPN is built over the Resnet architecture, and the pyramid is constructed from level R_3 to R_5 to obtain P_3 to P_5 in the FPN. P_5 is further downsampled twice to obtain to P_6, P_7 layers. The layer P_x has resolution 2^x times than the input. All pyramid levels have 256 channels.

Anchors

Retinanet uses predefined anchor boxes of sizes 32^2 to 512^2 on pyramid levels P_3 to P_7. At each level, anchors at 3 aspect-ratios $\{1:2, 1:1, 2:1\}$ and 3 sizes $\{2^0, 2^{1/3}, 2^{2/3}\}$ are used. Thus each level has $A = 9$ anchors per location and these cover a scale range of $32 - 813$ pixels with respect to the input image size.

Each anchor box is assigned a K one-hot vector of classification targets (where K is the number of classes). The assignment is done based on Intersection over Union (IoU) with the ground truth bounding box. Matches with IoU greater than 0.5 are labelled as positive samples, IoU between $[0.4, 0.5)$ are ignored, and rest are assigned to the background class. Each anchor box can be assigned at most one ground truth box based on the maximum IoU. Bounding box regressions are computed as the offsets between the ground truth and the assigned anchor box.

Classification Sub-network

The Classification Sub-network predicts the probability of an object's presence for each of the anchor boxes and K classes. The sub-network is an FCN composed of four 3×3 ConvLayers, each with C filters (C being the number of channels in each pyramid layer), followed by ReLU activation and ending with a 3×3 Conv with $K \times A$ filters. Finally, a sigmoid activation is attached to obtain $K \times A$ binary predictions per spatial location. This sub-network is attached to each FPN level; parameters of this sub-network are shared across all the pyramid levels.

Regression (Bbox) Sub-network

The Regression Subnet is a parallel FCN connected to each pyramid level and used to regress the offsets from each anchor boxes to the nearby ground truth. The architecture is similar to the Classification Subnet. However, it terminates in $4 \times A$ (A is the number of anchor boxes) coordinate offsets per anchor box. Note that the regression sub-network is class agnostic, i.e., it predicts bounding box offsets without classification information. The regression loss (L_{reg}) is added to the loss.

Focal Loss

Retinanet uses Focal loss for classification, which is a variant of the cross-entropy loss and is defined as:

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

where

$$p_t = \begin{cases} p, & \text{if } y = 1. \\ 1 - p, & \text{otherwise.} \end{cases} \tag{3.1}$$

$y \in \pm 1$ specifies the ground truth class (1 for positive, -1 for negative), $p \in [0, 1]$ specifies the model's estimated probability for the class with label $y = 1$, and α_t is the weighting factor that penalizes heavy class imbalances. When $\gamma = 0$, FL reduces to the cross-entropy loss. As γ increases, easy examples are discounted from the loss. Thus, Focal loss works better when there are large class imbalances between the foreground and background classes.

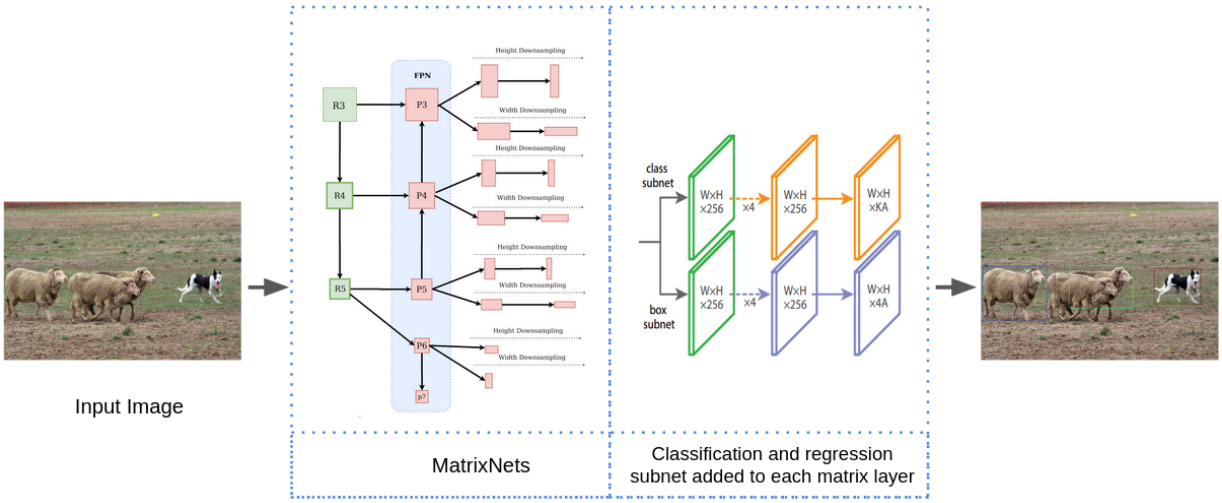


Figure 3.4: The x-Retinanet architecture .

Thus, the total training loss is:

$$L_{\text{total}} = \beta L_{reg} + FL \tag{3.2}$$

Here, β is a hyperparameter. Retinanet uses SGD to optimize this loss function.

3.2.2 Our Implementation: x-Retinanet

We follow the same architecture as Retinanet with two modifications.

1. We replace the FPN backbone with MatrixNets. Our MatrixNets uses the P_3 to P_7 as the diagonal, and the off-diagonal layers are computed by downsampling along the width and height as discussed in Section 3.1.

2. We have one anchor per location ($A = 1$). The classification and regressions sub-networks receive outputs from all 19 layers of MatrixNets.

Figure 3.4 shows the x-Retinanet architecture.

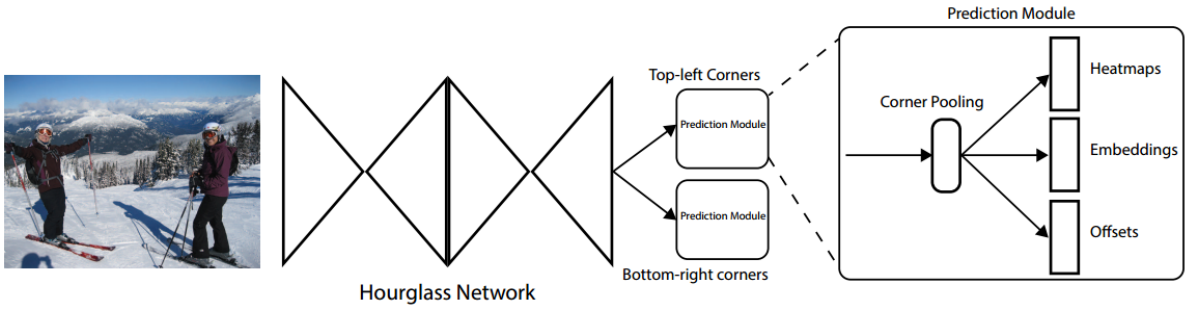


Figure 3.5: The Cornernet architecture taken from [27]

3.3 Cornernets

Cornernet [27] detects objects like a pairs of keypoints. The network uses an Hourglass [53] network as the backbone followed by two prediction sub-networks, one to predict the top-left corner and the other for the bottom-right corner. Each sub-network has a corner pooling layer that localizes corners. It then predicts heatmaps, embeddings and offsets. Cornernets does not use features of different scales and only uses the single output of the hourglass network. Figure 3.5 shows the Cornernet architecture.

3.3.1 Architecture details

Corner Pooling

Predicting corners for objects is challenging as the corners may lie outside the object. For example, a bounding box for a circle has corners that lie outside the object. Thus, there is not enough local information available to predict corners. Cornernet uses a corner pooling layer to localize corners. It takes the feature map and finds the top-left (or bottom-right) candidates by pooling horizontally (or vertically) along the top-left (or bottom-right) direction in each feature map.

Corner Detection

A CNN then predicts two sets of heatmaps, one for the bottom-right and another for the top-left corners. Each heatmap is of size $H \times W$ and has K channels where K is the

number of categories, and (H, W) is the size of the output from the hourglass network. Each channel is a binary mask indicating the location of the corners. For each corner, there can only be one ground truth positive location, and the rest are negative. During training, the penalty for a negative location is reduced based on the distance from the ground truth assignment. This is done as the authors assume that a pair of false corners close to the ground truth can still produce a valid set of corners. An unnormalized 2-D Gaussian $e^{\frac{-x^2+y^2}{2\sigma^2}}$ (σ is 1/3 of the size of the object) centred at the positive location is used to reduce the penalty.

The loss function for classification error is a specialized form of focal loss run over every location $\{(i, j)|i \in [1...H], j \in [1...W]\}$ on the heatmap for every class $k \in [1...K]$.

$$CFL = -\frac{1}{N}\sum_{k=1}^{K}\sum_{i=1}^{H}\sum_{j=1}^{W}\begin{cases}(1-p_{kij})^{\alpha}\log(p_{kij}), & \text{if } y_{kij}=1.\\ (1-y_{kij})^{\beta}p_{kij}^{\alpha}\log(1-p_{kij}), & \text{otherwise.}\end{cases} \tag{3.3}$$

$y_{ijk} \in \pm 1$ specifies the ground truth class (1 for positive, -1 for negative), $p_{ijk} \in [0, 1]$ specifies the model's estimated probability for the class k at location (i, j) with label $y = 1$, N is the number of objects and α, β are hyperparameters that encode the contribution of each point to the loss.

Corner Matching

The top-left and bottom-right corners of an object are grouped by using associative embedding as described in [36]. The network predicts an embedding vector for each detected corner. The embedding is constrained such that the distance between the corners' embeddings should be small if a top-left corner and a bottom-right corner belong to the same bounding box. Cornernet defines two embedding losses — L_{pull} to train the network to group corners, and L_{push} to train the network to separate corners. Embedding losses are then added to the total loss.

Offsets

As the input image is resized to the size of the output layer of the CNN backbone (which is usually smaller than the input size to save memory), the network's predictions need to be resized to the original image size. As precision is lost when resizing, the corners are slightly adjusted before mapping to the input resolution. The network thus predicts two

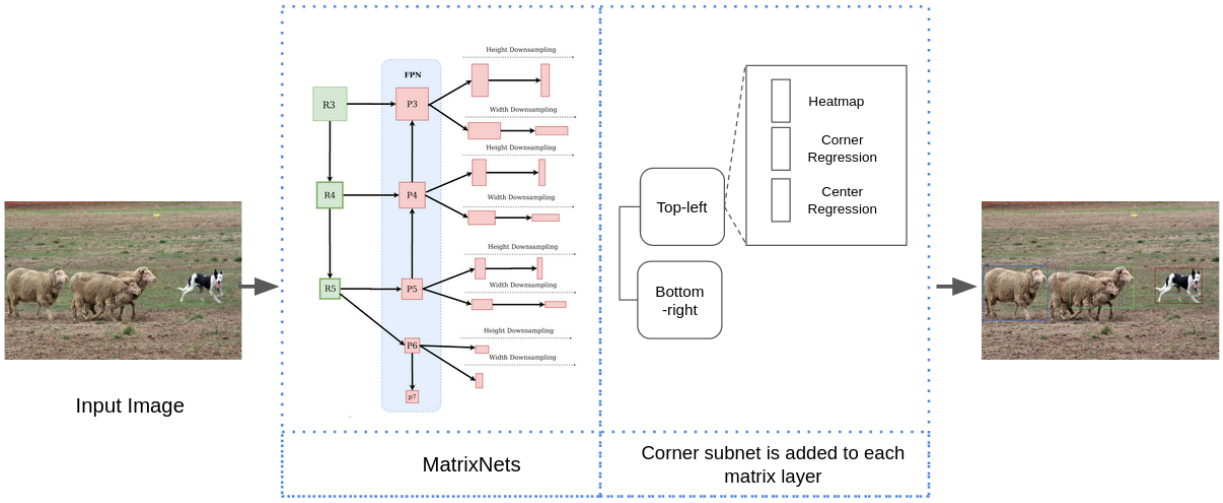


Figure 3.6: The x-Cornernet architecture.

pairs of offsets, one for each corner. The L1 loss between the ground truth offsets and predicted offsets (L_o) is added to the loss.

Thus the loss function of Cornernets is:

$$L_{\text{total}} = CFL + \alpha L_{pull} + \beta L_{push} + \gamma L_o \tag{3.4}$$

Here, α, β, γ are hyperparameters. Cornernet use ADAM [24] to optimize the full training loss.

3.3.2 Our Implementation: x-Cornernet

Cornernets has three main shortcomings:

1. Cornernet uses a single output layer from the backbone as input and then performs corner pooling to obtain predictions. However, pooling operations lose information. For example, if two objects of different sizes share the same location for the top corner, only the object with the dominant features will contribute to the gradient, so we can expect Cornernets to miss several objects. Using multiple outputs at different scales ensures that the context required for objects within a layer is bound by the single feature map's receptive field.

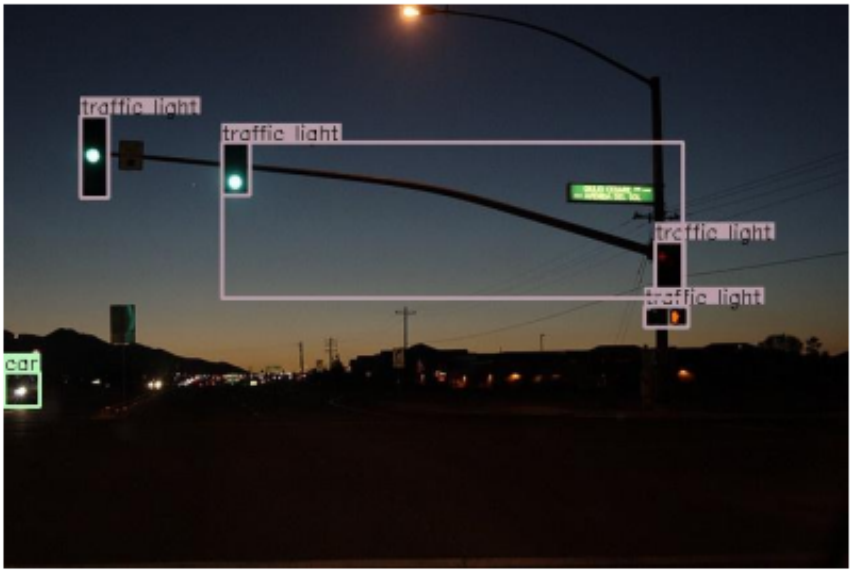


Figure 3.7: Example of a prediction where Cornernet makes an error matching corners and embeddings.

2. The top-left and bottom-right corners are matched using embeddings to find corners. Learning corners from embeddings is difficult, especially for tasks where similar objects are located together. For instance, Cornernet may may associate traffic lights of same size to different embeddings as seen in Figure 3.7.

3. Cornernet uses the Hourglass network, which has over 200M parameters and is computationally expensive to train.

x-Cornernets alleviates these issues by allowing learning from backbones with multiple outputs and eliminating the need for corner pooling. x-Cornernets consists of two components:

1. A MatrixNets backbone that extracts feature maps at various output sizes.

2. Two corner sub-networks, one for each corner that generates a corner heatmaps, then generate offsets for the centre and corner. This is done for each matrix layer. Then, a prediction module matches corners and applies offsets to produce the final predictions.

Figure 3.6 shows our x-Cornernets architecture.

Corner Heatmaps

The corner heatmap is generated in the same way as CornerNets. A CNN predicts a $K \times H \times W$ heat map for each corner where K is the number of classes, and (H, W) is the size of a matrix layer. A positive value implies ground truth, and the values are negative otherwise. An unnormalized 2-D Gaussian centred at each positive location is applied to reduce the penalty for locations closer to the ground truth corner. In x-Cornernets, multiple outputs from MatrixNets ensure that the context required for objects within a layer is bounded by the receptive field of a single feature map. As a result, corner pooling is no longer needed. Like Cornernet, we use the focal loss described in Equation (3.3) to deal with unbalanced classes.

Corner Regression

Just as in Cornernets, the input has to be resized to the dimensions of the matrix layers in x-Cornernet. As a result, when scaling down a corner from input dimensions to an (x, y) location in a layer, we predict offsets to scale up the corner to the original image size without losing precision. We keep the offset values between -0.5 and 0.5, and we use the smooth L1 loss Appendix C to optimize the parameters. The L1 loss for the top-left corner is denoted by L_{tl}, and the loss for the bottom-right corner is L_{br}.

Centre Regression

Since the matching is done within each matrix layer, the object's width and height are guaranteed to be within a specific range, which is the size of the layer. The centre of the object can be regressed easily because the range for the centres is small and allows us to remove the embedding entirely. At training time and for each corner location, we have a separate output head (centre regression head) that predicts the relative distance between the corner and the centre of the object. At test time, we match any two arbitrary top-left and bottom-right corners based on how far their centre predictions are from the actual centre. The actual centre between the top-left and bottom-right corners is the average of their coordinates. Once the actual centre is computed, we can compute the relative distance between the centre and each of the two corners. Using these distances as a reference, we can compute the error of the predicted distance by each corner with respect to the reference distance. We only match corners if both predict the relative distance to the centre with an error within 30%. To optimize the centre of the parameters, we again

use smooth L1 loss. Thus, we define the L1 centre regression loss for the top-left corner as L_{ctl}, and the one bottom-right corner as L_{cbr}.

Thus, the total loss for x-Cornernet is given as:

$$L_{\text{total}} = CFL + \alpha(L_{ctl} + L_{cbr}) + \beta(L_{tl} + L_{br}) \tag{3.5}$$

Here, α, β are hyperparameters. We use ADAM [24] to optimize the full training loss.

3.3.3 x-Cornernet-lite

We also propose a lighter variant of x-Cornernet called x-CornerNet-lite. In this architecture, we have the same MatrixNets backbone but a single output sub-network. The output sub-network produces centre heatmaps and regression offsets for the top-left and bottom-right corners. Thus, this architecture is similar to centre-based anchor-free methods, where each centre prediction acts as an anchor point. We compare the inference time-accuracy trade-offs in the experiments section.

3.4 Mask RCNN

Mask RCNN is a two-stage detector for the instance segmentation task. It is an extension of the Faster RCNN [43] framework that adds a parallel mask branch for predicting binary segmentation masks for each predicted object. We will discuss the architectural details below:

3.4.1 Architecture details

Backbone

Mask RCNN uses the traditional FPN from [31], i.e., it uses R_2, R_3, R_4, R_5 to create P_2, P_3, P_4, P_5 and then downsamples P_5 to obtain P_6.

Faster RCNN

Faster RCNN is made up of two stages.

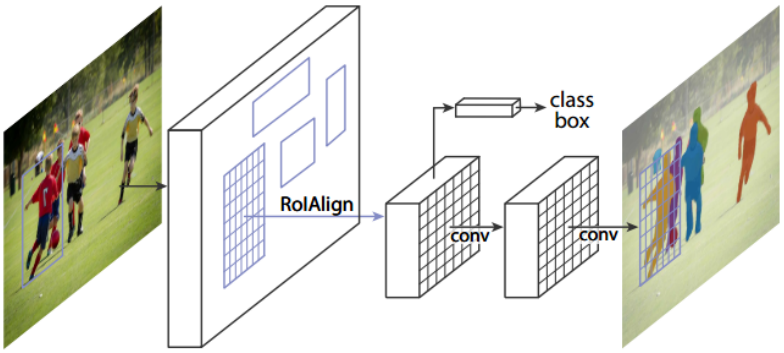


Figure 3.8: Mask RCNN architecture taken from [17]

1. **First Stage:** The first stage is a Region Proposal Network (RPN). The RPN takes an image of any size and outputs a set of bounding box proposals, each with an *objectness* score. Objectness measures a proposal's membership to one of the object classes vs. the background. The RPN works as follows: first, the backbone takes in the input image and calculates a feature map. Then, at each location of the feature map, we tile anchor boxes for predefined height and width over the feature map with each location of the feature map as the centre. If the dimensions of the feature map are $H \times W$, we have the total number of anchors given by $N_a = H \times W \times a$ anchor boxes, where a is the number of anchor boxes per location. Finally, two fully connected layers predict $4 \times N_a$ regression targets for each anchor box and $2 \times N_a$ probabilities, whether the anchor box has an object or not. The loss of this layer is the sum of the normalized classification and regression losses. Thus, the RPN produces a refined bounding box called Regions of Interest (RoIs) and ranks them by their objectness score.

2. **Second Stage:** The second stage is Fast RCNN [15]. This layer takes in the candidate boxes from the RPN, extracts features using RoIAlign and predicts the final classification and bounding box regression. Note that the first and second stage networks share features for faster inference.

Mask Branch

Mask RCNN adds a parallel mask branch in the second stage of Faster RCNN. Mask RCNN predicts a binary mask for size $m \times m$ for each RoI. Thus, the mask branch outputs a $K \times m^2$ dimension output for each RoI for the K classes. The mask loss is then defined as

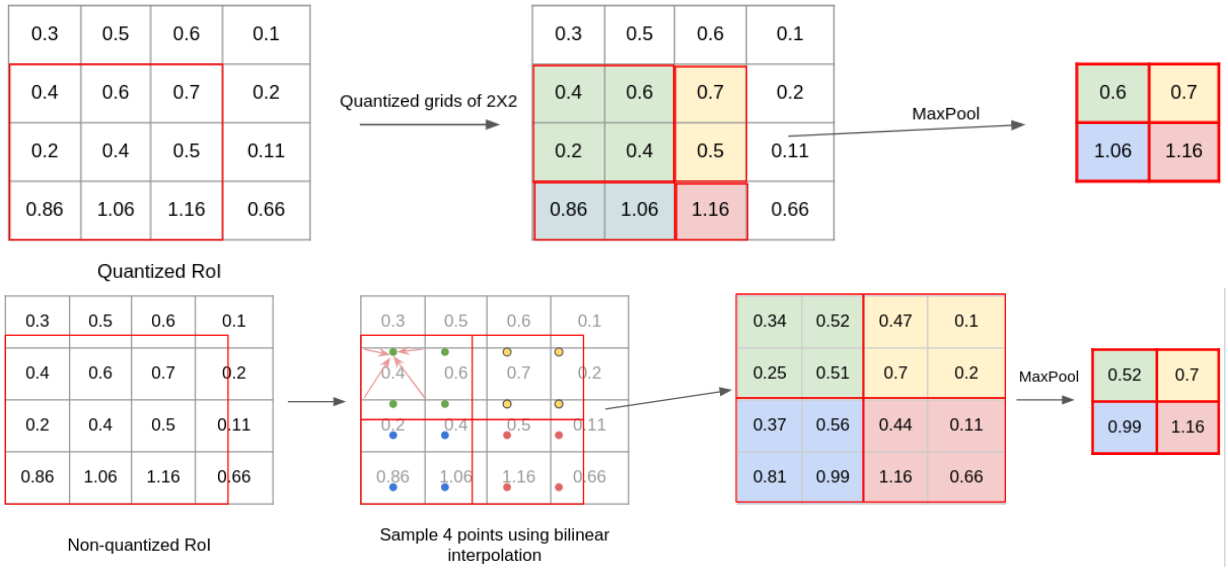


Figure 3.9: The top figure shows an example of RoIPool and bottom shows RoIAlign.

a binary cross-entropy loss over the predicted mask and ground truth binary mask scaled to size m^2. Note that the loss is calculated only for the predicted mask for class k of the ground truth. This mask loss is then added to the total loss.

RoI Align

RoIPool is a simple method that crops a region of the size of the RoI from the feature map. However, as the feature maps and input image are of different dimensions, RoIPool performs harsh quantization over the pixels to resize the RoIs to the feature map scale. For example, say we have an input of size $H \times W$. We obtain a feature map of $h \times w$ after some convolution layers, and we want to pool an area of size m^2. A region of size $h \times w$ decreases $(\lfloor H/h \rfloor \times \lfloor W/w \rfloor)$ times from the input to the feature map. As the downsampling in the FPN is square, $\lfloor H/h \rfloor$ is equal to $\lfloor W/w \rfloor$ and is called the stride, s, of the layer. Thus, for an RoI of shape $h_r \times w_r$, we have to compute $h_r * s \times w_r * s$. Next, the RoI in the feature coordinates is again quantized by the output size m^2, and maxpool is taken over each region. Figure 3.9 shows an example of RoIPool.

RoIAlign avoids this harsh quantization. Firstly, RoIALign does not use the floor function when it scales the RoI coordinates to the feature map scale. Thus, the coordinates are left as float values. The feature map is cropped with the coordinates, and the cropped

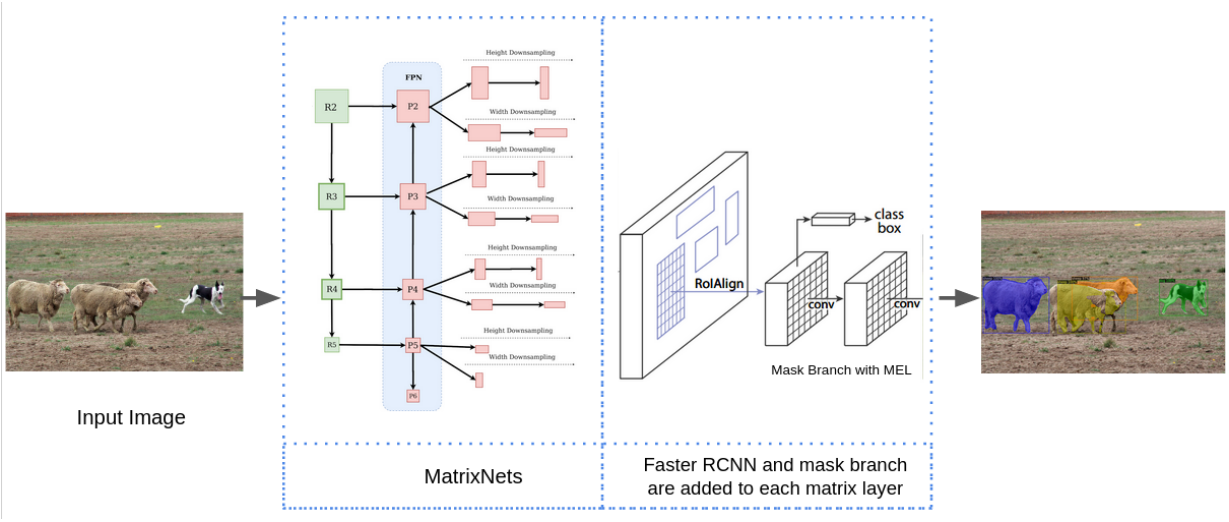


Figure 3.10: The x-Mask RCNN architecture. MatrixNets with R_2 is used in the backbone and MEL is added to the mask branch.

part is divided into a grid using the output size. Next, RoIalign chooses 4 points in each bin using bilinear interpolation. Then, the maximum of these four points is taken to obtain the final output. Figure 3.9 shows an example of RoIAlign.

3.4.2 Our Implementation: x-Mask RCNN

To adapt MatrixNets to Mask RCNN, we make three changes. We first modify the backbone FPN and add matrix layers. Then, we modify the RoIAlign to allow pooling from asymmetric matrix layers. Finally, we introduce Mask Edge Loss (MEL), which benefits from the larger number of RoIs coming from the additional matrix layers. Figure 3.10 shows the x-Mask RCNN architecture.

MatrixNets backbone

We recreate MatrixNets using the FPN with R_2 to R_5 as the backbone. Figure 3.10 presents the MatrixNets backbone with R_2.

RoIAlign and Adaptive Pooling

RoIAlign, as defined in [17], is implemented to pool from FPN layers. As FPNs use square downsampling, the stride along the height and width dimensions are equal for a layer. Matirxnets use asymmetric downsamplings, so the stride along the width and height can be different, especially along with the non-diagonal layers. We modify the RoIAlign to incorporate this functionality.

Faster RCNN [43] and then Mask RCNN [17] use the following rule to assign RoIs to layer (P_k) from which to pool:

$$k = \lfloor k_0 + log_2 \sqrt{w_r * h_r}/S_0 \rfloor \tag{3.6}$$

Here, k_0 is 4, and (h_r, w_r) are the height and width of an RoI. The equation implies that objects of size 244^2 are assigned to P_4. 244 is chosen as S_0, as the images in Imagenet are trained at an image size of 224^2. However, if the image dimension is 1024×1024, an object of size 224^2 would still be assigned to P_4, despite it being five times smaller than the input dimension.

Mask Edge Loss

Mask RCNN downsamples the ground masks to a fixed size of 28×28 in order to save memory during mask loss calculation. This downsampling leads to loss of information and is most pronounced around the edges. We add a subtle attention mechanism to the mask loss such that the loss function pays extra attention to the boundaries and penalizes coarseness. We call this new loss function *Mask Edge Loss (MEL)* .

Precisely, during loss calculation, we first extract the boundary from the ground truths. As ground truth masks are bitmasks, getting the boundary comes at minimal cost by merely taking the gradient along the x and y axes. We then apply a Gaussian blur (more details in Appendix B) to increase the thickness of the extracted contour. Then, we normalize the contour and multiply it with the pixel-wise cross-entropy loss between the predicted mask and the ground truth mask. Finally, we reduce the pixel-wise cross-entropy loss to a single value by taking the mean over all the values.

Chapter 4

Experiments

In this chapter, we present the experimental results for x-Retinanet, x-Cornernet and x-Mask RCNN. We train the network using the "train2017" subset of MSCOCO [33] dataset and test using the "val2017" subset. More details on the exact training and testing setup used are presented in each section.

4.1 x-RetinaNet

4.1.1 Experimental details

Training

We initialize the network as per the initialization in Retinanet [32]. We use synchronized SGD over 4 GPUs with a batch size of 16 images (4 per GPU) for 180K iterations. The initial learning rate is 0.01, which is divided by ten at 140K and 160K steps. We use a momentum rate of 0.09. We train using the MS COCO train2017 data set and resize images such that the smaller side of the image is at most 800 pixels. We do not use any other data augmentations at train time.

Inference

Inference involves a forward pass over the network. To improve performance, we decode box predictions from at most 1K top-scoring predictions per MatrixNet layer. The top

	Retinanet			x-Retinanet (Ours)	
# Anchors	1	3	9 (Default)	1	3
mAP	33.5	35.2	37.9	38.9	39.5
AP50	55.1	55.1	58.1	59.8	59.5
AP75	35.0	37.4	40.5	42.1	42.9
APs	21.0	21.8	22.8	23.4	23.3
APm	35.9	38.6	41.9	43.4	44.2
APl	42.4	45.3	48.3	49.7	51.4
1-1.5	38.5	42.3	43.5	43.0	44.3
1.5-2	36.2	38.9	41.2	42.1	43.0
2-3.5	31.5	32.6	36.5	37.5	38.0
3.5	12.6	6.5	9.1	14.8	15.9
Parameters	36.4M	36.8M	37.9M	37.5M	38.1M
Inference Time (ms)	74	83	106	140	159

Table 4.1: Retinanet Result with FPN and MatrixNets backbone. x-Retinanet implies MatrixNets with 19 layers. For #Anchors = 3 we add anchors at 3 scales and 1 aspect-ratio, and for #Anchors = 9 we add anchors at 3 scales and 3 aspect-ratio. We report the various precision metrics as discussed in Table 3.2.

predictions from all levels are merged, and Non-Maximum Suppression (NMS) with a threshold of 0.5 is applied to get the final detections.

4.1.2 Results

The results comparing RetinaNet to x-Retinanet are given in Table 4.1. We see that, without any bells and whistles, MatrixNets improves Retinanet by 1mAP. Moreover, MatrixNets significantly improves performance on skewed boxes. MatrixNets makes inference slower by $\sim$ 1.3 times.

4.1.3 Ablation

By adding MatrixNets to Retinanet, we only change two parameters: 1) We remove anchor boxes, and 2) we add matrix layers. Thus, we run ablation studies to understand the effect of these two parameters.

#layers	mAP	1-1.5	1.5-2.5	2.5-3.5	3.5+	Inference Time
5	33.5	38.5	36.2	31.5	12.6	74
13	37.9	42.5	41.1	36.0	12.5	115
19	38.9	43.0	42.1	37.5	14.8	140

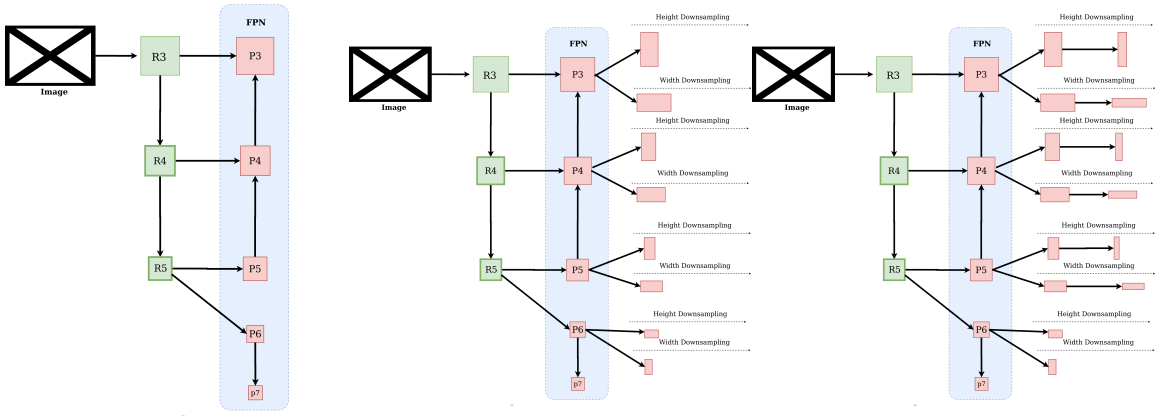
Table 4.2: Impact of the number of layers of MatrixNets on detection performance in x-Retinanet.

Adding Anchors to MatrixNets

Table 4.1 shows the effect of adding anchor boxes to Retinanet vs. adding anchor boxes to x-Retinanet. We see that going from 1 to 3 anchor boxes improves the performance of Retinanet by 2.7mAP. In contrast, the performance of x-Retinanet is improved by 0.6mAP, indicating that the matrix layers indeed capture some aspect-ratio awareness. This result also follows [32], which shows that more than nine anchors per location does not improve outcomes significantly. Moreover, x-Retinanet with one anchor box improves performance on boxes with a dimension ratio of 3.5+ by 5.7 AP.

Effect of Matrix Layers

We also run an ablation analysis on the effect of the number of matrix layers on performance. We run these tests for the number of layers in $\{5, 13, 19\}$. For a five-layer matrixnet, we remove all layers except the diagonal ones. As discussed earlier, this is equivalent to an FPN at $A = 1$. For 13 layers we remove layers $\{l_{1,3}, l_{2,4}, l_{3,1}, l_{3,5}, l_{4,2}, l_{5,3}\}$, i.e., we symmetrically remove the smallest layers from the top right and bottom left of the matrix. Figure 4.1 shows the various Matrixnet architectures. We see that adding layers has a similar effect as adding anchor boxes to FPN based RetinaNet. Results are available in Table 4.2. Going from 5 to 13 layers improves accuracy by 1 mAP, and going from 13 to 19 layers adds another 1 mAP to the accuracy.



(a) (b) (c)

Figure 4.1: (a) Matrixnet with 5 layers (b) MatrixNets with 13 layers and (c) 19 layer MatrixNets

4.2 x-Cornernets

4.2.1 Experimental details

Training

We train all of our networks on a server with 8 Titan XP GPUs. Our implementation is done in PyTorch [38]. The image is cropped to 512×512. As per [27], we use the following data augmentation techniques: random horizontal flipping, random scaling, random cropping and random colour jittering, adjusting the brightness, saturation and contrast. For optimization, we use the ADAM optimizer, set an initial learning rate to 5e-5, and cut it by 1/10 after 250k iterations, training for 350k iterations. For our matrix layer ranges, we set $l_{1,1}$ to be [24px-48px]x[24px-48px] and then scale the rest as described in Chapter 3.1.

Inference

For single-scale inference, we resize the max side of the image to 900px or the min size to 600px. Our images are slightly smaller than some others to lower inference time to account for the extra output layers introduced. We use ResNet-152 rather than ResNeXt-101 as our best model since it has fewer layers and 3/4 the inference time. Finally, we choose the top 100 detections according to their scores as the detector's final output. We use

SoftNMS [2] in favour of NMS for x-Cornernets as it has shown to improve accuracy in other keypoint-based detectors.

4.2.2 Results

The results for x-Cornernet are available in Table 4.3. We see that x-Cornernet can outperform Cornernet even with a smaller backbone. x-Cornernet R50 FPN improves on regular Cornernet R101 by 6 mAP. Furthermore, adding MatrixNets adds another 5.1 mAP to x-Cornernet. We also see that the relative improvement of adding MatrixNets for heavily skewed objects.

backbone	**Cornernet**		**x-Cornernet-Lite (Ours)**	**x-Cornernet (Ours)**
	res101	hourglass 104	res50 X19	res50 X19
mAP	30.2	40.6	40.3	41.3
AP50	44.1	56.4	60.8	60.9
AP75	32	43.2	43.7	45.5
APs	13.3	19.1	24.3	24.9
APm	33.3	42.8	44.9	45.8
APL	42.7	54.3	53.7	55.7
1-1.5		45.8	44.5	47.6
1.5-2.5		43.0	43.2	46.0
2.5-3.5		34.6	35.5	40.3
3.5+		16.9	18.7	19.6
Parameters	55.3M	201M	44.1M	48.7M
Image Size	ori.	ori.	600×900	600×900
Inference Time		301	166	282

Table 4.3: Cornernet Results. Blank indicates results were not available. ori. indicates original image size. We report the precision metrics as discussed in Table 3.2.

4.2.3 Ablations

Adapting Cornernet to use FPN based architectures allows us to run ablation studies on the effect of heavier backbones on x-Cornernets.

#layers	mAP	1-1.5	1.5-2.5	2.5-3.5	3.5+	Inference Time (ms)
5	35.1	41	38.1	30.8	12.4	85
13	38.2	43.4	42.5	35.5	10.2	125
19	40.3	44.5	42.4	39.8	18.7	139

Table 4.4: Impact of the number of layers of MatrixNets on detection performance in x-Cornernet-Lite.

Effect of Backbone

In Table 4.5 we show the impact of using larger backbones with the x-CornetNet-Lite architecture. We see that going from Resnet-50 to Resnet-101 gives us +1mAP, and going from ResNet-101 to Resnet-152 gives us another +1.5 mAP improvement. This result is expected as larger backbones have a positive impact on accuracy, as seen in most deep-learning architectures.

Effect of Crop Size

During the training, we use scale jitter to scale the image randomly. Then we use crops of fixed sizes to train the model. In Table 4.5, using a larger image crop during training leads to higher accuracy. This result is expected as a larger image would have more information and lead to better training.

Effect of Matrix Layers

Table 4.4 shows the result for ablation with the number of layers used in the MatrixNets backbone. We see that the performance improves as we increase the number of matrix layers. Moreover, the improvement is prominent for objects with skewed aspect-ratios. Thirteen layers MatrtixNets enhances the performance for objects with dimension ratio $2.5 - 3.5$ by 4.7 AP and 19 layers MatrtixNets improves the performance for objects with dimension ratio 3.5+ by 6.3 AP over the 5 layers MatrtixNets.

Qualitative Assessment of Predictions

Figure 4.2 shows sample predictions from x-Cornernets with FPN and MatrixNets. We see that adding additional matrix layers lead to tighter bounding boxes, especially for tall and thin objects.

Image Size	AP	AP_{50}	AP_{75}
512	40.3	61.4	43.5
640	41.3	60.9	45.5

Backbone	AP	AP_{50}	AP_{75}
Resnet50-X19	41.0	60.4	45.0
Resnet101-X19	42.3	62.1	46.4
Resnet152-X19	43.6	62.3	47.5

Table 4.5: Left table shows the impact of input image size and right table shows the impact of backbone on x-Cornernet-Lite performance. We see larger image size improves perfrormace.

Figure 4.2: Sample detection results for x-Cornernet object detection framework when using FPN (top), and 19 layers MatrixNet (bottom) as backbones. MatrixNet produces tighter bounding boxes especially for rectangular objects.

4.3 x-Mask RCNN

Most hyperparameters for x-Mask RCNN are set following Mask RCNN [17]. The modified settings are given below.

4.3.1 Experimental details

Training

Mask RCNN uses anchors at 5 scales, 32^2 to 512^2, and 3 aspect ratios, $[0.5, 1.0, 1.5]$. For the MatrixNets experiments, we set the number of anchors to 1 per scale. An RoI is considered positive if it has an IoU greater than 0.5 with the ground truth and negative otherwise. The Faster RCNN Branch takes 512 RoIs from the RPN, where the ratio of positive to negative samples is fixed at $1 : 3$. We train with 3 GPUs with 2 images per

	MEL	AP	Ap50	Ap75	Aps	Apm	Apl	1-1.5	1.5-2	2-3.5	3.5+	Parameters	Inference Time (ms)
Mask RCNN		34.1	55.4	35.96	16.5	36.7	49.0	37.7	35.7	31.4	14.0	49.0M	74
Mask RCNN	yes	34.4	55.7	36.5	16.8	37.0	49.7	38.2	36.5	31.5	14.2	49.0M	74
x-Mask RCNN		32.8	52.8	34.8	14.2	35.1	49.2	36.0	34.6	30.8	13.7	50.2M	100
x-Mask RCNN	yes	33.3	53.4	35.2	14.9	36.0	49.7	36.4	34.7	31.7	13.9	50.2M	100

Table 4.6: x-Mask RCNN results with and without Mask Edge Loss (MEL).

GPU for a total of 240k iterations. The initial learning rate is 0.00375 and is divided by 10 at 180K and 210K iterations following the linear scaling rule [16].

Inference

During testing, the RPN gives 1000 proposals over which the Faster RCNN branch is run. Then, NMS is applied, and the masks are predicted for the top 100 surviving boxes from the Faster RCNN branch. The m^2 floating-number mask output is then resized to the RoI size, and made binary at a threshold of 0.5. We set $m = 28$ as per the Mask RCNN paper [17]. The seg-AP is evaluated using IoU between the predicted and ground truth masks.

4.3.2 Results

The results for x-Mask RCNN are available in Table 4.6. We see that adding MEL adds 0.3 to 0.6mAP to Mask RCNN.

4.3.3 Ablations

We make two main changes to Mask RCNN. We change the backbone, and we change the loss function. As seen from Table 4.6, adding MatrixNets does not improve Mask RCNN as the RPN layer already optimizes the template boxes for the Faster RCNN layer, which then chooses a small subset of boxes. Thus, improving the RPN may not lead to any downstream improvement in Mask RCNN. The impact of MEL is discussed in the next section.

Impact of MEL

Table 4.6 shows the results for the effect of MEL on both Mask RCNN and x-Mask RCNN. We see that MEL consistently improves performance. The performance improvement in

Model	Acc w MEL	Acc w/o MEL
Mask RCNN	88.9	88.9
x-Mask RCNN	90.0	88.9

Table 4.7: Mask branch accuracy with and without MEL for Mask RCNN and x-Mask RCNN

x-Mask RCNN is higher owing to the larger variety of RoIs extracted from the matrix layers.

Detectron2 calculates the accuracy of the mask branch as the number of masks correctly predicted divided by the total number of ground truth masks. Table 4.7 shows the accuracy of the mask branch for Mask RCNN with MEL and without MEL. We see that MEL improves accuracy by 1% at virtually no additional cost. Finally, Figure 4.3 shows some examples of masks predicted with and without MEL. We see that MEL penalizes coarseness consistently.

Figure 4.3: Sample predicted masks from Mask RCNN with MEL (top), and without MEL (bottom). Adding MEL achieves finer mask predictions.

4.4 State of the art Comparison

We compare our current implementation, x-RetinaNet and x-Cornernets, to existing methods and state of the art in Table 4.8. We see that both with and without bells and whistles,

adding MatrixNets can achieve competitive results, while removing the need for anchor boxes (results marked with *Ours*).

Architecture	Backbone	Image Size	mAP	AP50	AP75	AP s	APm	APl
Previous Detectors								
RetinaNet[32]	ResNext-101	800x1333	40.8	61.1	44.1	24.1	44.1	51.2
NAS-FPN[14]	ResNet 50	1280x1280	45.4					
RefineDet[59]	ResNet-101	512x512	36.4	57.5	39.5	16.6	39.9	51.4
FreeAnchor[60]	ResNext-101	800x1333	44.9	62.4	48.1	25.6	47.4	57.4
RepPoints[54]	ResNet-101-DCN	800x1333	45	66.1	49	26.6	48.6	57.5
RefineDetΔ[59]	ResNet-101	512x512	41.8	62.9	45.7	25.6	45.1	54.1
FreeAnchorΔ[60]	ResNeXt-101	$\leq$ 2x	47.3	66.3	51.5	30.6	50.4	59
RepPointsΔ[54]	ResNet-101-DCN	$\leq$ 3x	46.5	67.4	50.9	30.3	49.7	57.1
Center-Based Anchor-Free								
FoveaBox[25]	ResNeXt 101	800x1333	42.1	61.9	45.2	24.9	46.8	55.6
FSAF[64]	ResNeXt 101	800x1333	42.9	63.8	46.3	26.6	46.2	52.7
FCOS[48]	ResNeXt 101	800x1333	43.2	62.8	46.6	26.5	46.2	53.3
FSAFΔ[64]	ResNeXt 101	$\leq$ 1.5x	44.6	65.2	48.6	29.7	47.1	54.6
SAPD [63]	ResNeXt-64x4d-101-DCN	800x1333	47.4	67.4	51.1	28.1	50.3	61.5
ATSSΔ [58]	ResNeXt-64x4d-101-DCN	800x1333	50.7	68.9	56.3	33.2	52.9	62.4
PPAΔ, † [23]	ResNeXt-64x4d-101-DCN	800x1333	51.4	69.7	57.0	34.0	53.8	64.0
x-CornerNet-LiteΦ Ours	ResNet-152-X19	600x900	43.7	62.7	47.8	22.7	48.2	57.4
x-CornerNet-LiteΔ, Φ Ours	ResNet-152-X19	$\leq$ 1.8x	46.1	64.7	42.5	26.9	49.9	59.6
Corner-Based Anchor-Free								
ExtremeNetΦ[62]	Hourglass 104	ori.	40.2	55.5	43.2	20.4	43.2	53.1
CenterNet[61]	Hourglass 104	ori.	42.1	61.1	45.9	24.1	45.5	52.8
CenterNetΦ[12]	Hourglass 104	ori.	44.9	62.4	48.1	25.6	47.4	57.4
CornerNetΦ[27]	Hourglass 104	ori.	40.5	56.5	43.1	19.4	42.7	53.9
CornerNetΔ, Φ[28]	Hourglass 104	$\leq$ 1.8x	42.1	57.8	45.3	20.8	44.8	56.7
ExtremeNetΔ[62]	Hourglass 104	$\leq$ 1.5x	43.2	59.8	46.4	24.1	46.0	57.1
CenterNetΔ[61]	Hourglass 104	$\leq$ 1.8x	45.1	63.9	49.3	26.6	47.1	57.7
CenterNetΔ, Φ[12]	Hourglass 104	$\leq$ 1.8x	47	64.5	50.7	28.9	49.9	58.9
CentripetalNetΦ[11]	Hourglass 104	$\leq$ 1.8x	46.1	63.1	49.7	25.3	48.7	59.2
CentripetalNetΔ, Φ[11]	Hourglass 104	$\leq$ 1.8x	48.0	65.1	51.8	29.0	50.4	59.9
CPNΔ, Φ, †[13]	Hourglass 104	$\leq$ 1.8x	49.2	67.4	53.7	31.0	51.9	62.4
x-CornerNetΦ Ours	ResNet-152	600x900	45.2	64.2	49.2	25.9	48.9	57.6
x-CornerNetΔ, Φ Ours	ResNet-152	$\leq$ 1.8x	47.8	66.2	52.3	29.7	50.4	60.7
Δ- Multi-Scale, Φ-Soft-NMS, ori- original image size, † results from ECCV'20								

Table 4.8: State-of-the-art comparison on COCO test-dev2017 set for object detection. A blank indicates methods for which results were not available. The image size for multi-scale results indicates the maximum size to which the image was scaled; for example, 2× means that the image was resized to twice the resolution for multi-scale testing.

Chapter 5

Conclusion and Future Work

In this thesis, we run extensive experiments to show that MatrixNets is a general-purpose CNN module that can add aspect-ratio awareness to existing architectures. We apply MatrixNets to three popular architectures — Retinanet, Cornernet and Mask RCNN. We demonstrate that MatrixNets mimics anchor boxes in the asymmetric matrix layers and improves performance on skewed objects. We further improve upon Mask RCNN by proposing Mask Edge Loss, a subtle attention mechanism that punishes coarseness in masks. Together, these experiments show that MatrixNets can make anchor-based architectures anchor-free, and add aspect-ratio awareness to anchor-free architectures.

Anchor boxes add aspect-ratio awareness by modifying the output network. By restricting receptive fields, MatrixNets refactors the properties of anchor boxes into the backbone without changing the output network. This allows architectures with customized output networks to become aspect-ratio aware. Thus, we observe an improvement in anchor-free architectures with generic backbones, but custom keypoint prediction mechanisms.

In Table 4.8, † marks the architectures released in July 2020. We note that our best method x-Cornernet with Resnet-152 is comparable to the state-of-the-art Centripetalnet [11] before July 2020. With the promise MatrixNets has shown with Retinanet and CornerNet, we speculate that MatrixNets can further improve these newer state-of-the-art methods. For instance, CPN [13] adds a corner proposal network to Conrnernets, but still uses the more massive Hourglass network. MatrixNets can replace the Hourglass network to push state-of-the-art even further.

Finally, we do not observe significant improvements in two-stage architectures like Mask RCNN. However, Mask Edge Loss, even though unique to x-Mask RCNN, is a general-purpose component that can improve the loss function of all architectures that predict

binary masks for instance segmentation.

This thesis further opens up new directions for studying MatrixNets and aspect-ratio awareness in object detectors. One potential future work is to add ATSS [58] or PPA [23] to MatrixNets and compare the "anchor-freeness" added by each method. We posit that both ATSS and PPA can supplement MatrixNets and improve performance even further. Another avenue for future work is to improve the speed of MatrixNets. MatrixNets is currently 1.4x slower than FPN based counterparts. Depth-wise Convolution [9] is a method that reduces the cost of convolution operations by breaking doing the matrix multiplication into groups. Adding Depth-wise Convolutions may make MatrixNets faster. On the other hand, the most time-consuming component in x-CornerNet is the NMS, which is currently computed on the CPU; we can move these calculations to the GPU to get a performance improvement. Finally, one limitation of Mask RCNN is that it requires the mask pooler to be square. This design decision stems from the square downsampling in FPNs. With MatrixNets, the pooler can now be asymmetric or even adaptive based on the layer assignment. Thus, asymmetric pooling is another area of future research that MatrixNets opens up.

References

[1] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[2] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017.

[3] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: real-time instance segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9157–9166, 2019.

[4] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[5] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, 1986.

[6] Hao Chen, Kunyang Sun, Zhi Tian, Chunhua Shen, Yongming Huang, and Youliang Yan. Blendmask: Top-down meets bottom-up for instance segmentation. *arXiv preprint arXiv:2001.00309*, 2020.

[7] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4974–4983, 2019.

[8] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[10] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.

[11] Zhiwei Dong, Guoxuan Li, Yue Liao, Fei Wang, Pengju Ren, and Chen Qian. Centripetalnet: Pursuing high-quality keypoint pairs for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10519–10528, 2020.

[12] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Object detection with keypoint triplets. *arXiv preprint arXiv:1904.08189*, 2019.

[13] Kaiwen Duan, Lingxi Xie, Honggang Qi, Song Bai, Qingming Huang, and Qi Tian. Corner proposal network for anchor-free, two-stage object detection. *arXiv preprint arXiv:2007.13816*, 2020.

[14] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7036–7045, 2019.

[15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[16] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Zhaojin Huang, Lichao Huang, Yongchao Gong, Chang Huang, and Xinggang Wang. Mask scoring r-cnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6409–6418, 2019.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[21] Amir Jamaludin, Timor Kadir, and Andrew Zisserman. Spinenet: automatically pinpointing classification evidence in spinal mris. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 166–175. Springer, 2016.

[22] Wei Ke, Tianliang Zhang, Zeyi Huang, Qixiang Ye, Jianzhuang Liu, and Dong Huang. Multiple anchor learning for visual object detection. *arXiv preprint arXiv:1912.02252*, 2019.

[23] Kang Kim and Hee Seok Lee. Probabilistic anchor assignment with iou prediction for object detection. *arXiv preprint arXiv:2007.08103*, 2020.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Tao Kong, Fuchun Sun, Huaping Liu, Yuning Jiang, and Jianbo Shi. Foveabox: Beyond anchor-based object detector. *arXiv preprint arXiv:1904.03797*, 2019.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[27] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[28] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. Cornernet-lite: Efficient keypoint based object detection. *arXiv preprint arXiv:1904.08900*, 2019.

[29] Hyungtae Lee, Sungmin Eum, and Heesung Kwon. Me r-cnn: Multi-expert r-cnn for object detection. *arXiv preprint arXiv:1704.01069*, 2017.

[30] Youngwan Lee and Jongyoul Park. Centermask: Real-time anchor-free instance segmentation. *arXiv preprint arXiv:1911.06667*, 2019.

[31] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[32] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[34] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[35] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[36] Alejandro Newell, Zhiao Huang, and Jia Deng. Associative embedding: End-to-end learning for joint detection and grouping. In *Advances in neural information processing systems*, pages 2277–2287, 2017.

[37] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

[38] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[39] Siyuan Qiao, Liang-Chieh Chen, and Alan Yuille. Detectors: Detecting objects with recursive feature pyramid and switchable atrous convolution. *arXiv preprint arXiv:2006.02334*, 2020.

[40] Abdullah Rashwan, Rishav Agarwal, Agastya Kalra, and Pascal Poupart. Matrixnets: A new scale and aspect ratio aware architecture for object detection. *arXiv preprint arXiv:2001.03194*, 2020.

[41] Abdullah Rashwan, Agastya Kalra, and Pascal Poupart. Matrix nets: A new deep architecture for object detection. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[42] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[44] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[46] Mingxing Tan, Ruoming Pang, and Quoc V Le. EfficientDet: scalable and efficient object detection. *arXiv preprint arXiv:1911.09070*, 2019.

[47] Zhi Tian, Chunhua Shen, and Hao Chen. Adelaidet. https://github.com/aim-uofa/AdelaiDet, 2020.

[48] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. *arXiv preprint arXiv:1904.01355*, 2019.

[49] Shaoru Wang, Yongchao Gong, Junliang Xing, Lichao Huang, Chang Huang, and Weiming Hu. Rdsnet: A new deep architecture for reciprocal object detection and instance segmentation. *arXiv preprint arXiv:1912.05070*, 2019.

[50] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. Solo: Segmenting objects by locations. *arXiv preprint arXiv:1912.04488*, 2019.

[51] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[52] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[53] Jing Yang, Qingshan Liu, and Kaihua Zhang. Stacked hourglass network for robust facial landmark localisation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 79–87, 2017.

[54] Ze Yang, Shaohui Liu, Han Hu, Liwei Wang, and Stephen Lin. Reppoints: Point set representation for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9657–9666, 2019.

[55] Hui Ying, Zhaojin Huang, Shu Liu, Tianjia Shao, and Kun Zhou. Embedmask: Embedding coupling for one-stage instance segmentation. *arXiv preprint arXiv:1912.01954*, 2019.

[56] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.

[57] Rufeng Zhang, Zhi Tian, Chunhua Shen, Mingyu You, and Youliang Yan. Mask encoding for single shot instance segmentation. *arXiv preprint arXiv:2003.11712*, 2020.

[58] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. *arXiv preprint arXiv:1912.02424*, 2019.

[59] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4203–4212, 2018.

[60] Xiaosong Zhang, Fang Wan, Chang Liu, Rongrong Ji, and Qixiang Ye. Freeanchor: Learning to match anchors for visual object detection. In *Advances in Neural Information Processing Systems*, pages 147–155, 2019.

[61] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[62] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krahenbuhl. Bottom-up object detection by grouping extreme and center points. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 850–859, 2019.

[63] Chenchen Zhu, Fangyi Chen, Zhiqiang Shen, and Marios Savvides. Soft anchor-point object detection. *arXiv preprint arXiv:1911.12448*, 2019.

[64] Chenchen Zhu, Yihui He, and Marios Savvides. Feature selective anchor-free module for single-shot object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 840–849, 2019.

APPENDICES

Appendix A

NMS

The algorithm given B, S and N_t, where, B the list of initial detection boxes, S, contains corresponding detection scores and N_t the NMS IOU threshold, and gets back the detections and scores that survive NMS D, S.

The NMS algorithm is given below:

Algorithm 1: The NMS Algorithm

Input : $B = \{b_1, .., b_N\}, S = \{s_1, .., s_N\}, N_t$
Output: Set of detections D which survive NMS and corresponding set of Scores S

$D \leftarrow \{\}$;
while $B \neq \phi$ **do**
  $m \leftarrow \text{argmax}(S)$;
  $M \leftarrow b_m$;
  $D \leftarrow D \cup M$;
  $B \leftarrow B - M$;
  for $b_i \in B$ **do**
    if $iou(M, b_i) \geq N_t$ **then**
      $B \leftarrow B - b_i$;
      $S \leftarrow S - s_i$;
    end
  end
end
return D, S

Appendix B

Gaussian Blur

Gaussian Blur is a a standard technique used in image processing to reduce the amount of noise in an image. The visual effect of this blurring technique is a smooth blur as seen in fig. B.1. Mathematically, applying a Gaussian blur to an image is the same as applying a 1D conv to the the image with weights taken from a Gaussian function of fixed standard deviation and then normalized.

Figure B.1: (Left) shows the image without blur and (right) shows the image with Gaussian blur.

Appendix C

Smooth L1 Loss

Smooth L1 loss is given as

$$L1_{\text{smooth}}(x) = \begin{cases} 0.5\frac{x^2}{\beta}, & \text{if } |x| < \beta \\ |x| - 0.5\beta & \text{otherwise.} \end{cases} \tag{C.1}$$

Here, β is a hyperparameter that divides the positive axis range into two parts: L2 loss is used for targets in range [0, β], and L1 loss is used $> \beta$ to avoid over-penalizing outliers. The overall function is smooth.