DETECTING NEW ZEALAND BIRDS IN REAL-TIME WITH YOLOV3

Ben Cravens

Victoria University of Wellington

ABSTRACT

I took a pretrained YOLOv3 object detection model and fine-tuned it to detect New Zealand bird species. The model works well and is able to accurately detect several bird species in both images and video. The YOLOv3 model is fast enough to deploy in real-time detection settings on a security camera or embedded device.

1. INTRODUCTION

For this assignment, I wrote a report on the paper "YOLOv3: An Incremental Improvement".[9] This paper was a tech report presenting improvements to the original object detection model YOLO presented in "You Only Look Once: Unified, Real-Time Object Detection"[10]. To understand YOLOv3 you must first understand the previous models, which it is a modification of, so I will explain v1 and v2[8] first. To begin, since YOLOv3 is a form of convolutional neural network, I will give a brief refresher on convolutional neural networks.

2. BACKGROUND THEORY

2.1. Convolutional Neural Networks (CNN)

A CNN is an architecture that is used primarily in computer vision tasks[3]. CNNs use two special types of layers, pooling and convolutional layers. I will explain these in depth below, but in simple terms, they alternate these convolutional layers (which find patterns) and pooling layers (used for downsampling) to perform feature extraction. These extracted features are then passed to a series of fully connected layers to deduce a classification or detection decision.

Convolution layer work by using filters to perform the convolution operation on a given input. The convolution operation is defined in the discrete case like so:

$$(f * g)[n] = \Sigma_{m=-\infty}^{\infty} f[m]g[n-m]$$

The convolution operation flips the image g horizontally. It then slides the filter f (which is a grid of numbers) across the image g, computing the element wise sum of the filter and the patch it overlaps at each point. A setting called "stride" determines how many grid spaces the filter moves at each point: a high stride corresponds to downsampling. Intuitively the convolution is calculating a spatially invariant measure of "similarity" of f and g. The values in the filter f are learned to detect patterns in the image. A large "activation" corresponds to a high likelihood the feature f represents occurs in g. At each convolution layer there are several filters which each learn different patterns.

After convolution come pooling layers, which have no learnable weights but simply downsample the image. This downsampling saves on memory and compute, and prevents overfitting, which gives the network a greater ability to generalise. The most common type of pooling is max pooling, which reduces a grid of points to the maximum value. When put after a convolutional layer, this is preserving the most important derived feature, i.e the one with the maximum activation.

CNNs are able to learn to detect or classify complex higher order patterns in images (like "dog" or "lamp"). They do this by learning a hierarchical representation: the filters at the initial layers learn the simplest features such as corners and lines. The activation of these simple features are combined as you go through the network from left to right, with the later activation representing higher level abstract features such as the "outline" of a face made by combining corners and lines. The last layers are then feed-forward layers which learn to combine the "presence" of different high-level features to determine a classification probability.

2.2. Object detection methods

Object detection is a subfield of computer vision in which the goal is to detect instances of objects in images. In practice this entails drawing a bounding box around detected objects, with an associated class and a confidence score (which ranges from 0-1). Object detection methods can be split into two types, two stage or one stage.[10] Two stage methods are slower and more accurate. They work by first running methods to detect regions where there may be an object. Once they have candidate regions they then run classifiers over these regions. One stage methods directly predict bounding boxes and their classes from images. YOLOv1 was the first one-stage detector. The speed of one-stage detectors makes them suitable for real-time detection.

2.3. YOLOv1

Before YOLO, two-stage detection methods were the standard. This meant that real-time detection on consumer devices was impossible. YOLO makes it possible by re-framing the object detection problem as a regression problem, directly predicting bounding boxes and their classes from input images.

It does this by first dividing the image into an $S \times S$ grid. Each grid cell is "responsible" for an object if its center is located inside the grid cell, although the object can be larger than the cell. YOLO then predicts B bounding boxes for each grid square. A bounding box is defined by [x,y,w,h]. These are relative to the grid square and range from 0 to 1. Each bounding box also has an associated confidence score, C that represents how confident YOLO is in an object really being there. Given this "object detected" confidence score, YOLO is able to calcuate a likelihood of each class of object being centered in a given grid cell. I will now show the math behind this. First we let $P(O)$ be the probability of an object occurrence and I(T,P) be the IOU loss between the true and predicted bounding boxes. The IOU loss is a way of measuring the geometric difference between predicted and true bounding boxes and is defined like so

$$I(T,P) = \frac{\text{BBs intersection area}}{\text{BBs union area}} = \frac{T \cap P}{T \cup P} \tag{1}$$

Given this, we calculate the aforementioned "object" confidence score

$$C = P(O) * I(T,P) \tag{2}$$

Once an object is detected, it must be assigned a class. The likelihood of each class y_i given that there is an object is $P(\mathrm{y}_i|O)$. This value is predicted for each grid cell. Given this, we can finally calculate a class confidence score C_i

$$C_i = P(y_i|O) * C \tag{3}$$

$$C_i = P(y_i|O) * P(O) * I(T,P) \tag{4}$$

$$C_i = P(y_i) * I(T,P) \tag{5}$$

This class specific confidence score takes into account both the likelihood the object is present $P(y_i)$ and the quality of the bounding box $I(T,P)$. When running inference, a specific threshold can be set for detection, if the confidence value is over that threshold then that class y_i has its bounding box detected and drawn to the image.

2.3.1. Architecture

YOLOv1 is a deep CNN that has 24 convolutional layers followed by 2 fully connected layers. The convolutional layers perform feature extraction, while the fully connected layers take the extracted features, combine them, and predict the bounding box regression variables. Throughout it uses 1x1 convolution layers to perform feature-wise dimension reduction. As this article explains[1] if we take a Conv2D input of shape [B,F,W,H] and perform a 1x1 convolution with N filters (where $N < F$) this gives a smaller (feature-wise) output of shape [B,N,W,H]. Subsequent YOLO object detection models have a similar structure. First, there is a feature extraction module (known as a backbone) which is block of convolutional layers, which may include pooling, residual connections, etc. After this is a section called the "neck", which is made of fully connected layers. It takes the features extracted by the backbone and combines them so a detection decision can be made. In YOLOv1, the neck is a single feed-forward layer, but in v3 it is a whole module. Lastly, the neck passes its combined features to the head (again in YOLOv1 a fully connected layer) which makes a final detection decision.

2.4. Loss function

YOLOv1 optimizes the following multi part loss function. It is based on taking the mean squared error between (respectively): the real and predicted location, shape, object confidence (in cells with an object), object confidence (in cells with no object), and class probability (in cells with an object).

$$\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{3}$$

Fig. 1. MSE loss function. Note that $\mathbb{1}_i^{obj}$ denotes this term applies to the object appearing in cell i and $\mathbb{1}_{i,j}^{obj}$ denotes the jth bounding box predictor in cell i. Taken from [10]

The authors found that because of the relative abundance of empty cells, the loss from empty cells dominated the gradient. Thus the cells with objects have a scaling factor $\lambda_{coord} = 5.0$ to scale up their contribution, and cells without objects have a scaling factor $\lambda_{noobj} = 0.5$ to scale down their contribution.

2.5. Impact

YOLO was the first object detection model to frame detection as a regression problem and directly optimize object detection performance. It immediately reached state of the art for object detection. YOLO was so fast that it made it possible to perform real-time object detection with limited compute (i.e a laptop or an embedded device). Starting with v1, the

YOLO object detection models have revolutionised the field of computer vision and they are now ubiquitous in every object detection application.

2.6. YOLOv2 (YOLO9000)

This section is based on the following article[12] and the YOLO9000 paper.[8] YOLOv2 introduced several tweaks to YOLOv1 to improve its accuracy and speed. It also increased the number of objects YOLO could classify. The primary change in YOLOv2 that is relevant to v3 is the move from bounding boxes to anchor boxes.

2.7. Anchor boxes

An important feature introduced in YOLOv2 that YOLOv3 shares is anchor boxes. In contrast to bounding boxes which are calculated fresh each time, anchor boxes are generated by predicting offsets from pre-defined bounding box (BB) templates. These templates are made for each class by performing k-means clustering on the BB labels in the training data. The regression model then predicts offsets $[t_x, t_y, t_w, t_h]$ from the template [x,y,w,h]. The authors did this because they found it was easier to learn offsets from a given template than it is to learn the standard BB template for each class by scratch. Because objects may commonly exist in a few different configurations (i.e you may see a truck side on, or from the front) the authors set k=5 so each template will have 5 variations. [1]

2.7.1. Anchor boxes: formal definition

The anchor box priors are defined by the centroid $[c_x,\ c_y]$ and the width and height priors $[p_w,p_h]$. Four co-ordinates are predicted which are used to calculate the bounding box co-ordinates as offsets from this template: $[t_x,t_y,p_w,p_h]$.

$$b_x = c_x + \sigma(t_x) \tag{6}$$
$$b_y = c_y + \sigma(t_y) \tag{7}$$
$$b_w = p_w \exp t_w \tag{8}$$
$$b_h = p_h \exp t^h \tag{9}$$

[1] https://stackoverflow.com/questions/52710248/anchor-boxes-in-yolo-how-are-they-decided?rq=1

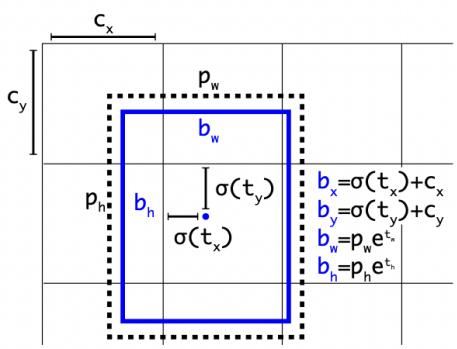

Fig. 2. Taken from YOLOv3 paper[9] (uses same anchor boxes as YOLOv2)

3. YOLOV3

3.1. Introduction

YOLOv3 was introduced in the paper "YOLOv3: An Incremental Improvement".[9] As the paper title suggests, it is a new version of YOLO with several new features added that modestly improve its performance. Firstly it has a new way of calculating the loss for class likelihood and confidence predictions. Secondly, it extracts features on three different scales allowing it to classify small, medium and large objects. Lastly, it has a new backbone network for feature extraction, Darknet-53. This backbone is significantly deeper and utilises residual connections for a more power object detection capability. In general YOLOv3 has higher accuracy but is slightly slower (more than offset by the increase in compute power since the earlier models).

3.2. Binary Cross-Entropy loss for object confidence and class probability

In YOLOv3, the loss function is slightly modified from the YOLOv1 loss function I explained previously. Firstly, as we saw in YOLOv2 we are predicting offsets from a template, not predicting coordinates and width. So in YOLOv3 the BB error is the MSE between the real and predicted offset terms. In YOLOv3, classes are not exclusive. You can have a dog that is also an animal, for example. Therefore the last three terms in the YOLOv1 loss function (confidence there isn't/is an object, as well as class probability) are treated as independent regression problems for each bounding box. Because we are using logistic regression we thus switch from the mean squared error loss function to categorical cross-entropy loss. The predicted output is also no-longer passed through the soft-max function to turn it into a probability distribution (where all class likelihood terms add to 1.0). Instead we use sigmoid activation function $\sigma(y)$ to map each class probabil-

ity to the range [0,1].

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{10}$$

This is because each class has an independent probability from 0.0 to 1.0 of occurring in a given bounding box that contains an object. As described above, the loss function of logistic regression is a piecewise version of cross-entropy called "binary cross-entropy". Cross-entropy is a standard way of measuring the difference between two probability distributions in classification tasks. We use the binary version because each class has two outcomes, it either appears or it doesn't. The BCE loss function can be written as a single equation in terms of T (1 if object exists in cell, 0 else), and y, which is the logistic regression output prediction

$$L(T, y) = -T \log(y) + (1 - T) \log(1 - y) \tag{11}$$

So for the object confidence term (T=1) we have the loss

$$L(1, y) = -\log(y) \tag{12}$$

And for the no-object confidence term we have the loss

$$L(0, y) = \log(1 - y) \tag{13}$$

Overall the loss function for a single scale (remember it is predicted on 3 scales, small, medium and large) can be written like this (I omit the $\mathbb{1}_i^{obj}$ object/noobject notation for clarity. Sums that apply to cells with objects have a $\lambda_{obj} = 5.0$ factor, sums over cells with no objects have a $\lambda_{noobj} = 0.5$ factor)

$$\lambda_{obj} \Sigma_{i=0}^{S^2} \Sigma_{j=0}^{B} [(t_x - \hat{t}_x)^2 + (t_y - \hat{t}_y)^2] \tag{14}$$

$$+\lambda_{obj} \Sigma_{i=0}^{S^2} \Sigma_{j=0}^{B} [(t_w - \hat{t}_w)^2 + (t_h - \hat{t}_h)^2] \tag{15}$$

$$+\lambda_{obj} \Sigma_{i=0}^{S^2} \Sigma_{j=0}^{B} [-\log(C_i)] \tag{16}$$

$$+\lambda_{noobj} \Sigma_{i=0}^{S^2} \Sigma_{j=0}^{B} [-\log(1 - C_i)] \tag{17}$$

$$+\lambda_{obj} \Sigma_{i=0}^{S^2} \Sigma_{c \in classes} [BCE(p_i(c), \hat{p}_i(c))] \tag{18}$$

The specific implementation of YOLOv3 that I used also uses focal loss[4] which adds a scaling term $(1 - p_t)^{\gamma}$ to the binary cross entropy loss. This shrinks the loss for "easy" examples where the estimated probability for the class p is higher than 0.5. Thus the error of rare, hard examples is emphasises and the network focuses more on correcting them (similar to MSE loss).

3.3. Darknet-53 backbone

In object detection, the "backbone" network is the part of the detection architecture that performs feature extraction (which is the majority of the compute, hence "backbone"). In the previous YOLO models, a convolutional network of < 20 layers was used in both YOLOv1 and YOLOv2. In YOLOv3, a 53 convolutional layer backbone model is used, hence the name Darknet-53. The model uses successive 3x3 (for feature detection) and 1x1 convolutions (for feature dimension reduction) as well as residual (or skip) connections which allow for the network to be deep and still train. Residual connections are simply connections that directly add the result of a earlier layer to the input of a later layer without processing it through the intermediate layers. Residual connections were introduced in this paper[2].

Residual connections are useful because in general, having a deeper model alows for greater representation complexity and generalisation. However deep models run into two problems while training that residual connections solve. Firstly, deep networks suffer the vanishing gradients problem. This is when gradients disappear while propagating through the deep network. Residual connections solve vanishing gradients by allowing the gradient to "skip" and avoid diminishing. Secondly, there is a problem called "gradient degradation" where adding more layers overwhelms the model and its accuracy goes down as it is too complex to train. Residuals solve gradient degradation by allowing the network to gradually ramp up the complexity of its representation by initially skipping layers and then integrating them over time.

	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
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

Fig. 3. Darknet-53 backbone architecture (taken from YOLOv3 paper)

When released, Darknet53 was more powerful than previous architectures, and similarly powerful to its main competitor resnet (more FPS, less FLOPS, same top-5 accuracy). The YOLOv3 paper has a good comparison table but it was too large to include in the report.[9]

[2]https://arxiv.org/pdf/1512.03385v1.pdf

3.4. Spatial Pyramidal Pooling

The version of YOLOv3 I used also had a special neck module after the feature extraction backbone, called "Spatial Pyramidal Pooling."[2] This module allows for arbitrary sized image inputs by applying a set of max pooling operations to the extracted features in parallel. These pooled outputs are then combined together to create an output of fixed size. SPP also increases the MAP of the YOLOv3 network by a few percentage points as the input images do not need to be cropped or reshaped which can degrade their quality.

4. EXPERIMENTAL METHODOLOGY

4.1. Downloading images

The first thing I did was procure a dataset of New Zealand bird images. The YOLOv3 documentation[11] suggested to have around 2000 images for each class. After looking around for a while I found a script on github[7] that I used (with modifications) to scrape images from the Macaualy Library, an online databaset of bird photos.[5]. I did this for the species Kea, Tui, and Takahe. I didn't want to do too many species because I would have to label 1000s of pictures by hand for each species which takes a long time.

4.2. Processing images, data augmentation

I was able to get between 500-1000 images for each class, which is not the recommended amount, so I decided I would have to do data augmentation to increase my training set size. I then split my data into train, test and validation sets to avoid information leakage. I then performed data augmentation on my training set to increase its size. To label the images, I used OpenLabelling[6], which is a open source GUI made to label YOLO images. To speed up the process I also wrote a shell script to create a data pipeline (included in submitted files).

4.3. Training

I followed the ultralytics yolov3 documentation[11] and fine-tuned a pretrained yolov3 model on my labeled training images using a GPU in colab. It took about two hours to train for 75 epochs using colab's premium GPU, a NVIDIA Tesla v100.

5. RESULTS

I ran detection on a test set of 111 images. I lowered the detection hyperparameters conf_thresh=0.2 and IOU_thresh=0.35, from the defaults of (0.25,0.45) as the model has slightly low confidence. There were a couple of false positives, but overall the detection was very accurate. The results can be viewed in the submitted results folder "images" in the parent folder "results" at the google drive folder linked in the abstract.

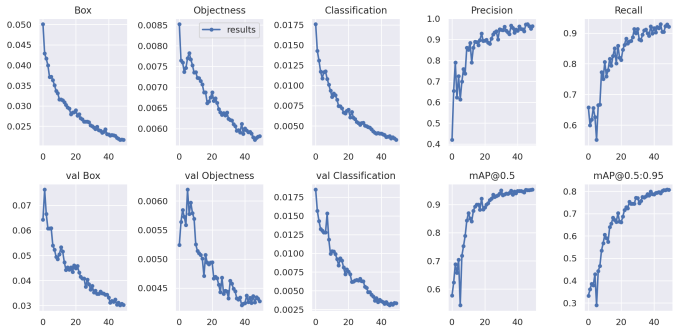

Fig. 4. Loss metrics for 75 epoch run

Bird	Proportion Correct Boxes	Percent Correct Boxes
Tui	59/60	98.3%
Kea	41/43	95.3%
Takahe	33/33	100%

5.1. Real-time detection

The model performs well at realtime object detection. I have prepared a demo in the file "v3video.mp4" in the submitted "results" folder. In it you can see the model running single shot detection on videos of birds. As the algorithm can be run in real-time it can effectively function as a bird "tracker".

5.1.1. Extra credit YOLOv7 model

When I originally started this project, I used the latest model in the series YOLOv7[**yolov7**] to create my bird model. Unfortunately when digging into the paper I discovered because it is a preprint there wasn't enough detail to make a report on. Therefore I switched to v3. I did this because v6 is made for embedded devices, v5 has no paper, and the v4 paper wasn't as good to write about as the v3 model (it was overly complex). In the future if I end up doing something with this project it will be extending the v7 model because of its performance. You can view a video demonstration of it in the submitted file "v7video.mp4". which is in the results folder.

6. CONCLUSION

I fine-tuned a pretrained YOLOv3 model to detect three New-Zealand bird species. The model works well at real-time video detection. Along the way I constructed an efficient data pipeline that could be extended given a budget for labelling and compute to encompass all NZ birds. This could have potential conservation, research, or consumer applications.

References

[1] *1x1 convolutions*. https://towardsdatascience.com/1x1-convolution-5219bbc09027. Accessed: 2022-09-21.

[2] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *Computer Vision – ECCV 2014*. Springer International Publishing, 2014, pp. 346–361. DOI: 10.1007/978-3-319-10578-9_23. URL: https://doi.org/10.1007%2F978-3-319-10578-9_23.

[3] Yann LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *NIPS*. 1989.

[4] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2017. DOI: 10.48550/ARXIV.1708.02002. URL: https://arxiv.org/abs/1708.02002.

[5] *Macaulay Bird Database*. https://www.macaulaylibrary.org. Accessed: 2022-09-21.

[6] *OpenLabelling*. https://github.com/Cartucho/OpenLabeling. Accessed: 2022-09-21.

[7] *Pittsburgh Bird Image Scraper*. https://github.com/ankurdave/macaulay-bird-species-pittsburgh/blob/master/scrape-macaulay-search-csv.py. Accessed: 2022-09-21.

[8] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. DOI: 10.48550/ARXIV.1612.08242. URL: https://arxiv.org/abs/1612.08242.

[9] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. DOI: 10.48550/ARXIV.1804.02767. URL: https://arxiv.org/abs/1804.02767.

[10] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640.

[11] *ultralytics YOLOv3 github*. https://github.com/ultralytics/yolov3. Accessed: 2022-09-21.

[12] *YOLO9000 explained*. https://towardsdatascience.com/review-yolov2-yolo9000-you-only-look-once-object-detection-7883d2b02a65. Accessed: 2022-09-21.