THE UNIVERSITY OF MELBOURNE

Poisonous Spider Recognition

COMP90055 Computing Project (25 Credits)

Type of Project: Software development project

Donghan Yang (680505)

Xueyang Ding (878297)

Zhenyuan Ye (883576)

Supervisor: Prof. Richard Sinnott

Semester 1, 2019

Abstract

With the rapid development of deep learning, people can use neural networks which mimic human neurons to solve problems related to artificial intelligence. Deep learning and neural networks have brought huge attention from all over the world and have been one of the most popular topics in modern computer science and information theory.

One of the most promising applications of deep neural networks is computer vision. Computers can complete simple tasks such as classifying an image or locating an object in an image just like a human. In this project, we explore various models that enable computers to do both tasks in the context of recognizing the most common 9 species of Australian poisonous spiders. Furthermore, we develop an iOS application that could recognize these spiders in real-time anywhere and anytime.

It is an extremely laborious process to collect images for spiders compared with dogs or cats. We still manage to obtain almost 2000 images for all species thanks to google image crawler. The model that we use for image classification is MobileNetV2 and for object detection, we use single-shot object detector (SSD) on top of MobileNetV2 feature extractor. MobileNet runs fast on a mobile device and has reasonably good performance.

We have achieved over 80% of accuracy for spider classification and over 65% average precision for object detection. Due to the hardware constraint of our testing mobile devices and the complexity of the object detection model, we could not deploy a more powerful model to the application without sacrificing the user experience.

Keywords: Australian poisonous spider, object detection, image classification, CNN, MobileNet, SSD, Core ML, Tensorflow, Keras.

We certify that :

- this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university, and that to the best of our knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.

- where necessary we have received clearance for this research from the University's Ethics Committee and have submitted all required data to the Department.

- the thesis is 6093 words in length (excluding text in images, table, bibliographies, and appendices).

Contents

Figures

Tables

Acknowledgment

We would like to express our deep appreciation for Professor Richard Sinnott for selecting this challenging and inspiring project for us. Also, we would like to thank him for his supervision, time and advice along the way. Without his help, we will surely struggle against how to approach this project.

We also appreciate the great efforts made by the developers in the open source community and the bloggers who are willing to share their wisdom with other people. Without them, many tools, APIs and software would not be available so are the awesome blog posts and tutorials.

Finally, we want to thank those who provide the spider images that we downloaded from Google Images. Our project is not possible without these precious photos.

1. Introduction

Spiders are ancient animals with millions of years of history. In Australia, there is a well-known saying 'Wherever you live, you are always close to a spider'. There are many different species of Australian spiders. Some of them are poisonous. For example, the Sydney Funnel-Web spider has the most toxic venom which contains neurotoxins and enzymes that can dissolve muscle tissue [1]. A bitten adult can be killed in 15 minutes without first aid. Every year many people are reported to be bitten by spiders. Table 1 indicates the number of spider bite calls from Victorian Poisons Information Centre (VPIC) in 2016 [2].

Table 1. Spider bite calls to the VPIC during 2016

Spiders	Calls
Redback spider	130
White-tailed spider	75
Spider bite: other/unknown	269

From the table, we can see that the number of accidents is substantial. There are two main reasons. Firstly, the public have limited knowledge to distinguish among various types of spiders. Secondly, some of these poisonous spiders are widely distributed in Australia.

Figure 1 below from BioMed research illustrates the distribution of the Sydney Funnel-Web spider (fatal) and Red Back spider (fatal) in 2014 [3]. This widespread distribution increases the chance of spider bites.

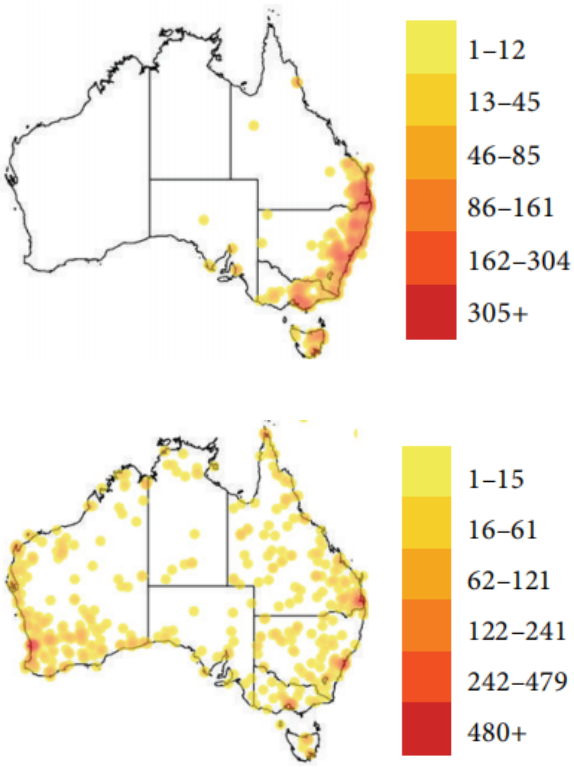

Figure 1. Distribution of the two most fatal spiders in Australia

To help Australian people recognize different kinds of spiders, especially poisonous ones, this report will propose an iOS spider recognition application which can recognize spiders in real-time and can be used anywhere and anytime.

In this report, we will first review some related works on computer vision technology. And we will introduce the dataset we used and how to pre-process these data. The core part will be our machine learning model design and architecture, including how to improve the model performance, what is the trade-off between system efficiency and accuracy. Then we will discuss the application design at a high level. Finally, we will purpose some future improvements to our application.

2. Background and Related Work

Our work is inspired by the rapid progress in deep learning techniques. Convolutional Neural Network (CNN) is one of the representative models of deep learning and it is the typical algorithm for image recognition. The earliest prototype of modern CNN can be traced back to 1998 called LeNet-5 which was invented by Yann LeCun and his colleagues [4]. LeNet-5 defines the basic structure of modern convolutional neural networks, which is built with alternate convolutional layers and pooling layers.

After 2006, with the development of deep learning theory, especially the invention of Layer-Wise Learning and Fine-Tuning technology, CNN began to develop rapidly with various learning and optimization theories introduced [5]. Some excellent CNN structures have been proposed. Since AlexNet in 2012, various convolutional neural networks have been the winners of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for many times. For example, the VGGNet and GoogLeNet in 2014 and ResNet in 2015 [6].

R-CNN is an algorithm focuses on regions with CNN features, which is known as an object detection technique. This technique can predict multiple objects in an image. The earliest idea was put forward in 2014 [7]. They use selective search to extract thousands of candidate regions from an image and put these regions into a CNN model. The outputs are fed into a multi-class SVM classifier to do prediction. Based on this, there are fast R-CNN and faster R-CNN for faster speed object detection, as well as mask R-CNN for object instance segmentation. And single-shot object detector (SSD), as well as the YOLO, etc. In 2019, a group of people developed a mobile application using object detection model (SSD Mobilenet_v1 FPN) for cat recognition. With enough training data of 14 cat breeds, the average prediction accuracy can reach more than 80% [8].

Tensorflow is Google's open source machine learning framework based on dataflow programming. Keras is a deep learning library based on TensorFlow and Theano. Keras is a high-level neural network API written by pure Python. It is famous for its modularization programming idea, which is simpler and faster for researchers to change the structure of the model or adjust the hyper-parameters. It also provides many typical models such as VGG19, ResNet50 and IncenptionV3. It

simplifies the fine-tuning step. And in this report, we choose Keras to build our single spider recognizer and TensorFlow Object Detection API to build our multiple spiders recognizer.

Currently, deep learning models can not only run on computers but also mobile devices. Core ML is a machine learning framework introduced by Apple. It helps iOS developers to integrate machine learning technology into their applications. Core ML supports natural language processing (NLP), image analysis and other ML tasks. It provides outstanding device performance with minimal memory footprint and power consumption. Core ML V2 will be used in this paper.

3. Deep learning dataset

The average prediction accuracy is the most important metric to evaluate the performance of a specific model. And different training datasets can influence final accuracy significantly. A good training dataset should contain enough high-quality data, and the data is balanced for each class.

Our target is proposing deep learning models for Australian spider recognition, especially for spiders which are poisonous and widespread. The Australian Museum official website provides detailed information about the most famous poisonous spiders. After doing in-depth research work, we select 9 spider species for training.

Table 2 below indicates the spiders we picked with their toxicity to humans.

Table 2. Spider species used for model training

Spider name	Toxicity
Daddy Long Legs Spider	Low Risk
St Andrews Cross Spider	Low Risk
Garden Orb Weaver Spider	Low Risk
Huntsman Spider	Low Risk
Australian Tarantula Spider	Toxic & Painful
Red-Headed Mouse Spider	Toxic & Painful
White-tailed Spider	Deadly
Australian Redback Spider	Deadly
Sydney Funnel-Web Spider	Deadly

These spiders are peculiar in Australia. The dataset of them is not provided by ImageNet, Open Images or Kaggle Challenge Platform. On the Australian Museum official website or other related websites, there are a few images of them, and these images are merely for description. The only platform to obtain enough images is Google Image Library. Because images inside are gained from multiple sources in advance. Hence, we use a crawler to crawl images from Google Image Library. One fact we found is, for each spider class, the image quality starts to drop after crawling more than 300 images. The "bad quality" is reflected by an increasing number of duplicate images and images do not belong to this class so they are irrelevant. For each class, we crawl 300 images and totally 2700. These raw images contain noisy data and we need to manually filter them to drop duplicate, irrelevant and inappropriate ones (see figure 2).

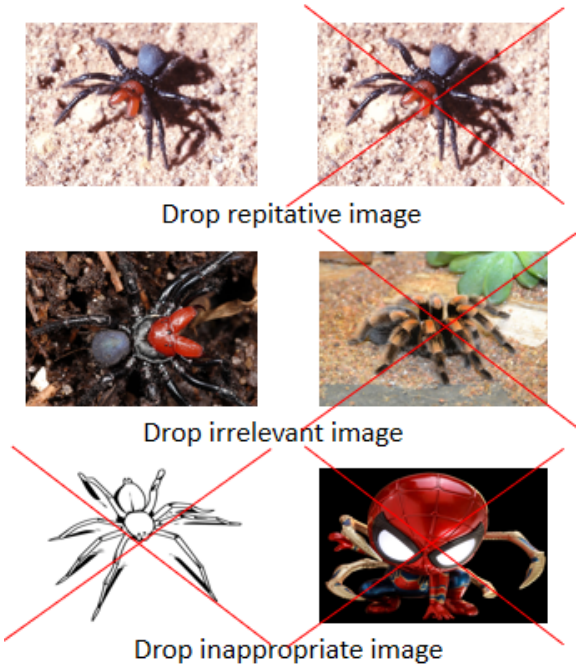

Figure 2. Manual filtering of the dataset

After filtering, we resize all these images into size 224×224, which is corresponding to our deep learning model's input layer (see figure 3).

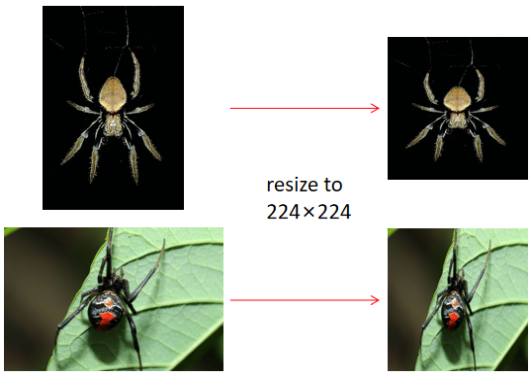

Figure 3. Images are resized to a uniform size

One method to reach high prediction accuracy is feeding enough data. Under the situation of limited raw data, one alternative solution is data augmentation. One research on image recognition using CIFAR10 dataset indicates data augmentation can decrease the prediction error. In order to improve the model robustness and accuracy [9], we increase the size of the dataset by rotating and

flipping each image (see figure 4). By rotating, the size of our dataset is quadrupled and flipping doubles the size of the dataset.

Another technique of data augmentation is random cropping. It can not only improve model accuracy but enhance stability [10]. We use 200×200 as the window size to randomly crop the images. And for newly generated data, we resize them to 224×224 (see figure 5). Using random cropping, the size of our dataset is doubled again.

Figure 4. Image rotating and flipping

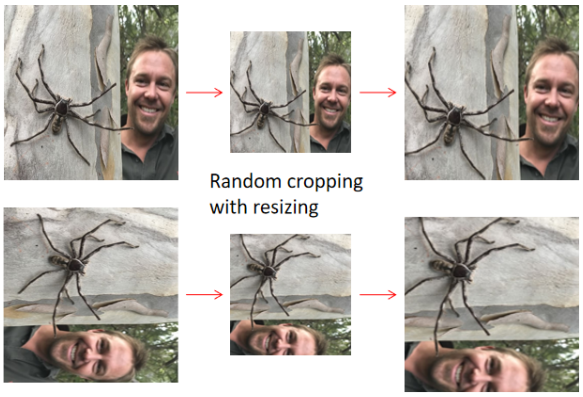

Figure 5. Random crop of images

After data augmentation, the number of images for each spider is shown in table 3.

Table 3. Number of images for each spider species

Spider Classes	Number of Images
Daddy Long Legs Spider	3296
St Andrews Cross Spider	3424
Garden Orb Weaver Spider	2656
Huntsman Spider	2880
Australian Tarantula Spider	2688
Red-Headed Mouse Spider	3264
White-tailed Spider	2464
Australian Redback Spider	3168
Sydney Funnel-Web Spider	3040

In addition, for object detection model training, we label each spider's location by drawing a bounding box around it for each image (see figure 6).

Figure 6. Spider localization using Labeling

The exported file for each image is an XML file that conforms to the Pascal VOC format.

```
<annotation>
        <folder>Sydney_Funnel_Web_Spider</folder>
        <filename>Sydney_Funnel_Web_Spider_175.jpg</filename>
        <path>C:\Users\vincent916735\Desktop\poisonous-spider-recognition\labelled_dataset\Sydney_Funnel_Web_Spider\Sydney_Funnel_Web_Spider_175.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>600</width>
                <height>600</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>Funnel Web Spider</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>90</xmin>
                        <ymin>31</ymin>
                        <xmax>510</xmax>
                        <ymax>565</ymax>
                </bndbox>
        </object>
</annotation>
```

Figure 7. An example labeled XML

The generated XML file essentially contains the height, width and depth (which is for an RGB image) of an image. In addition, it contains the necessary coordinate offsets for the bounding box. If there are multiple objects that are present in the image, there will be one box associated with each.

4. Model Design and Architecture

We consider two common computer vision tasks when designing deep learning models. One is image classification, which is classifying an image by the dominant type of the spider in the image and the other one is object detection which means locating all the spiders in the image and classifying them by their types. One could immediately tell that object detection is a more complicated process hence requires a more sophisticated approach.

A typical CNN generally has an input layer which accepts images represented as 3D tensors (multi-dimensional matrices) with each pixel represented by an RGB tuple.

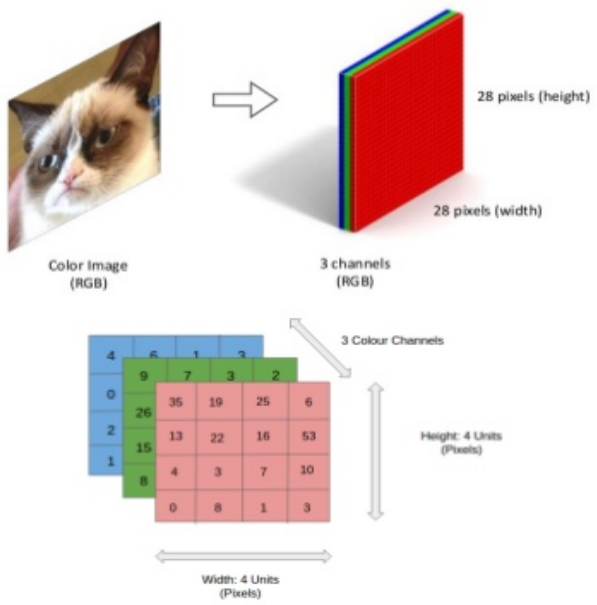

Figure 8. RGB image as tensor

They usually consist of different types of layers such as an input layer, multiple sets of convolutional, activation and pooling layers, several fully connected layers followed by the final output layer [11].

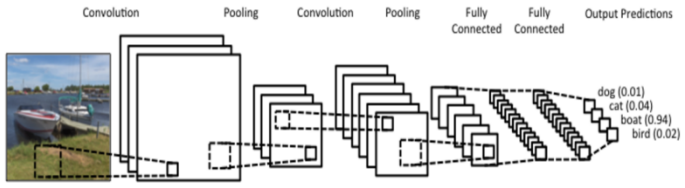

Figure 9. CNN structure

Convolutional layers are central to a CNN and they perform the convolution operation. It involves sliding sets of filters one by one across the input tensor and calculating the sum of the element-wise products along the way. The resulting tensor is then being fed into an activation layer and the output is the activated feature map of the input tensor.

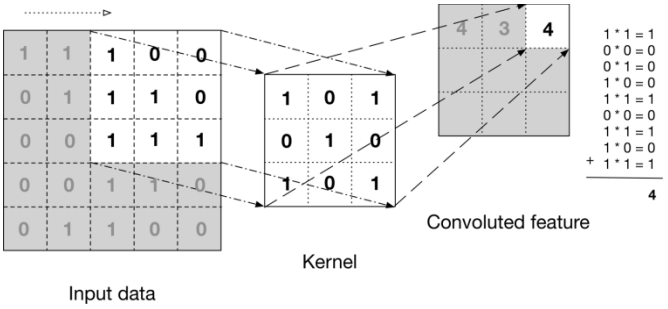

Figure 10. Illustration of the convolution step

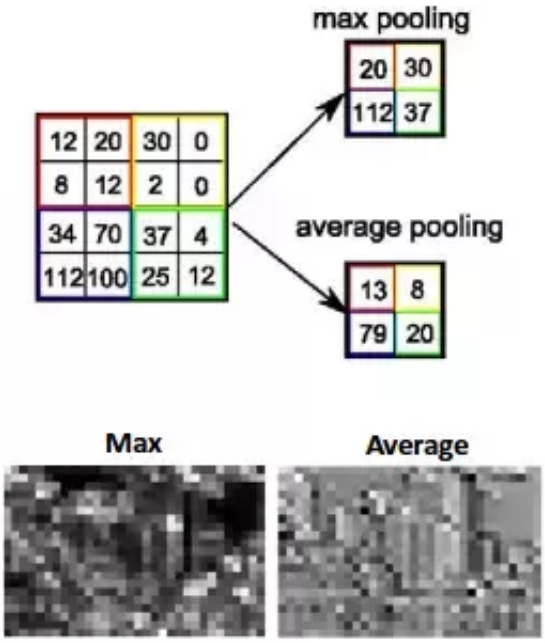

Figure 11. Illustration of the pooling step

Pooling layers are often introduced to reduce the dimensionality of the feature maps. Max pooling and average pooling are the most common techniques. Pooling essentially creates a more compact feature representation of the inputs and more importantly limits the total number of parameters of the network and prevents overfitting [12].

We aim at providing real-time experience for the app users in the sense that the models must be able to run smoothly on the device's camera feed and predict each frame in the pixel buffer at lightning speed. The prediction speed is usually measured in frames per second (FPS).

The trade-off is that although we could host a powerful model on a cloud-based server that offers more processing power and RAM, and potentially produce more accurate results, the accumulated delay associated with message exchanges between various components is quite considerable.

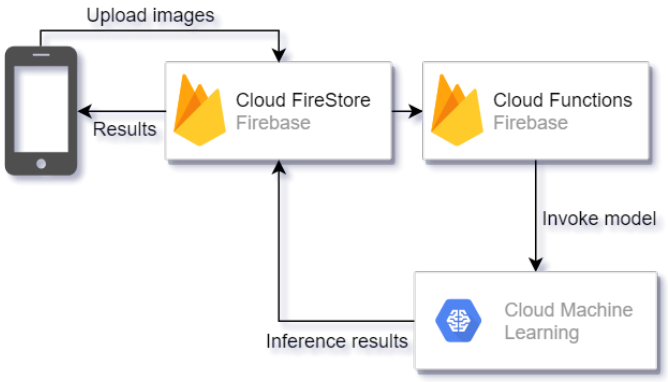

Figure 12. Illustration of a cloud-based ML architecture

Hence, the FPS achieved does not satisfy the requirement of being "real-time". Real-time prediction is only possible if we bundle the model into the app and run it on the mobile device. iOS provides a native Core ML framework that is hardware accelerated and offers API of interacting with the phone's built-in camera. It also comes with a toolkit that can convert any Keras models to Core ML models with ease. By moving the model to the application side, the spider recognizer can be used without an internet connection.

Finally, we considered the trade-off between size and running speed of the model. By placing the ML models in the app, the size of the app increases sharply and easily exceeds 100MB which is reasonably large for a mobile app. We also discovered that by deploying models that have over 10 million trainable parameters or have more than 100 layers, the FPS decreases dramatically, and the prediction is not real-time. It takes over a second or two to predict a single frame.

We compare the accuracy, running speed and sizes of different state-of-the-art CNN models and select MobileNetV2 as our model for image classification.

MobileNets replace a normal convolution by a depth-wise convolution followed by a pointwise convolution [13]. This reduces the total number of multiplication and sum operations required for a convolution block hence it is faster to compute. We achieve a much faster inference speed and up to 10 times reduction in model size without observing any significant drop in accuracy. The table below summarizes various state-of-the-art models' performances on our spider dataset. We split our entire dataset into a training set (80%), a validation (10%) set and a test set (10%). When we split the data, we applied the stratified option which ensures that the training data preserves the distribution of classes of the entire dataset. This is to prevent potential class imbalance caused by the random split. We then train various models using the same training data and evaluate on the same validation set.

We choose to evaluate 3 models. Xception is the successor of Google's InceptionV3, which uses depth wise separable convolution as MobileNet [14]. The difference is that it is a deeper network with larger size and more trainable parameters. The parameter setting that we use for Xception and both versions of MobileNet is ***epochs = 50, batch size = 16***. We can potentially have a larger mini-batch size for MobileNets but not for Xception since we only have one GTX1070 GPU with 8 GB memory for training. A over large mini-batch could trigger an out-of-memory warning.

Table 4. Model accuracy on the validation set

Model	Top-1 Accuracy	Top-5 Accuracy
Xception	0.9042	0.9583
MobileNet	0.8895	0.9821
MobileNetV2	0.8755	0.9821

The result is not surprising and suggests that there is a correlation between accuracy and model size. Xception has over 20 million parameters and is roughly 8 times larger than MobileNets in size. It is a different story after we deploy the model to the phone and run them in the app. Xception requires over 1 second to predict a single frame which creates intolerable lags when predicting

from the camera feed. MobileNets are extremely fast and can easily predict more than 20 frames per second on an iPhone 6.

The accuracy we obtain is for the hold-out validation set which is hidden during the training. The results seem surprisingly good for such a small-scale dataset. This is because the validation set is so small. If we could gather more unique spider images and construct a much larger test set, the accuracy will no doubt decrease.

Transfer learning [15] is another technique that we use. It involves using a pre-trained neural network (for task A) as the starting point on another task (task B). Transfer learning often works in computer vision because the pre-trained network has already gained the knowledge of how to extract low-level features and images from different domains that often share those features. A critical step of transfer learning is to reuse the weights as the initial weights instead of initializing weights randomly. As a result, it reduces our training time dramatically and the model is learning sharply during the first few epochs.

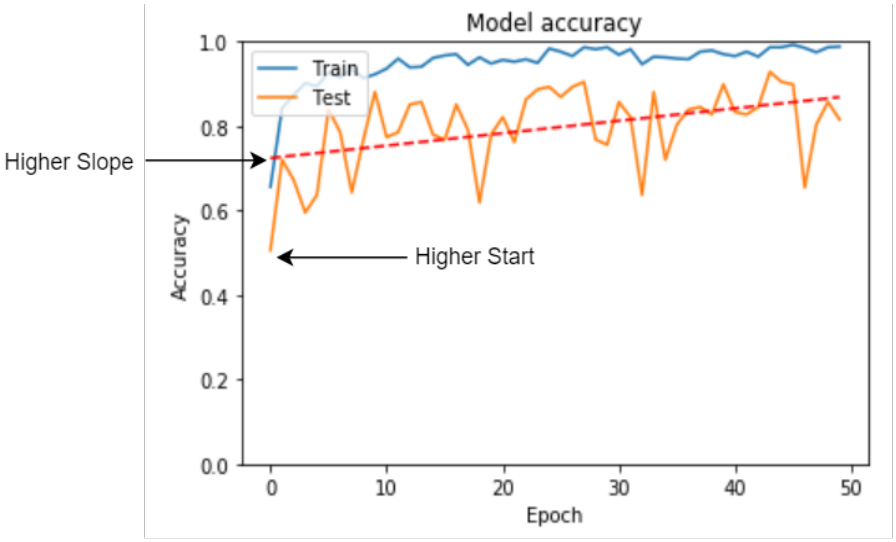

Figure 13. Illustration of how transfer learning helped us train the Xception model

So far, we have only considered the performances of various CNN models without mentioning the object detection task. It is a more trivial task than the image classification since it requires not only a base CNN but an algorithm to generate the locations of the objects and mark them on the image.

R-CNN is a family of algorithms based on region proposals as previously mentioned. It includes R-CNN, Fast R-CNN and Faster R-CNN. They both are based on the idea of generating the most likely regions of an image that might contain objects and looking at these regions to detect objects. YOLO [16] and SSD are the newly proposed object detection algorithms based on the concept of single shot detection. They do not generate regions but instead slide through the image or the convolved feature map directly and generate multiple bounding box predictions for each patch. Hence, they are faster than R-CNN algorithms since they do not require extra computations for region proposal. SSD [17] is considered being superior to YOLO since it takes the bounding box predictions on feature maps of various scales into consideration while the original YOLO algorithm only looks at the entire image once.

To evaluate various object detection systems, we utilize the TensorFlow object detection API which offers implementations of various state-of-the-art systems with pre-trained weights on large public datasets such as Microsoft COCO dataset [18] and Google Open Images dataset [19].

When we evaluate these systems, we keep in mind that the model ultimately needs to run on a mobile device. And it needs to perform real-time object detection with reasonable accuracy.

Furthermore, R-CNN based methods require custom layers which are not officially included in Core ML API. Therefore, the conversion from TensorFlow to Core ML requires removing the unsupported layers and manually implement these layers in iOS.

We evaluate one region-proposal based system and one SSD based system. The metric used for object detection task is not accuracy but mean average precision instead. It is an averaged precision over all classes. We use IoU to define positives and negatives in object detection since we are comparing generated bounding boxes to the ground truth boxes. The illustration of the definition of IoU measure is in figure 14.

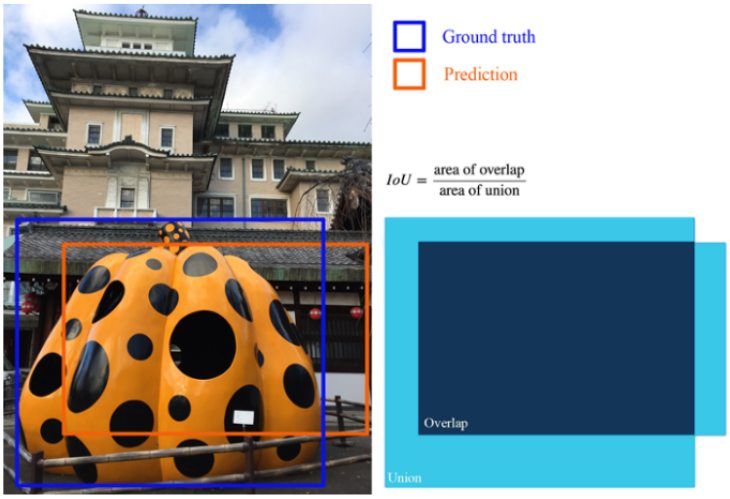

Figure 14. Illustration of the definition of IoU measure

A common threshold for this measure is 0.5. Any bounding box with a measure > 0.5 is positive.

Table 5. Average precision at .50 level of IoU for both settings

Feature extractor	Detector	$AP^{IoU=.50}$
Inception V2	Faster R-CNN	0.634
MobileNetV2	SSD	0.621

5. Hyper-Parameter Tuning

5.1 Learning Rate

Until now, we have been only experimenting with state-of-the-art models and their default configurations. It is worth tuning the various hyper-parameters and seeing their effects on the CNN performance.

Learning rate represents the rate or the order of magnitude and we adjust the weights using gradient descent. The objective of training a neural network is essentially minimizing a loss function. The image below perfectly illustrates this process. If the learning rate is too small, we need more steps of adjustments to reach the optimal weights that minimize the loss function. This is still better than having a over large learning rate since we may over adjust the weights when we are near the optimum and miss it. The result is either a suboptimum or even divergence. The perfect scenario is the middle case where we adjust the learning rate adaptively and gradually decrease the learning rate when we approach convergence.

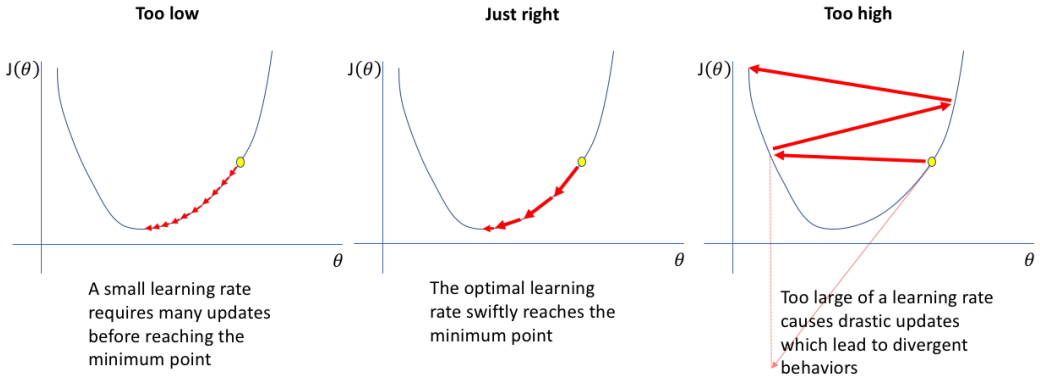

Figure 15. Effects of different learning rates

We explore the learning rate effects using the MobileNetV2 model with stochastic gradient descent (SGD) optimizer. For each experiment, we use the same train/validation split and the number of epochs. The only thing that varies is the learning rate (0.001, 0.01, 0.1).

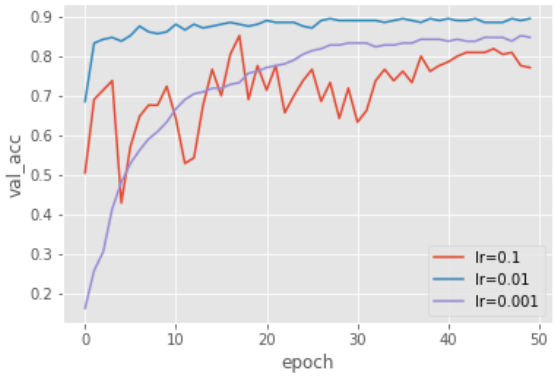

Figure 16. Validation set accuracy with various learning rates applied to SGD

The figures above clearly show the importance of having an appropriate learning rate for training. We can see that if we set the learning rate to 0.1, the model will fluctuate drastically and not necessarily converge to an optimum in given epochs. This is caused by the sharp adjustments of weights. Setting the learning rate to 0.01 makes the model learns sharply and converges to the highest asymptote. If we set the learning rate to 0.001 that would be too conservative, and the model learns very slow at the beginning but still manage to converge to a higher asymptote.

5.2 Mini-batch size

Batch size or mini-batch size is another parameter that often tuned during training. It describes how many training samples to use in each step of the epoch to update the weights.

Smaller batch size will result in noisy updates and noisiness will serve as a form of regularization. Hence the model may generalize better to unseen data [20].

We experiment with different batch size settings using the MobileNetV2 model with SGD and the results are shown below.

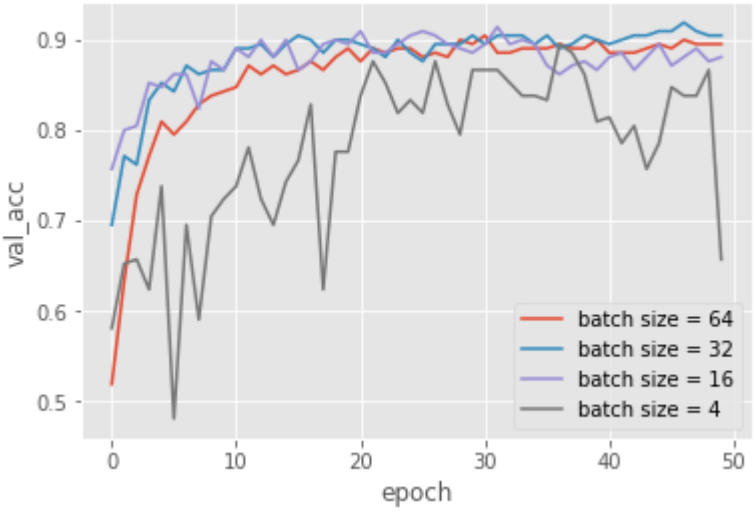

Figure 17. Effects of different batch sizes

The results demonstrate that a smaller batch size not always works better on our spider dataset. If we set the mini-batch size to be too small, the updates will be too noisy, and we do not necessarily descent in the steepest direction.

One thing to note is that a smaller batch size tends to make the learning curve fluctuate more since the updates are noisy. For our dataset, based on the experiment we can conclude that 32 is the most appropriate batch size to use.

5.3 Dropout

Dropout regularization is commonly used in deep learning models to control overfitting. It is easy for a neural network to overfit the training data since the neural network can approximate complex functions. Dropout means randomly dropping out nodes. A dropout layer takes the previous (usually a hidden layer) layer's outputs as inputs and for each of these, there is a chance that it is being set to zero and forward to the next layer (dropped). The dropout rate specifies the probability of any given input of the dropout layer being set to zero. It can be also interpreted as the fraction of inputs being dropped.

To see how dropout regularization has affected our training, we have built an experimental neural network based on the classic VGG16 [21] since Keras built-in models are somewhat trivial to customize. Our version is a compact version [22] of the original VGG net since the original network has too many parameters and is extremely slow to train. The results we obtained are general since almost all state-of-the-art models use dropout.

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 96, 96, 32)	896
activation_15 (Activation)	(None, 96, 96, 32)	0
batch_normalization_17 (Batc	(None, 96, 96, 32)	128
max_pooling2d_7 (MaxPooling2	(None, 32, 32, 32)	0
dropout_9 (Dropout)	(None, 32, 32, 32)	0
conv2d_16 (Conv2D)	(None, 32, 32, 64)	18496
activation_16 (Activation)	(None, 32, 32, 64)	0
batch_normalization_18 (Batc	(None, 32, 32, 64)	256
conv2d_17 (Conv2D)	(None, 32, 32, 64)	36928
activation_17 (Activation)	(None, 32, 32, 64)	0
batch_normalization_19 (Batc	(None, 32, 32, 64)	256
max_pooling2d_8 (MaxPooling2	(None, 16, 16, 64)	0
dropout_10 (Dropout)	(None, 16, 16, 64)	0
conv2d_18 (Conv2D)	(None, 16, 16, 128)	73856
activation_18 (Activation)	(None, 16, 16, 128)	0
batch_normalization_20 (Batc	(None, 16, 16, 128)	512
conv2d_19 (Conv2D)	(None, 16, 16, 128)	147584
activation_19 (Activation)	(None, 16, 16, 128)	0
batch_normalization_21 (Batc	(None, 16, 16, 128)	512
max_pooling2d_9 (MaxPooling2	(None, 8, 8, 128)	0
dropout_11 (Dropout)	(None, 8, 8, 128)	0
flatten_3 (Flatten)	(None, 8192)	0
dense_6 (Dense)	(None, 1024)	8389632
activation_20 (Activation)	(None, 1024)	0
batch_normalization_22 (Batc	(None, 1024)	4096
dropout_12 (Dropout)	(None, 1024)	0
dense_7 (Dense)	(None, 9)	9225
activation_21 (Activation)	(None, 9)	0

Total params: 8,682,377
Trainable params: 8,679,497
Non-trainable params: 2,880

Figure 18. Summary of our experimental VGG net

We train the same experimental VGG net twice and for 200 epochs each time. The only difference is that we switch off the dropout layers during the second round of training. The result is presented below.

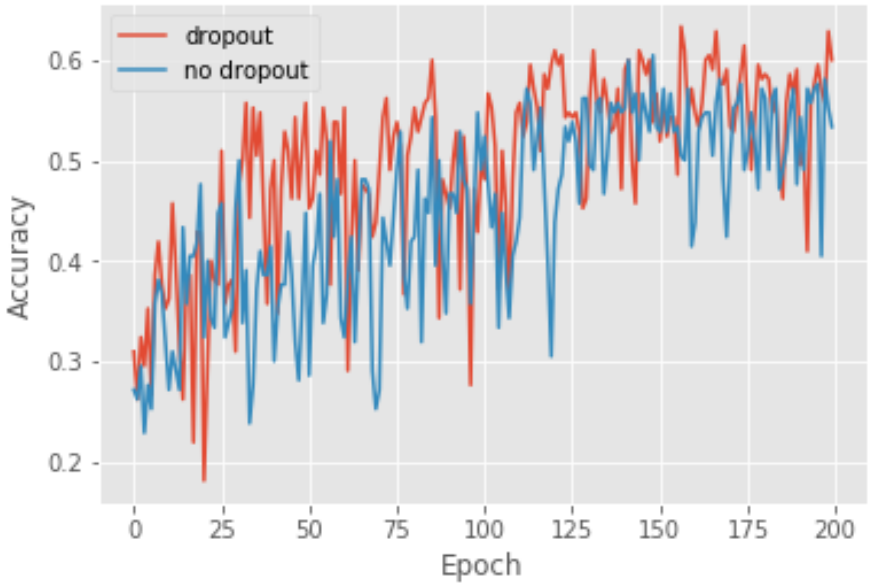

Figure 19. Effects of dropout regularization

We plot the validation set accuracy against the epochs and we can see from the figure that with dropout regularization, we achieve a higher accuracy for the validation set. In other words, the model is doing slightly better on unseen data with dropout turned on. Furthermore, we can see that our experimental model only achieves around 0.6 val_acc after 200 epochs which cannot compete with the performance of the Keras built-in models. It is always a good idea to explore the state-of-the-art models instead of reinventing the wheel when given a deep learning task.

5.4 Optimizer

We also tried different optimizers and run each for 50 epochs with MobileNetV2 for our spider classification. The setting is Adam optimizer with learning rate 1e-4 and SGD with learning rate 1e-2. The result is as expected as optimizer generally will not have a decisive effect over model performance as the learning rate. Although Adam [23][24] tends to converge faster than traditional

SGD since it has two parameters alpha and beta which adaptively adjust the learning rate, it is also one of the most popular choices of optimizers in deep learning.

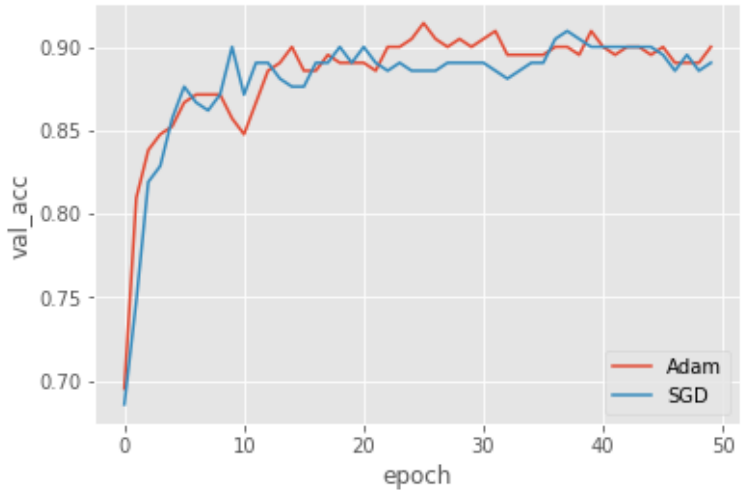

Figure 20. Comparison of different optimizers

6. Critical Analysis

We have presented various metrics in the previous sections of this report. They provide a big picture of the model performance. However, we have not conducted any thorough error analysis. We will look at the per class metrics in this section and the insights of where the model fails and why.

The per class metrics of image classification can be easily exported. It is presented below.

Table 6. Precision, recall and F1-score for each class

Spider name	Precision	Recall	F1-score
Daddy Long Legs Spider	0.93	0.96	0.94
St Andrews Cross Spider	1.00	0.93	0.96
Garden Orb Weaver Spider	0.90	0.82	0.86
Huntsman Spider	0.77	0.96	0.85
Australian Tarantula Spider	0.86	0.76	0.81
Red-Headed Mouse Spider	1.00	0.84	0.91
White-tailed Spider	0.87	0.95	0.91
Australian Redback Spider	0.95	0.90	0.93
Sydney Funnel-Web Spider	0.85	0.92	0.88

For the species with top 3 f1-score namely St Andrews Cross Spider, Daddy Long Legs Spider and Redback Spider. It is reasonable to examine the dataset closely. All three types have very clear distinctive features such as thin and long legs, red dot on the back or crawling on a web like a cross. This makes the feature extraction much easier than for example a Tarantula which is just a broad category of spiders with different colors and shapes. Hence, the choices of species may not be scientific enough as some of them are specific while some are too general.

Figure 21. Daddy Long Legs Spider, Redback Spider and St Andrews Cross Spider

Figure 22. Tarantulas come with all kinds of colors and patterns

The other deficiency that the system suffers is that the classification result is very sensitive to the distance between the camera and the image. A possible explanation is that our training images are all taken at positions very close to the spiders hence our model generalizes badly to images that taken at faraway positions. This reflects the fact that the number of training images is still quite limited, and the aspect ratios of these images are relatively similar. Although we apply data augmentation to the spider dataset, the augmented images do not bring much novelty to the training set since they are essentially the same image. The limited resolution and zoom ability of the mobile phone can also contribute to this phenomenon.

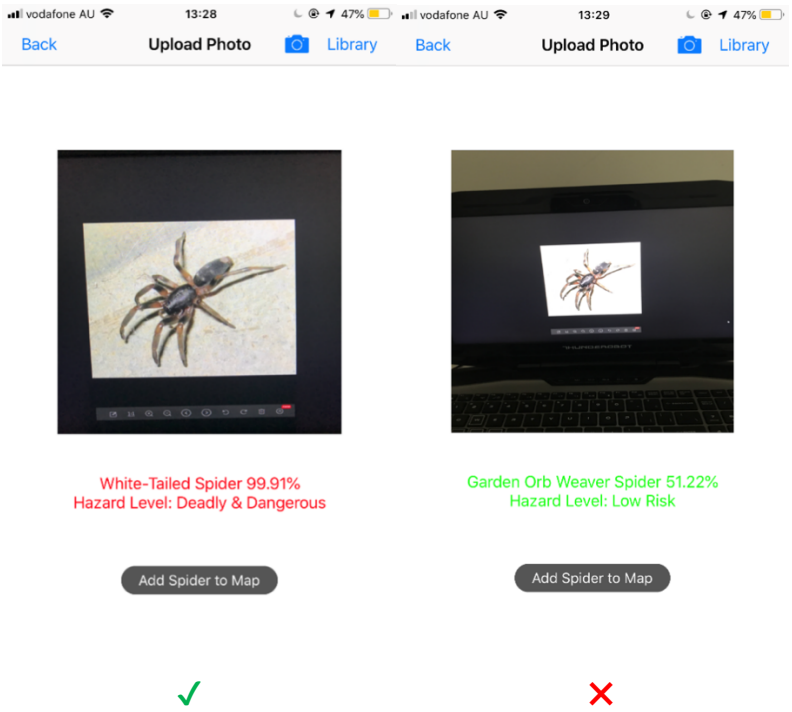

Figure 23. Different classifications generated from different distances

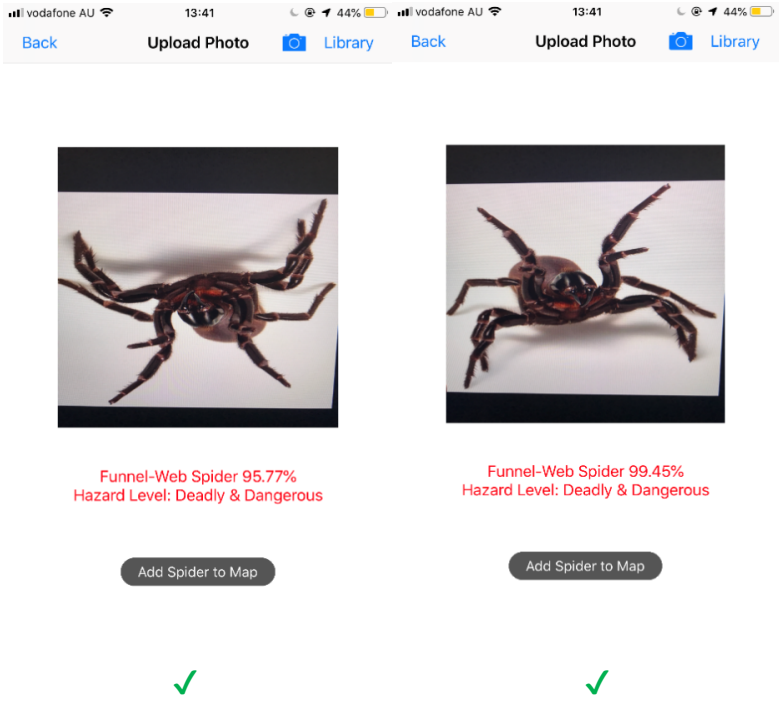

Figure 24. Correct identification from multi-directions

Object detection is inherently a more difficult task than classifying the image. Hence our object detection performance is worse than the classification. For both methods, we obtain just over 60% average precision across all species.

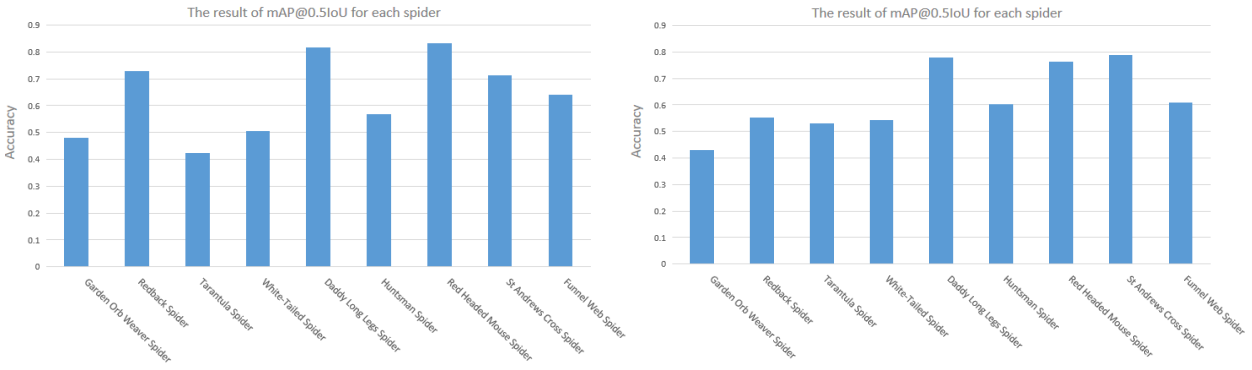

Figure 25. Average precision of InceptionV2 with Faster R-CNN and MobileNetV2 with SSD

The object detection results resemble the classification results in the sense that Tarantulas and Garden Orbs remain the most difficult species to predict. However, if we look at the negative predictions, two models reveal different capabilities. MobileNetV2 with SSD struggles with recognizing small objects and multiple objects but perform well with medium to large objects which is the case for most of our spider images. Therefore, its ability to detect real moving spiders from a distant location remains questionable. InceptionV2 with Faster R-CNN did slightly better in capturing multiple objects but is too demanding for a mobile device.

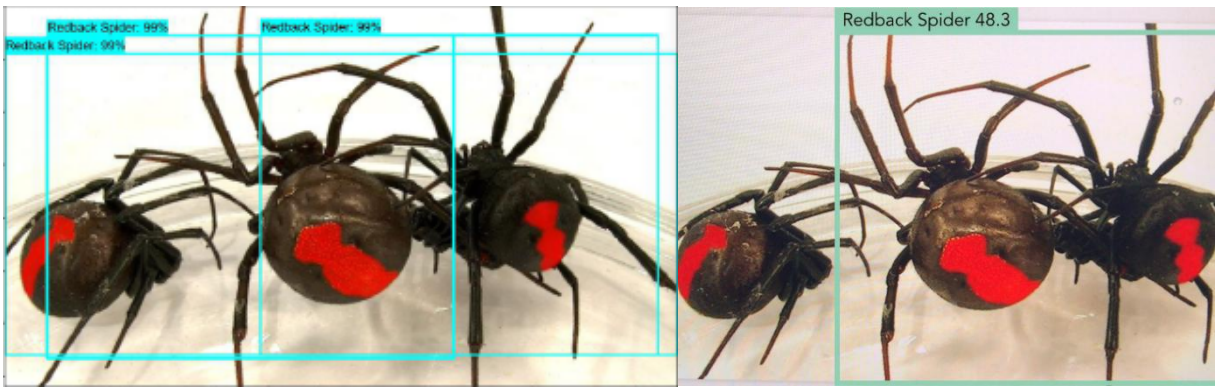

Figure 26. Prediction performance: R-CNN vs SSD

The problem with SSD is that it uses feature maps that are very deep in the network to locate objects and these feature maps are convolved many times and are essentially diluted versions of the original image. The important takeaway is that there is a trade-off between accuracy and running speed for any deep learning system.

7. Implementation of iOS Application

By using Core ML, we aim to design and implement an iOS application that is both user-friendly and efficient. The application consists of four parts: the basic mode, the advanced mode, Spider Gallery and Spider Map.

The basic mode and advanced mode correspond to image classification and object detection respectively. Both include the function of prediction from a photo or from the real-time camera feed and present the name of the species and the level of the poisonousness. In the advanced mode, object detection is implemented to locate spiders in the photo/frame. In simple terms, spiders in the photo/frame will be marked with colorful bounding boxes. For Spider Gallery, it introduced 9 different species of spiders in Australia, which are also the classes in our training model. The Spider Map allows users to view the locations of all spiders, uploaded by all the users of this application.

7.1 User Interface

When implementing the user interface, we follow the rule of ‘KISS’, which stands for ‘keep it simple and stupid’. The interface should be user-friendly, even for those who are using this app for the first time.

On the landing page, there is nothing but four buttons, which redirect users to four different parts of the application. We rename ‘image classification’ and ‘object detection’ as ‘basic mode’ and ‘advanced mode’ respectively since all jargons should be hidden from the users who most likely have no technical background in deep learning.

Figure 27. Launch screen & landing page

Once entering the basic/advanced mode, users have an option to upload a photo to the application, no matter it is a photo taken just now or a photo in the library. Another option is to scan and predict in real time. In basic mode, after a photo is uploaded, the result will be shown by a label at the bottom of the screen, including the name of the species, the level of poisonousness and the confidence. During scanning, it is similar, but the label is changing constantly since new frames keep coming in. In the advanced mode, there will be a bounding box surrounding each spider when a photo is uploaded. On top of the bounding box, the predicted class and the confidence will be labeled. When scanning, the bounding box and the label keep track of the spider in the screen with the changing frames.

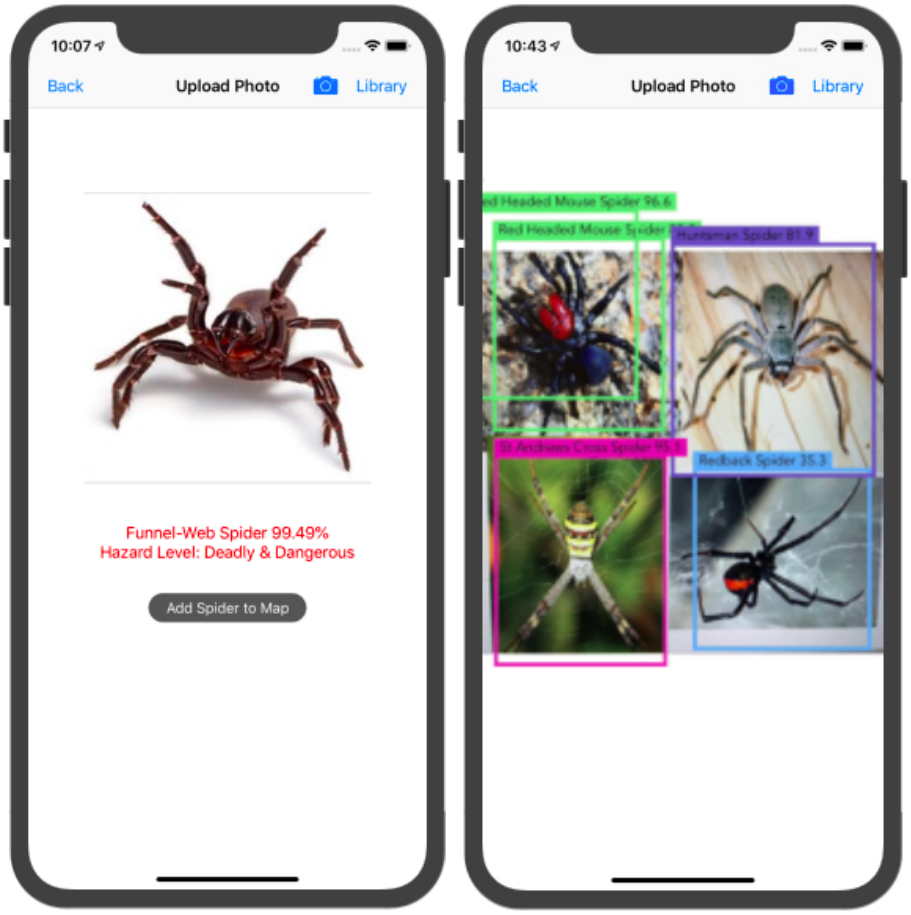

Figure 28. Predictions in both modes

7.2 Spider Map on Web

The Spider Map is the only feature that requires access to the Internet. After the prediction is generated and presented, the user is provided with an option to send the information to the server (Firebase). When this information, including the name predicted of the spider, the latitude and longitude of the user, is received by the server, it will be stored in an unstructured real-time database. Whenever users open the Spider Map, a web application deployed on GitHub Pages, it will automatically retrieve data from the server. Then a Google Map will be presented, with all spiders shown as markers on it. More information will be displayed once the user clicks on those markers. We carefully design the Spider Map so each species of spiders will have markers with a unique color.

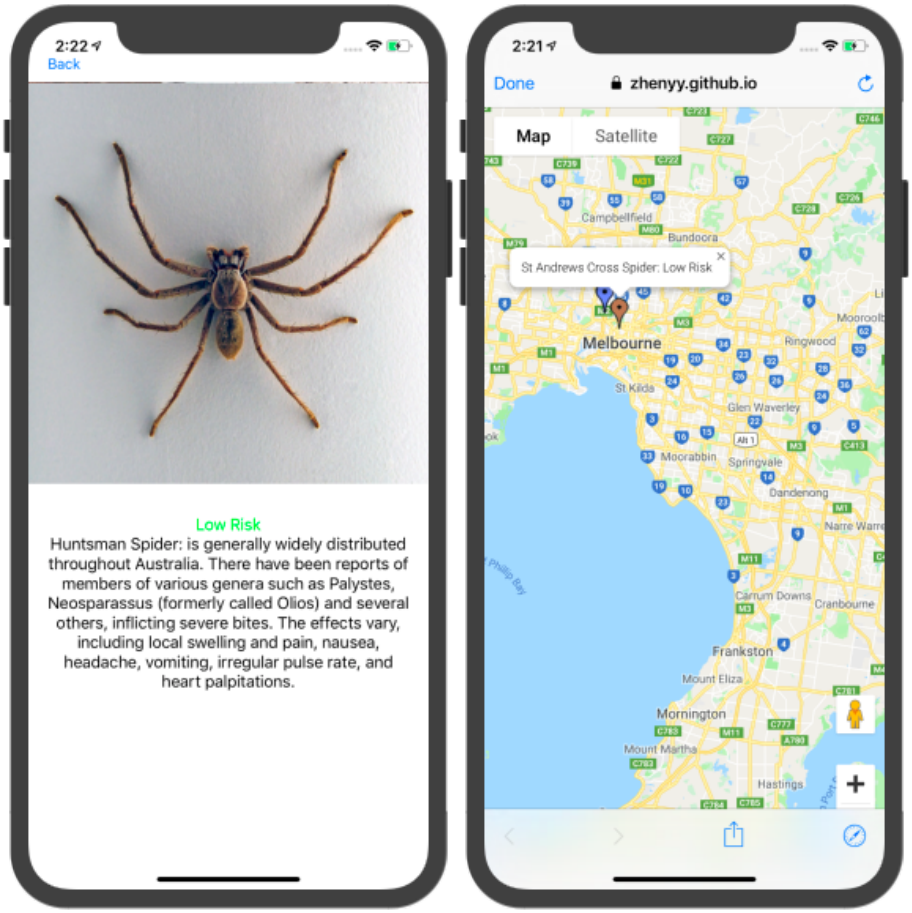

Figure 29. Spider Gallery & Spider Map

7.3 Design Trade-off

We faced some design trade-offs during the implementation of the application.

• SQL vs. NoSQL

First, which one is better, Firebase or SQL database. SQL database provides many database operations for sure, and it has better scalability. But we choose firebase because the data structure we need is simple. Only the name, latitude and longitude are needed. Moreover, an unstructured database provides a larger space for future enhancement. Consider that more information relevant to spiders will be uploaded and stored in the future, we do not need to create a new table or re-order the current structure of our database. We just simply upload an object including all the information, without considering the relationship.

• Embedded model vs. server-based model

Secondly, we have a trade-off between the embedded model and server-based model. As we have mentioned above, the cloud-based/server-based model may have better accuracy, due to its better computing power and RAM, which may easily trump the performance of mobile devices, especially an early version of iPhone. Moreover, the cloud-based/server-based model is of good extensible. Think about the case that when we build an Android version of Spider Recognizer, we are free from training/building a new model in Android environment, which is complicated and time-consuming. When we submit the photo/video to the server, a prediction will be quickly generated. We indeed try to build our server-based model using Django or Flask, but the first problem is the latency which may be longer than 10 seconds. Such a large latency may lead to very unsatisfied user experience and it is contradicted to our 'real-time' principle. Access to the Internet may cause the second problem. Given that the app may be used in the forest, where spiders usually distribute in, users are unlikely to access the Internet. Hence, the embedded model is the only option.

• Embedded map vs. Web-based map

Then it is the trade-off between the embedded map and web-based map. We know that Apple provides a map feature in every iOS device and allows developers to invoke any functions of it. Therefore, an embedded map seems to be a better choice since it allows the application to be more integrated. However, we finally select a web-based map for two reasons. We believe the iOS app should remain light-weight, and users from all platforms could be able to access our map, hence the Spider Map is built as a web-based app apart from our original iOS app.

8. Future Improvement

8.1 Increase Image Diversity

When training our models, less than 300 raw pictures are collected for each spider species. Although we expand our dataset to thousands of images by using data augmentation technique, we do not find enough high-quality images from Google Image Library. Besides, nearly all the images are closeup shots of spiders and taken from the top of spiders. This can cause a problem that our deep learning models can only recognize spiders when mobile devices are very close to them and

on top of them. We want to get more images with high diversity, which means from multiple angles of a spider and from different distances to it.

8.2 Cloud-based Model

We have already demonstrated the necessity of running the model locally. However, a cloud-based model can also be implemented. Due to the size constraint of a mobile app and the hardware constraint of a mobile phone, prediction using the embedded model alone is not accurate enough. Notice that, the application is supposed to support a range of iOS devices, for example, from iPhone 6 to the latest iPhone, while some of them have very limited processing power by today's standards. When a user is using a phone with access to the Internet, an optional verification step using the cloud-based model can be provided to the user. Moreover, the database stores the information of spiders, which is generated by the local model. A cloud-based model should be applied to double-check if the photo of the spider matches the result of the prediction. With the cloud-based model, the increase of the accuracy can be ensured.

8.3 Spider Map Enhancement

The current version of the spider map only supports displaying the marker of the spider with its name and toxicity. Various functions could be added to the map and turn it into an interactive database. For example, a heat map visualization will be very straightforward to implement and gives the viewers a visualization of the distribution of various types of spiders in different parts of Australia. Also, a search bar can be created to allow users to view any specific type of spider or filter spider types by their toxicity or locations.

Notice that, we have tried to upload more than 100 spiders to the database, and the map of Australia was totally covered by colorful markers. This may actually happen, especially when multiple users encounter the same spider and mark it on the map for multiple times. Possible solutions can be, divide the map into different areas by postcode, and mark each area with different colors based on the number of poisonous spiders, or a map similar to thermodynamic diagram, which is a better option for both normal users or relevant researchers.

9. Conclusion

This report puts forward an iOS application for Australian spider recognition using deep learning techniques. Our model can recognize 9 famous spider species with high performance, and it supports both single-spider and multi-spider recognition. The prediction accuracy for single spider classification using MobileNetV2 can reach above 80%. For multiple spider prediction with localization using SSD, the mAP can reach around 64%.

This report also refers to some data augmentation methods and model optimization techniques. For instance, using random crop in the data preparation phase to enhance model robustness, fine-tuning to speed up the training process.

After that, we discuss some trade-offs during the implementation of our iOS application. We want our application to be lightweight so it can run offline to suit complex environments (e.g. rainforest). Hence we use Core ML models. The sizes of models are small (8MB for MobileNet and 6.28MB for SSD). For real-time recognition, we test our model and the speed of prediction can reach 20 frames per second for MobileNet scanning and 10 frames per second for SSD scanning, which is efficient.

Appendices

Deep learning code: https://github.com/zhenyy/poisonous-spider-recognition

iOS app code: https://github.com/zhenyy/SpiderRecognizer

Spider Map web app code: https://github.com/zhenyy/SpiderMap

iOS app demo video: https://www.youtube.com/watch?v=LvMvKccfGkA&t=3s

Reference

[1] G. M. Nicholson, R. Walsh, M. J. Little, and M. I. Tyler, 'Characterisation of the effects of robustoxin, the lethal neurotoxin from the Sydney funnel-web spider Atrax robustus, on sodium channel activation and inactivation,' Pflügers Archiv European Journal of Physiology, vol. 436, no. 1, pp. 117–126, 1998.

[2] Victorian Poisons Information Centre. (2016). ANNUAL REPORT 2016. Retrieved from http://www.austin.org.au/Assets/Files/VPIC%20Annual%20Report%202016.pdf

[3] M. C. Hardy, J. Cochrane, and R. E. Allavena, 'Venomous and Poisonous Australian Animals of Veterinary Importance: A Rich Source of Novel Therapeutics', BioMed Research International, 2014. [Online]. Available: https://www.hindawi.com/journals/bmri/2014/671041/. [Accessed: 08-Jun-2019].

[4] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, 'Object Recognition with Gradient-Based Learning', in Shape, Contour and Grouping in Computer Vision, London, UK, UK, 1999, pp. 319–.

[5] G. E. Hinton, 'Reducing the Dimensionality of Data with Neural Networks', Science, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[6] K. He, X. Zhang, S. Ren, and J. Sun, 'Deep Residual Learning for Image Recognition', arXiv:1512.03385 [cs], Dec. 2015.

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

[8] X. Zhang, L. Yang, and R. Sinnott, 'A Mobile Application for Cat Detection and Breed Recognition Based on Deep Learning', in 2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile), Hangzhou, China, 2019, pp. 7–12.

[9] L. Xie, J. Wang, Z. Wei, M. Wang, and Q. Tian, 'DisturbLabel: Regularizing CNN on the Loss Layer', arXiv:1605.00055 [cs], Apr. 2016.

[10] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, ‘Random Erasing Data Augmentation’, arXiv:1708.04896 [cs], Aug. 2017.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Ha, ‘Gradient-Based Learning Applied to Document Recognition’, p. 46, 1998.

[12] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. The MIT Press, 2016.

[13] A. G. Howard et al., ‘MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications’, arXiv:1704.04861 [cs], Apr. 2017.

[14] F. Chollet, ‘Xception: Deep Learning with Depthwise Separable Convolutions’, arXiv:1610.02357 [cs], Oct. 2016.

[15] S. J. Pan and Q. Yang, ‘A Survey on Transfer Learning’, IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, ‘You Only Look Once: Unified, Real-Time Object Detection’, arXiv:1506.02640 [cs], Jun. 2015.

[17] W. Liu et al., ‘SSD: Single Shot MultiBox Detector’, arXiv:1512.02325 [cs], vol. 9905, pp. 21–37, 2016.

[18] T.-Y. Lin et al., ‘Microsoft COCO: Common Objects in Context’, arXiv:1405.0312 [cs], May 2014.

[19] A. Kuznetsova et al., ‘The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale’, arXiv:1811.00982 [cs], Nov. 2018.

[20] D. Masters and C. Luschi, ‘Revisiting Small Batch Training for Deep Neural Networks’, arXiv:1804.07612 [cs, stat], Apr. 2018.

[21] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv:1409.1556 [cs], Sep. 2014.

[22] A. Rosebrock, “Keras and Convolutional Neural Networks (CNNs),” PyImageSearch, 16-Apr-2018.

[23] N. S. Keskar and R. Socher, ‘Improving Generalization Performance by Switching from Adam to SGD’, arXiv:1712.07628 [cs, math], Dec. 2017.

[24] D. P. Kingma and J. Ba, ‘Adam: A Method for Stochastic Optimization’, arXiv:1412.6980 [cs], Dec. 2014.