A Mobile Application for Cat Detection and Breed Recognition based on Deep Learning

COMP90055 Computing Project (25 Credits)

Type of Project: Software Development Project

Supervisor: Prof. Richard Sinnott
Deep learning: Xiaolu Zhang 886161
Mobile Application: Luyang Yang 880199

Semester 2, 2018

Abstract

Deep learning is one of the latest technologies for computer science. It provides a way to achieve artificial intelligence allowing machines to solve problems in a manner similar to the human brain. Deep learning approaches have significantly improved the performance of visual recognition applications including image classification and image detection.

This project uses deep learning knowledge to build an object detection model to implement cat recognition and breed detection. There are many neural network models that could be used for this purpose such as Tensorflow, Faster R-CNN, SSD and more. By comparing six popular models, we eventually selected SSD Mobilenet_v1 FPN as the detection model because of its high accuracy and low processing time. Since there is no available dataset for our project, we made a dataset based on 14 kinds of cats using images from the Internet. The average accuracy of the finalized model was 81.74%.

We also developed an Android application to embed the model and predict the location and breed of a given cat using a mobile phone camera. This app had to identify and classify the image of the cat in a few seconds.

Keywords: Deep Learning, SSD, Mobilenet, FPN, Tensorflow, Cat recognition, Mobile App

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 8175 words in length (excluding text in images, table, bibliographies, and appendices).

Acknowledgement

We would like to thank our supervisor Professor Richard Sinnott for giving us the opportunity to work on this challenging project. He was always happy to provide us with fantastic suggestions on the problem we faced during the project. Our questions were always answered in a few hours. It helped us a lot to finish this project on time.

We would also like to thank all contributors in the Tensorflow community who provided excellent APIs and tutorial materials.

Last but not least, we would like to thank the owners of images used in our dataset and report. Thanks for sharing!

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Deep learning

Deep learning is one of the latest technologies extending the opportunities of computer science. Deep learning can be used in many areas such as image recognition, drug discovery, natural language processing and more. Deep learning was introduced in 2000 but has experienced a great revolution in the past two decades. It provides a way to achieve artificial intelligence which means allowing machines to solve problems in a manner similar to the human brain. It has been applied to many areas in people's daily life. For example, in the voice commander on smartphones like Siri it is used for speech-recognition and allows people to interact with devices by talking rather than inputs by hand.

Deep learning is a part of traditional machine learning. Machine learning methods were historically used for simple work using basic machine learning algorithms. Because of the limitation of these algorithms, traditional machine learning did not perform very well when dealing with more difficult problem such as image recognition. Deep learning solves this problem through use of neural network-based method. To be more specific, with traditional machine learning algorithms, feature selection and accuracy improvements are the main problems. Firstly, the neural network allows the system to identify the feature by itself rather than by manual input. Secondly, for traditional machine learning algorithms, finding more useful features is key, but up to now it was always very difficult to find better features manually. With deep learning, to get better system performance, users need to input more data and ask the algorithm to learn these data to extract features automatically. It is much easier and faster compared to traditional machine learning algorithms.

As Figure 1 shows, with the increases in the amount of input data, deep learning achieves much better accuracy than traditional machine learning when the size of input data is large. The final goal of machine learning is to achieve artificial intelligence. Deep learning brings significant progress in this regard [1].

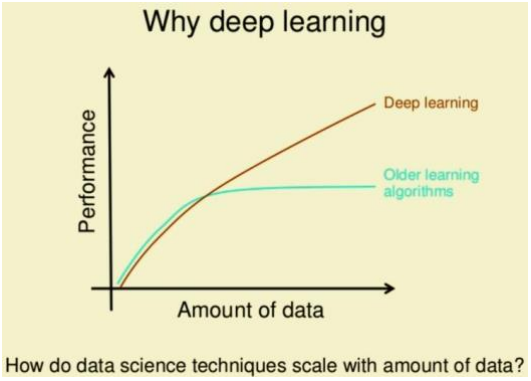

Figure 1. How do data science techniques scale with amount of data

1.2. Object detection and classification

Object classification means when the user inputs an image to a system, the system will classify the category of the image. For this project, the goal was to implement an object detection algorithm to recognize cats and cat breeds in an image.

Object detection means the system should classify the category of the image and get the location of the objects which have been classified by the system. For an object detection model, there are three main steps to the detection process: to determine the object location, to classify the category of the determined location, and to combine the result of the first two steps.

There are a lot of popular different object detection models, such as Faster R-CNN, SSD, YOLOv3 and more [28]. These models have different ways to support the detection process, and the differences between how these models working causes different performance results on the same dataset. The details of these models will be introduced in Section 3.2. To evaluate the performance of an object detection model, the processing time and detection accuracy are two main evaluation metrics. Figure 2 is a simple comparison of Faster R-CNN, SSD, and YOLOv3 model performance on the accuracy. For different types of object, these models have different level of performance on accuracy, but Faster R-CNN has better performance with regards to accuracy. However, SSD and YOLO generally have best performance on time costing.

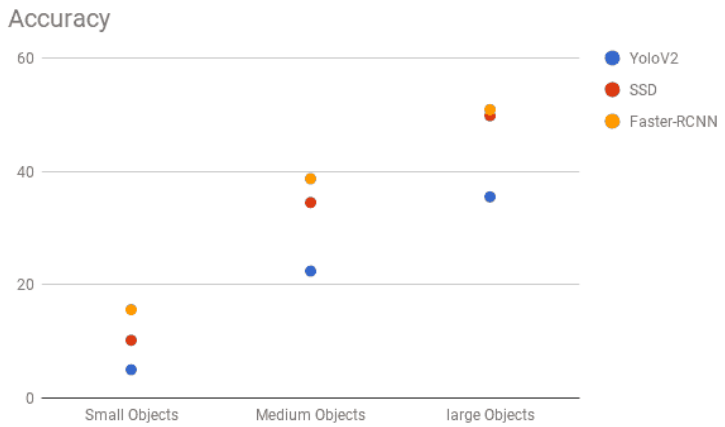

Figure 2. Comparison of the accuracy of Faster R-CNN, SSD, and YOLOv3 model performance

1.3. Existing applications on image detection

Many companies are developing applications on image detection. For example, face recognition is one of them, which is used to tag people on user's photos. It has been applied to Apple's iPhoto, Google's Picasa and Facebook as its entertainment and convenience. Microsoft's Kinect is another example of image recognition applications on gaming, which has been applied to Xbox 360. The users could play the game by sensing rather than hardware controllers. It is a new trend in gaming.

Moreover, face recognition is widely used in camera applications on smartphones, such as the AI camera. It works based on a face recognition model. The model is trained by numerous human face pictures with label information which contains the identity of the person on the image and relevant information about the person. After training, once the camera takes a new picture, the picture will be sent to the model. Then the model will predict the person's identity. Lastly, the information would be returned to the camera interface, and the prediction information would be shown on the image.

1.4. Tensorflow

To support the deep learning process, a suitable software framework is necessary. There are a lot of popular deep learning frameworks available such as Caffe, Tensorflow and more [29]. These are also several open source libraries for users to implement deep learning applications. These frameworks have their own advantages and limitations. For image classification and detection applications, Tensorflow has the better performance for several reasons. It is easy to transplant onto mobile devices and has support for the latest algorithms and more. Therefore, for this project, we select Tensorflow as the framework to train our object detection model and develop the mobile application.

1.5. Mobile Application

Mobile devices are very popular in people's daily life. According to recent surveys [30], there are around 5 billion users of mobile devices globally and this number is still increasing. Mobile devices are very popular for their convenience and use in many areas of people's lifestyle through targeted applications. For example, the map application is a very useful way to navigate around cities.

Our project is aiming to develop a cat recognition and detection system using a deep learning model. The model is trained on larger scale computers. Mobile applications have much less computing capacity. Therefore, we developed a performance oriented mobile application version for the detection model. The user just uses the mobile application to learn about the information about a range of cats.

According to a survey data [31], up to 2016, about 86% of smartphones are running on the Android system. That means Android phones are much more popular than other mobile operating systems. Based on these data, we decide to develop an Android application for our system.

2. Dataset

The quality of data used for deep learning is essential. A good object detection dataset should involve high resolution images, correct labels and bounding box information. However, there is no dataset available for cat breed detection which could be used directly. Therefore, we had to make our own dataset for the project.

According to CFA (i.e. The Cat Fanciers' Association), the number of common cat breeds is estimated at 45. As the coverage of all cat breeds was not the main concern for this project, our project only supports recognition of 14 breeds. One reason is we choose to narrow down the coverage of breeds is that we made a trade-off between large but poor-quality datasets between small but high-quality dataset. It takes a long time to make a high quality dataset. We need to manually label and highlight the areas of cats for each image in the dataset. So most of the effort is spent on enlarging the number and quality of images of a specific breed. The other reason is that there are only minor differences between some breeds listed in CFA as they have the same heritage, such as *European Burmese* and *Burmese*.

The following sections illustrate how the dataset was created including how we collected and preprocess the dataset.

2.1. Dataset Sources

There were four primary sources of data used: Oxford Pet Dataset, ImageNet, Flickr and Google Image. Oxford Pet Dataset contains 12 breeds of cats with 2,371 images [2]. However, we were not able to just use Oxford Dataset directly. Firstly, the number of images in the Oxford Dataset is not enough (on average there are about 200 pictures per breed). Secondly, the bounding box information does not satisfy the need of the project. For the Oxford Dataset, only the head area of a cat is recorded. It does not utilize other important characteristic information such as the body part or fur patterns. This causes issues with some breeds especially, e.g. it is difficult to detect the *Manx* cat since *Manx* cats have no tail.

The second source is ImageNet. ImageNet is an image database that contains about 14,197,122 images with 21,841 subnets indexed. It is a useful resources for researchers to collect labelled datasets (i.e. subnets). Some subnets have bounding box information which is ready for object detection. We chose five subnets of ImageNet to enrich our dataset [3, 4, 5, 6, 7, 8]. However, the quality of the subnets is quite low compared to the Oxford Dataset. Some images are labelled wrongly and some bounding box information is incorrect. To make sure our dataset was as accurate as possible, extra effort was required. We went through all the images of the subnets and deleted and modified incorrect ones. Due to the numbers, it is not possible to absolutely confirm that all errors were removed, but it is better to have human check and considerable effort was placed on this topic. The service offered by ImageNet is also not stable. We met a problem that it denied all requests to both the website and API access, which increased the difficulty of collecting images.

The last two sources used were Flickr and Google Image. Using their open API, we used a Python script to crawl images automatically. Images were collected if their text, tag or description parts had the keyword given by a cat breed name and their image quality satisfied the requirements we set, such as resolution limit. Following this, we used a tool named as *labelImg* [9] to create label and bounding box information. The main difficulty we met is that not all the images downloaded

correctly labelled the breed we expected. For example, if a user of Flickr uploads an image of a dog but writes "miss my persian cat a lot" in the description part, the image will also be collected. Just as what we did for the ImageNet dataset, we manually corrected labels of the downloaded images.

2.2. Dataset Integration

There are two main kinds of files used in our dataset: image files and corresponding *xml* files. xml files store supplementary information about the image, such as bounding box and path information. To make sure the training program can access them correctly, a uniform format was necessary. We did the following steps to integrate our dataset.

Firstly, uniform filenames were required since the four images sources have their own ways to name images. Secondly, some information in *xml* files needed to be changed, such as *folder name* and *path*. In this project, we use Python scripts to modify them automatically. We also needed to generate a file which recorded image-label information to guide the supervised machine learning. All the scripts used are listed in the Github.

2.3. Dataset Summary

The total number of images obtained was 12,451. As seen in Figure 3, the distribution of each breed is not even. The correlation between the accuracy and the number of images will be discussed in Section 5.1.

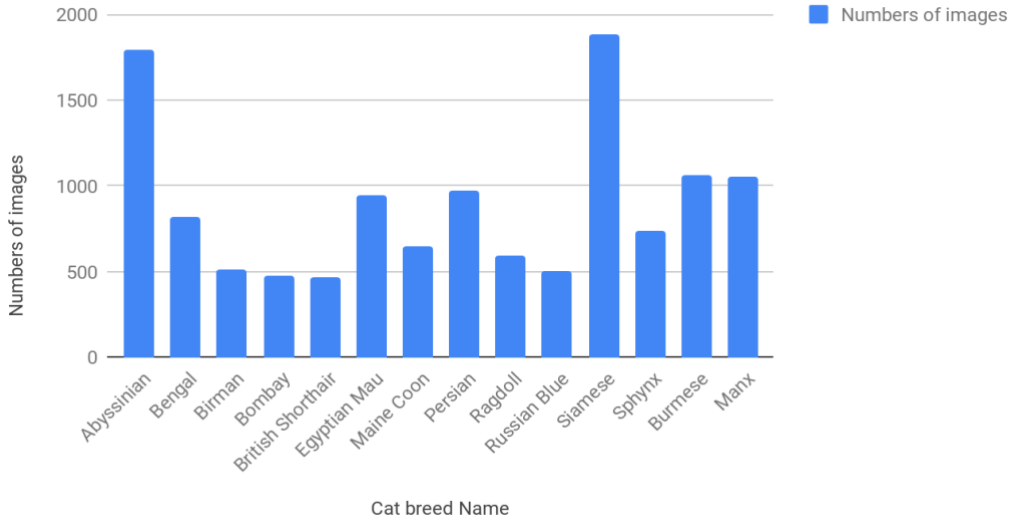

Figure 3. Summary of the dataset

The structure of the dataset is listed below in Figure 4. Bounding box information is stored in *xml* files and images are listed under the 'images' folder. '*pets_labels_list.txt*' records the 14 labels used in the project. These are used to create the label matrix during the training phase. 'trainval.txt'

records the corresponding relationship between the *image name* and *label*. For example, '*Abyssinian_10008 1*' means *Abyssinian_10008.JPEG* corresponds to label 1 which is for the Abyssinian cat.

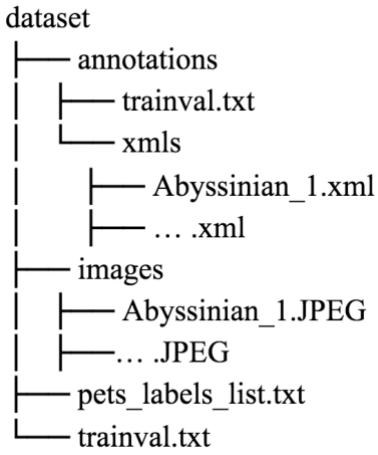

Figure 4. The structure of the dataset

Moreover, we split the dataset into a *training set* and a *validation set* in the ration 7:3. It is normal to have extra *test set* to test whether object detection models work well. However, in the project, we abandoned the *test set*. There are two reasons. Firstly, the *test set* is normally given at the end to avoid people tuning parameters depending on the *test set*. Thus, it is not necessary to have *test set* during the training and optimization phase. Secondly, as our project is to build a mobile app, it means we have no idea of what kind of images users will use. Therefore, the *test set* is not useful to test the accuracy of the actual usage. The more images in the *validation set*, the high possibility we can build a robust model. So we get rid of *test set* here.

2.4. Dataset Pre-processing

Even though we did a lot in dataset collection phase, such as manual labelling and formatting, there were still many things to be done before training our dataset. The following parts give a description of what we have done.

2.4.1. Data Augmentation

Even though there were 12,451 images of 14 breeds of cats in the dataset, this is not enough for a computer vision project generally. Therefore, data augmentation is used to "have more images". It is an effective way to expand a dataset by using transformations and deformations on images without changing their labels, resulting in additional data [10]. In general, data augmentation involves many methods, such as crops, rotations, translations, scaling, and mirroring. It has been shown that augmented data could enhance the discriminative and generalization ability of the model in training phase [10].

Rotation is used in the project. Apart from the purpose of enlarging the dataset, rotation is suitable for cats. A cat is a kind of animal that likes to stretch their bodies, which makes it hard to detect accurately as their shapes are changeable. By rotation with a random angle, convolutional neural network models can recognize a cat better. For example, Figure 5 shows the rotation of one image in our dataset by 90 degrees respectively.

Figure 5. Examples of rotation

2.4.2. Resize

Normally, resizing is necessary in data pre-processing for object classification. However, it is not so important in object detection. The reason is that bounding box information is already recorded in the *xml* files. Figure 6 is an example used in the dataset where the green rectangle is the bounding box. Our object detection model extracts bounding box information first and then learns features within the rectangle.

Figure 6. An example of images with bounding box information

However, we still use the resize method. The reason is that the size of image inputs is fixed in our detection model, so it is better to do resize operation before the training phase to increase the performance.

2.4.3. Shuffle

Shuffle is a good way to rearrange images randomly. This is useful for machine learning as it will increase the robustness of the model. In our dataset, images are ordered by their breeds, which

means the distribution of the dataset is uneven. Using shuffle, we have more confidence that our *training set* and *validation set* are representative of the actual distribution of the dataset. The method is critical to create *TF record file* which is a Tensorflow intermediate file generated from raw dataset files used to train our detection model.

2.4.4. Mean Normalization

It is normal for people to adjust the color and contrast of images to make images more beautiful nowadays. For human beings, it does not matter since we still can recognize the content of the given image. For example, it is easy to recognize there is a cat in a black-white image (Figure 7).

Figure 7. An example of a cat in a dark black-white image

However, it is difficult for computers to detect a cat since fur color is an important aspect used for detection. To improve the robustness of our model, mean normalization is used to overcome the exact color of images (i.e. absolute RGB values) but rather we focus on the contrast of colors in the image.

3. Training

This section illustrates what we do in the training phase and explains why we made the decisions. We introduce related knowledges used in the training phase (Section 3.1). Following this we introduce the general object detection framework (Section 3.2). We then show how to evaluate object detectors (Section 3.3) and explain the final model chosen (SSD Mobilenet_v1 FPN) in detail (Section 3.4).

3.1. Related Knowledge

In this Section, we explain the importance of transfer learning and overfitting challenges respectively.

3.1.1. Transfer learning

Transfer learning is an effective way to make use of knowledge learned from previous dataset to enhance a new model performance within the same domain [11]. According to Torrey and Shavlik,

there are three possible benefits when using transfer learning [12] (Figure 8). With transfer learning, we can have a high start, scope and asymptote, which shortens the training time and improves the outcome performance. Therefore, transfer learning is a popular method in deep learning.

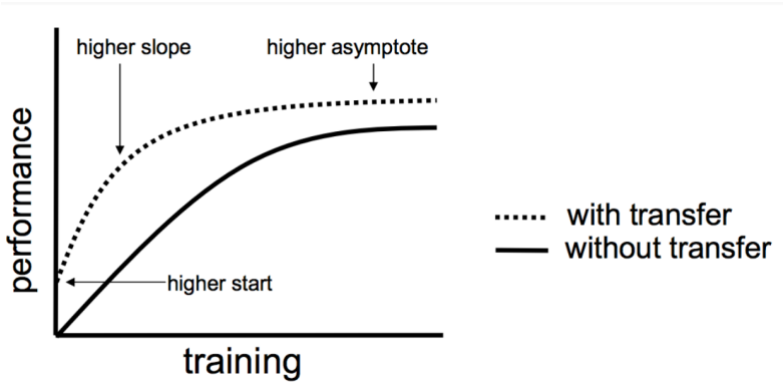

Figure 8. Three benefits using transfer learning [12]

Moreover, there are many pre-trained models available on popular machine learning platforms, such as Caffe and Tensorflow. As features of pre-trained detection models can be general (e.g. car and dog), these models are helpful to initialize a new model if the object interested are already in their dataset categories. By re-training a pre-trained model using our dataset, the training process will be significantly shortened.

The pre-trained models we used in the project are based on COCO dataset which includes 80 object categories (including 'cat' category) and more than 330K images [13]. This helps overcome the shortage of images in our dataset by using these models.

3.1.2. Overfitting Resistance

Overfitting is a problem that more terms and complex methods are used more than necessary [14]. It happens under the situation that the accuracy of the *training set* increases (i.e. the loss decreases) as more training time is spent, but the accuracy of the *validation set* does not grow with the *training set*. The reason is that the features we gain from the *training set* are much more than the ones required in prediction. In other words, the object detection model focuses on tiny features which is useless to detect objects. For example, it is normally for cats to have dark dots near their mouths. But it is not appropriate to use it as an index to detect a cat.

Using Tensorboard - a visualization tool of Tensorflow, we can visualize the trend of the result of loss and activation function during training. If we detect overfitting problems, we can reduce the number of training steps. Besides, we also use regularization techniques to prevent overfitting in our object detection model.

3.2. Object Detection Framework

In recent years, there have been many successful object detection solutions proposed [16]. Generally, an object detection model includes meta-architecture and object feature extractors. However, these solutions offer a fixed meta-architecture and feature extractors. For example, Fast R-CNN uses SPPnet as a feature extractor [15]. One of the key aspects of this work is that we split meta-architectures and feature extractors from current solutions and permutate them to make use of their advantages and disadvantages.

In the following part, we briefly introduce the meta-architectures and feature extractors used in the project and explain the object detection framework in detail.

3.2.1. Meta-architectures

There are many meta-architectures used in object detection. Examples include Faster R-CNN, SSD and YOLO. In this project, we chose Faster R-CNN and SSD as our meta-architectures. There are two reasons for this. One reason is that Faster R-CNN is an object detection architecture based on regions and SSD offers a single shot object detector like YOLO. The other reason is that both two meta-architectures are able to work with the feature extractors chosen in the Section 3.2.2 below.

Figure 9 gives the high level diagrams of these two architectures. SSD and Faster R-CNN use different ways to detect objects. For example, Faster R-CNN uses *Proposal Generator* before classification. However, both of them can use the same feature extractors such as *VGG* and *Inception*.

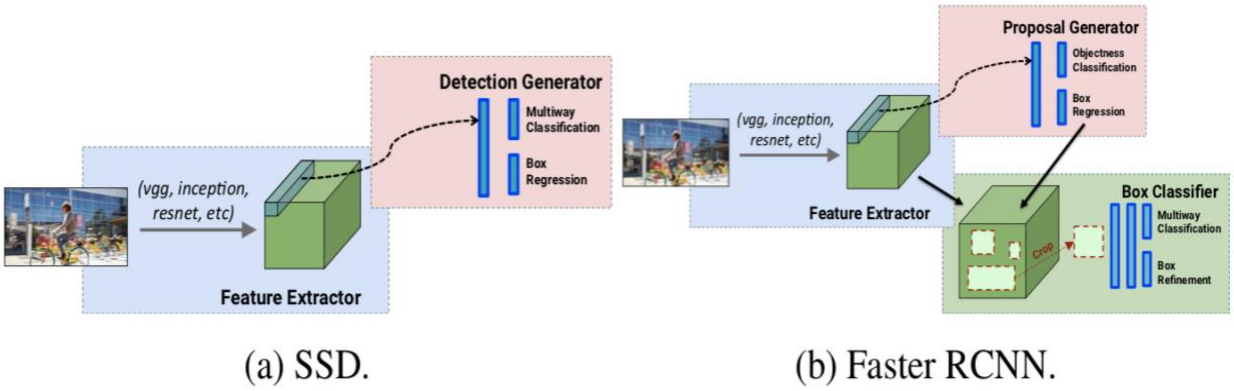

Figure 9. High level diagrams of meta-architectures used in the project[16]

Faster R-CNN

Faster R-CNN has two modules (Figure 10). The first one is a deep fully convolutional neural network which proposes regions (i.e. "region proposal network (RPN)") , and the second one is Fast R-CNN detector that uses the proposed regions [19]. The RPN module guides the detector where to look [19]. The implementation details can be found in [19].

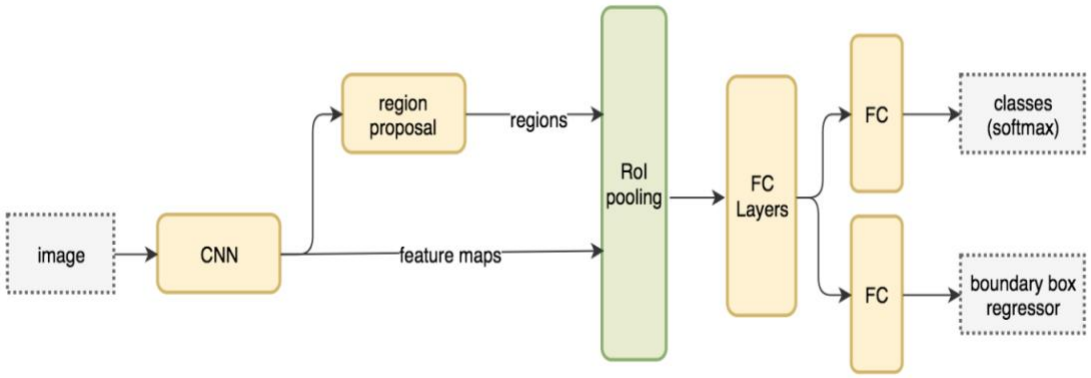

Figure 10. Network flow of Faster R-CNN [17]

SSD

SSD eliminates the RPN that Faster R-CNN uses. However, it offers improvements, such as multi-scale features and default boxes, which enables SSD to have the same accuracy as Faster R-CNN with lower resolution images whilst still meeting the speed required for real-time object detection [20]. As shown in Figure 11, SSD can use the VGG19 network as a feature extractor (similar to CNN in Faster R-CNN) [18]. It adds custom convolution layers (blue parts in Figure 11) and convolution filters (green parts in Figure 11) to make predictions [18].

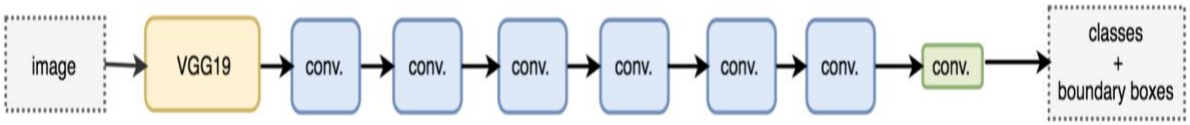

Figure 11. Network flow of SSD [18]

3.2.2. Feature Extractors

We chose the following four popular and high performing feature extractors in this work.

Resnet

Resnet (Residual Neural Network) is an architecture of CNN, which is a breakthrough for the development of deeper networks [21]. It solves the problem of vanishing gradients [22]. Most deep neural networks, such as AlexNet, often suffer gradient signals of error functions decreasing exponentially as they backpropagate to previous layers making it harder to learn small signals. With the help of shortcut connections, the gradient signal in Resnet can travel the way back to the previous layers which makes it possible to build more layers in the net. The implementation details can be found in [21].

Inception

If Restnet is about being deeper, Inception is about being wider. The key part of the Inception architecture is that it has a new block named “Inception module”. The Inception module computes

multiple transformations from the same input map in parallel and concatenates results into a single output. The model decides by itself which piece of the information it should use [23]. Inception module uses 1x1 convolutions to perform dimension reduction in order to solve the computational bottleneck when information density increases as the number of layers grows. By reducing the number of input maps, different layer transformations can be stacked in parallel, which results in deeper and wider nets [23]. The implementation details can be found in [23].

Mobilenet

Mobilenet uses depth-wise separable convolutions to build a lightweight deep neural network [24]. Depth-wise separable convolution has two layers, depth-wise convolutions and pointwise convolutions [24]. There is a single filter used for each input channel in the depth-wise layer. Pointwise convolution with a simple 1×1 convolution is used to create a linear combination of the output of the depth-wise layer [24]. The implementation details can be found in [24].

FPN

Feature Pyramid Network (FPN) is a top-down architecture with lateral connections used to build high-level semantic feature maps at all scales [25]. By replacing image pyramid features with in-network feature pyramids, there is no performance loss in the power, speed, or memory [25]. Moreover, it achieves improved single-model results on the COCO detection benchmark when used with a Faster R-CNN architecture [25]. The implementation details can be found in [25].

3.3. Comparison between different detection frameworks

By combining the meta-architectures and feature extractors mentioned above, we chose six object detection models as our detection model options. These included:

- Faster R-CNN Inception_v2
- Faster R-CNN Resnet50
- SSD Inception_v2
- SSD Mobilenet_v1
- SSD Resnet50 FPN
- SSD Mobilenet_v1 FPN

The following parts illustrate how these were evaluated and the results of the evaluation.

3.3.1. Methodology

Since our project aimed to build a mobile app to detect cat breeds, there were two important indexes: prediction accuracy and detection time. It is easy to understand that accuracy is a concern since we need to have a high confidence that the app can detect and recognize different cats correctly. Besides, the reason that detection time matters is that we want to provide a better user experience as our project is to solve a real-world problem. If it takes a quite long to detect a cat, target users will lose their patience and probably will not use the app again. Therefore, a trade-off must be made between accuracy and detection time.

During model evaluation, we use pre-trained models provided by *Tensorflow detection model zoo* as a start point of transfer learning. Using parameters set in the default configure file of each model, we re-trained models using our dataset. Notably, default values of the parameter "*num_steps*" (i.e. the number of training steps) of these configure files varied from 25,000 to 200,000. In this case, we choose 25,000 as the default value for each model to save training time. After re-training, the value of the mean Average Precision (mAP) and average detection time of each model was recorded. By comparing these values, we obtained a better understanding of the basic performance of each model of our dataset.

mAP was calculated with an evaluation script offered by Tensorflow (*eval.py*). The script uses images in the *validation set* to compute the overall average accuracy and each class's average accuracy. We chose *mAP@0.5IoU* as our evaluation metric. IoU means *intersection over union*, which is a metric used to measure the accuracy of an object detector. Generally, if IoU > 0.5, it is an acceptable prediction. Therefore, mAP@0.5IoU is a good index to measure these models' average accuracy.

The average detection time was calculated using the formula in Figure 12. Here, *object detection script* refers to the script that uses the model trained to detect cats for given images. The script is available in Github.

$$Average\ detection\ time\ =\ \frac{overall\ object\ detection\ script\ running\ time}{the\ number\ of\ images}$$

Figure 12. Formula to calculate average detection time

3.3.2. Result of comparison

Figure 13 shows the comparison of mAP@0.5IoU and the average detection time of the six detection models.

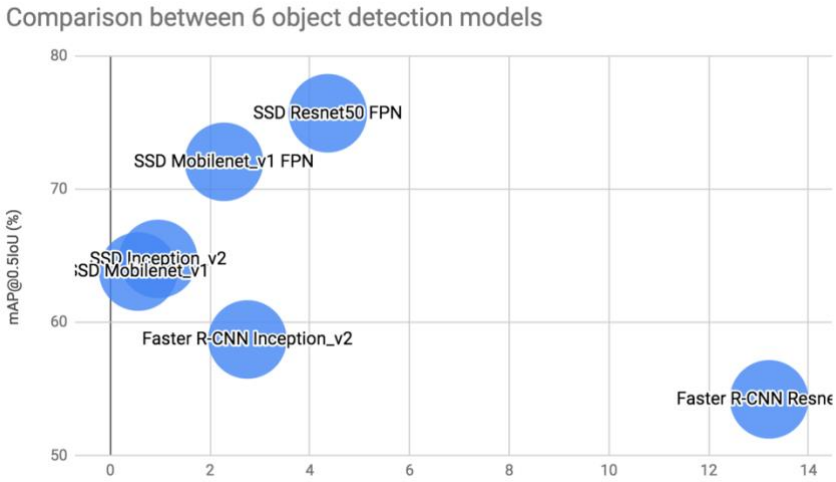

Figure 13. The comparison result of mAP@0.5IoU and detection time of the six models

It is obvious that the overall performance of SSD models is better than Faster R-CNN models. SSD is a one stage model while Faster R-CNN is a two stage model (Figure 9), hence it takes more time for Faster R-CNN to detect objects. Generally, Faster R-CNN can achieve higher prediction accuracy than SSD. However, with the new feature extractors, such as Mobilenet and FPN, the accuracy of compound SSD models is even higher than Faster R-CNN.

Both detection models using FPN feature extractor to achieve a better performance than other models, both in accuracy and detection time. The accuracy of *SSD Resnet50 FPN* was 75.71% and the average of detection time was about 4.36s. *SSD Mobilenet_v1 FPN* achieves 72.06% accuracy with about 2.279s average detection time. Even though the accuracy of *SSD Resnet50 FPN* is slightly higher than *SSD Mobilenet_v1 FPN*, taking detection time into consideration, we chose *SSD Mobilenet_v1 FPN* as our final object detection model.

3.4. SSD Mobilenet_v1 FPN architecture

SSD Mobilenet_v1 FPN is an object detection model which takes advantage of SSD, Mobilenet and FPN. Recently, Tensorflow provides TPU (i.e. Tensor processing unit) compatible version of *SSD Mobilenet_v1 FPN* model which accelerates the training process considerably.

Figure 14 is a basic structure of *SSD Mobilenet_v1 FPN* model. The model has four "*WeightSharedConvolutionalBoxPredictor*" (i.e. an independent unit to predict *Box* and *Class*) , which can compute image inputs in parallel. Above them, there is a layer to control and summarize the results from each *Convolutional Box Predictor* that is used to make the final prediction.

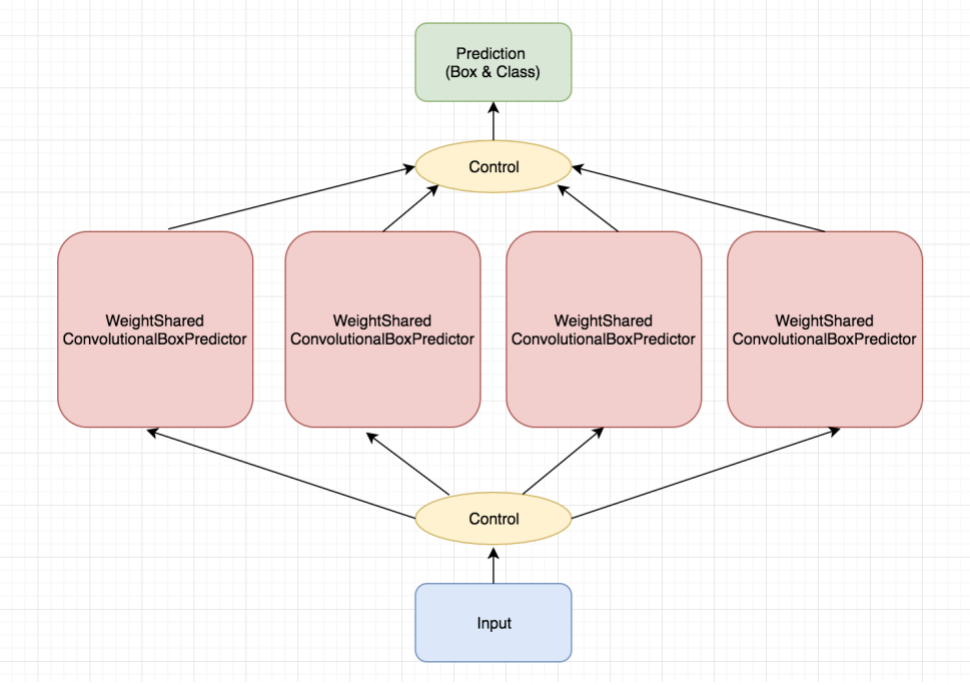

Figure 14. Basic architecture of SSD Mobilenet_v1 FPN model

Figure 15 shows one *WeightSharedConvolutionalBoxPredictor* from Tensorboard. Each *Convolutional Box Predictor* has two pairs of *PredictionTower* and *Predictor*, one for the Bounding Box and the other one for Class. There are four convolutional layers with the size of 80*20*20*256 pixels in the *PredictionTower*. There are three ReLUs (i.e. Rectified Linear Unit) between the two convolutional layers to make training faster. In *Predictor*, there is one convolutional layer (80*20*20*24) and one bias layer (80*20*20*24). The *Predictor* summarizes the prediction result from *PredictionTower* and sends it to the upper layers.

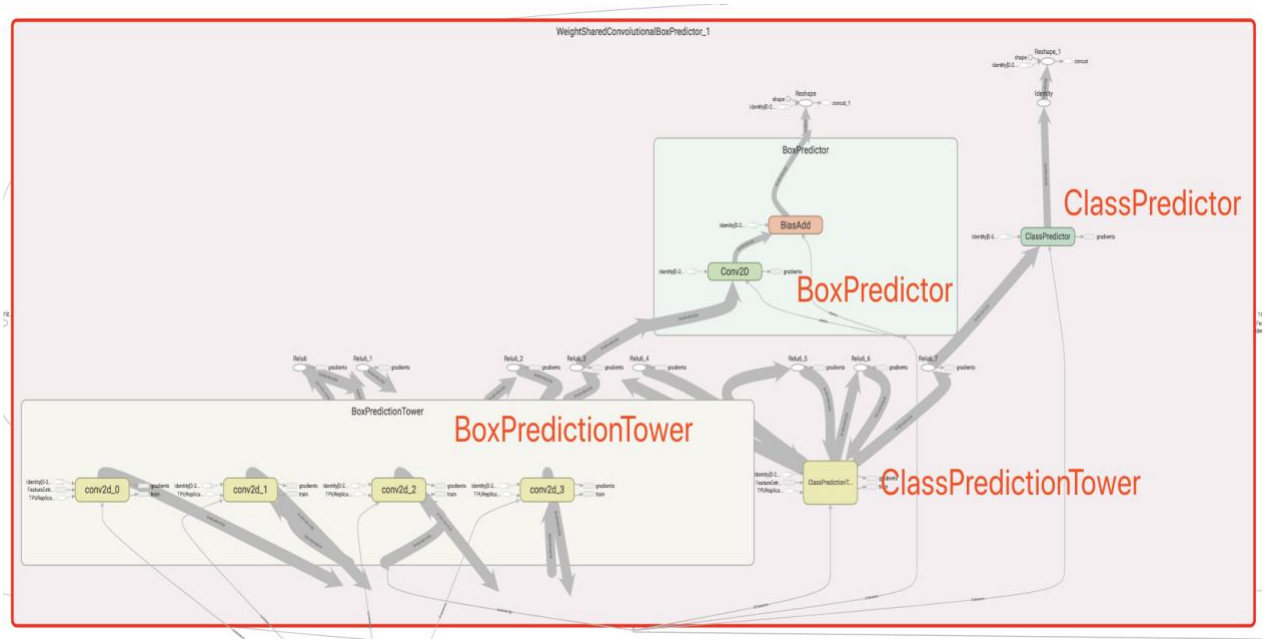

Figure 15. Diagram of WeightSharedConvolutionalBoxPredictor

4. Model Optimization

Even though the accuracy of the chosen model was acceptable, we still need to optimize the model. There are two main reasons. Firstly, since the accuracy we have comes from using a fine-tuned configure file in Tensorflow, there is no specific optimization operators based on our dataset. By tuning some parameters in the configure file, the accuracy can increase. Secondly, by model optimization we can have a better understanding of the effect of each parameter and usage of TensorBoard.

We optimize our chosen detection model from two aspects, dataset and configure optimization. The following two parts explain these aspects.

4.1. Dataset Optimization

Normally dataset optimization should not be a method to optimize a detection model as the dataset is generally given and fixed. However, our current dataset has a drawback due to the nature of cat

breeds. Therefore, by improving the quality of dataset, our object detector will also be improved. The following parts explain the drawback of the dataset and methods used to optimize the dataset.

4.1.1. Drawback

In terms of the nature of cat breeds, there are two main issues which impact on cat breed recognition. Firstly, no matter whether a pedigree or non-pedigree cats, they can have a variety of colours and patterns [26]. Figure 16 gives an example of *Bengal cat* with different colours and patterns. This is similar for other breeds.

Figure 16. Bengal cat with different colours and patterns [26]

Secondly, it is hard to use features extracted from kittens to detect their breed as the appearance of kittens is slightly different than their adulthood appearance. Figure 17 and 18 show a kitten and an adulthood *Birman* cat. They have common features in many parts, such as the colour of eyes and the dark edge of ears. The cat in Figure 19 also has these features and might be considered as a *Birman cat*. However, this is not the case. Figure 19 is a *Ragdoll* kitten. Therefore, the problem emerges: how to distinguish cat breeds if their kittens look similar.

Figure 17. Birman Kitten

Figure 18. Adult Birman

Figure 19. Ragdoll Kitten

Based on the above, there are more requirements for our dataset. The first one is the dataset should cover all of the appearances of each cat breed (i.e. the same breed but with different colours, patterns, etc.). The second requirement is the dataset should have more images to ensure each appearance of a cat breed has enough images. To achieve these two goals, professional knowledge of distinguishing each cat breed and further effort is needed to make the best possible dataset, which is infeasible in the project. Therefore, our current dataset has some limitations.

4.1.2. Methodology

We optimized our dataset by improving the "quality" of images in the dataset. In this context, "quality" means how well the images can represent the breed. If an image contains typical features of a cat breed, we assume it has a high "quality". It is reasonable to have this assumption for the object detection project. For example, a *Bengal cat* could have a snow colour [26]. However, since it is quite rare for this breed to have this colour, it is not a good option to add the image of a snow coloured *Bengal cat* to our dataset.

The solution of improving the quality of images we chose was to purify it by filtering images with ambiguous features (e.g. pictures without typical patterns of one breed or pictures with high similarity of other breeds) to achieve a higher accuracy.

Figure 20 shows the result of mAP@0.5IoU value for each cat breed before optimizing our object detection models. It is clear that the accuracy of *Ragdoll* is much lower than others (55.58%). *British shorthair* (62.61%) and *Egyptian mau* (65.1%) also have relatively low confidence. Therefore, we focus on filtering images of these three breeds. By deleting 161 ambiguous images, the mAP@0.5IoU of the breeds improves to (Table 1). The overall mAP@0.5IoU subsequently increases from 72.06% to 74.98%.

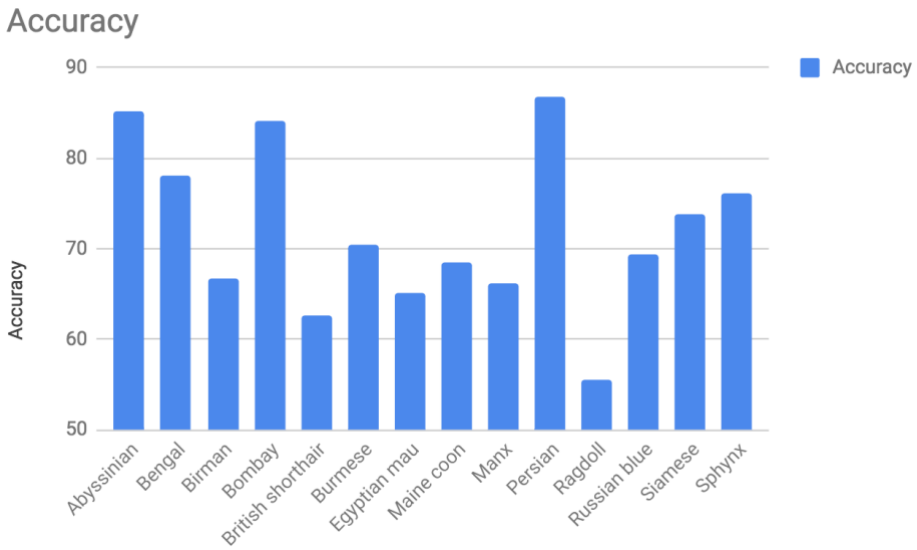

Figure 20. The result of mAP@0.5IoU value of each cat breed before model optimization

Accuracy (mAP@0.5IoU)	**British shorthair**	**Egyptian mau**	**Ragdoll**
Before dataset optimization	62.61%	65.1%	55.58%
After dataset optimization	78.66%	68.74%	58.94%

Table 1. The comparison of accuracy between before and after dataset optimization

4.2. Hyperparameter Optimization

As deep learning methods learn features autonomously, it is important to configure models' hyperparameters. However, the current status of deep neural networks is often a black box and hence it is difficult to tune parameters. Generally, there are several parameters which have impact on the performance of a model.

Firstly, learning rate. If the learning rate is too high, the loss function will not perform well. If the learning rate is too low, the model has a probability that it will not achieve convergence within the expected training steps.

Secondly, the batch size. The batch size defines the number of samples that will be propagated through the network. If the batch size is too big, there are two issues. Firstly, it requires more system memory to train. Secondly, the iteration time decreases, which means it will consume a lot of time to reach the same accuracy and hence weaken the effect of other hyperparameter tuning. If the batch size is too small, the model has a probability that it will not converge within the expected steps.

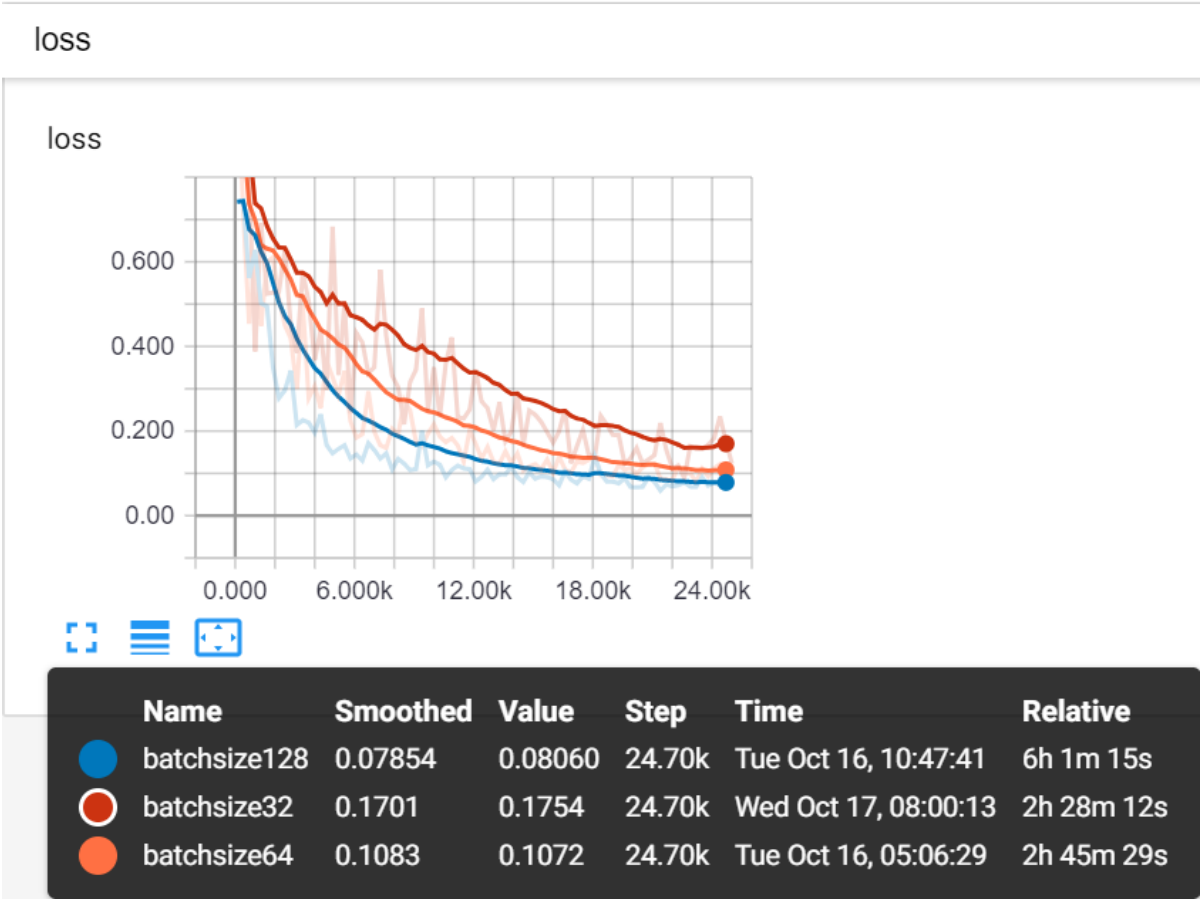

Figure 21. Loss curve changes with different batch size

From Figure 21, when the batch size increases from 64 to 128, it takes about 6 hours to complete the processing, while it only takes about 2 hours when the batch size is equal to 32 or 64. When the batch size decreases from 64 to 32, the loss function does not converge.

Thirdly, regularization is a very important technique in machine learning to prevent overfitting. The model uses weight penalty L2 to solve the overfitting problem by adding a weight to the parameters of the model. If the weight is too high, it will weaken features extracted from the training data. However, if the weight is too low, it cannot filter noise in the training data. Figure 22 shows the trend of the loss curve as the L2 weight changes. Table 2 shows the accuracy peaks at 81.8% when L2 weight is equal to 4.00E-03.

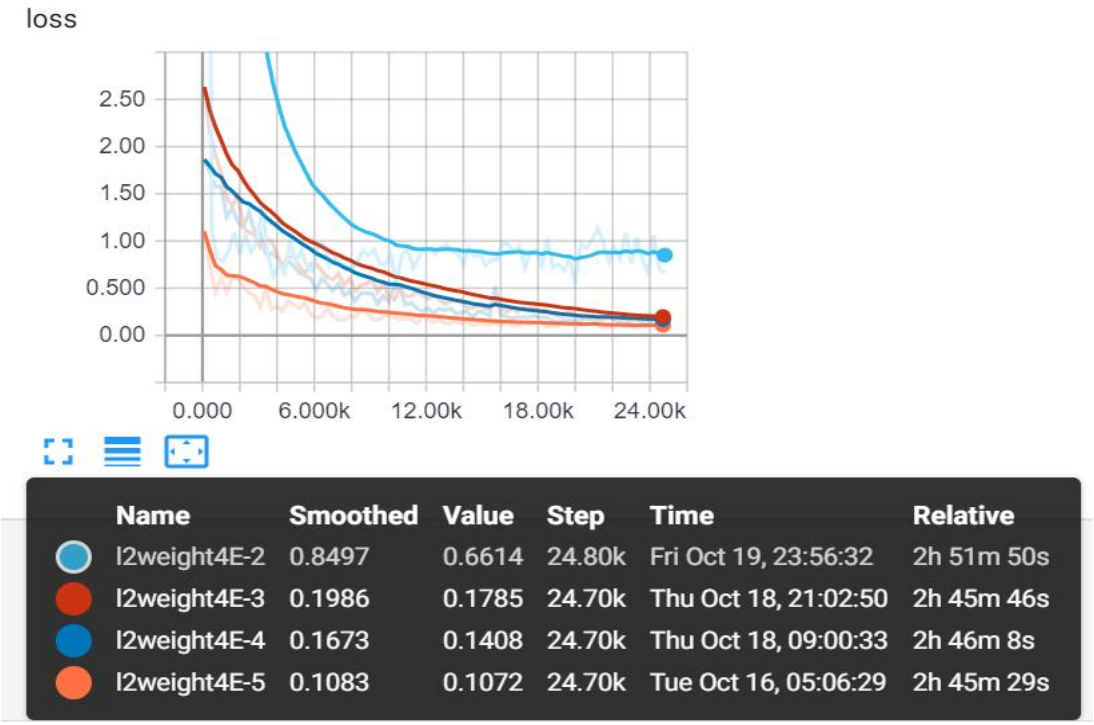

Figure 22. Loss curve changes with different L2 weight

weight/L2_Regular	mAP@0.5IOU
4.00E-05	0.749
4.00E-04	0.784
4.00E-03	0.818
4.00E-02	0.052

Table 2. Accuracy with different L2 weight

5. Results and Analysis

In this part, we analyse the overall performance of our detection model. Firstly, we illustrate the average prediction accuracy of each breed and discuss the underlying reasons. Secondly, we test the performance of our model under more complex images (e.g. images with many objects).

5.1. Prediction accuracy of each breed

Figure 23 show the relationship between the number of images of each breed and their accuracy. It is obvious that the accuracy (i.e. mAP@0.5IoU) of each cat breed is not even.

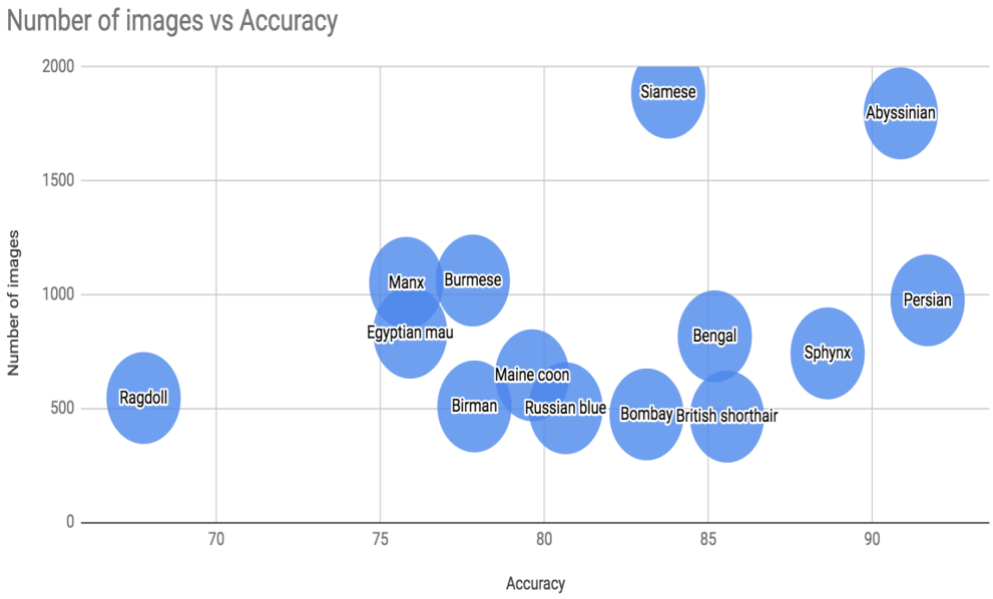

Figure 23. The prediction accuracy of each breed

Firstly, there are three breeds where the accuracy achieves around 90%, *Persian* (91.69%), *Abyssinian* (90.87%) and *Sphynx* (88.64%). Each of them have unique characteristics. A *Persian cat* has a fluffy fur and snub nose. An *Abyssinian cat* has beautiful almond-shaped eyes. The *Sphynx cat* is hairless. The results confirmed that our model works well when extracting cats with clearly noticeable features.

However, the model works less well when detecting *Ragdoll cats*. The accuracy is 67.78%. There are two reasons. One is the nature of the *Ragdoll cat*. The *Ragdoll* comes in 4 patterns with 6 colours [27], which means it is harder to extract typical features as with other cats like *Persian*. The other reason is that the number of *Ragdoll* images is not enough to represent all features (545 images). Compared with other cat breeds at the same level of sample images but with higher accuracy, such as *Birman* and *Bombay*, it is easy to find that there is no direct correlation between the number of images and the accuracy.

5.2. Complex Scenarios

Taken our app usage scenarios into consideration, there are four situations considered: one cat, more than one cat, many objects excluding cat and partial parts of a cat visible in an image (e.g. a cat partially shielded by a tree). Even though the last three scenarios are not typical, we still want to test our detection models under the complex scenarios.

We manually collected images from Google based on these scenarios to test the performance of our object detector. The reason that we chose the manual way is that it is much efficient to find "challenging" images, such as the fur color of a cat is almost the same as the background or many different cats are in one image. Even though this introduces bias due to manual selection, it is good

to give a taste that how the detector works in different situations. The following part lists a part of results we ran with our detector.

All in all, the results are acceptable. The accuracy of object classification is about 80%. However, the accuracy of object detection is quite low, often around 60%. The detector does a good job when detecting many objects with clear boundaries around objects, such as Figure 24.

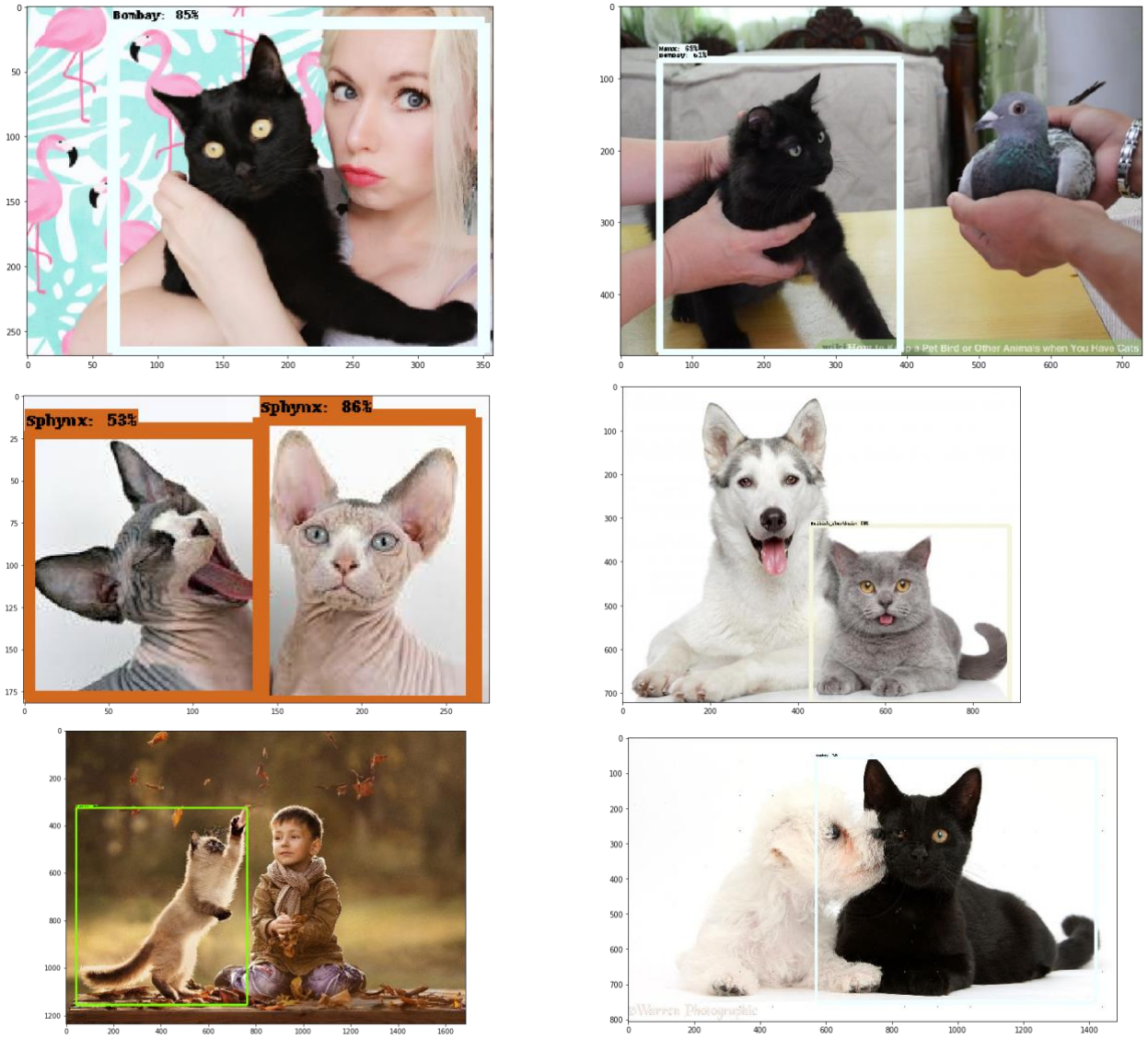

Figure 24. Good examples when detecting many objects

It is reasonable if we know how SSD is used to detect an object (Figure 25). SSD uses different size of anchor boxes in each level of the feature map to detect the location of an object. From Figure 25.b and c, we can see that the dog part has more information to predict since it is a larger part of the image. In other words, SSD works well when detecting large size objects rather than small objects. This explains why images in Figure 24 have a better performance since the size of each part of images is similar.

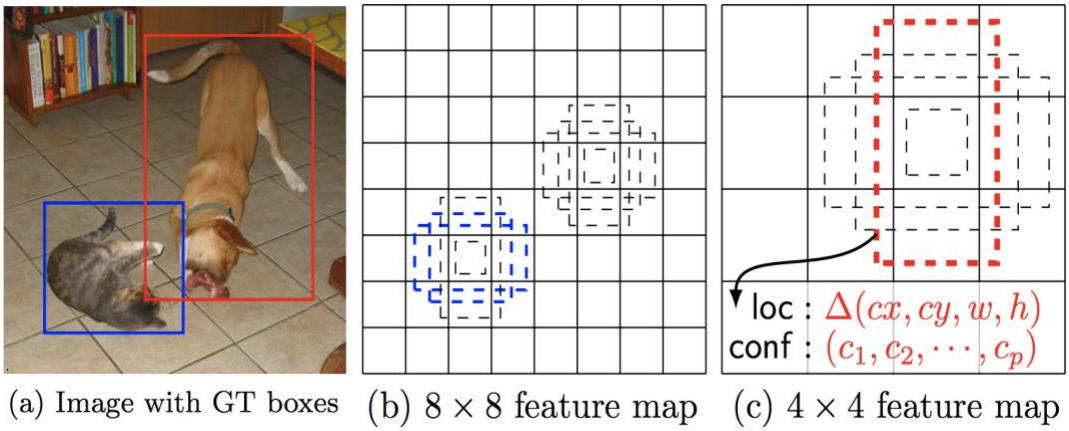

Figure 25. A diagram showing how SSD works

However, the drawback of the detector is also obvious. Figure 26 shows examples where the detector cannot successfully detect all objects. For the same reason mentioned above, SSD works poorly when detecting small objects. It focuses on the larger part of images and gives a broad bounding box information as much as possible.

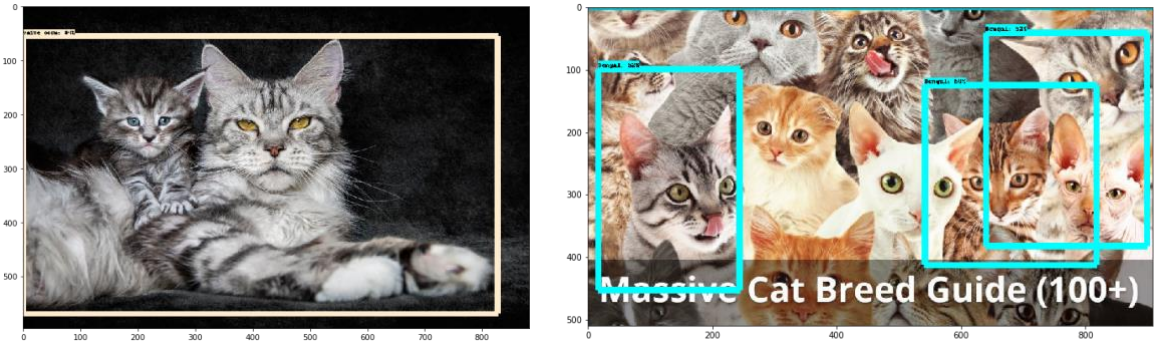

Figure 26. Bad examples when detecting many objects

Moreover, the detector works well when a cat has a similar colour as the background. For example, a cat on a tree in Figure 27. It works well even if the colour of fur is similar to the colour of the tree branch.trunk. The reason is that we use the mean normalization method to reduce the impact of background colour during training.

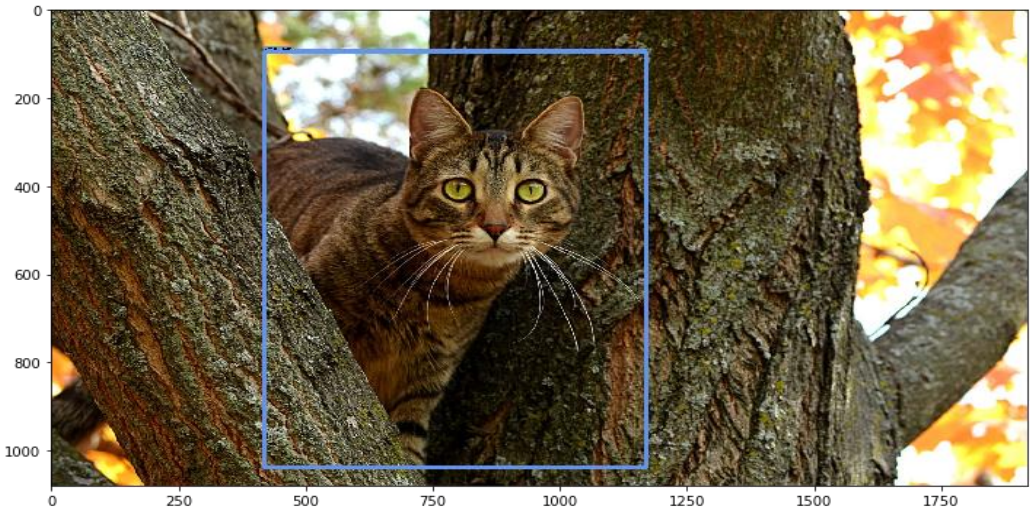

Figure 27. A good result when a cat on a tree

5.3. Performance of the Mobile App

All the statistical data mentioned above is based on the detector running on the server. Since the usage scenario of the project is to develop and use an Android mobile phone to detect a cat, extra evaluation is needed related to the mobile app.

There are three main differences between using the detection model on the server and the mobile app. Firstly, when a server detects a cat, the input image is static. However, when we use mobile app, the input image is dynamic. For example, the cat under detection is likely to move and slight device shaking is hard to avoid. Secondly, the computing power of a server and a mobile phone vary a lot, which could have an influence on the detection performance. Thirdly, the quality of the "input" are different. When using a camera on a mobile phone to take images for the target cat, the quality will be impacted based on many factors, such as the brightness and shooting angles.

After embedding the object detector generated on the server into an Android app, we test two scenarios: single object detection and multiple object detection. The accuracy of using the app is not always stable. Figure 28 gives an example showing how the accuracy varies with the changes of shooting angles. Generally speaking, the accuracy when using the app is lower than using the server, which is around 70%. Besides, the prediction time is about 2-3 seconds longer than using the server.

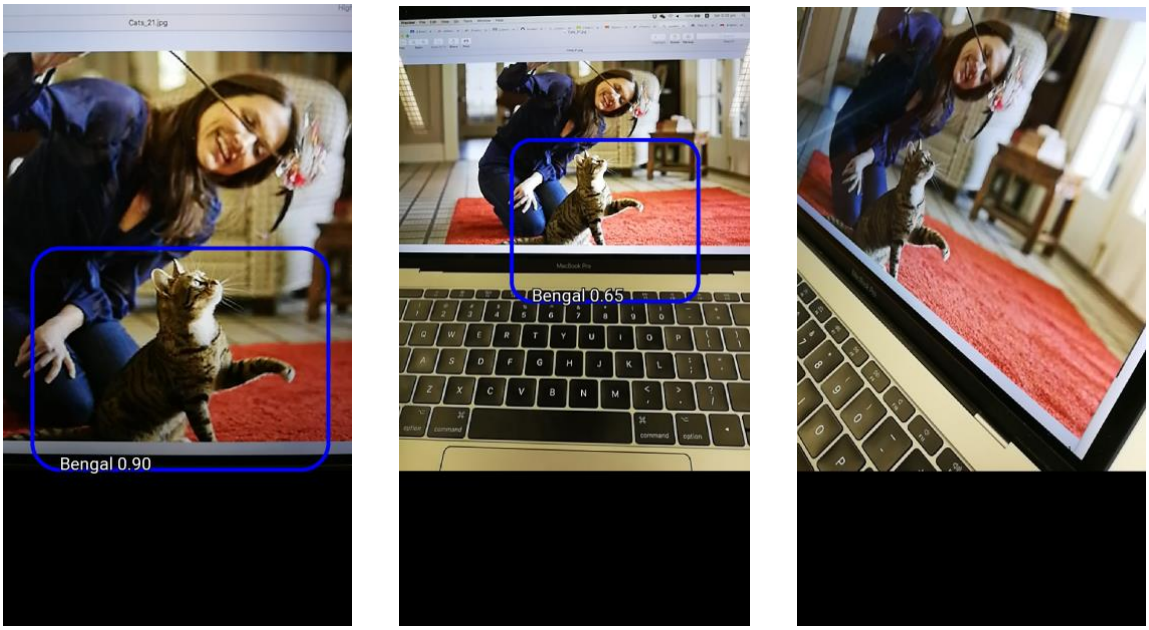

Figure 28. An example that the accuracy varies with the changes of shooting angels

6. Android Application Implementation

6.1. Design

As mentioned, this application is aiming to develop a convenient version of the cat recognition model. The model we selected for our project was based on *SSD Mobilenet_v1 FPN*. There are two ways to realise this. The first one is to develop a client server model application. Another one is to build an independent application to run the model. To be more specific, the client server model application means, on the smartphone side, we just need to develop an application to call the camera of the smartphone. On the server side, we need to build a server which is used to run the trained model to perform the detection process. The mobile application and the server could communicate, e.g. via an API. The mobile application needs to send the image which has been captured by the camera to the server. Then the server would run the model to analysis the received image and return the prediction result to the mobile application. The advantage of this way is that the server has more powerful computation power, and it can do the detection process in a shorter time.

However, there are also some limitations with this client server model. Firstly, the application and the server need to communicate which can take time. That means then the network is overloaded, the sending and receiving process can take a long time. When the server is down, the application would stop working completely. Besides, in this model, the server also needs to be maintained regularly to ensure the application can run smoothly.

The independent application means the trained *SSD Mobilenet_v1 FPN* model can be transferred into a mobile version and added to the mobile application. The application could run locally without any network limitations, and the user does not need to do any maintenance work to make the system run correctly. Furthermore, if the user would like to improve the performance of the trained model, they just need to replace the existing model with the new model directly. It would be much easier than the client server model application. The limitation is that it may require more time to do the detection process. For a mobile application, offering better service should be the final goal. Based on the comparison, we decided to develop an independent application to run the model for its stability and simplifying further development.

6.2. Tensorflow on Android

As discussed, we used Tensorflow to train our model for object detection. To develop the application, we need to transfer the Tensorflow to an Android version model. There is a mobile version of Tensorflow called *Tensorflow lite.* This is a lightweight version of Tensorflow. It could do the same work as the model on PC in a faster way. Because of the limitation of the smartphone computation power, *Tensorflow lite* is more suitable for smartphones for deep learning.

6.3. Interface

It is necessary for a mobile application to be user friendly. The user interface is an important component for any application. Our mobile application uses the camera of the mobile phone to capture real-time images and invoke the trained model to make predictions of the breed of cat on the image. At the same time, the trained model shows which part of the image has been identified. We designed a very simple interface for users. The user just needs to start the application, then it will automatically call the camera and the trained model directly. The user can see the real-time image on the screen immediately. After several seconds, there will be a rectangle area shown on the image. That area is used to show which part of the image has been predicted, and the result of the prediction will also be shown on the edge of the rectangle. This is a very simple interface and any user could run this application directly without any instruction.

As before introduced, this project is aiming to perform object detection process for images. This includes the combination of image classification and location determination. This makes the deep learning model much more complex than just the classification process. That means the model need to take more time to make the prediction of the image. Based on our testing, the classification process just needs to takes around 1 second, but the detection process needs about 8 seconds. Therefore, in addition to the interface for object detection, we also make an independent user interface to call the classification model. The user interface is almost the same as the object detection one, but it only offers the result of image classification without the location identification. It is used for future direction of this project, and the details will be introduced in the future work part.

To make the user be more familiar with our model, we also designed an independent interface which is used to show all types of cat which can predicted by our model. This interface shows some information about each type of cat which can be identified and some information about the model performance to be expected.

7. Future work

With the limitation of time and resources, there are also various ways this project could be improved in the future. The future work is mainly about two areas. One is to enhance the existing model performance, another is to develop other models.

As discussed, this project focuses on image detection through deep learning. The detection process can take a long time to get the result. There are two ways to improve the performance. One is about hardware and another is about the model improvements. For the hardware area, running the model on some device with more computation power like a GPU server would bring some improvements on the time. However, it would increase the project cost. For model improvements, as discussed, the deep learning model performs object detection by analyzing the image through a lot of features and layers. Offering better features and reducing the layers of the model could also reduce the time needed for processing. However, reducing the layers and changing the features may also influence the accuracy of the model. Therefore, reducing the processing time with high accuracy would be difficult. Deep learning models could summarize useful features from the training dataset. The accuracy of the model may also be improved by increasing the size and quality of the training dataset. Future developers could also collect more image with some pre-processing as the training dataset to improve the model performance.

For the new model area, as mentioned before, we have developed a mobile interface which is used to do the image classification process. We use an open source model which is trained by others to do the classification process. It is used to classify what is in the image, not for the cat breed classification. The future developers could develop a new model which is used to do the classification process with Tensorflow and replace the existing classification model with a new one.

8. Conclusion

This project focused on developing a cat detection model by deep learning through a mobile application. The trained model could recognize 14 types of cat with average 81.74% accuracy. The work compared six popular models with the final choice of *SSD Mobilenet_v1 FPN*. The model training tool was Tensorflow, and the platform delivered on the Android platform.

The first step of the work was data collection and preprocessing. We collected lots of image data

from different data sources and performed various processing on these images. Because data quality and quantity are both very important for a deep learning models to have better performance, data collection and preprocessing were essential.

The next step for this project should be model selection. As mentioned, there are a lot of models which can do object detection, but their performances vary. To find a suitable model, we tested different models and collected their performance data. Finally, based on the consideration of time costing, computation power needs, accuracy performance and complexity of implantation on Android phones, we selected *SSD Mobilenet_v1 FPN* as the model for our project.

The third step was to train and tune the model. We used our collected dataset as the primary source to train model and achieved average accuracy around 72.06%. By optimizing the dataset and hyperparameters, the accuracy improved to around 81.74%.

The last step of the project was to implement the trained model on the Android platform. To get a stable and user-friendly application, we developed an independent application for users. The application could run locally on the phone, without any influence of the network situation. The user could operate the application without any complex instructions. We identified various ways in which this work could be improved in the future.

Appendix

Statistical data used in the report:
https://docs.google.com/spreadsheets/d/1i2yfzJ_fb6Itoh2m88Y_wB4eTvmlyh-0b0I_HYOc7zw/edit?usp=sharing

Source code: https://github.com/tinalulu1327/Cat_Recognition.git

Demo video link: https://www.youtube.com/watch?v=7DSj4VKjEOU

References

[1] Parloff, R. (2016). The Deep-Learning Revolution. Fortune, 174(5), 96–106. Retrieved from https://ezp.lib.unimelb.edu.au/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=118302290&site=eds-live&scope=site

[2] Oxford Pet Dataset Available at: http://www.robots.ox.ac.uk/~vgg/data/pets/ (Accessed: 8 October, 2018).

[3] The subnet in ImageNet Available at: http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n02124313 (Accessed: 10 September, 2018).

[4] The subnet in ImageNet Available at: http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n02123917 (Accessed: 10 September, 2018).

[5] The subnet in ImageNet Available at: http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n02124075 (Accessed: 10 September, 2018).

[6] The subnet in ImageNet Available at: http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n02124484 (Accessed: 10 September, 2018).

[7] The subnet in ImageNet Available at: http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n02123394 (Accessed: 10 September, 2018).

[8] The subnet in ImageNet Available at: http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n02123597 (Accessed: 10 September, 2018).

[9] The source code and installation information Available at: https://github.com/tzutalin/labelImg (Accessed: 10 September, 2018).

[10] Ahmad J, Muhammad K, Baik SW. (2017) Data augmentation-assisted deep learning of hand-drawn partially colored sketches for visual search. PLoS ONE 12(8): e0183838. https://doi.org/10.1371/journal.pone.0183838

[11] Pan, S. J., & Yang, Q. (2010). A survey on heterogeneous transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345-1359.

[12] E. S. Olivas, J. D. M. Guerrero, M. M. Sober, J. R. M. Benedito, A. J. S. Lopez. (2010). Handbook Of Research On Machine Learning Applications and Trends: Algorithms Methods and Techniques - 2 Volumes. Hershey, PA:IGI Publishing, 2009.

[13] COCO Dataset Available at: http://cocodataset.org/#home (Accessed: 11 October, 2018).

[14] Hawkins, D. M. (2004). The Problem of Overfitting. Journal of Chemical Information and Computer Sciences 2004 *44* (1), 1-12. DOI: 10.1021/ci0342472

[15] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448).

[16] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... & Murphy, K. (2017, July). Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR* (Vol. 4).

[17] Jonathan Hui. (2018, March 28). What do we learn from region based object detectors (Faster R-CNN, R-FCN, FPN)? In Medium. Retrieved October 15, 2018, from https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9

[18] Jonathan Hui. (2018, March 28). What do we learn from single shot object detectors (SSD, YOLOv3), FPN & Focal loss (RetinaNet)? In Medium. Retrieved October 15, 2018, from https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d

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

[20] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21-37). Springer, Cham.

[21] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[22] Wikipedia contributors. (2018, October 8). Vanishing gradient problem. In *Wikipedia, The Free Encyclopedia*. Retrieved October 15, 2018, from https://en.wikipedia.org/w/index.php?title=Vanishing_gradient_problem&oldid=863125122

[23] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

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

[25] Lin, T. Y., Dollár, P., Girshick, R. B., He, K., Hariharan, B., & Belongie, S. J. (2017, July). Feature Pyramid Networks for Object Detection. In *CVPR* (Vol. 1, No. 2, p. 4).

[26] Bengal cats. (January 6, 2018). Bengal Cat Coat: Colors and Patterns. Retrieved from https://www.bengalcats.co/bengal-cat-colors-patterns/?log-gedout=true .

[27] Catster. (September 12, 2018). The Ragdoll Cat — All About This Fascinating Cat Breed. Retrieved from https://www.catster.com/cats-101/about-the-ragdoll-cat .

[28] Pranav, Dar. (2018). Top 10 Pretrained Models to get you Started with Deep Learning. Retrieved from https://www.analyticsvidhya.com/blog/2018/07/top-10-pretrained-models-get-started-deep-learning-part-1-computer-vision/

[29] Mitul, Makadia. (2018). Top 8 Deep Learning Frameworks. Retrieved from https://dzone.com/articles/8-best-deep-learning-frameworks

[30] Mobile phone users worldwide 2015-2020. Retrieved at October, 2018 from: https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/

[31] Natasha, Lomas. (2016). Gartner: Android's smartphone marketshare hit 86.2% in Q2. Retrieved from https://techcrunch.com/2016/08/18/gartner-androids-smartphone-marketshare-hit-86-2-in-q2/