Face Recognition Research

***INFO7390** – Advances in Data Sciences and Architecture
Research Project Paper
Lecturer: Nik Bear Brown
Team Member: Junxi Fan, Kaixin Gao*

Abstract

Last few years have witnessed the thrive of face recognition systems in researching areas and practical fields. Many companies or organization's algorisms or products are already reached usable accuracy in identifying faces, even some products of the face recognition system convinced people to use in their daily lives. Face recognition technology had three breakthroughs since the first working system in 1966. As the realize and use of deep learning in 2010's, a storm of Convolutional Neural Networks (CNN) Face Recognition research across the world raise the accuracy of face recognition significantly.

We hope we can take advantages of the CNN and big-scale face databases to practice face recognition on our own system. More importantly, our goal is attempting to achieve higher accuracy, which should be more than 90% accuracy on Microsoft's MSRA-CFW face database. Due to the face database is not very ideal, we need to adjust our image data using OpenCV and tuning our model using Keras. In this paper, we will also cover the process how we achieve 91% accuracy in detail and our thoughts on this research.

Keywords: CNN Face Recognition, Deep Learning, Keras, Databricks, Python

Introduction

I． Motivation

Face recognition has become one of the most widely used security mechanism recent years. Companies like Apple, who is famous for caution when using new technology, made a product using face recognition as a secondary security mechanism and selling point. Which means that the industry of technology companies has faith in the accuracy, and people are believed in that face recognition features are able to secure their devices. Face recognition is of great application value.

On the other hand, the use of deep learning thought and CNN, push face recognition to a whole new level. Since 1993, the error rate of automatic face-recognition systems has decreased by a factor of 272, with new algorism, the error rate decreased by one-half every two years. We want to experience the change by apply CNN on our face recognition system.

II． Background

Modern face verification pipelines mainly consist of two stages: extracting low-level features and building classification models.

New recent progress in this area are made because of the use of the convolutional neural network (CNN), and the availability of large-scale face data-sets of training. By today, the best result we can get in LFW data-set which is challenging is 99.15% of correctness. However, there is still plenty improvement can be made. For example, the most severe problem today in face recognition is the face recognition for some special face types and the face recognition in a video stream.

III． Our algorithm

Based on an appropriate framework, we will try different methods or parameters to get an appropriate machine learning model and get a higher point in the evaluation. Our system will have the ability to recognize specific people.

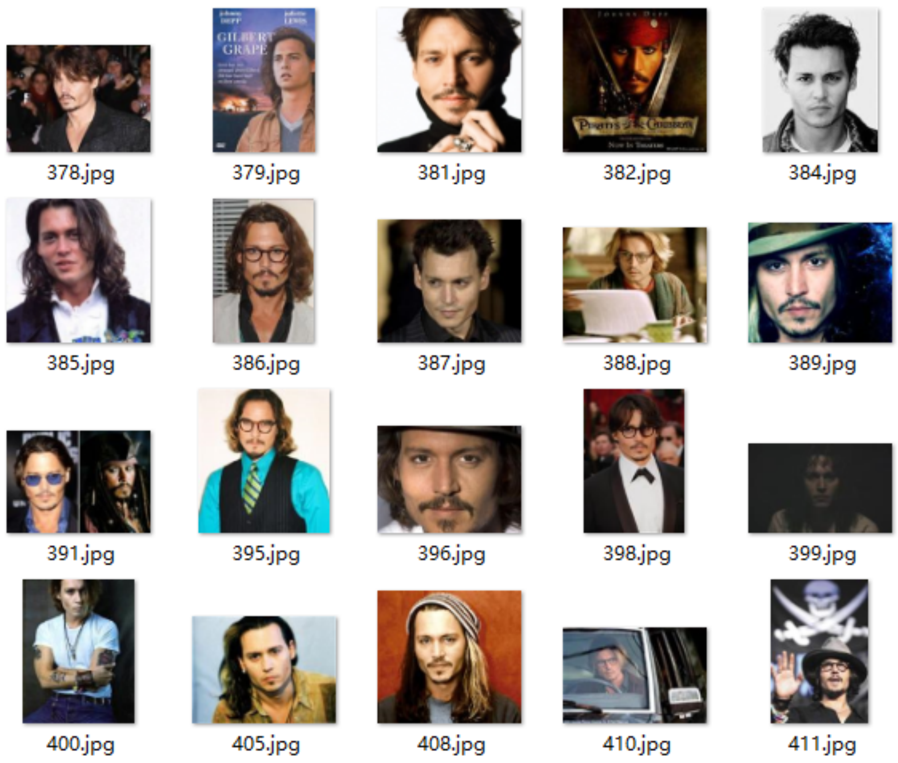
The database we are using is Microsoft’s MSRA-CFW, which is a set of celebrities’ pictures focused on the face. To be fair, we must say this database is challenging to train because some image is of poor quality and some image is not even the person we are training for. Second, we are using OpenCV to resize the image and adjust color so that we can process the same, then crop the image where is a human face. Thirdly, we are using Keras to build CNN model. Besides, considering making our work more attachable and can run on different ends, we chose to put our work on Databricks, a company founded by the creators of Apache Spark, that aims to help clients with cloud-based big data processing using Spark/Python. Users can easily build an environment to run our project.

Methods

I. Preparation

Before walk-through our code, here we briefly introduce our mythology from the beginning get the result.

To begin with, we have our training data ready by downloading Microsoft's MSRA-CFW face database and choose some person that specifically we want to train with. We also had considered other face training dataset, like LFW, FaceScrub, LFPW, but MSRA-CFW has more pictures under per person, that means we have more sources to train our model. And MSRA-CFW also provides jpg files other than a link, we don't need to write another code for bulk download them, and the most important thing is, we don't have to consider the invalid URL problem. As we can see that the images come with different sizes and quality, which means we need to process the images before we can use these to train our model.



(Picture as a face database example)

Databricks has a file system called DBFS, so just upload graphs we need on it. We can also build the environment easily on Databricks, just create new libraries by installing Pypi Package or uploading Python Egg, then Databricks will help.

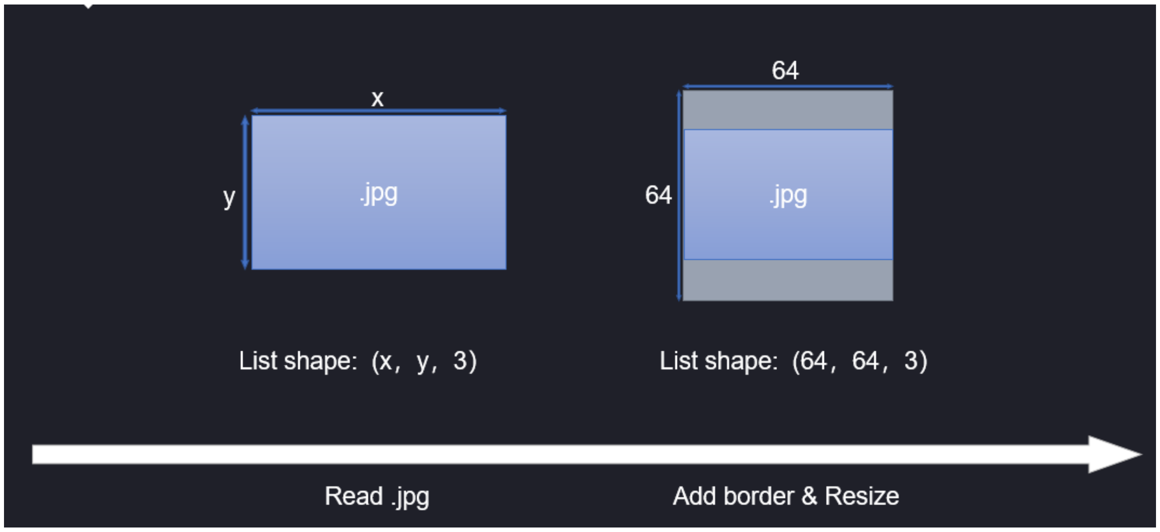
As we mentioned before, we will use Keras, OpenCV in our project. So, import these, our preparation is finished.

II. Image Processing

The input shape of a CNN has ruled, so before building our model, we need to process image dataset. We want to read graphs one by one, normalization, and add different labels to each of them according to the person shows on it, like ‘0’ for John and ‘1’ for Jane.

We use OpenCV, which has an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on. to adjust our images.

This picture shows our assumption of image processing:



(Picture how we want OpenCV process image)

When we read an image, suppose we have x pixel wide and y height image. OpenCV can read an image file and coverts it to a list (width × height × 3, 3 means three channels of colors as red, green, blue). To normalize the input data shape of our neural network model, for one image, we compare the width and height and add borders to the shorter sides. At last, we resized the picture to 32 × 32 pixels format.

How to read hundreds of images with labels into our project is the next problem. Now we have many graphs of two people, for example, Johnny Deep and Natalie Portman separated in two folders. Also, as it is a project about recognition, it’s necessary to do one-hot encoding to the labels. After doing this classification, we can get a list of images and a list of related labels.

We will do cross validation after training, so we split our dataset into three parts: Training, validation, and test sets. For now, the preparation before building CNN is finished. This is our data shape:

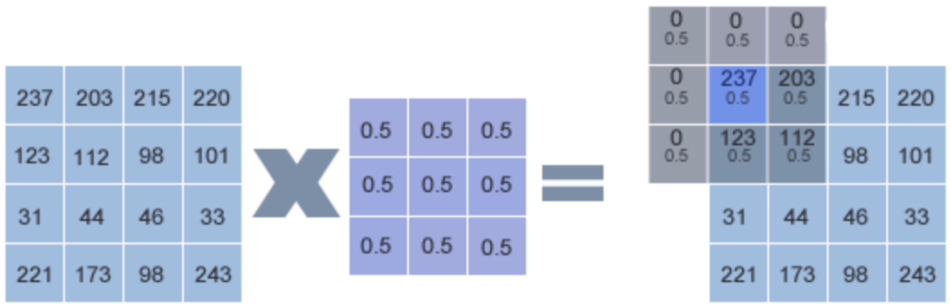
```
(1331, 'train samples')
(571, 'valid samples')
(951, 'test samples')
```

(picture of our data shape)

III. Build CNN model

Keras provides plenty of API for building neural network model. We can build a sequential convolutional neural network easily. First, here is an introduce of each kinds of layers in the CNN.

Convolution layer:
The emphasis here is on the function: Conv2D(). According to the Keras official document, 2D represents a 2D convolution, whose function is to perform sliding window convolution calculation on 2D input. Our facial image size is 64 x 64 pixels, which only contain length and width, so we are using a 2D convolution function to calculate the convolution. The sliding window calculation represents uses the convolution kernel to calculate pixels one by one in order.



(picture of 2D convolution process)

First, we will focus the convolution kernel on the first pixel of the image, here the pixel with a pixel value of 237. The area covered by the convolution kernel, and all pixels below it is averaged and then added together:

$$C(1) = 0 * 0.5 + 0 * 0.5 + 0 * 0.5 + 0 * 0.5 + 237 * 0.5 + 203 * 0.5 + 0 * 0.5 + 123 * 0.5 + 112 * 0.5$$

Then replace the first pixel in the image, then calculate the second, third…until we get a same size but convoluted image. As to the edge pixels, we fill over edge part with 0. We can use this setting in Conv2D() by code:

self.model.add(Conv2D(64, 3, 3, border_mode='same', input_shape = dataset.input_shape))

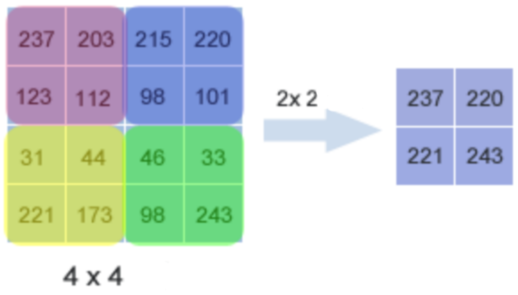
We also need to tell Keras the data we input, which is 64*64 in RGB color, code: input_shape(64, 64, 3)

Activate function layer:

Relu (Rectified Linear Units) function, the input is less than 0, the output is all 0, greater than 0 is equal to the input and output. The advantage of this function is its fast convergence. The keras library also supports several other activation functions: Softplus, Softsign, Tanh, Sigmoid, Hard_sigmoid, Linear. We tried all of this activation functions and decide to use Relu.

Pooling Layer:

The purpose of the pooling layer is to reduce the input feature map, simplify network computational complexity, and simultaneously compress features, highlighting key features.



*(Picture of 2*2 pooling)*

We establish the pooling layer by calling the MaxPooling2D() function. This function uses the maximum pooling method. This method selects the maximum value of the coverage area as the main feature of the area to compose a new reduced feature map. Thus, we will get a 32*32-pixel image after pooling.

Dropout Layer:

The Dropout layer randomly disconnects a certain percentage of input neuron links and consciously reduces the model parameters, making the model simple to prevent overfitting. Dropout() function use float parameter from 0-1 to define drop out percentage.

Flatten layer:

After many times of convolution, pooling, and Dropout, here you can enter the full connection layer for final processing. The fully connected layer requires that the input data must be one-dimensional, so we must "squash" the input data into one dimension.

Dense layer:

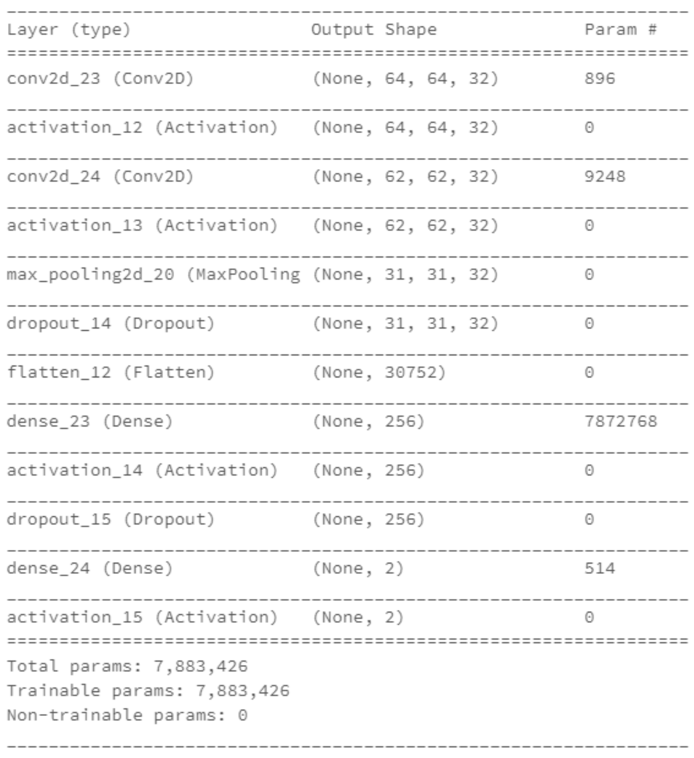
The role here is for classification or regression, which is classification. We define dense layer through the Dense() function. One of the required parameters of this function is the number of neurons, which it is to specify how many outputs the layer has. In our code, the first dense layer specifies 512 neurons, that is, retains 512 features output to the next layer.

Classification layer:

The goal of the dense layer is to complete our classification requirements: 0 or 1.

self.model.add(Dense(nb_classes))
self.model.add(Activation('softmax'))

In the first row of code, we define classification requirement number of neurons, which is 2 for us. And in next layer, we use Softmax() to finish final classification. From the perspective of classification, the greater the output value of the neuron, the greater the likelihood that its corresponding category is a real category. Therefore, after the Softmax(), the upper N inputs are mapped to N probability distributions, and the sum of the probabilities is 1. The highest probability is the model predicted by the model.
Our structure of CNN model:



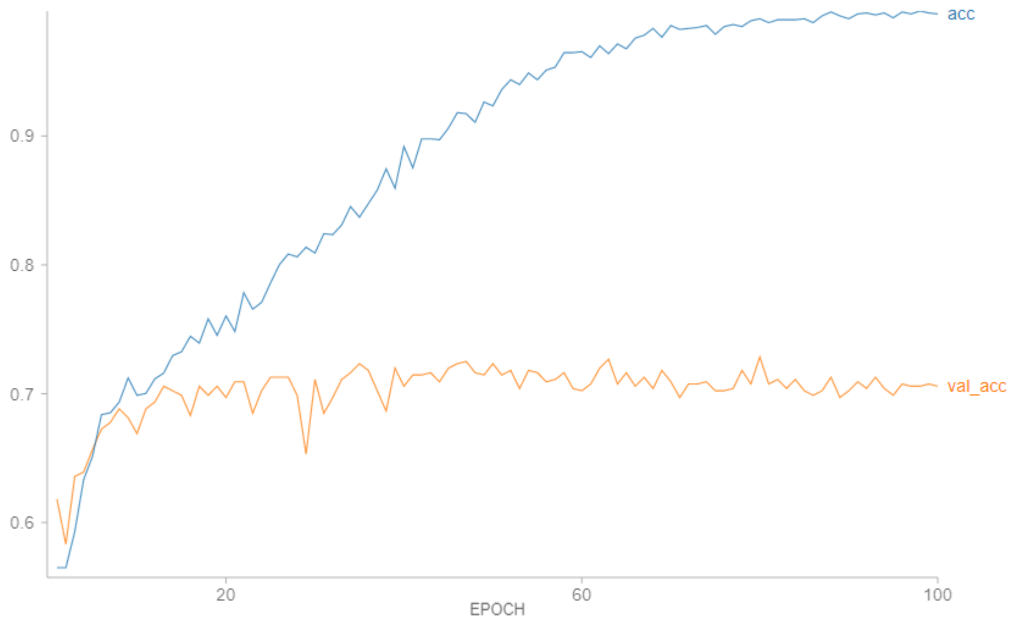
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_23 (Conv2D)           (None, 64, 64, 32)        896       
_________________________________________________________________
activation_12 (Activation)   (None, 64, 64, 32)        0         
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 62, 62, 32)        9248      
_________________________________________________________________
activation_13 (Activation)   (None, 62, 62, 32)        0         
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 31, 31, 32)        0         
_________________________________________________________________
dropout_14 (Dropout)         (None, 31, 31, 32)        0         
_________________________________________________________________
flatten_12 (Flatten)         (None, 30752)             0         
_________________________________________________________________
dense_23 (Dense)             (None, 256)               7872768   
_________________________________________________________________
activation_14 (Activation)   (None, 256)               0         
_________________________________________________________________
dropout_15 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_24 (Dense)             (None, 2)                 514       
_________________________________________________________________
activation_15 (Activation)   (None, 2)                 0         
=================================================================
Total params: 7,883,426
Trainable params: 7,883,426
Non-trainable params: 0
_________________________________________________________________
```

(CNN model Structure)

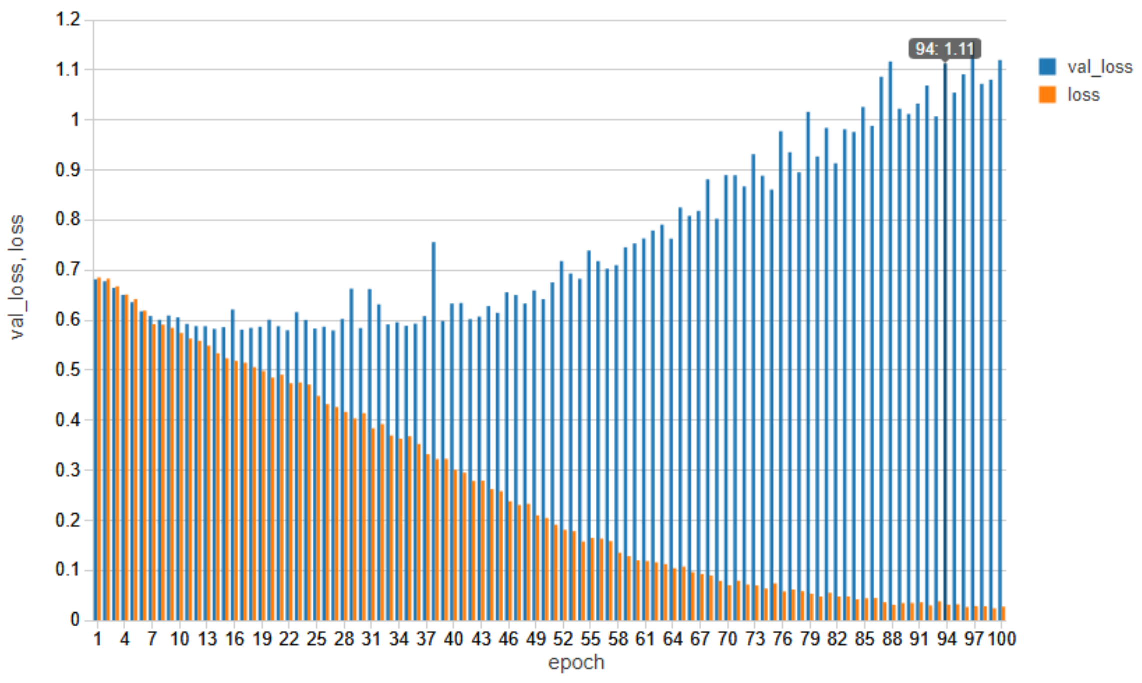
The total params in this model is about 8 million, not bad.

IV. Training

Next is let the dataset we have prepared to train this model and do cross-validations. We set the cost function as ‘categorical_crossentropy’, a typical function dealing with the categorical project. The optimizer is SGD (Stochastic gradient descent optimizer), at the same time, Keras includes support for momentum, learning rate decay, and Nesterov momentum, which means we can take multiple advantages from them. Using SGD only, the decent direction depends on batch data completely, so add a momentum can keep original decent direction to some degree, make the optimizer works faster, but more stable. We set the epoch as 100, and at last, we get a graph using Spark DataFrame, which shows the cross-validation result (accuracy and loss) of each epoch, for both training set and validation set. We run our project on Databricks so it is more convenient to generate carts with Spark DataFrame. If you run this project locally, like Anaconda, just use pandas, Seaborn, to process the result.



(Accuracy of each epoch (Trian & Validation Set))



(Loss of each epoch (Trian & Validation Set))

(Code of this project can be found here: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5131058779162399/1344617025196396/1086196905023500/latest.html)

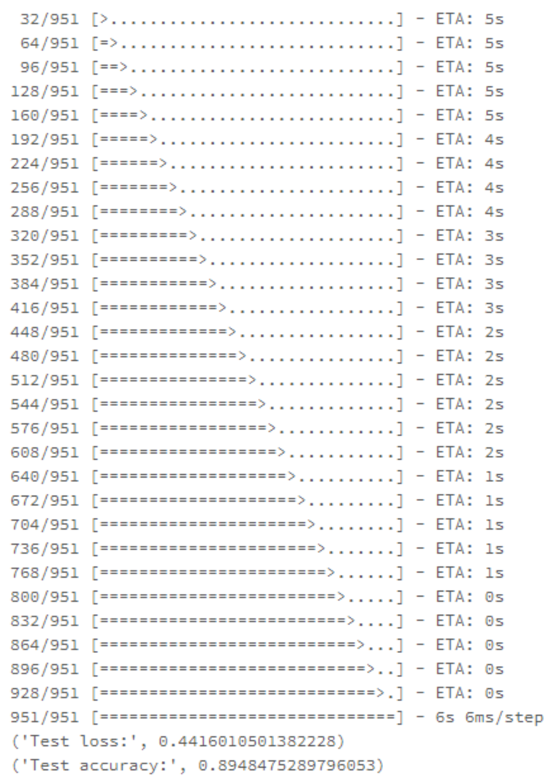
Results

Before evaluating our model, we need to realize the standards, or conditions of the face recognition research area. As we know there are many outstanding solutions of face recognition, like Fisher Vector Faces, DeepFace, Fusion, FaceNet, etc.

No.	Method	Images	Networks	Acc.
1	Fisher Vector Faces	-	-	93.10
2	DeepFace	4M	3	97.35
3	Fusion	500M	200	98.37
4	DeepId-2,3		1	99.47
5	FaceNet	200M	1	98.87
6	FaceNet + Alignment	200M	1	99.63

(Test Result of some Face Recognition methods)

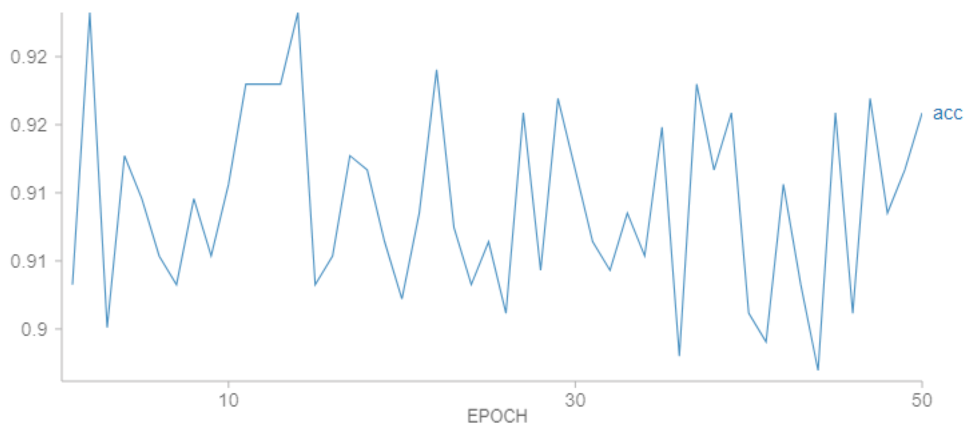
Using test dataset, we generate before to evaluate our training result. Keras provides an evaluate function, use trained model and test set as parameters, it will return the value of loss and accuracy.



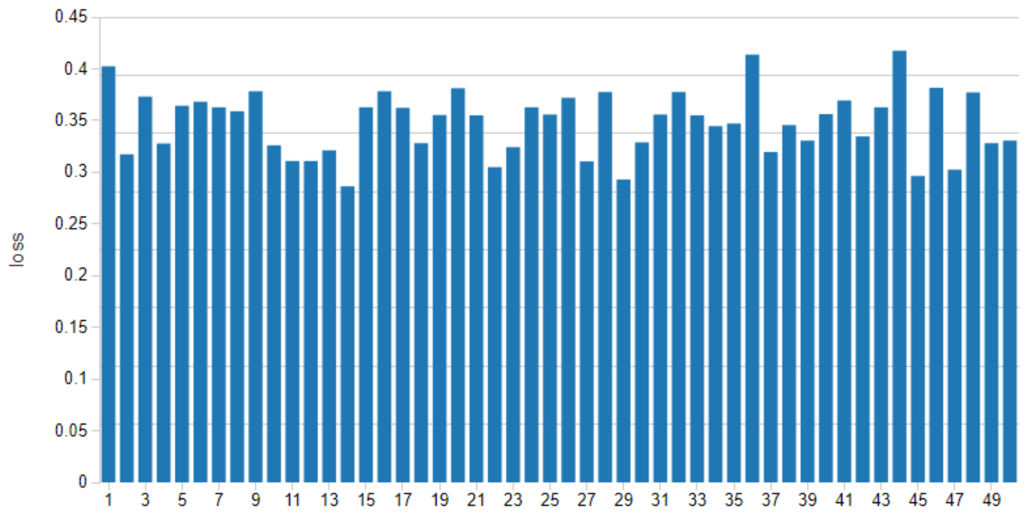
```
 32/951 [>.............................] - ETA: 5s
 64/951 [=>............................] - ETA: 5s
 96/951 [==>...........................] - ETA: 5s
128/951 [===>..........................] - ETA: 5s
160/951 [====>.........................] - ETA: 5s
192/951 [=====>........................] - ETA: 4s
224/951 [======>.......................] - ETA: 4s
256/951 [=======>......................] - ETA: 4s
288/951 [========>.....................] - ETA: 4s
320/951 [=========>....................] - ETA: 3s
352/951 [==========>...................] - ETA: 3s
384/951 [===========>..................] - ETA: 3s
416/951 [============>.................] - ETA: 3s
448/951 [=============>................] - ETA: 2s
480/951 [==============>...............] - ETA: 2s
512/951 [===============>..............] - ETA: 2s
544/951 [================>.............] - ETA: 2s
576/951 [=================>............] - ETA: 2s
608/951 [==================>...........] - ETA: 2s
640/951 [===================>..........] - ETA: 1s
672/951 [====================>.........] - ETA: 1s
704/951 [=====================>........] - ETA: 1s
736/951 [======================>.......] - ETA: 1s
768/951 [=======================>......] - ETA: 1s
800/951 [========================>.....] - ETA: 0s
832/951 [=========================>....] - ETA: 0s
864/951 [==========================>...] - ETA: 0s
896/951 [===========================>..] - ETA: 0s
928/951 [============================>.] - ETA: 0s
951/951 [==============================] - 6s 6ms/step
('Test loss:', 0.4416010501382228)
('Test accuracy:', 0.8948475289796053)
```

(Accuracy & loss of one test set)

However, in order to make our test result more convincing, we capture test set from original image sources randomly (the size is 951) and evaluate one by one for 50 times. Here is the result.



(Accuracy of 50 test sets)



(Loss of 50 test sets)

The mean of Test Accuracy is 0.91. The mean of Test Loss is 0.35. Considering the amount of sample and the time we cost for training, this result is not bad, which achieves our goal.

Discussion

I． Result Conclusion

As we mentioned in result section, we achieved 91% accuracy, why can’t us get higher accuracy? We think there are 2 most important elements for the result of face recognition deep learning.

First is the Structure of CNN, including hyperparameters, layers, optimizers, etc. It seems that an appropriate network is based on specific conditions, many things can affect the training result. The CNN model we showed in this paper is the final version after countless adjustment, like changing the learning rate, cost function, activation function, etc.

Second, the situation of data source, like the size of the dataset, quality of photos, proportion of noise, wrong matching of people and photos. To be honest, although MSRA-CFW provides a large number of photos, however many photos in MSRA-CFW are not suitable, for example, under one celebrity’s folder there are some photos of other people, or the viewpoints of some photos are terrible. These will all take bad effect on training.

II． Future Improvement

During the work, we found some places that we can improve and some idea that we are not able to achieve.

First, we found dataset quality is affecting the evaluation result in a negative way, we found images in data set is not having face towards us, or text covering some part of the face, person’s age in the dataset images is varying a lot from youth to more than mid-age, or we even have person that is not one we train for. If we will have more time to deal with thousands of images, we probably getting better results.

Second, we found our less time to learn more about CNN and tools we use. For example, when we built our model, we tried dozens of functions hoping to get better evaluation which we thought we did, but we don’t have the chance to know how every function performed in detail. That also a field that we could do better.

III． Others

The project is about implementing CNN to recognize faces, though we found it challenging sometimes, we still feel happy that we are facing a problem that has great application value.

Talking about the tools, we found due to the upper-layer API of Tensorflow, Keras is much more user-friendly for deep learning. We've tried to build the same CNN model using Tensorflow, the code became much more complicated. Also, as we know Tensorflow has a session, everything must "maintain the status quo" until the session start, in some cases, it will make the code confusing.

Now CNN is widely using in the image recognition area, and the results of our project prove that even a CNN model with a simple structure can recognize specific person well (The accuracy rate is more than 90%). For further works, we can improve this model and use it for recognition of more people, or try to analyze faces in videos, or the face lock on mobile phones… After all, this is one of the key areas of deep learning.

References

[1]. Facial recognition system. (2018, April 22). Retrieved from https://en.wikipedia.org/wiki/Facial_recognition_system

[2]. Chen, D., Cao, X., Wen, F., & Sun, J. (2013). Blessing of Dimensionality: High-Dimensional Feature and Its Efficient Compression for Face Verification. 2013 IEEE Conference on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.2013.389

[3]. Paluszek, M., & Thomas, S. (2016). Face Recognition with Deep Learning. Machine Learning, 89-112. doi:10.1007/978-1-4842-2250-8_7

[4]. MSRA-CFW: Data Set of Celebrity Faces on the Web. (n.d.). Retrieved from https://www.microsoft.com/en-us/research/project/msra-cfw-data-set-of-celebrity-faces-on-the-web/?from=http://research.microsoft.com/en-us/projects/msra-cfw/

[5]. Databricks. (n.d.). Retrieved from https://www.wikiwand.com/en/Databricks

[6]. OpenCV (Open Source Computer Vision) (n.d.). Retrieved from https://docs.opencv.org/3.4.1/d1/dfb/intro.html