NTHU animal classifier: an application to help user identify animals in NTHU

Ke, Yi-Syuan
department of computer science
Tsin Hua University
Hsin Chu, Taiwan
ryanke91@gmail.com

Tsai, Chih-Tai
department of computer science
Tsin Hua University
Hsin Chu, Taiwan
erictsai90@gmail.com

Chen, JiaHui
department of computer science
Tsin Hua University
Hsin Chu, Taiwan
1741954267@qq.com

Huang,Sheng-Yang
department of computer science
Tsin Hua University
Hsin Chu, Taiwan
frog910219@gmail.com

Chen, Yu-Hsun
department of computer science
Tsin Hua University
Hsin Chu, Taiwan
yuxun90140@gmail.com

Yeh, You-Xin
department of computer science
Tsin Hua University
Hsin Chu, Taiwan
t555555561@gmail.com

***Abstract*—Inside National Tsin Hua University, there are various kinds of animals including dogs, cats, turtles and other creatures. Not to say there are actually fourteen dogs lives in the campus currently, each with different names. Distinguishing them is a quite challenging task since some of the dogs just look the same.**

Unfortunately, there exists no tools to help us identifying them. Most of the animals are to be known as "the black dog" or "the yellow dog". Therefore, we want to utilize the knowledge learned throughout the machine learning course to build an application that can help people distinguish the animals easily and correctly.

We present NTHU animal classifier, an app that can generates the animals name by simply uploading picture of them on it. It can distinguish 11 animals including dogs, cats, birds and squirrels. Moreover, the application is currently available on android mobile platform.

Keywords—image recognition, app, machine learning, transfer learning, convolution neural network

I. INTRODUCTION

Inside National Tsin Hua University, there are various kinds of animals including dogs, cats, turtles and other creatures. Not to say there are actually fourteen dogs lives in the campus currently, each with different names. Distinguishing them is a quite challenging task since some of the dogs just look the same.

These animals, especially dogs, play a significant role inside the campus. They are great helpers when it comes to stress relief. Unfortunately, most of the students do not know their names as some dogs look just the same. We believe it is crucial to let people know more about them.

We present NTHU animal classifier, an app that can generates the animals name by simply uploading pictures of them on it. Inside this project, we use transfer-learning with the model MobilenetV2 to do the image recognition. The model is built by TensorFlow and later put onto android studio, where we use Kotlin to build our app.

We've spent a lot of time choosing the model we want to transfer. In the latter chapters, we'll show the comparison of different models like MobilenetV2, ResNet50 in number of parameters, accuracy, and other details. Plus, why we finally chose MobilenetV2 as our model. Connecting the model to Android is also not a straightforward task, we need to use the Tensorflow lite transformer to generate the lite file. We'll explain this later, too.

The data is collected by taking photos inside the campus by ourselves and finding pictures on the internet, such as the fan page of the NTHU-dogs club. We struggle to get a sufficient amount of data because it's not easy to find the specific animals inside the campus or on the internet. In the support of data preprocessing and augmentation. We'll show the evaluation result that our model actually does a quite accurate classification among the animals.

To evaluate the model, we randomly select three uncollected pictures on the internet to see the classification result. The chart is shown in the latter chapter. All source code of the NTHU animal classifier is at https://github.com/frog0219/ml-final.

II. METHODOLOGY

Designing a model isn't a simple task, we've tried different models and training methods. In this chapter we'll introduce the challenges and the final decision in each aspects.

A. Overall Structure

We use Tensflow to build the model in python file, puts the model into Android Studio, where we use Kotlin to construct the app.

B. Data Collection and Preprocess

In total, we have 11 classes with 9 dogs, 1 cat, 1 squirrel and 1 bird. There are 1500 images evenly distributed in each class overall. The pictures is mostly collected manually by ourselves. The second source is websites like NTHU-dog Club and Other animal sites. The validation rate is 0.8, therefore we'll have 1214 images for training and 296 images for validating the model.

As for data preprocessing, we didn't process the image in the beginning. However, the validation accuracy reached only 0.4 to 0.5. After trying different models, the accuracy was still not ideal. Moreover, when we evaluated the model, we found out that the prediction was significantly influenced

by the background. So, we decided to preprocess the images, manually cutting out most of the background as illustrated in figure 1.

We also do the data augmentation to enhance our data. The images are later rescaled to 224*224. Below is the parameters for the data augmentation.

Horizontal_flip = True

width_shfit_range = 0.1

height_shfit_range = 0.1

C. Model Comparison

Choosing a reasonable model is extremely important for transfer-learning. We've tried various kinds of models. Since we're applying our model onto mobile devices, an important requirement is that the model cannot be too heavy. This includes diverse aspects like the model size, use of GPU, model computation speed, memory cost, etc. At first, we aimed to use MobilenetV3, a lightweight convolutional neural network architecture designed for efficient on-device image classification and object detection. It is well suited on resource-constrained devices such as smartphones and embedded devices. However, we only got 0.4 accuracy, although we had added extra fully-connected layer, the accuracy couldn't got over 0.7.

To verify whether the problem came from model implementation or lack of well input images. We've tried another well-known model: Resnet50. To our surprise, without doing any adjustment, the accuracy could reach 0.9. This brought us good news and bad news. The great news is that the input data and the model is workable. However, the bad news is, for mobile devices, the parameter amount of Resnet50 is too large. If we put this model into our mobile phone, we may encounter poor computation efficiency, insufficient memory space and other bottlenecks.

Later, we found MobilenetV2, the previous version of Mobilenet, had a really great performance on prediction accuracy. It reached 0.8 at first. After adding fully-connected layer, dropout layer and batch layer, we got 0.9. Eventually, by finetuning the weight, we had overall 0.95 validation accuracy. Mobilenet V2 thus became our final decision. Figure 2 shows the final structure of our transferred MobilenetV2 model.

Excitation block" (SE block). The technique is based on the idea that not all features in an image are equally important for classification. SE uses a mechanism called "squeeze" to create a global representation of the feature maps in a CNN, and an "excitation" mechanism to adaptively recalibrate the feature maps based on this global representation. However, our input images has a lot of noise, as a result, the focus of the model might be misdirected by them. Moreover, after researching on some papers and report. Some say that MobilenetV3 performs worse in images with complicated background or lots of noise, which supports our speculation. However, the paper doesn't produce any practical experiment. We can't draw the conclusion.

The second speculation is about the layer connection of MobilenetV3, also called "skip connect" or "short cut", , refer to the direct connections between layers of a neural network that bypass one or more intermediate layers. In MobileNet V3, skip connections are used to help the network better preserve spatial information as it passes through the layers. This can help the network make more accurate predictions and improve its overall performance. Therefore, we could only finetune the latter layers. The consequence is that the weight of the former layer can't correctly present the features and the effect of finetuning is less significant. We've also discussed this issue with Prof. Kuo. He affirmed the possibility of this speculation.

Another insight is that the structure of MobilenetV2 learns from Resnet50. This may be the reason of them having similar result.

As for training speed, to our surprise, we found that bothMobilenetV2 and MobilenetV3 computed only faster than Resnet50 for 1 to 2 minutes in training 50 epochs. After we read the paper of them, we observed that Mobilenet uses "Depthwise seperable convolution", which turns the parallel computing into sequential computing. Under GPU environment, there would be no significant differences in training performance because it lacks parallelism. However, when it comes to mobile device, which is a CPU environment. The speedup will be outstanding. Therefore, Mobilenet benefits edge-devices. In table I, we show the comparison between MobilenetV2 and Resnet50. We can see the parameter amount of MobilenetV2 is ten times less than Resnet50, the Size is seven times less and the speed is 1.7 times faster.

D. Training details

The training details is shown below.

Optimizer: Adam

Loss: CategoricalCrossentropy

Epoch: 50

Batch size: 32

We also use tf.keras.callbacks.ModelCheckpoint() to store the weight at every epoch then choose the best model. The code is shown below.

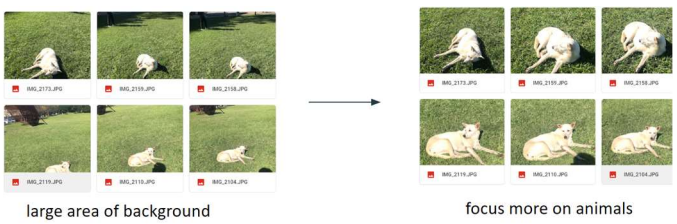

Figure 1: Before and after of the input images. The images were made more focus on the animals.

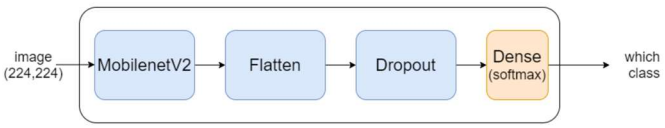

Figure 2: the structure of the transferred model.

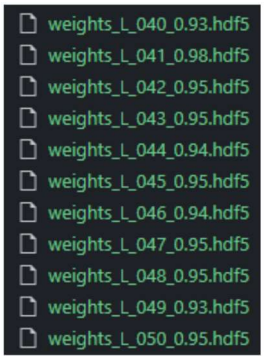

Figure 3: The files containing weight at every epoch, 50 in total.

```
My_callback = [tf.keras.callbacks.ModelCheckpoint(
        Filepath='weights_save'/weights_L_{epoch:03d}_
        {val_accuracy:.2f}.hdf5,
        Monitor= 'val_accuracy',
        Save_wights_only=True,
        Mode='max'
)]
```

E. *Connecting to Android*

Connecting to Android is not a straight-forward task. We cannot directly put our Tensorflow model onto Android because the model would be too large and heavy. Instead, we use TensorFlow Lite, a light-weight mobile library for deploying models on mobile and other edge devices. Figure 4 shows the process to generate Tensorflow lite file. We can use Tensorflow lite converter, which is offered by Tensorflow to output the light-weighted version of our trained model, then put it into Android Studio.

Another challenge is, rather than Python, there's no libraries supporting the model to directly take png or jpg files as input. Consequently, we have to transfer the image into pixel array manually then put it into the model. Figure 5 shows the code to accomplish this task.

III. DEMONSTRATION

We'll show the app flow and its function in this section. Figure 6 is the app flow of our product. The leftmost figure is the first page of our app. When press the magnifier icon, user can take photo instantly, then the photo taken will be the prediction input. The shopping cart button on the right means to select photos from the album. After choosing photos, the model will give us the result immediately as shown in the middle figure. The result screen shows the dog's name and the image is the picture that we just uploaded. There's also a magnifier button next to the dog's name, which can give users more details about the dog including characteristic, found place and traits. Shown in the rightmost picture.

IV. RESULT

In this section, we will present the learning results of MobilenetV2, MobilenetV3 and Resnet50 in two perspectives: efficiency and feasibility.

```
val intValues = IntArray(imageSize * imageSize)
image.getPixels(intValues, 0, image.width, 0, 0, image.width, image.height)
var pixel = 0
//iterate over each pixel and extract R, G, and B values. Add those values i
for (i in 0 until imageSize) {
    for (j in 0 until imageSize) {
        val `val` = intValues[pixel++] // RGB
        byteBuffer.putFloat((`val` shr 16 and 0xFF) / (255f))
        byteBuffer.putFloat((`val` shr 8 and 0xFF) / (255f))
        byteBuffer.putFloat((`val` and 0xFF) / (255f))
    }
}
```

Figure 5: The codes to transfer image into pixel array

A. *Efficiency*

First, for efficiency, we recorded the model training time, cost of memory space and number of parameters(Table 1). The reason why we use efficiency to evaluate the three models is that these models will eventually load into the mobile devices, which are very light weight and lack of sufficient memories or computing resources compared with computers. In other words, if we don't consider the limitation of mobile devices, our model may work badly or even can't work on mobile devices due to, memory overflow ,etc.

B. *Feasibility*

For the other perspective, feasibility. We recorded accuracy curve, loss curve and the final test results of these three models (Figure7, Figure 8, Figure9). In order to show the credibility of our models, we use photos which were chosen randomly on the internet and are not in our datasets. The reason why we also use feasibility as an evaluation of our models rather than just only use efficiency is that we want our models to keep high accuracy in any prediction on mobile devices (accuracy higher than 0.9, specifically). After all, fast but incorrect predictions are meaningless. Figure10, Figure 11, Figure12 show one of the prediction of these three models

Because the images in our dataset have different ratio of background and dogs (cause by the different photographing distances), we want to know how the ratio of background and dogs will influence the model. We recorded the learning results of MobilenetV2 before(Figure 13, Figure 14) and after(Figure 15, Figure 16) cutting the background smaller. Here is how we cut the images: we cut

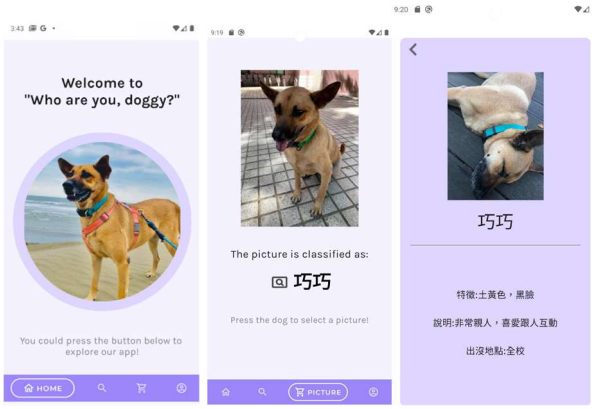

Figure 6: The app flow. Leftmost one is the first page, middle one is the prediction result, rightmost one is the details of the dog.

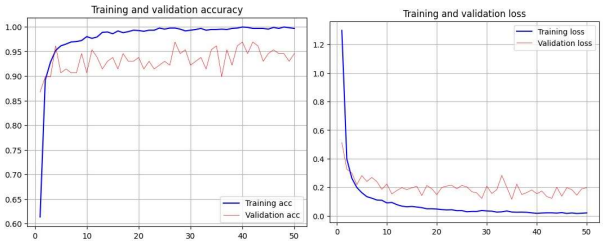

Figure7: Accuracy curve(left) and loss curve(right) of MobilenetV2

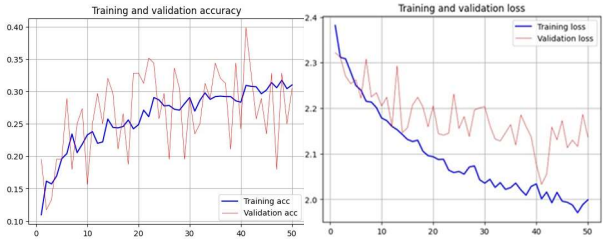

Figure8: Accuracy curve(left) and loss curve(right) of MobilenetV3

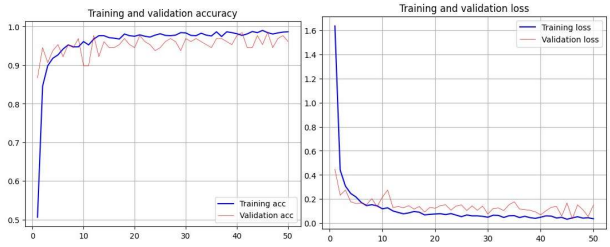

Figure9: Accuracy curve(left) and loss curve(right) of Resnet

the images that contains dog body less than one-third of the image area, so the dog body have at least one-third of the image area for every image in our second datasets. Note that we removed a few photos which the dog body area is too small, otherwise the resolution will be too low after cutting off the background.

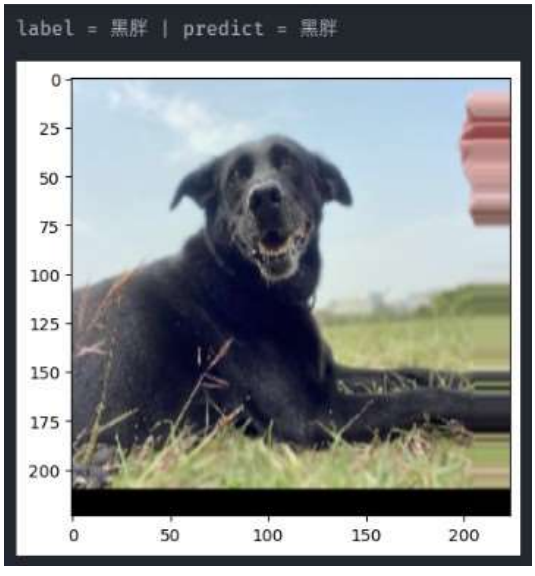

Figure10: Predicion of MobilenetV2

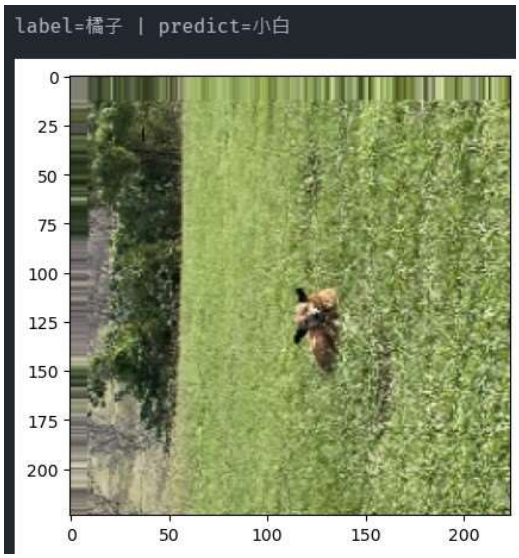

Figure11: Predicion of MobilenetV3

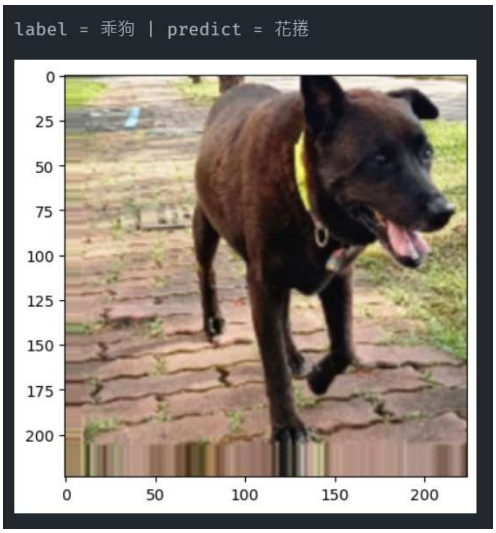

Figure12: Predicion of Resnet

	MobilenetV2	MobilenetV3	Resnet
Memory space	14MB		98MB
Learning time	33m30s	32m19s	35m32s
Parameters number	2,257,984	2,996,352	23,587,712
Accuracy	0.95	0.34	0.96

Table 1: Comparison between MobilenetV2, MobilenetV3 and Resnet

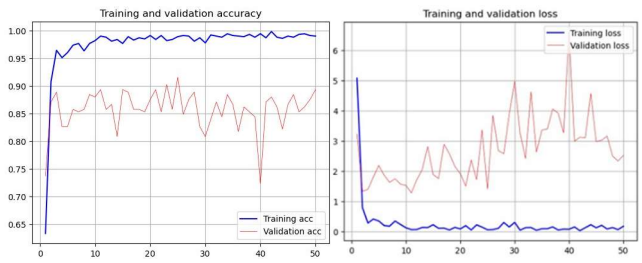

Figure13: Accuracy curve(left) and loss curve(right) of MobilenetV2 – before cutting

```
Train loss: 0.0003599768679123372
Train accuracy: 1.0
Validation loss: 2.8640975952148438
Validation accuracy: 0.8883928656578064
```

Figure 14: Final accuracy and loss of MobilenetV2 – before cutting

V. DISCUSSION

In this section, we will do some discussion based on the results that we have shown above. Same as the previous part, we will discuss in two perspectives: efficiency and feasibility.

A. *Efficiency*

First, for the efficiency, we can see that MobilenetV2 use the least memory space among the three models, use only one-seventh of memory and about one–tenth of number of parameters of Resnet, but its accuracy is 0.1 lower than Resnet-(Table 1). Intuitively, Resnet use more parameters so its accuracy curve is more stable and it has a higher accuracy. Although Resnet is more accurate than MobilenetV2, MobilenetV2 is obviously much more efficient than Resnet. So we chose MobilenetV2 considering the restriction of mobile devices.

Now the question is why MobilenetV2 can achieve that high accuracy just use not much parameters? The answer is behind the structure of the model. The two most important techniques to reduce the number of parameters are "Depthwise Separable Convolution" and "Inverted Residual With Linear Bottleneck".

1) Depthwise Separable Convolution

The meaning of depthwise separable convolution is, like its name, separate convolution neural network into two part: depthwise convolution and pointwise convolution, compute them respectively and combine them to build the model. By doing so, we can reduce the computatation. Figure17 shows the result of using depthwise Separable Convolution.

2) Inverted Residual With Linear Bottleneck

The main concept of the other technique "inverted residual with linear bottleneck" is like dimension reduction. The layers receive compressed representations, increase them to high-dimension representations, and then do convolution using these high-dimension representation. After finishing convolution, the layers compress the high representations to low dimension representation again, and then pass it to next layers. By doing so, we can reach the goal: transfer informations in low dimensions, extract features in high dimensions.

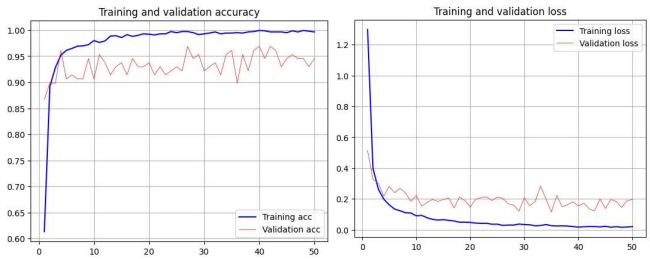

Figure15: Accuracy curve(left) and loss curve(right) of MobilenetV2 – after cutting

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Figure17: Experiment result of using depth separable convoluton

B. *Feasibility*

For the feasibility, Table 1 show that MobilenetV3 only have the accuracy 0.34, which is like randomly guessing. For the reason of its bad performance, we observe that its model may focus on background too much, which is not the target we want to predict. According to the above, we choose MobilenetV2 to be our model.

To know the influence of the ration between background and dog body, we cut the images and use it to train the model again, the accuracy increases by 0.06. We observe that MobilenetV2 also have the issue that focus on background too much, but the affection is not that much compare with Mobile-netV3. The reason why we didn't cut all the background off is that we think the background is also features of the images, because different dogs indeed have different habitats. In other words, the background can also help the model to do predictions.

VI. CONCLUSION

The most important requirement for our model is about the limitation of memory space and computing resource of mobile devices. After analyzing the pros and cons of these three pre-trained models by doing many expiriments evaluting their performances, we chose Mobi-lenetV2 to be our final model.

Our NTHU animal classifier can offer accurate prediction on animals in NTHU. We believe this app is able to support people to know more about the creatures in the campus. Consequently, when we are more familiar with them, we can also show our kindness to those lovely animals. We hope to give the whole campus a nicer and more friendly environment.

VII. AUTHOR CONTRIBUTION

Below is the division of work, we think we split the contribution evenly.

Chen, Yu-Hsun: data collection, building models

Chen, JiaHui: data collection, building models

Tsai, Chih-Tai: data collection, building apps

Huang,Sheng-Yang: data collection, building apps

Ke, Yi-Syuan: data collection, presentation and report

Yeh, You-Xin: data collection, presentation and report

```
Train loss: 0.03247630596160889
Train accuracy: 0.9983108043670654
Validation loss: 0.6305931806564331
Validation accuracy: 0.9513888955116272
```

Figure16: Final accuracy and loss of MobilenetV2 – after cutting

VIII. REFERENCE

[1] Siying Qian; Chenran Ning; Yuepeng Hu, “MobileNetV3 for Image Classification”.

[2] Andrew Howard; Mark Sandler; Bo Chen; Weijun Wang; Liang-Chieh Chen; Mingxing Tan; Grace Chu; Vijay Vasud, “Searching for MobileNetV3”.

[3] Muhammed Mutlu Yapıcı; Adem Tekerek; Nurettin Topaloğlu, “Performance Comparison of Convolutional Neural Network Models on GPU”.

[4] Ke Dong; Chengjie Zhou; Yihan Ruan; Yuzhi Li, “MobileNetV2 Model for Image Classification”.

[5] https://zhuanlan.zhihu.com/p/80177088?utm_id=0

[6] https://www.cnblogs.com/hepeiqi/p/15965657.html#/c/subject/p/15965657.html

[7] https://www.facebook.com/NthuCarelife

Yeh, You-Xin: data collection, PTT & report

Ke, Yi-Syuan: data collection, PTT & report