Snake Detection and Classification using Deep Learning

Zihan Yang
School of Computing and Information Systems
The University of Melbourne
zihany1@student.unimelb.edu.au

Richard O. Sinnott
School of Computing and Information Systems
The University of Melbourne
rsinnott@unimelb.edu.au

Abstract

Object detection is a major task in computer vision. With the rapid development of machine learning in the past few decades, and more recently deep learning, it is now possible to utilise complex machine learning models to automatically detect and classify objects from potentially complex images. In this paper we consider machine (deep) learning networks suitable for detection and classification of (Australian) snakes and their deployment and performance in a mobile environment. We explore state of the art Convolutional Neural Networks (CNNs) and their use for transfer learning. We develop an iOS application supporting an offline (model-embedded on the device) approach and an online version where images are sent to a Cloud-based server for classification. We present the results and discuss the performance differences as well as the impact on the accuracy and time for classification for the two environments.

1. Introduction

Computer vision has long been a popular research direction. Its primary goal is to provide automated visual information extraction and image analysis. Unlike text information, graphical data usually requires a higher-level of interpretation and processing [1]. In the past few years, the advancement in hardware now provides sufficient computational power for much more complicated calculation operations required for image processing. Moreover, the rapid development of machine learning (ML) has influenced the evolution of computer vision applications. There has been an increasing interest in applying machine learning techniques for object detection tasks. These have been shown to have superior performance over existing heuristic-based methods [2]. However, in order to fully extract inherent features from input images has historically required complex Artificial Neural Networks (ANN) demanding extensive computational resources for data processing. With the general trend in portable devices and especially mobile phones, advanced machine learning technology is increasingly available for mobile applications. A key challenge is to optimise the image recognition task and achieving adequate performance in varied environments.

This paper focuses on the exploration of several object detection models as well as the underlying classification backbone networks that can be used in a mobile setting. To ground the work we consider a case study comprising 11 different Australian snake species including poisonous snakes.

2. Background and Related Work

Computer vision aims to provide computers with the human-like visual capability of understanding graphical information with no accompanying textual description [1]. Common tasks include *object detection* and *object classification*. Object classification involves categorising an input image into several predefined classes [3]. This provides the foundation for multiple more advanced tasks such as semantic and instance segmentation [4]. Object detection aims to rapidly locate the region of an image where an object of interest may appear [5].

To resolve aforementioned problems, researchers have been using classical pattern matching in 1970s to extract meaningful interpretation from graphical data [6]. This feature-based approach is generally combined with classical machine learning algorithms, including Support Vector Machines and K- Nearest Neighbours [7]. These methods perform well especially on simple classification with very promising efficiency. However, for complex classification in real life, it usually requires extremely complicated models to describe potential solutions, which limits both speed and accuracy of traditional machine learning method.

In recent decade, the most popular approach for visual recognition is based on Convolutional Neural Networks (CNN). Even though there are multiple

URI: https://hdl.handle.net/10125/70760
978-0-9981331-4-0

alternatives to achieve image recognition, CNN have been demonstrated to have superior performance for image classification over them and even humans [8]. Several popular architectures of CNN are often used in practice, including *VGGNet* [9], *Inception* [10], *Residual Network* (ResNet) [11] and *MobileNet* [12].

VGGNet was first published in 2014 [9]. It contained approximately up to 19 layers in total. The main contribution of VGGNet was that it theoretically demonstrated the correlation between the network depth and accuracy. There are two commonly used variants of VGGNet: VGG16 and VGG19 with the number indicating the depth of the network. VGGNet utilizes multiple small convolutional kernels, usually 3×3 to achieve the same size of feature map.

Inception network was the winner of ILSVRC in 2014 [10]. It was specifically designed to optimize the computation power inside the network. With network depth of 22 layers in total, the Inception network maintained a fixed cost of the computational resources by making use of an inception layer. The principal concept of Inception is to capture and repeat the optimal local correlation patterns using dense clusters. Following the ideas in [13], multiple filters within the Inception layer are used to represent the correlation of the previous layer. These filter banks are then concatenated to form inputs to higher layers.

The architecture of modern neural networks has become substantially deeper for better performance. However, as the layers of network increase, the complexity of the training process grows, while the final accuracy may not always be improved due to the vanishing gradient [14]. The goal of ResNet was to solve the training problem of very deep networks without incurring degradation problems. ResNet addresses the degradation problem by explicitly reformulating the structure of networks via residual functions. It has been comprehensively shown that ResNet can be trained using a network eight times deeper than VGGNets, whilst maintaining lower complexity [11].

The general trend of CNNs is to make deeper networks to achieve higher accuracy [9]. This tendency has resulted in a more complicated architecture of CNNs and makes them unsuitable for many realistic environments, especially mobile and embedded computer vision applications. MobileNet approaches this problem from a different direction. It provides a class of more straightforward but more efficient CNN models designed for environments with constrained computational resources [12]. Instead of using a standard convolution kernel, MobileNet utilizes a depth-wise separable convolution filter, which consists of a depth-wise convolution layer and a 1×1 point-wise convolution layer. By providing two global hyper-parameters, MobileNet can trade off latency and accuracy based on the application requirements.

Numerous deep learning frameworks have been put forward for object detection. They are generally organized into two major classes: *two-stage detection models* and *one-stage detection models*. In the former, the detector firstly generates all potential region proposals likely to contain the object of interest and these proposals are sent to a specific classifier for further detection and classification [15]. The most predominant two-stage model is Regions with CNN Features (RCNN) [15]. For one-stage detection modes, the detectors are region-free, i.e. there is no separate procedure for proposal generation during the detection. Instead, the detection process is designed to be unified to predict the bounding box of the image of interest and the corresponding class probabilities [16]. Representative detectors include You Only Look Once (YOLO) [17] and Single Shot Detector (SSD) [18]. RetinaNet [19] was specifically designed to improve the accuracy of one-stage approach via focal loss.

In undertaking any deep learning it is essential to have a large, pure and feature-rich data set used for training and validating the models. As discussed in this work the focus was on Australian snakes.

3. Dataset

In order to ensure a sufficient quantity of training data, the ImageNet database was for data collection. ImageNet provides approximately 3.2 million images over more than $5,000$ categories [20]. However, most of the Australian snake species are not extensively covered in the ImageNet dataset. Therefore, Google Image was utilized to augment the data. For each breed, images were retrieved via a Python crawling script using the breed name as the keyword. However since the quality of data can have a dramatic impact on the model performance, it was necessary to first filter out images with low resolution or irrelevant information. Due to the mobile application requirements, all image data was required to be greater than 224×224 pixels. A summary distribution of all data is shown in Table 1.

For object detection, these images need to be labelled with the location information of the object and the corresponding class (snake type). LabelImg was used for this purpose and data stored in PASCAL VOC format.

For deep learning, more data is required. To address this, data augmentation techniques were applied to increase the size of the training dataset. Augmented

Table 1. Distribution of Australian Snake Data Used

Snake Breed	Number of Images
Bandy Bandy	88
Carpet Python	149
Coastal Taipan	111
Common Death Adder	76
Eastern Brown Snake	147
Lowland Copperhead	62
Mulga Snake	97
Red-bellied Black Snake	65
Spotted Python	73
Tiger Snake	60
Western Brown Snake	99
Total	1027

data can also improve the performance of models when dealing with deformed images. To this end, random data augmentation was employed to accelerate the training speed and improve the accuracy of the model performance. The augmentation process was randomly performed on mini-batches of data at runtime. After that, the augmented data were then input into the model for parameter learning. The actual augmentation methods used included rotation, flipping, scaling and adjustment of the colour contrast.

The collected data and annotations need to be split into different subsets, e.g. training and evaluation datasets. The latter is used for model evaluation and should not be input into the model during training. In this report, the ratio of the training data and evaluation data was set to $9:1$ respectively. The distribution of the test data over 11 classes is shown in Figure 1. In evaluation experiment, there are 100 images used for test in total.

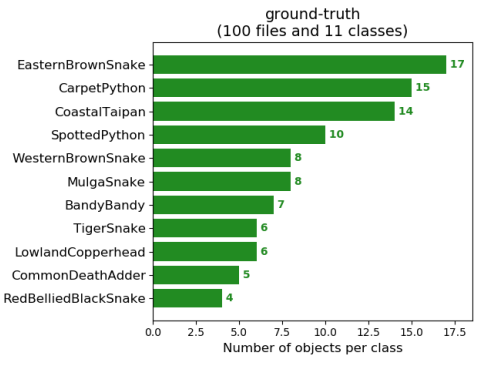

Figure 1. Distribution of the Snake Testing Data

4. Methodology and Experimental Set-Up

To achieve better performance of snake classification, various CNN models were examined. Specifically, they were used as the backbone networks of different object detection models in the experiments. Specifically the work considered VGGNet, ResNet, Inception and MobileNet as introduced previously.

Examples of both two-stage and one-stage frameworks were trained and evaluated. RCNN [15] was chosen as a representative example of a two-stage framework. Specifically a more advanced region-based strategy Faster RCNN [21] was used. The work also considered different backbone networks with Faster RCNN: ResNet and Inception were used for the case study.

For a unified approach, various detectors were selected including SSD [18], YOLO v3 [22] and its less commputationally demanding variation Tiny YOLO, and RetinaNet [19]. They were compared with Faster RCNN in terms of both accuracy performance and inference speed. Since a potential reduction in accuracy was expected, RetinaNet was trained during the experiment to examine the accuracy using a one-stage framework. The differences among these one-stage methods were also taken into consideration.

All base networks were pre-trained using the Microsoft Common Objects in Context (COCO) dataset [23] to ensure basic recognition of common objects. Due to its comprehensive coverage of everyday objects this provides a suitable approach for pre-training in order to provide a baseline model. Obtaining reasonable weights for the base network lessens the burden of the task-specific tuning required.

These baseline models were trained using the collection of Australian snake images as described previously. A cloud server provided by Google CoLab was used for this purpose. The SSD MobileNet, SSD VGG16, Faster RCNN Inception, Faster RCNN ResNet and RetinaNet model training used the TensorFlow framework and Keras APIs, while YOLO v3 and Tiny YOLO used the Darknet neural network [17].

The loss value of the Faster RCNN with ResNet during the training process is shown in Figure 2. A significant reduction of loss is found at the beginning of the training. As the number of epoch increases, it is evident that the loss function decreases (gradually). At around 200 epochs, the decrease in loss value becomes less pronounced. The training loss of the network reaches a plateau, which means that the weights of the network have completed the convergence process and will not benefit from further learning.

Additional optimization can be made to the model

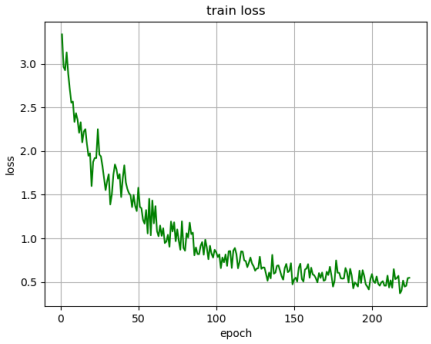

Figure 2. Loss value of Faster RCNN using ResNet during Training

hyper-parameters to accelerate the training and further reduce the training loss. Several parameters were tuned to obtain a suitable configuration providing optimal performance including the batch size and the optimizer. These were found to have a significant influence on the training and accuracy of the model. Figure 3 shows an example visualization of the hyper-parameter optimization for SSD VGG16 model.

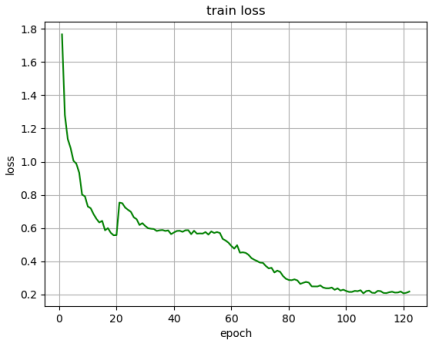

Figure 3. Loss value of SSD VGG16 during Hyper-parameter Tuning

The SSD VGG16 network used a batch_size of 16 and the Adaptive Moment Estimation (Adam) optimizer [24]. During the training, the training loss stopped decreasing at epoch 20, giving a loss value of approx. 0.6. To further reduce the loss, the optimizer was changed to Nesterov-accelerated Adaptive Moment Estimation (Nadam). Nadam provided an improved variant of the Adam optimizer by replacing the vanilla momentum in Adam with a superior Nesterov accelerated gradient (NAG) [25]. From Figure 3, the training loss immediately rises to 0.8 after the change in optimizer but as the training progresses, the loss continues to drop. Furthermore, at epoch 60 the batch size was changed from 16 to 8. Generally, large batch sizes can result in poor generalization whilst a smaller batch size may slow down the training process. It can be seen from the graph that the training loss becomes smoother with batch_size $= 8$. The final loss converges to approximately 0.2 at epoch 120, which is a reasonable value.

The similar procedure was applied to the rest of models with minor adjustment based on their different features to achieve better performance. Overall, batch_size tended to have the greatest effect on the loss value. For given training dataset, larger batch size could greatly accelerate the training process but also led to a higher loss value. In contrast, a small batch size could reduce the final loss while may bring the risk of overfitting [26].

5. Experimental Results

The work used the following experimental environments. YOLO was trained on the Google Colaboratory (CoLab) platform with the following environment: NVIDIA Tesla K80 GPU with 24GB and 12GB RAM. The other models were trained on a desktop computer with an Nvidia GeForce GTX 1060 CPU with 6GB and 16GB RAM. The actual detection experiment was conducted on a standard iPhone X with 256GB storage and 3GB RAM.

5.1. Mean Average Performance

For detecting and classifying Australian snakes, multiple models were trained and evaluated. Specifically, YOLOv3, Tiny YOLO, SSD MobileNet, SSD VGG16, Faster RCNN ResNet and RetinaNet were compared with regards to their mean average precision (mAP). During the testing, one unexpected finding was the poor performance of Faster RCNN Inception model on the test data set. Despite the low plateau of the Faster RCNN Inception train loss during training, the trained model could not detect anything and produced nearly 0 true positive results. This result was probably due to the large number of parameters in the Inception network leading to possible over-fitting. Therefore, the evaluation focused on the other trained models.

The performance of all trained models is shown in Table 2. In order to eliminate variable factors, all base models were pre-trained on the same COCO training-evaluation data set. These base models were then trained and adjusted on the Australia snake training data.

Table 2 shows the summary performance of each detection model. As seen, there is a large difference among the mAP for all models: from 33% to over 81%. This is not surprising given the different complexities

Table 2. mAP on the Australian Snake dataset for all Trained Models.

Model	mAP Performance
YOLO v3	77.08%
Tiny YOLO	66.23%
SSD MobileNet	63.24%
SSD VGG16	55.56%
Faster RCNN ResNet	81.62%
RetinaNet	33.94%

and diverse structures of these models. Generally, it has been considered that a more complicated model achieves better accuracy. For example, Faster RCNN ResNet, uses a significant number of parameters, achieves the highest mAP (over 81%). For simpler one-stage detection models such as SSD and RetinaNet, the mAP is lower. Within the one-stage methods, YOLOv3 has the best performance (over 77%), whereas the RetinaNet model performs significantly worse.

The average precision (AP) of each class is visually displayed in Figure 4. From the chart, it can be seen that for the *Carpet Python* and *Bandy Bandy* classes, almost all models achieve a high AP over 80%. However, for some classes such as the *Western Brown Snake* and *Eastern Brown Snake*, the AP of the models is relatively low and at times zero, which implies that no ground truth of this class could be successfully classified. Other classes show a wide range of AP values with each model. The class *Lowland Copperhead* is one of the most representative examples, with its lowest AP being 0 while the highest AP exceeding 90%.

5.2. Inference Time

The inference time and the corresponding FPS of each model is extremely important. For a mobile application, the inference time can be more important than accuracy to end-users because it can directly affect user experience. Figure 5 shows a boxplot of the original inference time measured in milliseconds. As the figure shows, the inference time of Faster RCNN ResNet is considerably greater than the other models. The maximum value of Faster RCNN ResNet exceeds 14 seconds, while the peak value of other models is no more than 4 seconds.

To compare Faster RCNN with other approaches, a logarithmic scale is used and shown in Figure 6. After data normalisation, it can still be seen that the average time of Faster RCNN is much larger that the other models. The inference time of YOLO series is the least, which makes YOLO the best performing model (in terms of the speed). For the SSD model, the choice of backbone CNN network has a minor impact on the final

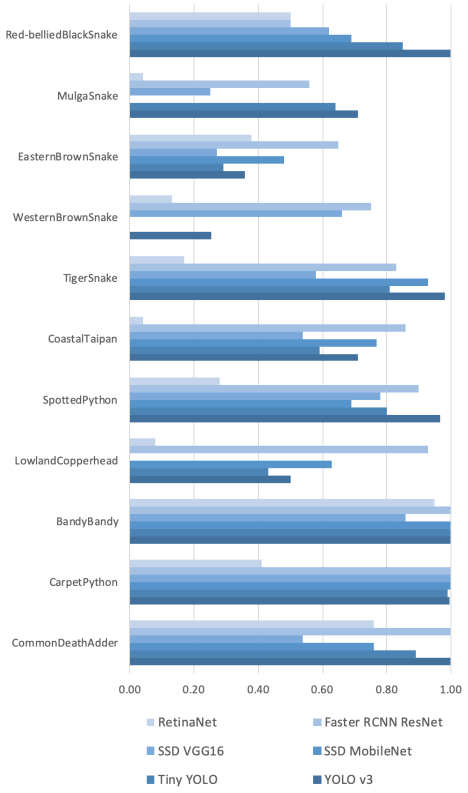

Figure 4. Average Precision Performance for all models with each Class.

inference time. An SSD with a light-weight network such as MobileNet has a superior speed than the SSD using VGG16. While the variation of SSD MobileNet is larger than SSD VGG16, which may imply some instability during the inference process.

The number of outliers in some one-stage models is clearly more than the two-stage model (Faster RCNN). This phenomenon can be explained by model initialisation. The SSD model with older VGG16 networks requires more resources during the launch phase, however it does not affect the later inference process of the model when the network is fully established. What is surprising in Figure 6 is the long inference time required by RetinaNet at more than 2.4 times the SSD model.

The average inference time of all models is shown in Figure 7. All data is visualised without logarithmic normalisation. To avoid the overwhelming skew introduced by the Faster RCNN model, the mean inference time of this model is removed in Figure 7. As with the previous boxplot (Figure 6), the second-longest inference time is found in RetinaNet. It is also clearly shown that YOLOv3 and Tiny YOLO have the best speed during evaluation with Tiny YOLO the best

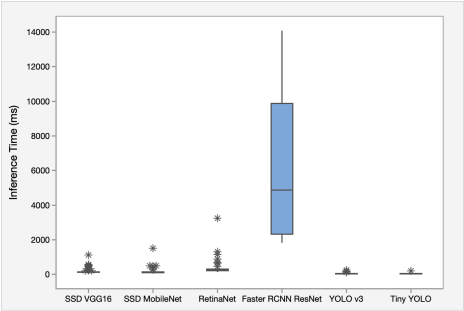

Figure 5. Boxplot of the Inference Time (in ms).

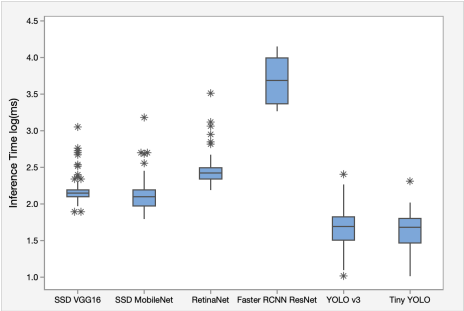

Figure 6. Boxplot of the Inference Time Measured in log(ms).

performance (speed).

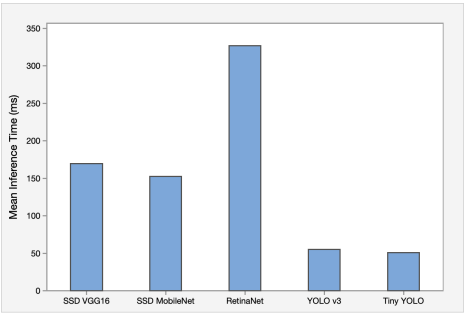

Figure 7. Bar Chart of Mean Inference Time (ms)

5.3. Detection Experiments

From the accuracy and latency performance results given in Section 5.1 and Section 5.2, the models with the top performance are implemented in the iOS application for comparison. Considering the constrained memory space and computational power of mobile devices, only Tiny YOLO was embedded in the application for truly offline detection. At the server-end, Faster RCNN ResNet was deployed to detect the online experiments. A summary of the performance of both models is shown in Table 3.

Table 3. Summary Table of the Tiny YOLO and Faster RCNN ResNet Models.

Model	Inference Time	mAP
Tiny YOLO	49.93 ms	66.23%
Faster RCNN ResNet	5882.43 ms	81.62%

In order to compare the different approaches, two kinds of model implementation were evaluated. Offline (with the model running just on the mobile phone) and online (where the phone was used to send an image to the Cloud server for detection/classification and returning of results).

5.3.1. Single Object Detection To demonstrate the accuracy of these two different methods, an initial experiment was to detect and classify a picture containing a single snake (where the snake image was not in the training data set). In these experiments, the Faster RCNN ResNet model successfully detected nearly all snake instances inside the image, achieving almost 100% recall. In contrast, the embedded Tiny YOLO performed relatively poorly (see Table 2).

One of the failed examples of Tiny YOLO is shown in Figure 8 (a), where Tiny YOLO successfully detects a snake but predicts the incorrect class. In Figure 8 (b), the correct prediction result produced by Faster RCNN ResNet is shown. As seen the snakes have a similar colour to their environment. The ambiguous edge make it difficult for the network to distinguish the snake from its background.

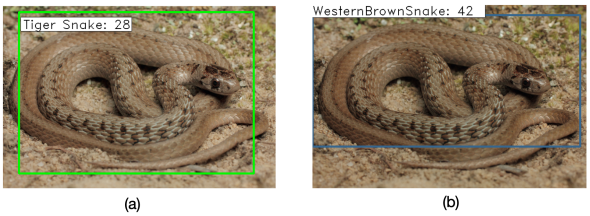

Figure 8. Detection Experiment with a Single Object. (a) Tiny YOLO (b) Faster RCNN ResNet

5.3.2. Multiple Object Detection This experiment is designed to test both approaches when facing a number of snakes within a single input image. It is noted that pictures containing multiple, non-overlapping Australian snakes are hard to find in a natural setting. Therefore, a screenshot of several test images was used as a substitute, as shown in Figure 9.

During this experiment, the recall accuracy of the

offline Tiny YOLO and the online Faster RCNN ResNet were similar. Figure 9 compares the experimental results of the two models. As seen, both algorithms were able to detect and classify the instances, while Tiny YOLO outperformed Faster RCNN with regard to multiple detections. For each sub-image, Tiny YOLO gives very high confidence score of its prediction. This is probably due to the small number of the sub-images within the input. The reason for the poor performance of Faster RCNN could be the unnatural white space between each sub-image. This part of the input greatly affects the feature extraction and thus leads to a reduction in the final accuracy.

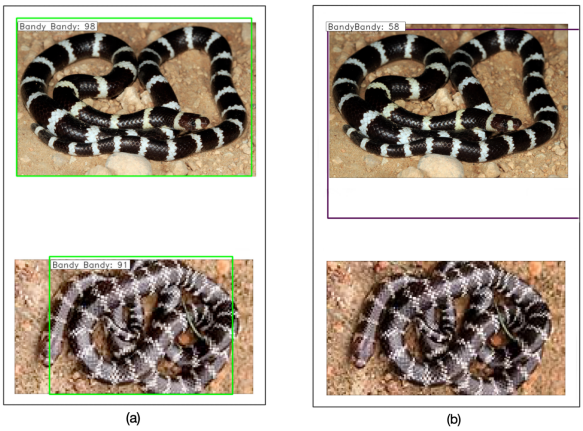

Figure 9. Detection Experiment with Multiple Objects. (a)Tiny YOLO (b)Faster RCNN ResNet

5.3.3. Overlapping Object Detection Instead of a collection of separate pictures, it is much more common to see snakes wrapping around each other. In such a case, most of the snakes would have overlapping parts within the picture, which causes challenges to detection.

In this experiment, the embedded model was found to have limitations in detecting snakes close to one another. For Faster RCNN, the accuracy of the result was greatly decreased. As displayed in Figure 10, Tiny YOLO was unable to clearly separate two overlapping snakes. Instead, it only produced one (imprecise) bounding box that included both snakes. Moreover, the labelled result of Tiny YOLO was also invalid. Faster RCNN could successfully detect two distinct instances within the input. However, the prediction process was affected by the overlapping parts resulting in one snake being incorrectly classified.

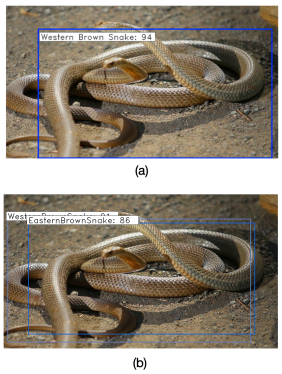

Figure 10. Detection Experiment with Two Overlapping Objects. (a)Tiny YOLO (b)Faster RCNN ResNet

5.3.4. Deformed Object Detection Images with different scales and ratios were also utilised to explore the models in various situations. Overall, the accuracy performance of both Tiny YOLO and Faster RCNN was promising. Both models were able to detect most deformed objects given as input. This was likely due to the data augmentation approach used in the training phase, where the scale and ratio of the labelled data were adjusted and fed into the neural network. This procedure not only increased the quantity of the training dataset but also augmented the object detection.

A sample result for deformed object detection is shown in Figure 11. It can be seen that both the online and offline approach can precisely localise the bounding box of the snake and give an accurate classification result. However the confidence score of Faster RCNN (81%) is higher than the Tiny YOLO (52%), which is consistent with the fact that Faster RCNN achieves a higher overall mAP (Table 2).

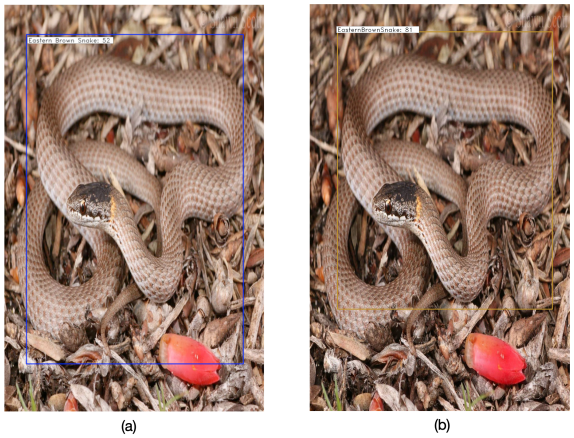

Figure 11. Detection Experiment with Deformed Objects. (a)Tiny YOLO (b)Faster RCNN ResNet

5.3.5. Detection Latency Detection latency refers to the total time that the model requires from taking an image to outputting a final prediction result. The network delay and the model loading time are included as they reflect the end user experience. Table 4 presents the mean inference time of the embedded Tiny YOLO model and the online Faster RCNN ResNet model. The size of the model file is also listed in the table.

Table 4. Summary Table of Two Different Approaches

Approach	Latency	Model Size
Embedded Model	53.81 ms	58.9 MB
Server-end Model	6441.58 ms	114.2 MB

Considering Table 4, the mean latency of the server-based Faster RCNN ResNet is significantly higher than Tiny YOLO at over $6,000$ms. This table can also be compared with Figure 5, which shows approximately the same tendency where the two-stage Faster RCNN demands much more time to process images. What also stands out in this table is the growth in inference time in the mobile environment. It can be seen that the server-based model requires additional time (around 500 milliseconds) for prediction, whilst Tiny YOLO only results in a slight increase. In addition to the network communication delay, this is likely due to the different time in loading the model.

The Faster RCNN model file is 114.2 MB, which is almost twice as big as the Tiny YOLO file. This difference can be attributed to the complex architecture of Faster RCNN and the two stages approach. In contrast, Tiny YOLO is a light-weight one stage detector specifically designed for constrained computational resources and limited storage.

6. Discussion

6.1. Detection Model Comparison

From Table 2, it can be seen that Faster RCNN is the most accurate model at over 80%. This result may be explained by the fact that the two-stage architecture provides better accuracy with its region-based network [21]. The extra proposing procedure eliminates possible missed cases, but demands larger computational resources. Therefore, the pre-processing algorithm improves the overall accuracy but at the cost of the overall detection speed.

MobileNet and SSD provide comparable performance for both speed and accuracy. However, the mean inference time of SSD models still falls behind that of YOLOv3 and Tiny YOLO. The reason behind this phenomenon could be distinct architectures. SSD produces more than one detection from multiple features maps with different scales and ratios, which may be an obstacle to the speed of inference. In YOLO, feature maps are partially flattened and concatenated together into low-resolution maps [17]. The final prediction is then calculated from those maps through a linear regression. The simple regression problem greatly reduces the overall time consumed. The mean inference time decreased by approximately 100ms using YOLO instead of SSD. With the addition of FPN for feature extraction, YOLOv3 also achieved a higher mAP than SSD.

Tiny YOLO has a much smaller size and provides faster inference speed during the experiments. It is designed to further accelerate model prediction by sacrificing some accuracy. According to [22], Tiny YOLO is 4 times faster than YOLOv3, achieving around 200 FPS on the COCO dataset. However the accuracy of Tiny YOLO drops to only 20% mAP. This also accords with the observations in Table 2 and Figure 7, which shows that the Tiny YOLO reports the smallest average inference time among all models, with a 10% lower mAP than YOLOv3.

One unanticipated finding was that no statistically significant improvement in the performance was found when using RetinaNet. RetinaNet adopts focal loss to address the foreground-background class imbalance, which has been shown to have a positive effect on accuracy in previous research [19]. However, this work has been unable to repeat the enhancement in using focal loss. The overall performance, including both speed and accuracy of RetinaNet, was found to be poorer than other one-stage models. Possible reasons could be inappropriate hyper-parameter configurations or low data quality.

6.2. Choice of Backbone Network

Faster RCNN has been the predominant two-stage framework for several years [16]. However, Faster RCNN with the Inception network gave an mAP of approx 0 during the experiments. This finding is unexpected and suggests that the choice of the backbone network for Faster RCNN may have a dramatic impact on the final performance. This finding is consistent with [16] who also identify the key role of the backbone network in object feature representation. The Inception network focuses on increasing the width to ameliorate performance, especially on large-scale computer vision tasks [10]. This measure effectively improves the final accuracy but also introduces a larger feature space,

which has more demands on the quantity of input data. A limited dataset is more likely to result in possible over-fitting. This problem becomes even more pronounced in transfer learning. In contrast, ResNet increases the depth of the network in order to improve the robustness of the model and therefore leads to a better representation of non-linear features. As a result, Faster RCNN ResNet achieves improved accuracy when compared to other models, including Faster RCNN Inception.

There are some CNNs specifically designed for improving speed, such as MobileNet. The difference between VGG16 and MobileNet is shown in Figure 7. The mean inference time of SSD MobileNet is less than SSD using the backbone network VGG16. This finding is consistent with that of Howard et al. (2017), who claims that MobileNet is much less computation-intensive than VGG due to its optimised architecture. In terms of accuracy, SSD with MobileNet also achieves a higher mAP value than SSD VGG16, which is unsurprising. As mentioned in the literature review, MobileNet can be 32 times smaller than VGG16 in terms of size yet it provides the same accuracy [12]. Previous studies also show that the size of VGGNet could be compressed using pruning techniques while still retaining the same performance [27].

6.3. Accuracy and Latency Trade-off

As seen above, there is a negative correlation between accuracy and latency. The models that are able to achieve higher accuracy should inherently consume much greater computational power and thus result in increased latency. However, light-weight networks with fewer operations are typically faster but have poorer performance, especially when detecting small or overlapping objects.

When selecting a suitable model for a mobile device, the trade-off of performance and speed should be carefully balanced. The decision is highly dependant on the specific requirements at hand. For example, if the accuracy is the primary concern, a two-stage detector such as Faster RCNN should be used to provide improved performance. If speed of inference is more important, then Tiny YOLO could be a better choice for embedded-models.

6.4. Quality of Data

The overall performance of deep learning models is greatly affected by the quality of training data. Since the task here focused on Australian snakes, the feature representation of each category was highly related to the appearance of the different species. Some snake breeds may have distinctive patterns or colours on their skin, which can be easily identified by a neural network. There are also several species that look similar to each other, which is likely to result in almost identical feature maps. In such a situation, accuracy would be impacted. For example, the AP value of the *Eastern Brown Snake* is generally lower than the AP of *Bandy Bandy* (Figure 4) even though the latter class contains less training data (Table 1). The reason the vivid characteristics of the *Bandy Bandy*, namely the sharply contrasting black and white rings around their bodies. Therefore *Bandy Bandy* is rarely mislabelled, thus guaranteeing the performance of the model. In contrast, it can be relatively hard to distinguish the *Eastern Brown Snake* and the *Western Brown Snake* - at least for non-herpetologists.

Moreover, the imbalanced distribution of the training dataset may also lead to a wide variance in AP values. As displayed in Figure 4, the AP values of each class can be very different. A larger amount of training data is likely to be associated with a higher AP value, e.g. as seen with the *Carpet Python*. In contrast, the *Lowland Copperhead* has a high variance for the AP, ranging from 0 to around 90%. Such results could be explained by the insufficient quantity of *Lowland Copperhead* images for training. This makes it's AP value highly sensitive to the architecture of the model framework. This issue could be mitigated by constructing a high-quality dataset with well-balanced distribution among each class.

7. Conclusion and Future Work

In this paper we presented two approaches for Australian snake detection and classification: a mobile approach and use of a server. We explored multiple state of the art models. The results of this work indicate that different models can greatly vary in terms of accuracy and their inference speed. Generally it can be said that a more complex network would increase accuracy compared a simpler model, while the latter may achieve swifter inference and require fewer resources.

To get the application useful in the real life, there are many possible future directions to improve the work: extending the data collection, utilising the spatial information. A more diverse training dataset could potentially lead to a significant improvement in accuracy, especially for those models based on a relatively simple architecture. We would also need more images of diverse species to make application capable of recognising different kinds of snakes. For now, it is only able to distinguish 11 snake breeds given the limited training data, while there are over 100 species of land snakes in Australia according to official record.

This information gap would need to be filled with a more comprehensive bioinformation database of snakes.

We could also make the use of geographical information associated with image data in the future. During the classification, users can be localised using GPS sensor in their mobile phone. This location data is critically valuable for prediction, especially when users are using the application to classify the photo they just shoot. For example, Dugites, a venomous snake, is native to Western Australia, which means that it is not likely to see Dugites in the east side of Australia. Thus we can factor this information into the application and adjust confidence score of classification result based on user's real-time location.

References

[1] N. Sebe, I. Cohen, A. Garg, and T. S. Huang, *Machine learning in computer vision*, vol. 29. Springer Science & Business Media, 2005.

[2] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

[3] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[4] A. Karpathy, "Convolutional neural networks for visual recognition," 2016.

[5] P. Druzhkov and V. Kustikova, "A survey of deep learning methods and software tools for image classification and object detection," *Pattern Recognition and Image Analysis*, vol. 26, no. 1, pp. 9–15, 2016.

[6] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[7] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," in *Science and Information Conference*, pp. 128–144, Springer, 2019.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[13] S. Arora, A. Bhaskara, R. Ge, and T. Ma, "Provable bounds for learning some deep representations," in *International Conference on Machine Learning*, pp. 584–592, 2014.

[14] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5353–5360, 2015.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[16] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[19] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[22] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[24] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[25] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[26] A. Rosebrock, "A gentle guide to deep learning object detection," 2018.

[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.