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

Xiaolu Zhang
University of Melbourne, Australia
tinalulu1327@gmail.com

Luyang Yang
University of Melbourne, Australia
838595401@qq.com

Richard Sinnott
University of Melbourne, Australia
rsinnott@unimelb.edu.au

***Abstract*—Deep learning is one of the latest technologies in computer science. It allows using 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. In this paper we benchmark different deep learning models and present an Android application used to predict the location and breed of a given cat using a mobile phone camera. The average accuracy of the finalized model was 81.74%.**

Keywords—Deep Learning, Cat recognition, Android

I. Introduction

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 provides a way to achieve artificial intelligence 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 manual input.

In this paper we develop a mobile application applying deep learning technologies to tackle image recognition problems. Specifically, we show how it can be used for cat detection in complex images and cat breed recognition. This is based on approaches for object detection and classification. Object classification is used to identify the category of a given image. For this project, the goal was to implement an object detection algorithm to recognize cats in an image taken by a mobile phone. 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 many different object detection models, such as Faster R-CNN, Single shot detectors (SSD), YOLOv3 and more [1]. These models have different ways to support the detection process, and the differences between how these models working impacts on the performance. Based on a comparison of Faster R-CNN, SSD, and YOLOv3 model performance and accuracy [1], it was identified that Faster R-CNN has better performance with regards to accuracy, however SSD and YOLO have the best overall performance.

To support deep learning, a suitable software framework is necessary. There are a lot of popular deep learning frameworks available such as Caffe, Tensorflow and more [2]. 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. Therefore, for this project, we select Tensorflow as the framework to train our object detection model and the basis for the mobile application.

Furthermore, since about 86% of smartphones are based on the Android system, we develop an Android application for our cat detection and breed recognition system.

II. Data

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, as is typical for many deep learning efforts, there is no dataset available for cat breed detection which could be used directly. Therefore, we had to create one.

According to The Cat Fanciers' Association (CFA), 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, we focus on 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 and between small but high-quality datasets. It can take a long time to make a high quality dataset since it requires manual labelling and highlighting areas of cats in each image. There are also only minor differences between some breeds listed in CFA as they often have the same heritage.

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 [3]. However, we were not able to just use Oxford Dataset directly. Firstly, the number of images in the Oxford Dataset is insufficient (on average there are about 200 pictures per breed). Secondly, the bounding box information is typically not given. For the Oxford Dataset, only the head area of a cat is recorded. It also does not utilize other important information such as the body parts and fur patterns. This causes issues with some breeds especially, e.g. it is more difficult to detect the Manx cat since Manx cats have no tail.

The second source was ImageNet [4]. 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 can be used for object detection. We chose five subnets of ImageNet to enrich our dataset, 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 sheer numbers of images, it was not possible to absolutely confirm that all errors were removed, but human checks are currently required for training data quality.

The last two sources used were Flickr and Google Image which have APIs. 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 set, e.g. resolution limit. Following this, we used a tool (labelImg) [5] to create label and bounding box information. The main difficulty we met was 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. Manual correction was thus required.

In total 12,451 images were obtained. However, the distribution of each breed was uneven. The structure of the dataset included bounding box information stored in xml files and images listed in an 'images' folder. 'pets_labels_list.txt' recorded the 14 labels (cat breeds) used in the project. These were used to create the label matrix during the training phase. We split the dataset into a training set and a validation set in the ratio 7:3. It is normal to have an extra test set to test whether object detection models work well. However, we abandoned the test set for several reasons. Firstly, a test set is normally given at the end to avoid tuning parameters dependent on the test set. Secondly, as we develop a mobile app, we have no idea what kind of images users will actually use. Therefore, a test set is not useful to test the accuracy of the actual usage. Also, the more images in the validation set, the more robust the model.

Even though there were 12,451 images from 14 breeds of cat in the dataset, this is not enough. Therefore, data augmentation was used to produce more images. This is an effective way to expand a dataset by using transformations and deformations on images without changing their labels, resulting in additional data [6]. In general, data augmentation can involve many methods, such as cropping, rotation, translation, scaling, and mirroring. Apart from enlarging the dataset, rotation is especially 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. Through rotation, convolutional neural network models can recognize a cat better.

Normally, resizing is necessary in data pre-processing for object classification. However, it is not so important in object detection since bounding box information is already recorded in the xml files. Our object detection model extracts bounding box information first and then learns features within the rectangle. However, we still use the resize method since the size of image inputs is fixed in our detection model, hence it is better to perform resizing operations before the training phase to increase the overall performance.

Shuffling used to rearrange images randomly. This is useful for machine learning as it increases the robustness of models. 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 realistic. The method is critical to create Tensorflow intermediate files generated from raw dataset files that are used to train the detection model.

It is normal for people to adjust the color and contrast of images. 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. However, it is difficult for computers to detect a cat since fur color is typically an important aspect of detection. To improve the robustness of our model, mean normalization is used to overcome the exact color of images, i.e. absolute RGB values.

III. Model Training

Transfer learning is an effective way to make use of knowledge learned from previous datasets to enhance a new model performance within the same domain [7]. According to Torrey and Shavlik [8], there are three possible benefits when using transfer learning: with transfer learning, we can have a high start, scope and asymptote, which shortens the training time and improves the outcome performance. There are also many pre-trained models available on popular machine learning platforms, such as Caffe and Tensorflow. As features of pre-trained detection models can be generalised, e.g. cat and dog, these models are helpful to initialize a new model. By re-training a pre-trained model, the training process can be significantly shortened.

The pre-trained models we explored in the project are based on the COCO dataset which includes 80 object categories (including 'cat' category) and more than 330K images [9]. This helps augment the limited number of images in our dataset.

Overfitting is a further problem whereby more terms and complex methods are used than is necessary [10]. This happens when the accuracy of the training set increases and hence the variability decreases and even if more training time is spent, the accuracy of the validation set does not improve. The reason for overfitting is that the features gained from the training set are much more than the ones required in prediction. In other words, the object detection model focuses on smaller 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. We also use regularization techniques to prevent overfitting in our object detection model.

In recent years, there have been many successful object detection solutions proposed [11]. In general, an object detection model includes a meta-architecture and object feature extractors. For example, Fast R-CNN uses SPPnet as a feature extractor [12]. One key aspect 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. There are many meta-architectures that have been used in object detection, e.g. 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 model like YOLO. The other reason is that both two meta-architectures are able to work with different feature extractors.

SSD and Faster R-CNN use different ways to detect objects. For example, Faster R-CNN uses a Proposal Generator before classification. However, both of them can use the same feature extractors such as VGG and Inception. Faster R-CNN has two modules. The first one is a deep, full convolutional neural network which proposes regions (region proposal network (RPN)), and the second one is a Fast R-CNN detector that uses the proposed regions to guide the detector where to look [13]. 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 [15]. SSD can use the VGG19 network as a feature extractor. It adds custom convolution layers and convolution filters to make predictions [16].

We explored several popular and high performing feature extractors in this work. Resnet (Residual Neural Network) is one architecture of CNN, which supports development of deeper networks [17]. Specifically, it solves the problem of vanishing gradients [18]. 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 back to previous layers, thus making it possible to build more layers in the net.

Whilst Restnet is about being deeper, Inception is about being wider. The key part of the Inception architecture is that it has a new Inception module block. The inception module computes multiple transformations from the same input map in parallel and concatenates results into a single output. The model decides which piece of the information it should use [19]. It uses a 1×1 convolution 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, resulting in deeper and wider nets [19].

Mobilenet uses depth-wise separable convolutions to build a lightweight deep neural network [20]. A depth-wise separable convolution has two layers: depth-wise and pointwise convolutions [20]. A single filter is used for each input channel in the depth-wise layer. Pointwise convolution using a simple 1×1 convolution is used to create a linear combination of the output of the depth-wise layer [20].

Feature Pyramid Network (FPN) is a top-down architecture with lateral connections used to build high-level semantic feature maps at all scales [21]. By replacing image pyramid features with in-network feature pyramids, there is no performance loss in the power, speed, or memory [21]. Moreover, it achieves improved single-model results on the COCO detection benchmark when used with a Faster R-CNN architecture [21].

By combining the meta-architectures and feature extractors mentioned above, we explore six object detection models as the basis for the detection model options, namely: Faster R-CNN Inception_v2; Faster R-CNN Resnet50; SSD Inception_v2; SSD Mobilenet_v1; SSD Resnet50 FPN and SSD Mobilenet_v1 FPN. The following parts illustrate how these were evaluated and the results of the evaluation.

Since our project focuses on building a mobile app to detect cat breeds, there were two important aspects to consider: speed and accuracy. It is easy to understand that accuracy is a concern since we need to have a high degree of confidence that the app can detect and recognize different cats correctly. However, if it takes a long time to detect a cat, users will lose patience and probably not use the app again. Therefore, a trade-off must be made between accuracy and the amount of time for detection.

During model evaluation, we use pre-trained models provided by Tensorflow detection model zoo as a start point for transfer learning. Using parameters set in the default configure file of each model, we re-trained models using our dataset. Default values of the parameter "num_steps" (number of training steps) of the configure files varied from 25,000-200,000. We chose 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 with the dataset.

The mAP was calculated with an evaluation script offered by Tensorflow that 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, where IoU is the intersection over union, and used as a metric to measure the accuracy of a detector. 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, hence it takes more time for Faster R-CNN to detect objects. Generally, Faster R-CNN can achieve a higher prediction accuracy than SSD. However, with new feature extractors, such as Mobilenet and FPN, the accuracy of compound SSD models can be even higher than Faster R-CNN.

Both detection models using the FPN feature extractor to achieve a better performance than other models. The accuracy of SSD Resnet50 FPN was 75.7% and the average detection time was about 4.36s. SSD Mobilenet_v1 FPN achieves 72.1% accuracy with an average detection time of 2.28s. Even though the accuracy of SSD Resnet50 FPN was slightly higher than SSD Mobilenet_v1 FPN, taking detection time into consideration, we chose SSD Mobilenet_v1 FPN as our final object detection model.

IV. Model Optimization

Even though the accuracy of the chosen model was acceptable, it still required optimization, since the accuracy we have comes from using a fine-tuned configuration file in Tensorflow, i.e. there is no specific optimization operators based on our dataset. By tuning some parameters in the configuration 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 the chosen detection model from two considerations: the dataset and configuration optimization. Normally dataset optimization should not be a method to optimize a detection model as the dataset is generally given and fixed. However, our dataset has a drawback due to the nature of cat breeds. Therefore, by improving the quality of the dataset, the object detector will also be improved. However, given the nature of cat breeds, there are two main issues which impact on cat breed recognition. Firstly, no matter whether pedigree or non-pedigree cats, they can have a variety of colours and patterns [22]. Secondly, it is hard to use features extracted from kittens to detect their breed as their appearance is often different than their adulthood appearance. Based on this, 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 enough images to ensure each appearance of a cat breed has sufficient images.

A. Methodology

We optimized our dataset by improving the "quality" of images in the dataset. In this context, "quality" means how well the images 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 [22]. 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 the 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 to other breeds, to achieve a higher accuracy. Fig. 1 shows the result of mAP@0.5IoU for each cat breed before optimizing the 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 as shown in Table 1. The overall mAP@0.5IoU subsequently increases from 72.06% to 74.98%.

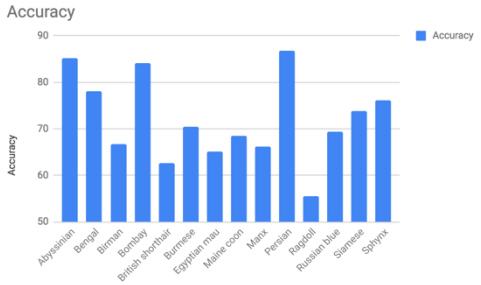

Fig. 1. Result of mAP@0.5IoU for each cat breed before optimization

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. Accuracy before and after dataset optimization

B. Hyperparameter Optimization

As deep learning methods learn features autonomously, it is important to configure a model 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 impact on the performance of a model including the learning rate, batch size and regularization. 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. 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 tunings. If the batch size is too small, the model has a probability that it will not converge within the expected steps.

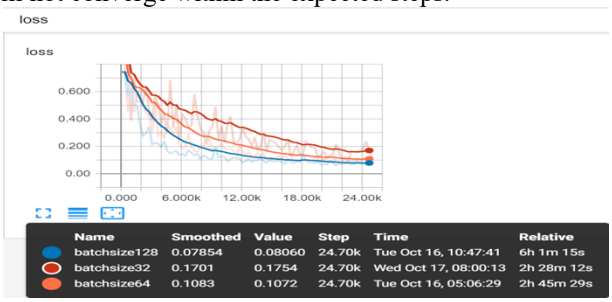

Fig. 2. Loss curve changes with different batch size

From Fig. 2, 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.

V. Results and Analysis

In this part, we analyse the overall performance of our detection model. We present the average prediction accuracy of each breed and the performance of our model with more complex images (e.g. images with many objects).

A. Prediction accuracy of each breed

Fig. 3 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.

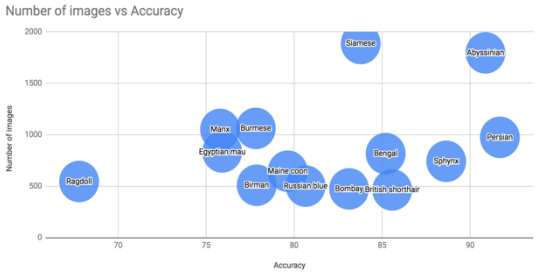

Fig. 3. The prediction accuracy of each breed

As seen, there are three breeds where the accuracy achieves a confidence level around 90% (Persian (91.69%),

Abyssinian (90.87%) and Sphynx (88.64%)). Each of these cats has unique characteristics. A Persian cat has a fluffy fur and snub nose. An Abyssinian cat has 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, where 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 [23], which means it is harder to extract typical features as with other cats. 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.

B. Complex Scenarios

We also consider more complex scenarios: one cat, more than one cat, many objects excluding cats and parts of cats visible in an image. We manually collect images from Google based on these scenarios to test the performance of the object detector. Even though manual selection may introduce bias due to the manual selection, it is good to give a taste of how the detector works in different situations.

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%, i.e. the detector does a good job when detecting many objects with clear boundaries around objects. SSD works well when detecting large size objects rather than small objects.

VI. Android Application Implementation

As mentioned, this application is aiming to develop a convenient mobile version of a cat recognition model. The model we selected for our project was based on SSD Mobilenet_v1 FPN. There are two ways to realise this: develop a client server model application or build an independent application on the phone to run the model. 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 period. However, there are also some limitations with the client server model. Firstly, the application and the server need to communicate which can take time. That means that when 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.

An 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 can 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. 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. We use a mobile version of Tensorflow called Tensorflow lite for the mobile app.

A key aim is for the mobile application to be user friendly, hence the user interface is an important component for any application. The mobile application uses the camera of the mobile phone to capture real-time images and invoke the trained model to make predictions on 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 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. 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 introduced, the app was required to support object location and classification. This makes the deep learning model more complex than just a classification process. This means the model needs to take more time to make the prediction of the image. Based on our testing, the classification process just needs to take 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 location identification. We also designed an independent interface which is used to show all types of cat which can predicted by the app. This interface shows information about each type of cat which can be identified and some information about the overall performance.

VII. Conclusions and Future Work

This project focused on developing a cat detection model by deep learning and delivery through a mobile application. The app has been trained to recognize 14 types of cat with an average accuracy of 81.74%. In undertaking this, the work compared six popular models with the final choice of SSD Mobilenet_v1 FPN. The model training tool was Tensorflow, and the platform ultimately delivered on the Android platform. To support the work, we required many images from different data sources and performed various processing on these images. Because data quality and quantity were both very important for deep learning model performance, data collection and preprocessing were essential. In addition, we tested the different models and

collected their performance data. Based on the consideration of time, computation power needs, accuracy and complexity of implementation on Android phones, we selected SSD Mobilenet_v1 FPN as the model for the app.

We used the collected dataset as the primary source to train model and achieved an 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 achieve 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.

There are also various ways this project could be improved. It is possible to enhance the existing model performance as well as developing other models. One way is through hardware and the other is model improvements. For hardware, running the model on a device with more computation power such as a GPU server would bring some improvements on the time. However, it would increase the project cost. For model improvements, the deep learning model performs object detection by analyzing images 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. We developed a mobile interface used to support the image classification process which is based on an open source model trained by others to do the classification process. It is used to classify what is in the image, not for the cat breed classification. Future work could also develop a new model that is used to do the classification process with Tensorflow and replacing the existing classification model with a new one.

The source code for this mobile application and associated model implementation is available at: https://github.com/tinalulu1327/Cat_Recognition.git.

REFERENCES

[1] 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/

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

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

[4] ImageNet Datasets Available at: http://www.image-net.org/ (Accessed: 8 October, 2018).

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

[6] 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

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

[8] 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.

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

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

[11] 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).

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

[13] 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).

[14] 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

[15] 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.

[16] 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

[17] 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).

[18] 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

[19] 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).

[20] 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.

[21] 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).

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

[23] 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 .