Photo Source Classification

1. Abstract

Machine learning offers a great variety of algorithms that can be applied for classification. The main objective of our paper is to use the methods to correctly identify and categorize which phone one photo was taken by. So this paper deals with the Convolutional Neural Network for identifying the source of the image. The main metric used in our study to evaluate the performance of our model is the accuracy, and the best result we finally obtained is 82.23%. The results showed that our model has an effect in photo source classification.

2. Introduction

Nowadays, people tend to take photos with their phone to record their life, and share them on social media. Since different phone companies have their own way in designing cameras, photos may come out with different effects when they're taken by different phones. Due to the popularity and the diversity of photos, we want to help those people who care about the effect of their photos identify the brand of the phone used to take a photo, when they see a photo with some effect they desired. This way they can take it as reference when they're buying a new phone.

Thus, we will train a model that can classify the brand of a photo input, and output the name of the brand. In this project, to gain a better quality of classification under limited time and relatively less resources, we will train a binary classifier, and limit the testing data and output of our model to the two most popular phone brands - Apple and Samsung.

3. Method

3.1 Scheme for Image Source Classification

The scheme for image source classification shown in the figure below. Given an image, I, we first separate it into n parts, with fixed size 224 by 224. After that, put these n small images into our machine learning model, f, obtain outputs. The model we train takes an p by p image as input, and output a brand, which indicates that image was taken by the phone of that brand. The details of the model would be described in section 3.3. Finally, apply the majority vote on those outputs, take the most frequent output as the final answer.

There are some hyperparameters that can be adjusted: the size of small images, the number of small images that split from an image, and the method to choose these small images from the image. And most importantly, should the image be resized or be compressed? This is another factor that can affect the efficiency of our model.

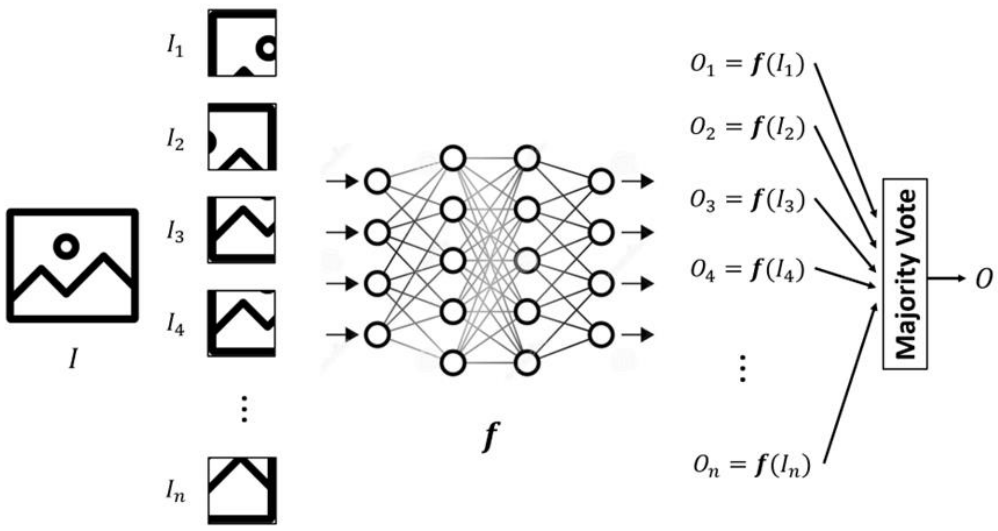


Figure 1. Scheme for Image Source Classification

3.2 Data Preprocessing

In this section, we proposed a method to obtain 224 by 224 images from the dataset to solve the problem that these images may have different sizes and the problem that data is not enough for the model to have a good convergence, and also reduce the number of parameters to train.

Here we have an assumption that a small image I_k and the original image I should have the same label (brand). Based on this assumption, we separate the original image, I, into n pieces, with the size of each small image being 224 by 224. Then we can increase the data size by floor (original image's height / 224) * floor (original image's width / 224) times.

Let r_h be the remainder of (the original image's height)/ 224, r_w be (the original image's width) / 224, if the original image's height or width isn't divisible by 224, we will not use the pixels whose x coordinate is between (width - r_w) and width, or pixels whose y coordinate is between (height - r_h) and height.

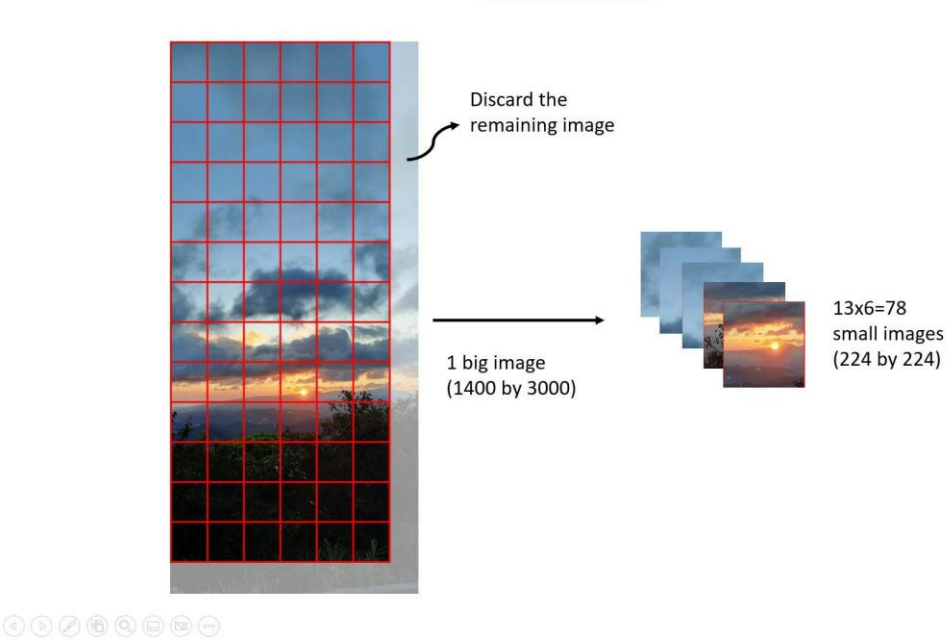


Figure 2. Split an image into 224 by 224 images.

We save the 224*224 images as new files and separate them into two sets: training data set and validation data set, where the size of the validation data set is 20% of the original set size. Next, we create and save the labels of these 224*224 images into two separated .npy files, one for training data and one for the validation data, where the first column is the file names, and the second column is the label(0 represents Apple and 1 represents Samsung).

3.3 Training Model

We choose the series of Convolutional Neural Networks, which was first proposed in the paper "Gradient-Based Learning Applied to Document Recognition" (Yann LeCun et al., 1990s) as our model. Taking 224 by 224 images as input, and outputs the label of the brand of phone, which 0 represents Apple and 1 represents Samsung. Since there are only two classes, we used the Sigmoid function to be the last activation layer.

We use Mini-Batch Gradient Descent to find the optimal model, using Adam as the optimizer (Adam: A Method for Stochastic Optimization. Diederik P. Kingma, Jimmy Ba. *arXiv:1412.6980*, 2014), in the progression. For loss function, we use Binary Cross Entropy, which is one of the best loss functions for binary classification problems.

We build our CNN model by using the PyTorch package in Python. In the model, we construct many layers to improve the performance, e.g. convolution layer, pooling layer, batch normalization layer, activation layer and dense layer. The architecture of our model is given in table A.

Table A. ConvFCNetv3 architecture

Layer	#filters	Output Size	Kernel Size	Activation
Conv2D	16	224x224x16	3x3	ReLU
BatchNorm2D				
Conv2D	32	224x224x32	3x3	ReLU
BatchNorm2D				
MaxPool2D		112x112x32		
Conv2D	64	56x56x64	5x5	ReLU
BatchNorm2D				
Conv2D	64	56x56x64	3x3	ReLU
BatchNorm2D				
Conv2D	128	28x28x128	5x5	ReLU
BatchNorm2D				
Conv2D	128	28x28x128	3x3	ReLU
BatchNorm2D				
MaxPool2D		14x14x128		
Conv2D	256	7x7x256	5x5	ReLU
BatchNorm2D				
Conv2D	256	4x4x256	5x5	ReLU
BatchNorm2D				
Conv2D	256	2x2x256	2x2	ReLU

BatchNorm2D		
MaxPool2D	1x1x128	
Dense	128	ReLU
Dense	32	ReLU
Dense	1	Sigmoid

3.4 Test Model

We test our model by two different methods. For the first method, we use the validation dataset with the same format of training dataset (224 by 224 images with labels) and the accuracy as an evaluation metric to test the performance of our model. For the second method, we use the test dataset with original size and test the performance by the way described in section 3.1, evaluating it by using an accuracy metric.

3.5 Classification

In this section, there are three main steps to make a prediction. After importing the model we have trained before and reading the target image, we first crop the image into small pieces with the size of each one being 224 by 224, using the way described in section 3.1.

Then we feed the preprocessed images to the model in units of 64. Since the last layer is the Sigmoid function, outputs are between 0 and 1. To obtain the correct results, we make the original results whose values are larger than or equal to 0.5 be 1, and otherwise be 0.

At last, we make a majority vote. Counting the amount of 0s and 1s in the list of results made by the model, we choose the number which has a relatively large amount as our final prediction.

4. Data Reference

In our project, we are going to use existing photos on our phone or more available data which are the pieces cut from completed images to be the input data (Apple: 31451 pieces, Samsung: 35360 pieces). Therefore, we would gain a classifier that is able to automatically categorize data. In this way, whenever applying a new photo as an input data, the classifier has the ability to distinguish what phone the photo was taken with. To put it another way, we would get the phone brand as output which might be iPhone or Samsung.

5. Results

Taking the validation dataset as our input, we gained an accuracy of 92.71% and the loss about 0.003 in the ConvFCNetv3 model. Moreover, taking the test dataset as input, we finally gained an accuracy of 82.23%. The figures below are the accuracy and loss curve of the validation dataset. (We start training with the model ConvFCNet. To improve the accuracy, we modified ConvFCNet to ConvFCNetv2, ConvFCNetv2 to ConvFCNetv3 by adding more layers and also transplanted the trained parameters to the new one. Finally, we use ConFCNetv3 as our final model.)

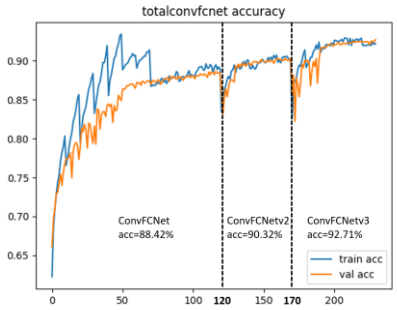


Figure 3. Accuracy curve.

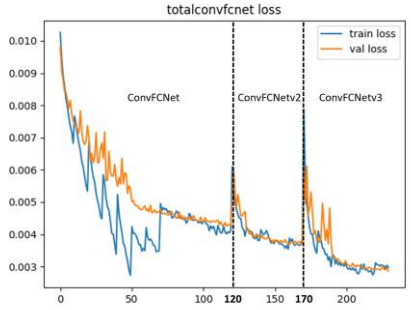
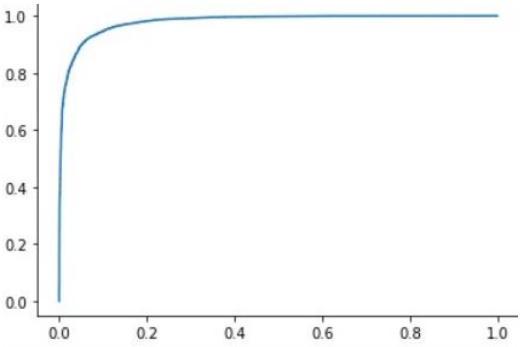


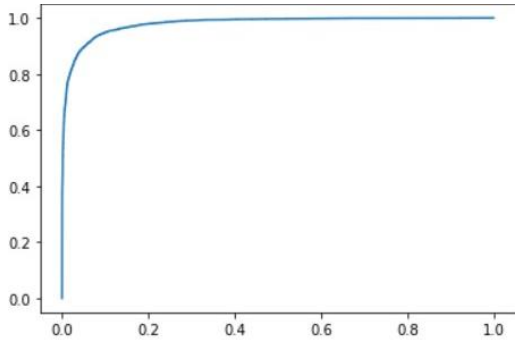
Figure 4. Loss curve.

The figures below are the ROC curves (receiver operator characteristic) of Apple and Samsung of the validation dataset, respectively.



Apple

Figure 5. Apple ROC curve.



Samsung

Figure 6. Samsung ROC curve.

6. Difficulties

Since we cut the images in small pieces to obtain more training data, we discovered that the amount of input data was too large which caused a lot of time cost for training. Therefore, we separated input data into seven sets of data, and respectively put them into the ConvFCNetv3(ours) model to train. In this way, we saved more time on training and have more time to train to gain better accuracy on this project.

7. Discussion (explain result)

7.1 Benefits of separating the training dataset into several chunks

Due to the huge training dataset, our training dataset was separated into several datasets. There are 2 benefits we have obtained from the training process. First, the time of a single training epoch becomes less since only one chunk of the training dataset was used to update the parameters (weights and bias) of NN. It is also good for tuning the hyperparameters quickly. Second, as shown in the training/validation curves, the trained model would not overfit the whole training dataset.

Even if the result model would not overfit the whole training dataset, it still may overfit to a single chunk of training dataset. That is the reason why the regularization techniques such as data argumentation, weight decay, are needed.

7.2 Gap between validation accuracy and testing accuracy

As shown in our results, we get validation accuracy up to 92.71%, but our testing accuracy is only 82.23%. There is about a 10% gap between them. The possible reason we came up with is discussed in this section.

Although we use validation dataset to observe and prevent the phenomenon of overfitting problem, overfitting still occurs in our model. As shown in the data preprocessing section, training dataset and validation dataset are separated after images were cropped into small pieces. That means there may be some common points in training and validation datasets since they are in the same original images. Eventually, our model may learn how to recognize those common points, which causes overfitting to the training and validation datasets.

7.3 Shortcut learning prevention

Since the dataset is collected by us and the problem of shortage of datasets is obvious, shortcut learning may occur such as our model may classify the image by the content of the images. To prevent shortcut learning, in addition to collecting more images from different people, we also cropped the images to prevent the content of images learned by our model. Finally, we used familiar images of both iPhone and Samsung to test, and our model can classify them correctly.

8. Conclusion

This project provides useful insights from existing techniques. As a result, all the provided insights can be seen as the first step towards the development of a classification model that classifies the image and applies in various fields. We hope in future our design could classify more brands and phone models precisely.

9. Author Contribution Statement

Yen-Yun Kuo (20%): data collection, train model, final presentation

Chao-Ju Chen (20%): data collection, train model, final presentation

Mei-Ching Chen (20%): data collection, prediction implementation, final presentation

Cheng-Yu Sie (20%): data collection, build model, final presentation

Ya-Han Chang (20%): data collection, image preprocessing, final presentation

10. Reference

[1] ResNet | Pytorch, https://pytorch.org/hub/pytorch_vision_resnet/

[2] Implementing ResNet18 in PyTorch from Scratch, https://debuggercafe.com/implementing-resnet18-in-pytorch-from-scratch/