Department of Computer Science


Submitted in part fulfilment for the degree of BSc.

# Detecting and Tracking Wild Animals in Video Sequence

Asim Poptani

2021-May

Supervisor: Dr. Kofi Appiah

To my family and friends who have gone above and beyond
and have supported me throughout life.
Thank you.


## Acknowledgements

I would also like to extend my thanks to the one and only Dr Kofi Appiah for his deep patience, knowledge and guidance.

# Contents

# List of Figures

# List of Tables

# List of Equations

# Executive Summary

Detecting and tracking animals has been used for a vast number of tasks, most notably for tracking where different types of animals live and the quantity of the number of animals using trap cameras.

Detecting and tracking animals using trap cameras has been very important for researchers, hobbyists and scientists as it allows them to see what ecology is living in an environment. With this information, these users of trap cameras are able to look after the environment and make decisions that could preserve endangered animals.

Unfortunately, detecting and tracking using track cameras are not perfect as the camera is set up usually to take videos when there is a change of a large number of pixels or a difference in infrared light, which leads to a large number of false positives.

Additionally, when a user of a trap camera is viewing through video footage, it can be hard to locate the exact position of the animal in the video due to the animal's ability to hide in foliage which, can lead to the video being discarded as a false positive. Or another possibility is that the image is not clear, so it is hard for the user to find what animal is in the video frame.

In this project, we aim to look for a modern object detector that can be used to track and detect animals with the objectives: of being able to process video at real-time of 30fps or faster, able to locate animals by partitioning the image from the foliage from the animal and being able to identify what the animal is.

To find a model capable of identifying and locating an animal within a video frame, we look at existing literature such as one-stage detectors, two-stage detectors and segmentation detectors. We find that in previous literature, we see there is no perfect solution for our problem.

From previous literature, there are two types of outputs: bounding boxes and segmentation images.

Models which produce bounding boxes such as YOLO V3 [1] detect objects by using a grid to separate the image. Inside each grid, there are cells inside each cell - a prediction is made if there is an object in this cell.

This methodology is very fast and offers high performance in categorization

and localization of where the animal is in the video frame. Unfortunately, these models do not allow for close outlining of an animal which is what our project objectives requires us to do to segment animals from the foliage.

The alternative solution that the literature suggests is using models such as SegNet [2] which generates close segmentation images, which segment the outline of where objects are across the whole video frame. The results of these models are slower than in real-time but it generates high accuracy in both categorical and localization.

In this project, we suggest a new model called YOLO-PV (You Only Look Once Polygon Version), where we combine these two approaches to allow multiple objects to be detected in a video frame quickly. This is done by using the YOLO V3 [1] architecture, and instead of outputting boxes at each cell, the output points. When joining these points together they form a segmentation mask that can be used to just identify the animal from its surroundings.

This new approach did mean that the training data for localizing YOLO-PV had to be created or converted from another dataset, as no dataset currently supports this new technique. In this project, it was decided to convert an existing dataset as making a new dataset is time-consuming and illegal during COVID-19 times. So it was decided to convert the [3] dataset [3] a free and public dataset, into the new format of points.

Our test results show for YOLO-PV that the model is able to find, identify and localize animals in a video frame faster than in real-time. The accuracy of the model is backed up when running the COCO [3] benchmark against the model we got an AP (Average precision) of 24.0. This score is less than other modern detectors but it works as a proof of concept.

There are improvements to be made to the model. This project, suggests that future work should focus on creating better datasets for night-time as well as datasets that allow obstructions over objects to increase the robustness of the models. Additionally, we also suggest that future works may consider using the technique of creating segmentation's using points as it has been shown to work in this project for other applications of object detection.

There are some ethical concerns when dealing with this model. Such concerns if this model was used by criminals to locate endangered animals, or use the model to track valuable owned animals using trap cameras to steal or kill animals. Or additionally, this model could be repurposed to be used to track other objects.

# 1 Introduction

## 1.1 Motivations

Knowing an animal's habitat is essential for a variety of sciences. In recent years there has been a steady decline in the number of animals [4]. Technology has played a big part in keeping track of animals as well as locating them.

It informs, of what the ecology of a site is like without disturbing the wildlife living there. Tracking and locating animals using technology has been used by researchers to keep note of the population of certain species [5]. This has been helpful in finding out which animals are endangered and which ones are thriving. Using technology to help track and locate animals gives an advantage at some sites where dangerous animals are life-threatening to humans.

Typically tracking animals is done using trap cameras[6]. These trap cameras capture high-end videos when triggered. The triggering mechanism is usually set when the set number of pixels changes drastically, or thermal heat is detected. This setup ends up with lots of false negatives and ends up filling up the storage, which means there are many false positives videos to go through manually [7].

An idea to solve this issue would be to automate the sorting through these false positives using an efficient model that detects when an animal has entered the video frame and allows the user to see precisely which animal was detected by partitioning the image in the video frame thus highlighting the animal. The model will need to process images faster than in real-time so that a user does not have to spend much time waiting.

Therefore in this project, we will aim to look for a modern object detector that can be used to track and detect animals with the objectives of being able to: process video at real-time of 30fps or faster, able to locate animals by partitioning the image from the foliage find the animal and able to identify what the animal is.

While keeping these objectives in mind, we will investigate previous methodologies and see what models or techniques work and what models or techniques do not work and see what datasets we can use for our project

that reflect the animal kingdom. We will then attempt to create a fast, accurate model which can outline and identify animals.

## 1.2 Background

Early works of detection, can be seen in the early works of background removal tools such as the one created by Elgammal [8]. Unfortunately, these frameworks had consistent problems with noise and sub-optimal lighting conditions due to the fact they were based on rules and so were not robust.

As time progressed, newer techniques were discovered, such as the Viola-Jones [9] object detection framework. The Viola-Jones model was used to make a face object detection framework to identify if there was a face.

The Viola-Jones framework [9] worked by using Haar Features Selection. However, the Viola-Jones [9] framework is not robust to detect faces if there is sub-optimal lighting or if the face in question is tilted.

Newer techniques for object detection do not suffer from these issues. This is shown by new object detection models such as YOLO V3 [1], which runs approximately 30 frames per second. Additionally, the newer models are far more robust and have high accuracy scores.

Because older techniques/methods are obsolete, I will focus more on newer models techniques as old techniques are not robust, which solve the tracking and detection problem.

# 2 Literature Review

Over the past few decades, object detection has become better and more mature thanks to discoveries and more, high-quality datasets, [3], [10], [11] for training and testing than in the past. Additionally, there has been improvement in computational power, which has helped to create bigger more complex models. In this chapter, we will look to find the best techniques for object detection and what can be applied to the problem of identifying and outlining animals quickly. To achieve looking for the performance for a model , this chapter will review the traditional and current literature surrounding Object detection; datasets and how they impact the quality of object detection; Two-stage detectors and One-stage detectors; and then Image segmentation.

## 2.1 Traditional Methods

Traditional Methods, although old, can be useful in gaining an understanding of potential pitfalls and give us a foundation for understanding and appreciating newer models. For this subsection, I will start with the least capable and move onto the most capable.

Starting with the Segmentation in Color and Spatial Domain [12], this model shows how a model can group together pixels that were similar and then remove them to leave an image of the target.

The model managed to do group pixels together by using various methods such as Max Quantization Error, a greedy algorithm that reduces global colour quantization by creating new clusters at the place where maximum quantization happens. However, as a downside, this model could not handle complex environments where there was complex lighting.

LeNet [13] developed by LeCun could be argued as a pinnacle of neural networks; it used a new technique called a convolution neural network (CNN) to recognize numbers written by the US postal service. Identifying written numbers is a tricky proposition if pursued in the classical mathematical way as each person has their way of writing numbers, so a technique to optically translate the numbers to text must be robust. The CNN, when trained was incredibly robust as it has very high complexity. LeNet [13] was revolutionary as it beat the performance of a simple Multi-layer perceptron

with backpropagation [14] for images. LeNet [13] shows us a model needs complex rules to be able to robustly detect objects.

Finally, onto more later work created by Viola-Jones [9] in 2001, they use lots of Haar features with an integral image to detect faces. This was a breakthrough as it was very swift compared to neural networks at the time as it used degenerative trees. These trees had 38 stages and over 6000 features, which managed to pick up human faces most of the time; however, they failed when human faces were distorted.

Viola-Jones [9] is an incredibly important paper as it shows that although Convolution Neural networks are very good at recognizing new classifications, they are not optimized as Viola-Jones [9].

These early methods with LeNet [13] being an exception, which had lots of noise from the low-quality cameras, have still managed to form some segregation between objects if they have high contrast. Here we can conclude that an object detection technique does not have to be a complex model to work in simple environments. However, they do need complexity otherwise they will not be able to adjust to complex environments. But having a complex model such as LeNet [13] we show that a model is robust by default as it has more complex rules are formed. Having this knowledge allows us to understand why modern models are more complex to allow for more robustness. For our application, it is desirable to have a robust model so we will now look at modern models and see what knowledge we can use for our own model.

## 2.2 Modern Detectors

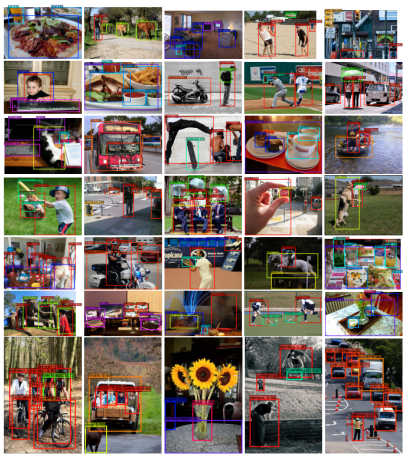


Figure 2.1: Single Shot Detector(SSD) [15] predictions on the COCO dataset [3] showing bounding boxes over images.

Now we have established that we need to use complex models to be able to be robust, we will now need to have a look at newer techniques such as SSD [15] as seen in Figure 2.1. Which, are used currently to solve the tracking and detecting problem. We will then see what techniques or models are used to enhance object detection, and what we can use to make our model.

Modern Object detection use CNNs to detect and identify objects due to there ability to find complex relationships. Modern object detection and localization models use CNNs and fall into two categories: two-stage detectors and one stage detectors. In this section, I will go through each and explain why they are useful and the downsides of each.

A two-stage detector is where we take an image and take snapshots of different areas of an image by a selection method. And then, we perform an analysis on that section. R-CNN [16] is a good example of such a neural network where their approach was to use a selective search [17] to find areas of interest. And then selection search passes areas of interest to a CNN with two outputs: to an SVM [18] and an offsets table for a bounding box. However, this solution has some issues in that the selection search is slow and generates many areas to be tested. Newer techniques, such as Faster-RCNN [19] have solved some of the issues of R-CNN [16], such as reducing the number of regions proposed. Reducing the number of regions proposed was done by creating a Region Proposal Network (RPN). An increase of accuracy was done by replacing the SVM [18] and the offsets for bounding boxes with two fully connected layers; one is responsible for classifying what the object is while the other is responsible for telling what the offsets of the bounding box are. Faster-RCNN [19] has high accuracy and five frames per second speed.

Further improvement has also been seen by using an SPP Network [20], which allows images to maintain their aspect ratio, which has helped performance by around 2%. An SPP network [20] performs multiple pooling but keeps its filter size and step size in proportion.

As we can see from above these models are very accurate but very slow and have two stages. At 6 frames per second these models would be suitable for offline analysis but not for real-time analysis.

Another approach is to use a One-stage detector that can analyze the whole image at once. Then the model would propose which areas which have objects in them.

CenterNet [21], YOLO V3 [1] and SSD [15] network are examples of one-stage models.

YOLO V3 [1] and SSD [15] both work of the same principle, that an image is divided up into grids and inside these grids are cells. Each cell is

responsible for detecting an object if and only if that cell is in the center of the object in question.

YOLO V3 [1] describes that each of these cells predict a probability if there is an object, the coordinates of a bounding box and class probabilities of what the object is. While SSD [15] has each cell predict coordinates of an object and it also predicts the class probabilities plus the background.

Both YOLO V3 [1] and SSD [15] are fast at 45 fps and 46 fps respectively when running on Pascal Voc evaluation [22]. And so can be used for real-time video analysis which is precisely what is needed for our application.

CenterNet [21], has a different approach to the other two models, instead of having grids the model splits the image after processing it through the backbone into three outputs a Heatmap Head, Dimension Head and a Offset Head. Each one of these heads has a specific job. The Heatmap head is used to estimate the center points of objects. The Dimension head is used to predict the sizes of the boxes of the objects and the Offset head is to recover from distorting the image during the downsampling. This technique is quite slow as the model is quite large at 7.4 fps [21].

| | backbone | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|
| *Two-stage methods* | | | | | | | |
| Faster R-CNN+++ [5] | ResNet-101-C4 | 34.9 | 55.7 | 37.4 | 15.6 | 38.7 | 50.9 |
| Faster R-CNN w FPN [8] | ResNet-101-FPN | 36.2 | 59.1 | 39.0 | 18.2 | 39.0 | 48.2 |
| Faster R-CNN by G-RMI [6] | Inception-ResNet-v2 [21] | 34.7 | 55.5 | 36.7 | 13.5 | 38.1 | 52.0 |
| Faster R-CNN w TDM [20] | Inception-ResNet-v2-TDM | 36.8 | 57.7 | 39.2 | 16.2 | 39.8 | **52.1** |
| *One-stage methods* | | | | | | | |
| YOLOv2 [15] | DarkNet-19 [15] | 21.6 | 44.0 | 19.2 | 5.0 | 22.4 | 35.5 |
| SSD513 [11, 3] | ResNet-101-SSD | 31.2 | 50.4 | 33.3 | 10.2 | 34.5 | 49.8 |
| DSSD513 [3] | ResNet-101-DSSD | 33.2 | 53.3 | 35.2 | 13.0 | 35.4 | 51.1 |
| RetinaNet [9] | ResNet-101-FPN | 39.1 | 59.1 | 42.3 | 21.8 | 42.7 | 50.2 |
| RetinaNet [9] | ResNeXt-101-FPN | **40.8** | **61.1** | **44.1** | **24.1** | **44.2** | 51.2 |
| YOLOv3 608 × 608 | Darknet-53 | 33.0 | 57.9 | 34.4 | 18.3 | 35.4 | 41.9 |

Figure 2.2: Average precision (AP) accuracy metrics from YOLO V3 paper[1]. $AP_s$ , $AP_m$ $AP_l$ means how well the models did with small, medium and large sized objects respectively.

One-staged and two-staged detectors both have their upsides and downsides in Figure 2.2. We can see that the accuracy of one stage detectors comparable to the two-stage detectors. This can be seen in the AP levels. AP levels dictate how good a detector is based on its classification of what is in the image as well as how close it is to an IoU metric (how close the predicted box is close to the actual box).

When comparing two-stage and one stage detectors, we can see, there is a speed penalty when picking two-stage detectors with a small benefit of being slightly more accurate as can be seen in Figure 2.2. The major issue with all the models seen is that they do not segregate the individual objects instead they produce bounding boxes to surround objects and so are not as precise. Being not precise can be bad because animals which camouflage themselves are hard to spot [23].

When looking at these models they all have their pros and cons, and there needs to be a choice of what is the most important speed, class accuracy or localization accuracy as each model compromises one to give the other.

## 2.3 Object detection using segmentation

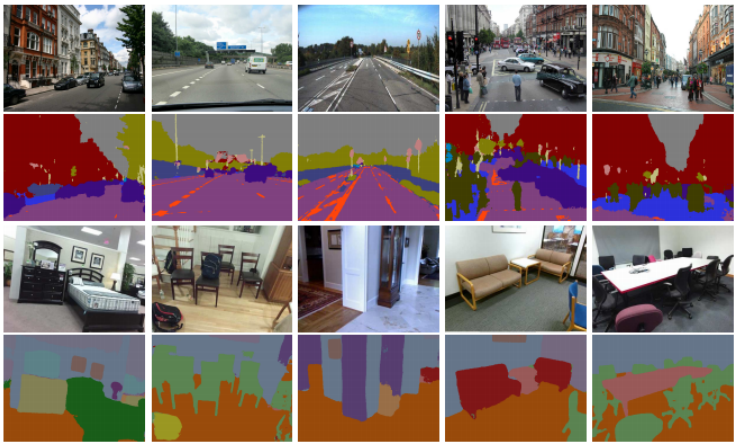


Figure 2.3: SegNet [2] prediction of images.

| Model Name | FPS | Average AP |
|---|---|---|
| SegNet [2] | 16.7 | 0.46 |
| Deeplab[24] | 8 | 0.51 |

Table 2.1: Benchmark results from Deeplab [24] and SegNet [2] on the COCO[3] benchmark

Another approach to Object detection is using segmentation. Segmentation is an approach where an image is taken in and masks are generated for an image therefore separating objects from each other. Having this separation could be useful when locating and tracking animals.

Segmentation was initially shown to work by using fully convolutional networks for semantic segmentation [25]. Segmentation is a new concept, and early works have been made by using autoencoders. Autoencoders are not a new concept the date back to the 1990s [26], they work on downscaling the dimension of data and then upscaling the data thus compressing information in the center of the model.

Deeplab [24] and SegNet [2] both use convolutional autoencoders to take an image input and then output an mask output as seen in Figure 2.3 each colour represents a class. Convolutional Autoencoders work by first convoluting using abacus filters/filters the image into smaller and smaller

images then upscaling and deconvolution the image to create an overall result where all the objects are segmented.

The key differences between Deeplab [24] and SegNet [2] is in there architecture and speed.SegNet [2] uses normal pooling ,in contrast, Deeplab [24] uses a particular type of pooling called atrous pooling, which helps find and keep spatial data together. Speed is an issue for Deeplab [24] as it can only manage 8 fps while SegNet [2] has a speed of 16.7 fps.

Segmentation seems to be promising for using to track and detect animals. However, convolutional segmentation networks suffer from speed issues, on the other hand it has the ability to be able to analyze the whole image which is excellent in the localization of animals. Which can be seen in Table **??**.

## 2.4 Datasets

Having a good model is important but if the quality of the dataset is bad then the quality of the result will be bad too. So it is important to review what is a good dataset and what datasets which are relevant to detecting animals and object detection.

To look for a dataset we must consider the quality of the dataset. According to Dodge and Karam [27] the best datasets must be mixed in terms of image quality to get the best performance from a neural network to perform well. Hence a dataset of images must either contain high quality images which are then preprocessed to generate a mixed quality dataset or a dataset must contain a mixed quality dataset, to begin with.

Youtube-8M [28] is a well known largest video classification dataset holding at the time of writing over 50 years worth of video fully annotated. Youtube-8M [28] managed to do this by using computer-generated and asking humans to verify a large amount of what is in a video frame, with a high level of accuracy, this does mean that some of the dataset is inaccurate in some places. However, the majority is accurate, and Youtube-8M [28] has been shown to improve neural networks.

ImageNet [10] has been used to test train a whole host state of the art neural networks such as AlexNet [29] and has been proven to be high quality and well maintained. ImageNet [10] can be described as an ontology, where each category can be broken down into subgroups and groups, creating a model between images. It was generated by downloading images of the internet using search engines and then humans manually verifying them.

COCO dataset [3] - has high quality data which are have object segmentation's on every object. It has 80 categories and 330,000 datapoints .

Humans spent over 70,000 hours verifying: images, image labels and segmentations. COCO [3] has been used to train many models [1], [30]

All three of these datasets are excellent, but the two which stand out are ImageNet [10] and COCO dataset [3] since they are high quality and have been verified by humans. Additionally, they both have high-quality animal data that can be used to train a model.

## 2.5 Chapter Summary

Tracking and detection are growing more important, as more critical systems now rely on object detection, such as car manufacturers like Tesla [31] who have started to make autonomous cars. Autonomous cars need to be able to use object tracking to be safe on the road, otherwise the results could be life-threatening.

From current literature, we have seen that object detection is possible, but you have to compromise between class accuracy, localization accuracy and speed. Currently, there does not seem to be a model which does all of these well.

If there was a model that could do all of these object detections well, it could allow real-time tracking of animals using cameras and a variety of other applications such as tracking the number of cars on the road for traffic management.

The model would have to be able to identify and outline individual objects even if they were small or close together. In the next section, I will investigate how to make a modern one-stage object detector which has the benefit of having a fast speed and class accuracy and having the ability of object segmentation of closely outlining an object.

# 3 Methodology

In this chapter, we will go through the fundamentals needed to take on object detection. We will investigate how we can create an object detection system using a modern detector that has the ability to predict segmentation masks.

We will first go in-depth on how a CNN works, as all of our modern detectors we reviewed uses CNNs in their models. Having an understanding of a CNN will allow us to compare and choose the correct model.

Then we will go into the design stage and look at what modern single-stage detectors are helpful to our problem and what choices we can make, and why we made them.

After we have reviewed what model we want to use. We will then go into the implementation phase, where we will start making decisions on how we will calculate loss, what dataset we will use, what preprocessing we need , how we can speed up the training of our network, and how we can find how we can find how accurate our model is.

## 3.1 CNN

Convolutional neural networks (CNNs), introduced by LeCun [13], are widely used for vision purposes, due to the fact that they have only a few parameters to train compared to a more traditional feed-forward network. Having fewer parameters allow convolutional networks to be trained faster for the same result of a more extensive feed-forward network . The reduction of the number of parameters to be trained is due to data being convolved by a matrix of parameters. We will be using CNN due to its ability to find complex non-linear patterns in data without human supervision. Having this ability allows us to care about the input and the output solely and not worry about the rules the CNN comes up with to solve our problem.

A CNN takes in an input, typically an image, and outputs a prediction which is a range of values; this is done via training the CNNs weights. A CNN has the ability to detect complex patterns and output our predicted outcome by using three distinct layers which are repeated. These three distinct layers are called the: Convolution layer, Pooling layer and Activation layer.

Furthermore, recent developments in using deep learning have allowed us to make predictions more accurate by using: mini-batching, batch normalization [32], and shortcut layers [33]. Using these techniques will help us build a strong resilient neural network.

### 3.1.1 Convolution Layer

An image contains a matrix of pixels that have a height, width and depth. The depth can be considered colour channels. Most images have colour channels of three: red, green and blue. However, others exist, such as greyscale, which has one channel.

A convolution operator is a, Generalized Linear Model (GLM) used to extract features from images. It does this by using many filters which perform matrix multiplication with an image. A filter is a small matrix with a height and width that is usually odd and small, usually 1x1, 3x3 or 5x5. These filters are applied to an image by stepping through the image matrix with a stride determined through a hyperparameter; the number of filters is now the new matrix's depth. The filters matrix values are learned using backpropagation.

### 3.1.2 Activation Function

The activation function usually happens after the convolution layer; its job is to add a non-linear aspect to the network to learn non-linear features. The activation function is applied to constrain an input.

$$ReLU(x) = max(0, x) \tag{3.1}$$

$$LeakyReLU(x) = max(0.001x, x) \tag{3.2}$$

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

There are many activation functions,the most common and the most used are called Rectified Linear Unit (ReLU), Leaky Rectified Linear Unit (Leaky ReLU) and Sigmoid. These functions can be seen in Equation 3.1,Equation 3.2 and Equation 3.3 respectively.

As you can see, these functions are non-regular, which adds complexity to our model.

### 3.1.3 Pooling

Pooling is the method of reducing the size of the matrix to make it more manageable. To make a matrix smaller, we create a window with a width and height and then convolve it through each of the channels (or depth) of a matrix. The numbers inside the window are then operated on and are reduced to one number. There are two main types of pooling average pooling and max pooling. Average pooling takes the average of the window's values, while max takes the most substantial values inside the window.

### 3.1.4 Mini-Batch

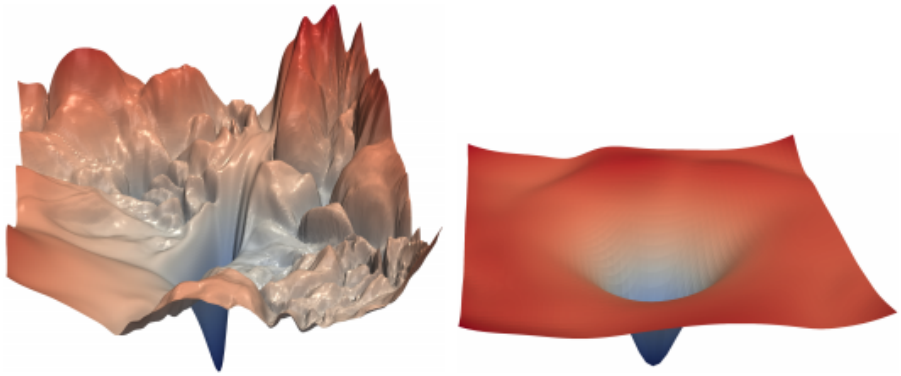
Mini-batch or Mini Batching data is the method of grouping training data for training. The grouped data does not need to be in any order, but they must not be repeated. By doing so you can create a bias towards a particular data point(s).

As an example if there was a group of 100 training data points a suitable strategy would be using 10 data points * 10 mini-batches. Mini-batching has the advantages of been trained faster as you update the gradient of every data point and requiring less memory than training all the data on the model at once.

### 3.1.5 Skip Layer

A skip layer otherwise known as a shortcut layer allows an output from a layer in the model to skip layer(s) to another layer bypassing layers. Shortcut layers are quite prominent in ResNet [33] and give flexibility to the model.

By having shortcut layers, it combats the vanishing gradient problem. The vanishing gradient problem happens when you have very deep nets and the gradient is very small that little change happens in the top layers. Skip Layers combat this by skipping layers of the network thus allowing a larger gradient loss to effect the top of the network. As an added effect having shortcut layers speed up the performance of a network as the gradient valleys morph as can be seen in Figure 3.1.



(a) without skip connections (b) with skip connections

Figure 3.1: Visualizing the Loss Landscape of Neural Nets [34]. This figure shows what happens when you use skip connections compared to when no skip connection are in the model.

### 3.1.6 Batch Normalization

Batch Normalization [32] is the method of normalizing an output from a layer in a network. By having this normalization every layer has a standard input, this has an effect of signals in the network not being overpowered or too small. Due to the fact, the output of the layer is not overpowered or too small. Batch normalization speeds up training the network.

## 3.2 Choosing a model

Now that we have established how a CNN works we can now deep dive into finding what modern detector we should use and modify so we can build a model which predicts segmented images.

I have decided, to use an already existing model and modify it as the existing models created by academia have already been proven compared to creating a model from scratch.

I have also decided to constrain myself to be choosing between one-stage models, YOLO V3 [1] and SSD [15] as they have been proven to have reasonable classifications and localization accuracy.

To compare these two models I will first outline there similarities and then go into detail on how they work and what modifications we would need to do so they outputted a segmentation. I will then compare and decide on which model we should use.

### 3.2.1 Similarities

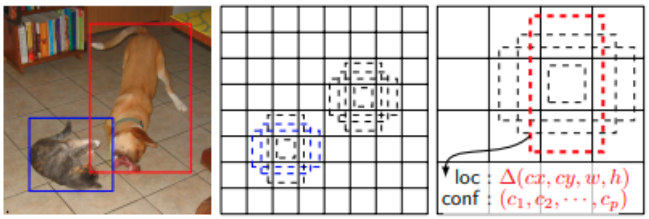



Figure 3.2: SSD [2] grid for predicting objects in an image.

YOLO V3[1] and SSD [15]are one stage detectors; when you run an image through them, they output a prediction. In this case both YOLO V3[1] and SSD [15] output bounding boxes which are boxes which surround an object of interest.

Both YOLO V3[1] and SSD [15] detect objects using a grid system as can be seen in Figure 3.2. In each cell, inside the grid is responsible for detecting if there is a center point of an object is in that cell. If the center point of an object is inside that cell, the cell is responsible for predicting the localization of the object as well as the classification of that object.

In terms of localization both SSD [15] and YOLO V3[1] use anchor boxes. Anchor boxes can also be seen in the last two images in Figure 3.2. The Anchor Boxes are box sizes that have been manually been calculated by pre-computing the distribution of ground truth boxes. When the prediction happens, the closest anchor box is chosen this is then predicted as a bounding box. Anchor boxes are useful when there are multiple center points of objects inside a cell and allow multiple detections to take place.

Another similarity between YOLO V3 [1] and SSD [15] is that they both use Non-Maximum Suppression to limit the number of boxes generated.

The way that Non-maximum Suppression works is in multiple steps:

1. Selects predictions which have a higher confidence score than a manually selected prediction score

2. Iteratively loops through each class and

    a) Picks the highest confidence scoring prediction

    b) Compares this prediction with the rest of the predictions and removes the other prediction if too similar

    c) Puts the chosen prediction into a list

    d) This repeats until no predictions are left in this class

$$IoU(A,B) = \frac{Intersection(A,B)}{Union(A,B)} \tag{3.4}$$

Non-Max Suppression decides on what predictions are similar by using a technique called Intersection over Union (IoU). IoU works by calculating the area of intersection over two shapes and calculating the area of union as can be seen in Equation 3.4.

One final similarity is that both models use all the advantages of Batch Normalization, Mini-batching and Skip-layers.

### 3.2.2 YOLO V3

| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 × 3 | 256 × 256 |
| | Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| 1× | Convolutional | 32 | 1 × 1 | |
| | Convolutional | 64 | 3 × 3 | |
| | Residual | | | 128 × 128 |
| | Convolutional | 128 | 3 × 3 / 2 | 64 × 64 |
| 2× | Convolutional | 64 | 1 × 1 | |
| | Convolutional | 128 | 3 × 3 | |
| | Residual | | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 | 32 × 32 |
| 8× | Convolutional | 128 | 1 × 1 | |
| | Convolutional | 256 | 3 × 3 | |
| | Residual | | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 | 16 × 16 |
| 8× | Convolutional | 256 | 1 × 1 | |
| | Convolutional | 512 | 3 × 3 | |
| | Residual | | | 16 × 16 |
| | Convolutional | 1024 | 3 × 3 / 2 | 8 × 8 |
| 4× | Convolutional | 512 | 1 × 1 | |
| | Convolutional | 1024 | 3 × 3 | |
| | Residual | | | 8 × 8 |

Figure 3.3: YOLO V3 [1] architecture

YOLO V3 [1] architecture is not based on any previous models; it is made of 53 convolutional layers, as seen in Figure 3.3. The architecture outputs to 3 scales, small (13x13 grid), medium (26 x 26 grid) and large (52 x 52 grid) and its input must be 416 pixels by 416 pixels to get those grid sizes.

In YOLO V3 [1] prediction, each cell predicts if the center of an object is in that cell. It does this by using anchors, anchor offsets, a confidence score and a class score. This means that the grid depth can be calculated as seen in Equation 3.5.

$$D = NA * (NC + 4(anchor\ offsets) + 1(confidence\ score)) \tag{3.5}$$

Where D is depth, NA is the number of anchors, NC is the number of classes.

As a modification to YOLO V3[1], to predict a mask. We could use points instead of an actual segmentation image. These points, when joined up together, would form a mask. These points would be used instead of boxes. Additionally, because masks are dynamic, there are no clusters segmentation's which could be used as anchors. Therefore, if we modify YOLO V3 [1] each cell would only detect one object. Therefore the depth would be calculated as seen in Equation 3.6.

$$D = NC + N(X\_Points + Y\_Points) + 1(confidence\ score)) \quad (3.6)$$

Where D is depth, N is the number of points, NC is the number of classes.

After this we could use Non-max Suppression to find the best masks.

### 3.2.3 SSD

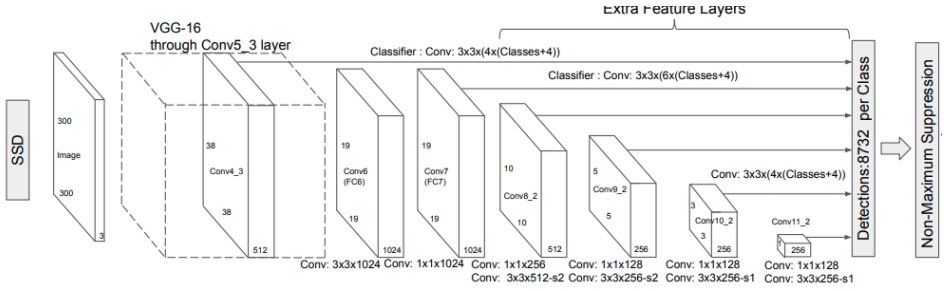



Figure 3.4: SSD [15] architecture

SSD [15] is based of the VGG-16 network [35] as seen in Figure 3.4. SSD [15] takes feature maps from convolutions and convolves them to grid sizes 1x1,3x3, 5x5,10x10,19x19 and 38x38. Each one of these grids has cells. Each cell is responsible for predicting if an objects centre point is in the cell and classifying what the object is.

In each of the SSDs grids like YOLO V3 [1] each cell predicts if the center of an object is in that cell. It does this by using anchors, anchor offsets and a class score + a background score. As we can see, the prediction is slightly different from YOLO V3 [1]. It does not have a confidence score; instead it has a background score. This means that the grid depth can be calculated as seen in Equation 3.7.

$$D = NA * (NC + 1(background) + 4(anchor\ offsets)) \quad (3.7)$$

Where D is depth, NA is the number of anchors, NC is the number of classes.

As a modification to SSD [15] prediction to predict a mask we could use the same technique of using points. Therefore, we would remove the anchors from SSD [15] and the box offsets and we would end up with a grid depth as seen in Equation 3.8.

$$D = NC + 1(background) + N(X\_Points + Y\_Points)) \tag{3.8}$$

Where D is depth, N is the number of points, NC is the number of classes.

And then same as YOLO V3[1] modification, we could use Non-Max Suppression to find the masks.

### 3.2.4 Comparison and decision

To find the best network which we can use for object detection, we must compare YOLO V3[1] and SSD [15]. We need to have a look at their problems, their performance and the amount of effort needed to adapt the network to create points to make a mask.

SSD [15] has one key issue which is that it does not account for focal loss [36] as YOLO V3 [1] does. Focal loss [36] allows the network to address class imbalance and reduces the loss of when the network makes a wrong class choice. While reported in the YOLO V3 [1] paper that it does not have the same issue as SSD [15] probably because YOLO V3[1] uses a confidence score rather than a background class score in SSD [15]. Not accounting for Focal Loss[36] means that the accuracy of the SSD [15] suffers for some classes.

The next metric we can look at to compare YOLO V3[1] and SSD is performance. Performance can be split into two parts accuracy of the model and the models' fps. The accuracy of YOLO V3 [1]and SSD [15] can be comparable by their AP which can be seen in Figure 2.2. We can see that the YOLO V3 [1]outperforms SSD[15] in all AP except for large objects however SSD [15] is one frame per second faster than YOLO V3[1].

Finally, we can look at the effort needed to modify the networks into ones that would output points that could be converted into a mask. This metric is subjective and cannot be measured beforehand and is based on 'gut-feeling' as both YOLO V3 [1]and SSD [15]need modifications. In my opinion,

they both would require the same amount of effort as they both need their losses changed and their network modified.

Based on the fact that YOLO V3[1] is just as fast as SSD [15]and more accurate than SSD[15] and does not suffer from class imbalance I have chosen to use YOLO V3 [1]and modify it so that it outputs points, class accuracy and a confidence score.

## 3.3 Implementation

Now that we have chosen to modify YOLO V3 [1]as our model, we now need to consider how we will implement our loss function so our model can learn. We also need to consider which dataset we are going to use and if we need to do any preprocessing to turn masks into points so that our model can learn as well as what optimizer would be most suitable to train our network.

I have also chosen to name this modified network YOLO-PV as short for You Only Look Once - Polygon Version.

### 3.3.1 Loss function

To get started on choosing the loss functions for YOLO-PV we first need to consider what the previous losses of YOLO V3[1] are. The previous losses can be as seen below:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \tag{3.9}$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \tag{3.10}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \tag{3.11}$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{3.12}$$

These losses above were taken from the YOLO V3[1] paper directly.

Where $\mathbb{1}_i^{obj}$ encodes whether any object appears in cell $i$. And $\hat{x}_i$, $\hat{y}_i$ are the predicted offsets of the bounding box. And $x_i$,$y_i$ are the true values of the bounding box offsets. $w_i$ and $h_i$ are the true width and height of the bounding box while $\hat{w}_i$ and $\hat{h}_i$ is the predicted value of the bounding box. $C_i$ and $\hat{C}_i$ are true confidence score and predicted confidence if the object is inside a cell. $\hat{p}_i(c)$ and $p_i(c)$ is the predicted class and the actual class respectively.

$\lambda_{noobj}$ and $\lambda_{obj}$ are hyperparameters to set when there is an object and when there is no object.

Since we are keeping the classes and confidence score from YOLO V3[1]and YOLO V3 [1]has been able to be trained on, we will keep those losses (Equation 3.15,Equation 3.14). As for the box predictors (Equation 3.10,Equation 3.13) we don't need them so we can remove them.

Instead we need to come up with another loss function to compare the predicted points $\hat{x_{ip}}$ and $\hat{y_{ip}}$ to the actual points $x_{ip}$ and $y_{ip}$. There are a number of standard loss functions out there, we must be careful though not to use a loss function which is not differentiable like IoU. Loss functions need to be differentiable as it gives the model a direction of where to adjust the weights.

Getting inspiration from YOLO V3 [1]and its Equation 3.13 we could use the squared error to work out the difference between the points, or we could use a mean squared error approach. The mean squared error approach would reduce the amount of loss of detecting and outlining an object thus making it less important than classification and confidence, however, on the flip side, the squared error approach would enforce the outlining of an object far greater.

Both approaches are good however we can only pick one as we don't want to double count our loss. Based on the fact that in our problem we are facing objects that have complex shapes, the suitable choice would be to use the squared error approach. Therefore our losses would be as seen in equations

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \sum_{p=0}^{P} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_{ip})^2 + (y_i - \hat{y}_{ip})^2] \tag{3.13}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \tag{3.14}$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{3.15}$$

Where $\mathbb{1}_i^{obj}$ encodes whether any object appears in cell $i$. And $\hat{x}_{ip}$, $\hat{y}_{ip}$ are the predicted points of a mask. And $x_{ip}$,$y_{ip}$ are the true values of the points of a mask. $C_i$ and $\hat{C}_i$ are true confidence score and predicted confidence if the object is inside a cell. $\hat{p}_i(c)$ and $p_i(c)$ is the predicted class and the actual class respectively.

$\lambda_{noobj}$ and $\lambda_{obj}$ are hyperparameters to set when there is an object and when there is no object.

### 3.3.2 Dataset

Considering what dataset to use, we need a dataset that uses segmentation. Moreover, it has the qualities of being accurate and high quality. There is only one dataset which we have reviewed which has the features we want, and that is COCO [3].

The COCO[3] dataset has some advantages and disadvantages. The COCO [3]dataset has a higher quality dataset and is readily trained upon by major models already. Additionally, the COCO [3]dataset has metrics that we can use to compare our model to the other models, readily giving us a metric of seeing if our model performs. The disadvantage of using the COCO[3] dataset is that there are only ten animal categories listed.

Due to the limited amount of quality datasets and the ability to compare multiple models, we will be using COCO [3]for our project.

### 3.3.3 Preprocessing dataset

Our output will be a list of points, classification, grid position, and confidence level in our model YOLO-PV. Furthermore, as seen in our loss section, we need points to compare the predicted points and the true points. To get the true points, we need to convert a segmentation mask to a point list. We can see an example of this here at Figure 3.5 where the high accuracy mask is turned to a low accuracy mask with 32 points.

Figure 3.5: The figure on the left is without any point processing the one on the right is with 32 points

To convert segmentation masks into training points we can use the following method.

Taking an mask as shown in Figure 3.6 and find out its bounding points. We call these $x$,$x_2$,$y$,$y_2$ as seen Figure 3.7. We work out the interval between $x$ and $x_2$ and the same with $y$ and $y_2$ based on the number of points we want to generate per side. In this case, it was three so we placed three points on each side of the mask and then project them to the mask. By doing this, we create several points mapping to a mask. We can see this in Figure 3.8.

By converting a mask into points, we retain 99% of the IoU when using 8 points per side that is 64 points in total. In our model we will be using the maximum amount of points we can fit. The number of points we can fit is 43 points per side (178 points). When mapping this to a mask we get almost $100\%$ IoU so we should not have any issues with loss of accuracy.

To decrease training time, we want to use a precomputed dataset to not wait on masks to be converted to points. Therefore we will precompute the training and the validation data and store it into a TensorFlow TFRecord [37].

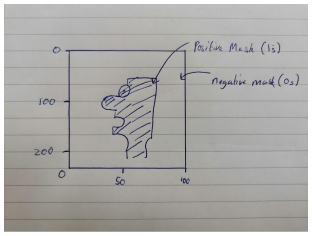



Figure 3.6: This figure shows an example mask

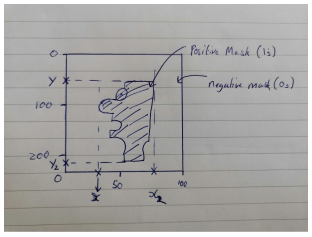



Figure 3.7: This figure shows the projected max/min height and max/min width.

### 3.3.4 Converting model outputs to detections

To understand how we get detections from our model we first need to understand the outputs. From the model we will get an outputs of 13x13x255 , 26x26x255 and 52x52x255 values, which are float32 values.

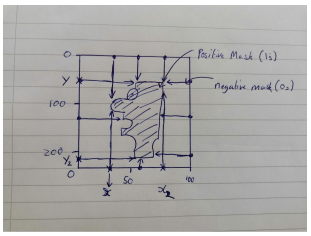



Figure 3.8: This figure shows the projection of the points onto the mask.

Each of these outputs represents a grid where 0,0 is on the top left of an image. From each grid cell, there are 255 values, the first value is the confidence level if there is an image inside a grid cell, the next 80 values are of a binary classification of which classes are active in the grid cell. And finally the next 172 values are grouped into two giving 86 points which joined together form a mask.

However we do not form these masks until the confidence level is above a certain threshold for this paper I will choose 0.6 (60%)initially as this is what is used in YOLO V3 [1] and SSD [15].However this can be experimented on to get the best results.

Once all these predictions have been found which are above the confidence level we will use Non-maximum suppression to only show the best results and filter multiple detections.

### 3.3.5 Transfer Learning

The idea of transfer learning is that networks have already been trained on datasets and have been successful. To speed up creating new experimental or custom networks, you can transfer the learnings from one network to another. Transferring the learnings is done by copying over the previously successful network weights into a new network with parts with the same structure. The copying of the weights onto the new network has the side effect of copying filters, picking up valuable structures in an image. Using transfer learning aids in training a new network as the network does not need to relearn everything from scratch.

For our model we will use the precomputed weights created for YOLO v3 [1]for YOLO-PV as they have exactly the same architecture [38].

### 3.3.6 Optimization techniques

To improve the speed of the YOLO-PV we need to take into account vectorization. Vectorization is the method of streamlining code so large pieces of

data can have their operation processed at the same time rather than going through . Where possible I will remove for loops and use vectorization to improve speed and performance.

To decrease the time the model needs to train we can use mini-batching as it has been proven by many networks that it improves the speed of how quickly the model will converge to a minima. Additionally to speed up the time it takes to train the model we will use a mirrored distributed strategy between two GPUS. Having two GPUs and a mirrored distributed strategy means that when one GPU is processing one batch another GPU can process another batch and then sum up those losses and apply those losses using an optimizer to the model.

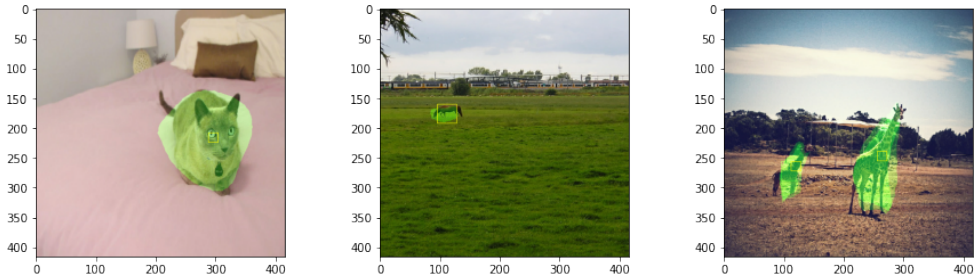
Finally to further decrease the time the model needs to train, the pre-computed dataset will be loaded onto a nvme-drive to enhance read performance.

### 3.3.7 Training

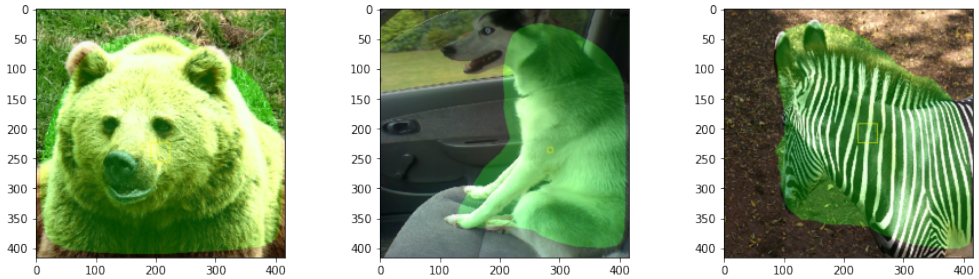
To train the model we will continue our epochs until we start to over-train our model. This is very different to how the YOLO V3 paper [1] suggests we should train our model as they have set it to 300 epochs. However, I believe that we can judge when we start to overtrain our model by when the validation data goes down over time. We do not want to overtrain our model as this causes the network to be less accurate when looking at other images than the training set.

To decrease training time we will use mini-batches. Mini-batches allow us to step in the correct direction and overall this is a faster technique than a stochastic technique. However on every GPU, there is a set number of mini-batches you can use before you run out of memory. There are ways to increase the mini-batch by sharing system memory [39], but this is more costly as it involves moving allot of memory which takes allot of time and is overall slower. This is why we will be using a mini-batch size of 64 which fills up our GPUs memory.

# 4 Results & Evaluation



(a) Cat 88% confidence, Cow 67% confidence, Giraffe x 2 (78% and 89% confidence)



(b) Bear 73% confidence, Dog -82% confidence , Zebra -99% confidence

Figure 4.1: Detected animals using YOLO-PV

In this chapter, we will review how our model (YOLO-PV) performed. As well as, what decisions were made after implementation and evaluate how the model aligns to our aims and objectives. We will additionally talk about the key weaknesses of this model and how future work could elevate these issues.

## 4.1 Modifications

As the model was being trained a few unforeseen modifications had to take place for the model to be trained.

Initially when the model was being trained the model learning rate was set to $\lambda$ 0.0005, this value was chosen as it is the learning value used to train YOLO V3 [1] at the end . Unfortunately, after a small period of time

the model started to diverge, this was shown by the loss factor and the inaccuracy increasing exponentially.

To counter this unexpected divergence, I manually decreased the models learning rate when I saw that the loss factor and inaccuracy increased exponentially. In the end three learning rates were used $\lambda = 0.0005, 0.00005, 0.000005$. These weights lowered the loss and decreased and the accuracy increased.

When looking at the value for what values the non-max Suppression would be best I used a linear search to find the best value starting from 0 and working to 1 in 0.1 increments. It was found that the best value is 0.61 which is very close to my or value of 0.6.

## 4.2 Aims & Objectives Evaluation

When we started this project, we aimed to find a model which could act as a modern object detector with the objectives of: being able to process video at real-time of 30fps or faster, able to locate animals by partitioning the image from the foliage from the animal and being able to identify what the animal is. Below we will compare how our model compares to these objectives and how well they compare to other models.

### 4.2.1 Detection accuracy

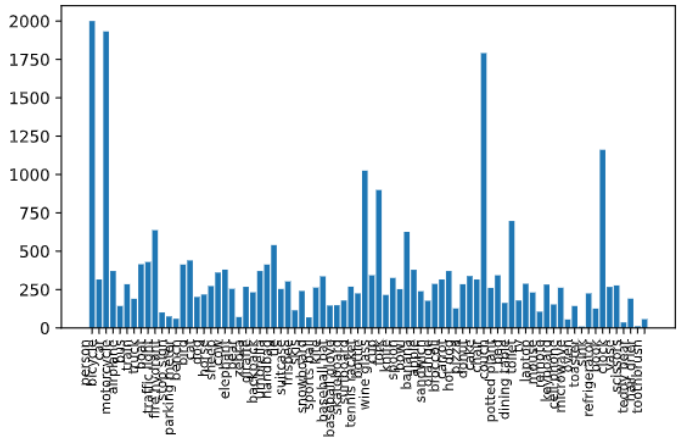



Figure 4.2: [3] validation set distribution

In this subsection, we will analyze how well the model was able to capture the accuracy of categorizing the COCO [3] categories and how well our model can segment our image into masks. To analyze how well we performed, we used the COCO[3] validation set which contains 5000 images!

Each image may have multiple categories inside those images we can see the distribution of those images in the Figure 4.2. In this distribution we can see there are quite a few animals as well as other objects.

From this dataset we can perfrom a COCO [3]benchmark, in this benchmark we got values for the average precision which we can use to compare to other models. We can see the result here in Figure 4.3.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.240
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.520
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.140
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.230
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.340
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.416
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.270
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.141
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.141
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.150
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.250
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.340
```

Figure 4.3: Result of average precision of YOLO-VP calculated using the COCO [3]validation set

| Model Name | Average AP |
|---|---|
| SegNet [2] | 0.46 |
| Deeplab[24] | 0.51 |
| YOLO V3 [1] | 0.33 |
| YOLO V2 [40] | 0.21 |
| SSD [15] | 0.31 |
| YOLO-PV (our model) | 0.24 |

Table 4.1: Benchmark results on the COCO [3]benchmark comparing AP

When testing to see if we have met our objective of detection categories and detection using segmentation we can look at other models to compare and see where we are. Table 4.1 shows that we are not on par with many of the other models but as a proof of concept we do achieve our objective. By inspection on Figure 4.1 we can see the model can separate the animal from the foilage and be able to categories the animal.

When analyzing the Figure 4.3 we can see that YOLO-PV struggles with detecting and classifying small objects. This is shown by the low AP for small objects.

The inaccuracy of detecting small objects could be due to the fact that large objects have a greater chance of being detected due to the fact they span a large amount of the grids while small objects have less opportunity to be detected due to the fact they span less cells.

After analyzing further why the AP was low overall I discovered that it was down to the metric where the categorical precision was very high but when comparing a segmented mask generated from the model to the actual

model the loss was much higher for images which were of small size as the model overestimated/underestimated the size of these objects, this had a knock-on effect of making the general AP low.

Overall, we can say that we have met the objectives of having a good detection accuracy and segmenting the animal from the image, although there is scope to improve this accuracy. We will explore this in the weaknesses section 4.3.

### 4.2.2 Detection Speed

Another crucial section to our model was to see how if our model could process trap-camera footage faster than real-time. Trap-cameras typically operate at 30fps or lower and have a typical resolution of 1920x1080 pixels (1080p).

| Model Name | Average FPS |
|---|---|
| SegNet [2] | 16.7 |
| Deeplab[24] | 8 |
| YOLO V3 [1] | 45 |
| YOLO V2 [40] | 40 |
| SSD [15] | 46 |
| YOLO-PV (our model) | 43 |
| YOLO-PV batched (our model) | 110 |

Table 4.2: Average FPS tested on a Tesla V100 on 10 videos downloaded from Youtube-8M [11]

To test if our objective of running or matching in real time, 10 videos were downloaded from Youtube-8M [11] where animals could be seen, and the video was 30fps and operating at 1080p. The videos were then passed to the YOLO-PV sequentially frame-frame as well as all the other models listed in Table 4.2. Each frame was downscaled using the nearest-neighbours approach. This was taken into account when counting the fps. As we can see from the Table 4.2 we are able to keep up with the rest of the state-of-the-art detectors.,

However, we can use a trick to speed up detection further; since we know the data is offline, we can mini-batch the data and pass it to our model; this allows the GPU (Nvidia v100) to perform multiple processes at once. We managed to speed up processing to an average of 110 fps which is 275% faster than processing it sequentially.

Based on this test, we can safely say that we have objective of making our model faster or equal to real-time. It is possible to make the detection

speed faster however it comes with drawbacks as discussed in section 4.3.

## 4.3 Weaknesses

In this project we have shown a variety of success with YOLO-PV, however it should be noted that there are some key weaknesses in this project which we could not address.

One major weakness are that we were unable to locate suitable night-time datasets with animals thus, our model is unable to detect and track animals. This flaw in our model is important as trap-cameras can be setup during the day or night and some animals are nocturnal such as the owl. Future work could be made into creating nighttime datasets. Fortunately, there are some solutions to this problem already such as using a model which converts a night image to a day image [41] using a CNN, this output can then be fed to our model to be predicted. When we try this method we lose some accuracy but we are able to still find the animals.

Another key weaknesses is that our model, cannot support different resolution of images and only supports an image input of 416x416 pixels. This means that whenever we feed data into our model we lose data as we have to downscale our image to this size using the nearest-neighbors technique which could lead to less accurate prediction. A possible approach to counter this is to use GlobalMaxPooling which will pick the biggest values from the channels section and converting them into grid cells, however this has not been tested and the required amount of channels would be very large which would mean that the model would grow very large and would take more time to process. Another more promising approach is to have a large image resolution where images are unscaled to fit the resolution, this way no information would be lost from different resolution images. Unfortunately this would mean that the model would take allot more time to process the information. Future research is needed in this field.

Although speed is very fast in our model we could make it go faster by pruning weights and making our weights use 8 bit integers instead of 32 bit floats. As integers are faster to process and quicker to transmit to the device doing the computation due to the fact they are very small 1 byte compared to 4 bytes. However there is a large trade off, we have a loss in precision and accuracy which could be critical in some applications in animal detection such as detecting and counting endangered animals and a miss count could cause damage.

One last weakness of this model is in the dataset there were not many areas of where animals were obscured by another object. And so when our model observes an image where an object is obscured it may not

accurately outline the animal. Instead it outlines where the rest of the animal would be. This is a good and bad feature as the model is showing that it understands what the shape of animal should look like on the other hand it is not detecting exactly the shape of the animal on an image. If there was a dataset which contained images where an animal was obstructed this could potentially solve the issue as it would force the network to learn not to shape the general shape of animals.

## 4.4 Failure Cases

When analyzing our model I have found where there are complex scenes our model fails to predict objects accurately. These scenes are usually inside homes where there are lots of reflections and artificial light as seen in Figure 4.4.

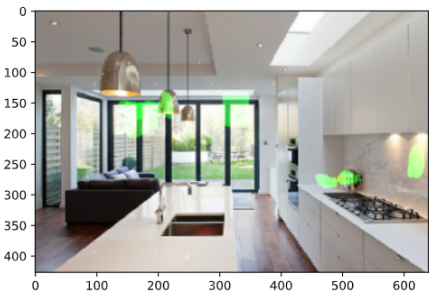



Figure 4.4: Model predicition failing to detect indoor potentially due to complex shadows

Since our model is a CNN we cannot accurately say what the issue is internally, however if I was to have an educated it would be down to reflections and refractions of light of different surfaces. The reason I suggest this is outside light is less reflected and refracted on different substances than internal lighting. Therefore the animals are evenly shaded regardless of the time of day, making it easier to detect. However, since these failure cases are mainly inside and are unlikely places for trap cameras to be setup we can ignore these results.

But we can try to improve these results so that our model is more robust to complex environments, there are several techniques which have been shown to help. Accounting for focal loss [36] , data augmentation [42] and using dropout has been shown to improve accuracy and reduce loss. Unfortunately these techniques have been shown to take a longer time to train and process images.

# 5 Conclusion

In this project, we wanted to help enable detection for trap cameras so that we could detect their positives and only outline when animals is additionally, we wanted the features of the model to be able to identify what and where the animals is in an frame, so that a user could quickly identify these animals from foliage.

We have managed to achieve each of these tough objectives. However, there is plenty of room for improvement, as shown in the weaknesses section.

There are some key implications and learnings which can be taken away from this project. We have shown we can actively reduce the amount of time analyzing trap camera videos from day time footage, meaning that researchers, hobbyists and users who use a trap camera can use YOLO-PV to drastically reduce the amount of human interaction to analyze a video, therefore enabling more time to be spent on more important activities.

Another key implication which can be taken from this project is that you can use this technique not just for animals but for other objects allowing the network to outline the object precisely, this could allow for applications such as traffic management and recognizing when there are hazards on the road and warning drivers ahead of time.

The technique of using points instead of bounding boxes could also be used not just for the YOLO-V3 [1] but other smaller architectures which utilize YOLO-LITE [43] which is used on an old laptop with no GPU. Which has the implication of running on a trap camera to do onboard recognition.

There are large gaps where future work could proceed, making smaller models to generating better datasets. But overall, this technique should be used in the future as it is faster than most other models. Therefore this project suggests that researchers should start making models which specifically outline specific categories rather than whole images or bounding boxes.

# Bibliography


[1] J. Redmon and A. Farhadi, 'YOLOv3: An Incremental Improvement,' *arXiv e-prints*, arXiv:1804.02767, arXiv:1804.02767, Apr. 2018. arXiv: 1804.02767 [cs.CV].

[2] V. Badrinarayanan, A. Kendall and R. Cipolla, 'SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,' *arXiv e-prints*, arXiv:1511.00561, arXiv:1511.00561, Nov. 2015. arXiv: 1511.00561 [cs.CV].

[3] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, 'Microsoft COCO: Common Objects in Context,' *arXiv e-prints*, arXiv:1405.0312, arXiv:1405.0312, May 2014. arXiv: 1405.0312 [cs.CV].

[4] WWF. (). 68% average decline in species population sizes since 1970, says new wwf report, [Online]. Available: https://www.worldwildlife.org/press-releases/68-average-decline-in-species-population-sizes-since-1970-says-new-wwf-report#:~:text=9%2C%202020%20%E2%80%93%20Globally%2C%20monitored,an%20average%20decline%20of%2094%25 (visited on 01/04/2021).

[5] T. O'Brien, 'Camera traps in animal ecology,' in. Jan. 2011, pp. 71–96, ISBN: 978-4-431-99494-7. DOI: 10.1007/978-4-431-99495-4_6.

[6] wildlifetrust. (). Using a camera trap in your garden, [Online]. Available: https://www.wildlifetrusts.org/how-use-camera-trap (visited on 01/03/2021).

[7] MongaBay. (). A snapshot of camera traps reveals user frustrations and hopes, [Online]. Available: https://news.mongabay.com/2019/02/a-snapshot-of-camera-traps-reveals-user-frustrations-and-hopes/#:~:text=The%20passive%20infrared%20sensors%20commonly,blank%20images%20(false%20positives) (visited on 01/04/2021).

[8] A. Elgammal, D. Harwood and L. Davis, 'Non-parametric model for background subtraction,' in *Computer Vision — ECCV 2000*, D. Vernon, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 751–767, ISBN: 978-3-540-45053-5.

[9] P. Viola and M. Jones, 'Rapid object detection using a boosted cascade of simple features,' in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, Dec. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[10] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, 'Imagenet: A large-scale hierarchical image database,' in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[11] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan and S. Vijayanarasimhan, 'YouTube-8M: A Large-Scale Video Classification Benchmark,' *arXiv e-prints*, arXiv:1609.08675, arXiv:1609.08675, Sep. 2016. arXiv: 1609.08675 [cs.CV].

[12] T. Q. Chen, Y. L. Murphey, R. Karlsen and G. Gerhart, 'Color image segmentation in color and spatial domain,' in *Developments in Applied Artificial Intelligence*, P. W. H. Chung, C. Hinde and M. Ali, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 72–82, ISBN: 978-3-540-45034-4.

[13] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, 'Gradient-based learning applied to document recognition,' *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, ISSN: 1558-2256. DOI: 10.1109/5.726791.

[14] D. E. Rumelhart, G. E. Hinton and R. J. Williams, 'Learning representations by back-propagating errors,' in *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, pp. 696–699, ISBN: 0262010976.

[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, 'SSD: Single Shot MultiBox Detector,' *arXiv e-prints*, arXiv:1512.02325, arXiv:1512.02325, Dec. 2015. arXiv: 1512.02325 [cs.CV].

[16] R. Girshick, J. Donahue, T. Darrell and J. Malik, 'Rich feature hierarchies for accurate object detection and semantic segmentation,' *arXiv e-prints*, arXiv:1311.2524, arXiv:1311.2524, Nov. 2013. arXiv: 1311.2524 [cs.CV].

[17] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers and A. W. M. Smeulders, 'Selective search for object recognition,' *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, Sep. 2013, ISSN: 1573-1405. DOI: 10.1007/s11263-013-0620-5. [Online]. Available: https://doi.org/10.1007/s11263-013-0620-5.

[18] T. Evgeniou and M. Pontil, 'Support vector machines: Theory and applications,' vol. 2049, Jan. 2001, pp. 249–257. DOI: 10.1007/3-540-44673-7_12.

[19] S. Ren, K. He, R. Girshick and J. Sun, 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,' *arXiv e-prints*, arXiv:1506.01497, arXiv:1506.01497, Jun. 2015. arXiv: 1506.01497 [cs.CV].

[20] K. He, X. Zhang, S. Ren and J. Sun, 'Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,' *arXiv e-prints*, arXiv:1406.4729, arXiv:1406.4729, Jun. 2014. arXiv: 1406.4729 [cs.CV].

[21] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang and Q. Tian, 'CenterNet: Keypoint Triplets for Object Detection,' *arXiv e-prints*, arXiv:1904.08189, arXiv:1904.08189, Apr. 2019. arXiv: 1904.08189 [cs.CV].

[22] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman, 'The pascal visual object classes challenge: A retrospective,' *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[23] M. Stevens and S. Merilaita, 'Animal camouflage: Current issues and new perspectives,' *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 364, no. 1516, pp. 423–427, 2009. DOI: 10.1098/rstb.2008.0217. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rstb.2008.0217. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2008.0217.

[24] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, 'DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs,' *arXiv e-prints*, arXiv:1606.00915, arXiv:1606.00915, Jun. 2016. arXiv: 1606.00915 [cs.CV].

[25] J. Long, E. Shelhamer and T. Darrell, 'Fully Convolutional Networks for Semantic Segmentation,' *arXiv e-prints*, arXiv:1411.4038, arXiv:1411.4038, Nov. 2014. arXiv: 1411.4038 [cs.CV].

[26] G. E. Hinton and R. S. Zemel, 'Autoencoders, minimum description length and helmholtz free energy,' in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, ser. NIPS'93, Denver, Colorado: Morgan Kaufmann Publishers Inc., 1993, pp. 3–10.

[27] S. Dodge and L. Karam, 'Understanding How Image Quality Affects Deep Neural Networks,' *arXiv e-prints*, arXiv:1604.04004, arXiv:1604.04004, Apr. 2016. arXiv: 1604.04004 [cs.CV].

[28] S. Dodge and L. Karam, 'Understanding how image quality affects deep neural networks,' in *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*, Jun. 2016, pp. 1–6. DOI: 10.1109/QoMEX.2016.7498955.

[29] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks,' in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[30] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, 'YOLOv4: Optimal Speed and Accuracy of Object Detection,' *arXiv e-prints*, arXiv:2004.10934, arXiv:2004.10934, Apr. 2020. arXiv: 2004.10934 [cs.CV].

[31] Tesla. (2021). Tesla home page, [Online]. Available: https://www.tesla.com/ (visited on 01/04/2021).

[32] S. Ioffe and C. Szegedy, 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,' *arXiv e-prints*, arXiv:1502.03167, arXiv:1502.03167, Feb. 2015. arXiv: 1502.03167 [cs.LG].

[33] K. He, X. Zhang, S. Ren and J. Sun, 'Deep Residual Learning for Image Recognition,' *arXiv e-prints*, arXiv:1512.03385, arXiv:1512.03385, Dec. 2015. arXiv: 1512.03385 [cs.CV].

[34] H. Li, Z. Xu, G. Taylor, C. Studer and T. Goldstein, 'Visualizing the Loss Landscape of Neural Nets,' *arXiv e-prints*, arXiv:1712.09913, arXiv:1712.09913, Dec. 2017. arXiv: 1712.09913 [cs.LG].

[35] K. Simonyan and A. Zisserman, 'Very Deep Convolutional Networks for Large-Scale Image Recognition,' *arXiv e-prints*, arXiv:1409.1556, arXiv:1409.1556, Sep. 2014. arXiv: 1409.1556 [cs.CV].

[36] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, 'Focal Loss for Dense Object Detection,' *arXiv e-prints*, arXiv:1708.02002, arXiv:1708.02002, Aug. 2017. arXiv: 1708.02002 [cs.CV].

[37] Google. (). Tfrecord and tf.train.example, [Online]. Available: https://www.tensorflow.org/tutorials/load_data/tfrecord (visited on 02/03/2021).

[38] J. Redmon. (). Yolov3 coco weights, [Online]. Available: https://pjreddie.com/media/files/yolov3.weights (visited on 01/03/2021).

[39] IBM. (2021). Tensorflow large model support, [Online]. Available: https://github.com/IBM/tensorflow-large-model-support (visited on 01/04/2021).

[40] J. Redmon and A. Farhadi, 'YOLO9000: Better, Faster, Stronger,' *arXiv e-prints*, arXiv:1612.08242, arXiv:1612.08242, Dec. 2016. arXiv: 1612.08242 [cs.CV].

[41] N. Capece, U. Erra and R. Scolamiero, 'Converting night-time images to day-time images through a deep learning approach,' Jul. 2017, pp. 324–331. DOI: 10.1109/iV.2017.16.

[42] A. Mikołajczyk and M. Grochowski, 'Data augmentation for improving deep learning in image classification problem,' in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, 2018, pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338.

[43] J. Pedoeem and R. Huang, 'YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers,' *arXiv e-prints*, arXiv:1811.05588, arXiv:1811.05588, Nov. 2018. arXiv: 1811.05588 [cs.CV].

# A Further Acknowledgments

This project was undertaken on the Viking Cluster, which is a high performance compute facility provided by the University of York. We are grateful for computational support from the University of York High Performance Computing service, Viking and the Research Computing team.