CS373
Project Report

Author:
Haobo Wang

Xiao Hu

Boxuan Zhu

12/01/2018

Abstract

The purpose of this project is to apply machine learning into cases in real life. In our experiment we will apply machine learning into computer vision, mostly in object identification and classification in images, to classify one specific person from other people. We will use George W. Bush as the unique person to detect, and we will use YOLOv3, a real-time object detection application to train, test a model for face detection from the dataset we have, then . We will also implement a We will use k-fold cross validation method to do the process of our images, in which k will be 5. In the experiment the hyperparameters of the two methods will be changed to compare and see the effect of different values of parameters for our dataset.

Additionally, we will use PCA+SVM as a second method to recognize George W. Bush among all the other people in the lfk dataset. Principal component analysis (PCA) is a technique which can reduce dimensionality of arrays and keeps several most important characters of people's face. Then we use SVM to classify different people through these characters.

Introduction

With the rapid and unprecedented development in image processing technology, face recognition has become an important member acting significantly in the related field. In Beijing, traffic police began to use this technology to identify jaywalkers and automatically issue them fines by looking their identification information in the database. Impressed by the potential ability of facial recognition technology, our team decided to use our knowledge of machine learning and image processing to create a application that detect a certain personal in images. In this paper, we will discuss our model, our results, and potential future work.

Related Work

There are numerous existing algorithms for object identification and classification, and to begin with we want to use Yolov3 solely to do the face recognition and detection, mainly because of the following reasons: First, it is an open source from Github to download and use. Second, it is really easy to train your own custom models and test with different inputs. In terms of speed, according to Yolov3's published paper, algorithms such as RCNN, Fast RCNN, and Retina-Net, has a slightly better performance but 4X faster.

However, Yolo has also downsides and we also verified during our training process with images of George Bush. One downside that is critical to our experiment is that it could not generalize to different ratios and configurations, which is the most significant part of human faces. Since human faces have distinct features to identify, and the dataset we use also has

different setting of lighting levels, face directions and some even provide obstacles that block the full face from being measured. (See experiment section for details). Thus, a more robust algorithm for face detection is needed to do the further algorithm of our project.

Finally we decide to include Facenet, an open 3rd-party library especially focuses on face recognition. We will run Yolo for face detection in the image, and then we will provide the images with metadata to Facenet the face classification with the specific person, in our experiment, George Bush.

SVM Part
In the SVM face recognition part, sklearn plays an important role. We use open library sklearn.model_selection.KFold to implement k-fold algorithm and select best classifier, sklearn.datasets.fetch_lfw_people to fetch and save information of all the people in the fold, sklearn.decomposition.PCA to delete unnecessary data in the pictures and sklearn.svm to create SVM classifier.

Dataset and Feature

For this project, we will use datasets in the Labeled Dataset in the Wild - lfw-funneled and lfw-bush. In lfw we will filter 4 people . lfw-bush contains 530 images of George W. Bush. The size of all images that we train and test with Yolo, Facenet and SVM are unified with RGB color space and 250 x 250 pixels dimension. For Yolo we will use only 1 class, George W. Bush mainly and use 530 images to train the model. For svm and Facenet, we will use 5 classes/people to train the classfiier.

Environnment Set Up, Methods and Algorithms

We will use Windows, Mac OSX and Ubuntu as our environment setting. We use Windows for training and testing model from Yolo, since we need opencv and CUDA v9.1 to accelerate the process of training, so a Windows laptop with Nvidia GTX970m 3G is used for training in the experiment. Microsoft Visual Studio 2015 toolkit (v140) is also needed to build the darknet project to Windows executable. For Ubuntu we will test with our models and do the images preprocessing, which includes images labeling and converting, k-fold cross validation dataset splitting. Mac OSX will run Facenet with tensorflow and also SVM algorithm to do the lfw images classification.

First we will label the faces that our 530 George Bush images have. To do so we use an open source called BBox Label Tool, which could create a txt files that contains information of the bounding box for each image we label. In order to use the text files for Yolo to train, we also run a python file, convert.py to translate the format for Yolo to use.

To create a 5-fold cross validation for training, validating and testing in our images dataset, we run a python script to parse the images in 5 subsets, each subsets contains 530/5 = 106 images with increasing order. Since number orders in the images are random assigned and does not have any meaning so split the data with order will not affect generalization. We create 5 train and test text files for yolo to train and test, so that after training we will select the one with most precision or least MSE.

For Facenet, Tensorflow-gpu toolkit is used to speed up the training process using GPU. Since Facenet uses MTCNN for the framework, we need to crop our images to 160X160 image thumbnails and make alignment to people's faces to feed the input to MTCNN. Thankfully, facenet/src/align/ the author provides the python file align_dataset_mtcnn.py for us to fulfill this task. We will also split our data set into 2 groups, one is training which will be used to train a classifier, and the rest part will be used to validate the precision. To train our own model, we will use train_softmax.py which will trains a softmax classifier, and we will run validate_on_lfw with our lfw datasets to test with our training model.

SVM Part

In the PCA+SVM part, firstly we read all the unlabeled photos in the data/.../lfw_funneled folder. If there is no readable data, the program will download lfk database and related files automatically from the website. Next, we convert data of each face photo into an numpy array. The reason we save data into arrays is that we need these information to calculate eigenfaces then train SVM. Now we use k-fold cross validation, splitting all the train data into 10 (n) parts, randomly training 9 (n-1) out of 10 (n) parts and test with the last part. Every time we record the performance score of SVM, and finally find the best one. We use the best SVM to test and predict Bush's pictures in the dataset and observe its performance.

Experiments and Discussion

Unfortunately, as we followed our original plan and trained our models with the first 2 models in our k-fold cross validation, the outcomes was disappointing. From the training curve, although we could get a reasonable average training loss (<= 0.18 or even smaller) and prevent the issue of overfitting by selecting the proper iterations number that go convergence

(see image below):

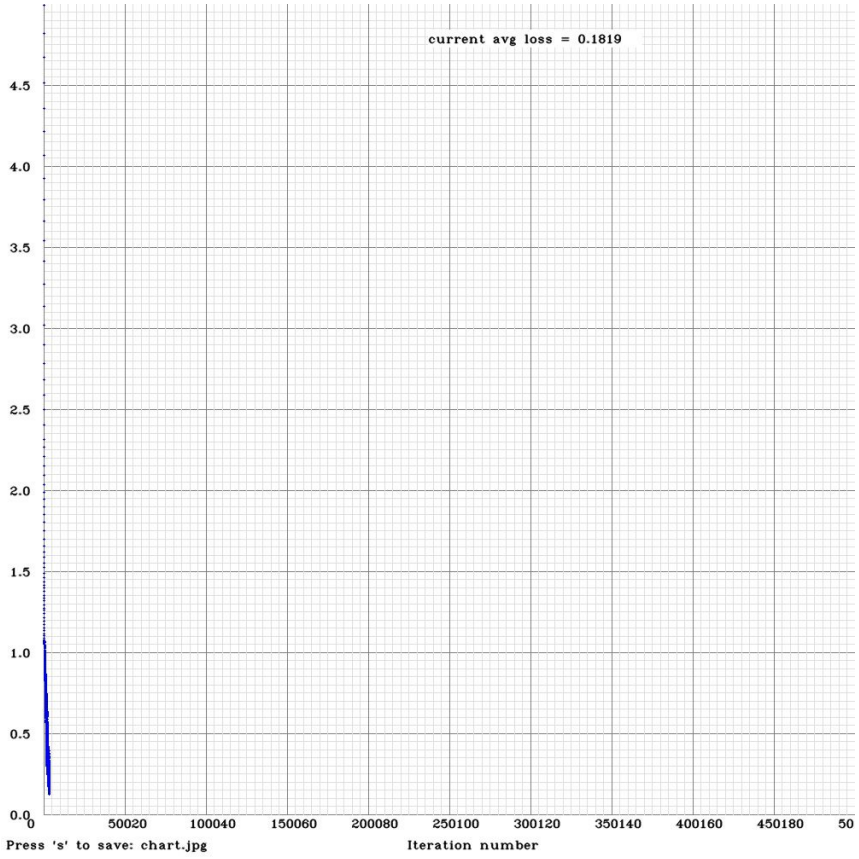


Figure Training Loss VS. Number of Iteration

And we also gained a reasonable precision, recall and IoU with the model selected with the model we trained with our test set:

```
detections_count = 140, unique_truth_count = 107
class_id = 0, name = Bush,          ap = 90.91 %
 for thresh = 0.25, precision = 0.96, recall = 0.98, F1-score = 0.97
 for thresh = 0.25, TP = 105, FP = 4, FN = 2, average IoU = 77.10 %

 mean average precision (mAP) = 0.909091, or 90.91 %
Total Detection Time: 1.000000 Seconds
```

Figure Testing Based on Training Model

However, the model we trained does not classify well for the different people in the database set of lfw:


```
Enter Image Path: data/trump.jpg
data/trump.jpg: Predicted in 12.818000 milli-seconds.
Bush: 94%
```

Figure Sucess Face Detection


```
Enter Image Path: data/lfw/5.jpg
data/lfw/5.jpg: Predicted in 11.594000 milli-seconds.
Bush: 50%
```

Figure Failed Prediction With Our Model

Since we do not want to waste more time training with our own models without a good generalization and specialization, we changed our method to Facenet, which does a nice prediction on lfw training set according to lfw's result page.

Model Selection and Overfitting Prevention on Yolov3:

For Yolov3, on the progress of training a plot with current average training loss VS. number of iterations will be popped up, and we could figure out when to stop our loop and start picking the ideal model. We start picking models with loss that less or equal to 0.15, and this usually takes at least from 1600 iterations. After that we will start running the command for testing:

```
C:\Users\whb19\Documents\GitHub\darknet\build\darknet\x64> darknet.exe detector map data/obj.data yolo-obj.cfg yolo-obj_2400.weights
```

And it will return the outcome of our model:

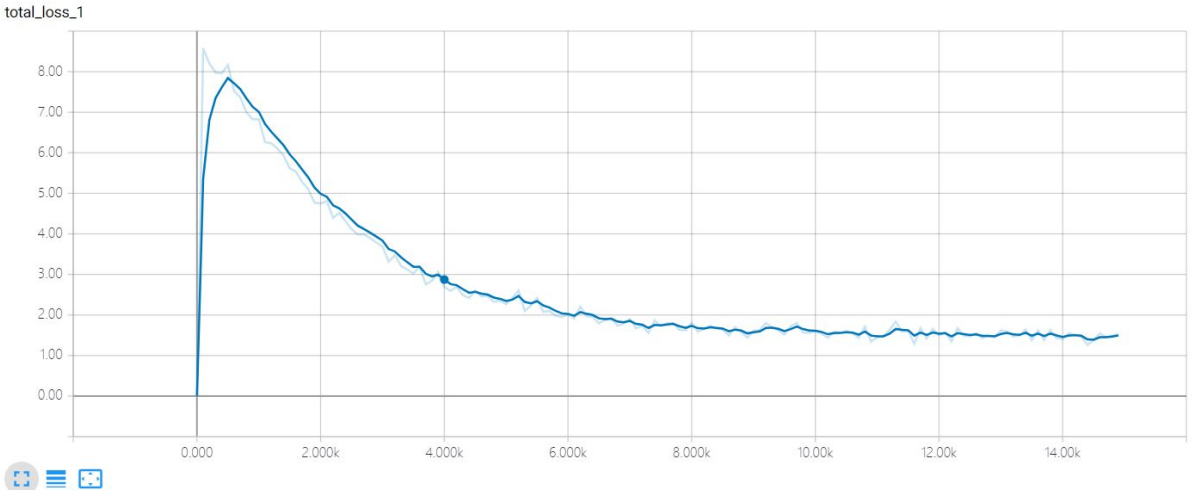
```
detections_count = 140, unique_truth_count = 107
class_id = 0, name = Bush,          ap = 90.91 %
 for thresh = 0.25, precision = 0.96, recall = 0.98, F1-score = 0.97
 for thresh = 0.25, TP = 105, FP = 4, FN = 2, average IoU = 77.10 %

 mean average precision (mAP) = 0.909091, or 90.91 %
Total Detection Time: 1.000000 Seconds
```

Then we will choose the model that returns the maximum mean average precision, in this case we choose the model that has the maximum mAP (90.91%) and has the least number of iterations to prevent the problem of overfitting, in this case, is 2400.

Facenet

We use facenet to train classifier to detect the accuracy of the input images.

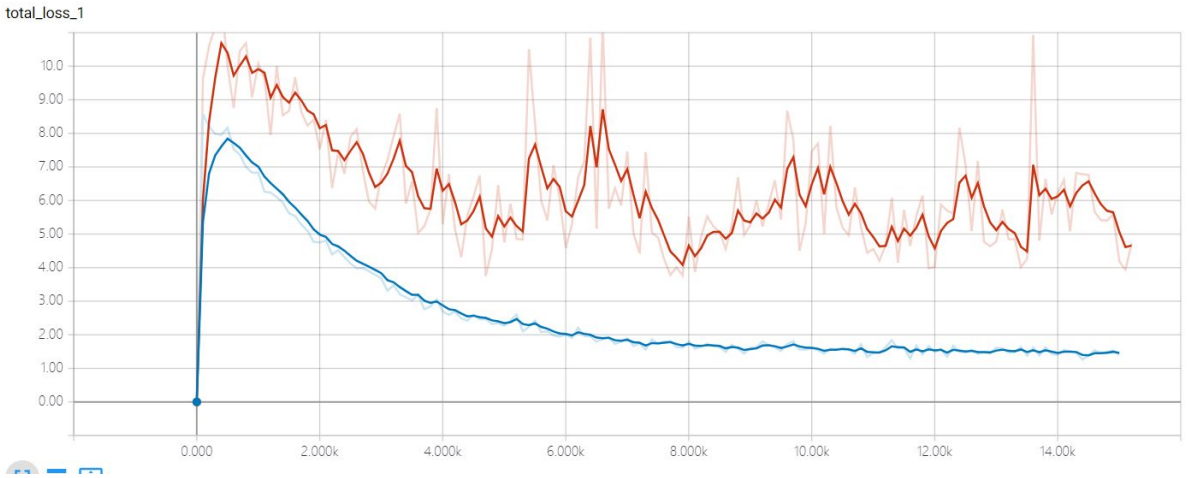
Due to the limitation of our hardwares we trained the model with 15,000 steps(15 epoches each with 1000 iterations). The learning rate starts at -1, and the batch size is set to 1.



1st Training Total Loss VS. Number of Steps

After this we changed our batch size from 1 to 5, and the graph of total loss VS. Number of iterations is shown as follows. From the graph we could see that with batch size = 5, within the same amount of steps the loss function are harder to converge as batch size=1, this might

because of the insufficient number of images we used in our dataset (approximately 200). The precision which also shown below also implicates that the precision of the prediction at batch size = 5 is much more slower than 1. For batch size = 5. we should continue with more epoches to get



more

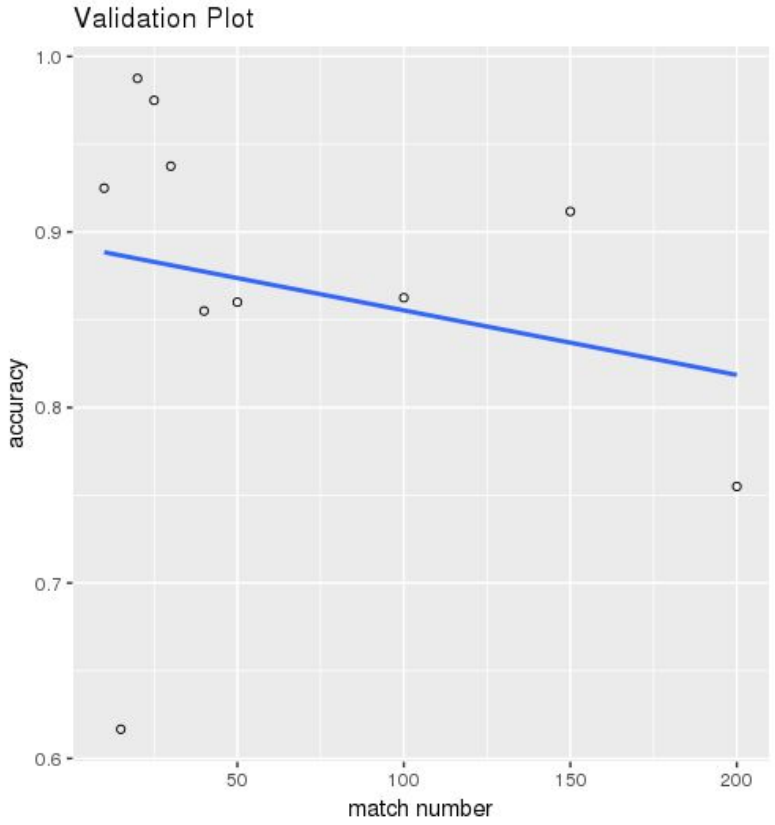
2nd Training Plot

Validation:

After training, we used other 100 images of five personals' faces we selected to validate our model 20181201-181023.pb stored in the model folder "20181201-181023". The following is the result from running Facenet validate_on_lfw.

```
2018-12-01 21:04:17.932607: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports i
nstructions that this TensorFlow binary was not compiled to use: AVX2 FMA
Model directory: /Users/Sion/Documents/GitHub/models/20181201-181023/
Metagraph file: model-20180408-102900.meta
Checkpoint file: model-20180408-102900.ckpt-90
Runnning forward pass on LFW images
..........................................
Accuracy: 0.97500+-0.02500
Validation rate: 0.13333+-0.13333 @ FAR=0.00000
Area Under Curve (AUC): 1.000
Equal Error Rate (EER): 0.016
```

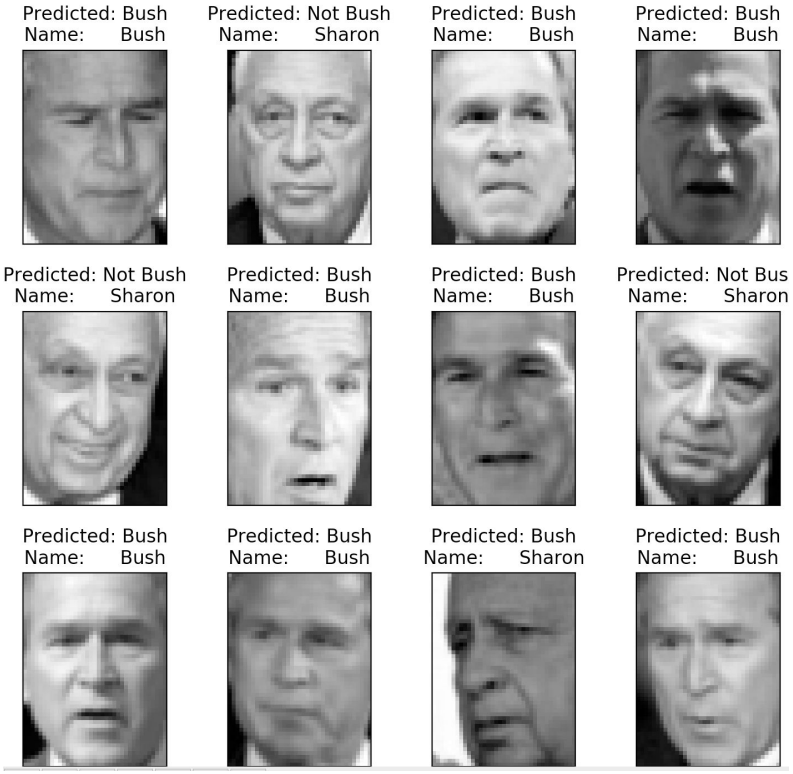
We chose to use 2-fold validation with different number (mis)match pairs to test our model, finding the best values to generate the highest accuracy. (mis)match indicating the number of matched pairs per set (equal to the number of mismatched pairs per set).



As shown, in the scatterplot above, the increasing of (mis)match number will lead to decreasing in accuracy in general, but the association is not that strong. Therefore, the number of (mis)match will work the best with the model we trained when it's between 10 to 30.

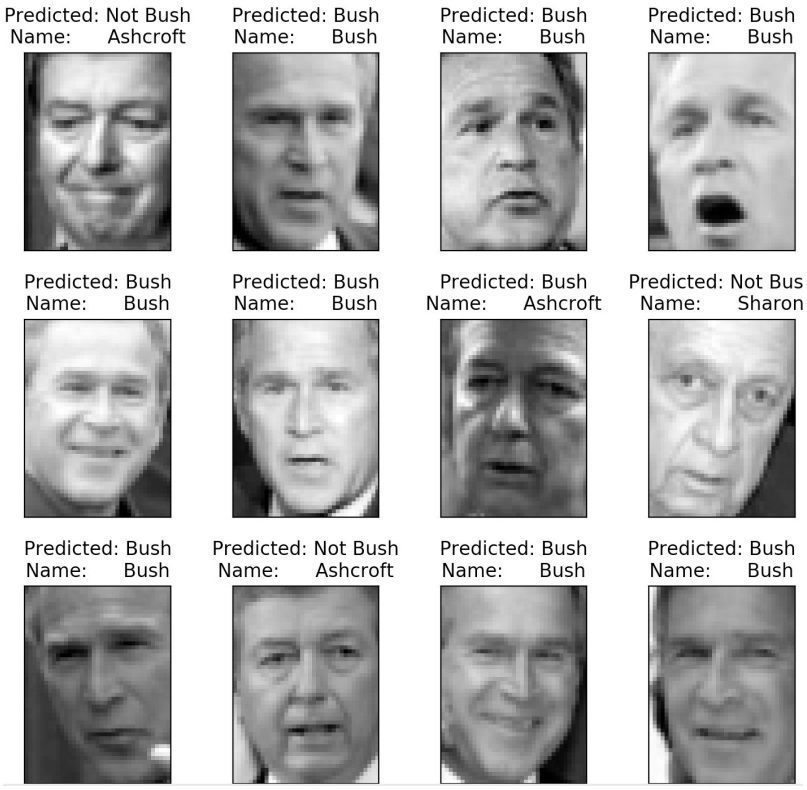
SVM Part

SVM classifier worded well in the experiment. Firstly, we trained the whole lfk dataset with K-fold cross validation (k = 8), the final performance scores were high in each iteration with the accuracy more than 95%.




```
Extracting the top 150 eigenfaces from 303 faces
done in 0.164s
Projecting the input data on the eigenfaces orthonormal basis
done in 0.009s
Fitting the classifier to the training set
done in 3.197s
Predicting people's names on the test set
Score is: 0.9802631578947368
done in 0.013s
```

Next, we selected five people (Ariel_Sharon, Bill_Clinton, George_HW_Bush, George_W_Bush, John_Ashcroft) as a new dataset and tried to recognize Bush among them. If we run the SVM with K-Fold, the accuracy was always around 86% (without K-Fold it was 84%) which was good . It was less accurate than training the whole lfk dataset because the number of pictures was much fewer. We investigated the effect of n_slipts in K-Fold, C and gamma parameters in SVM and n_components in PCA in the following experiment.

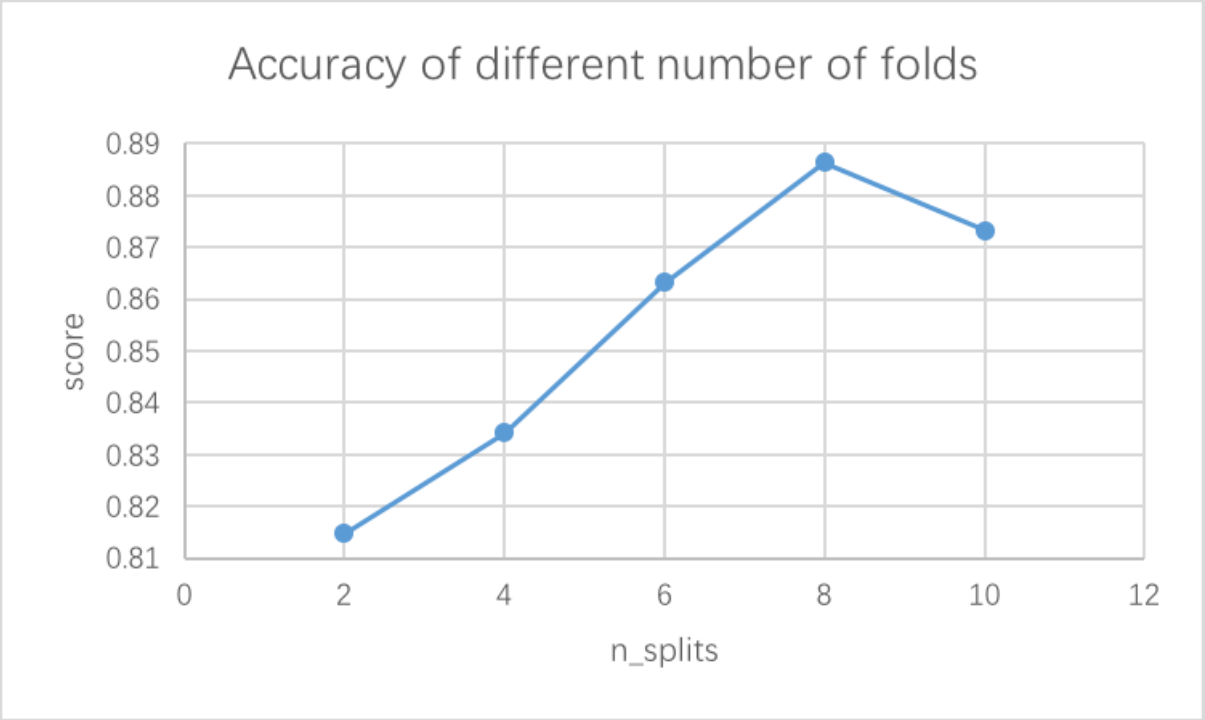



```
Predicting people's names on the test set
Score is: 0.8390804597701149
done in 7.866s

Maxmium SVM with corss validation score is: 0.8863636363636364
```

K-Fold

We used SVM with cross validation function in this experiment. We ran our program with different number of folds from 2 to 10 (keep other variables as default) to see how the accuracy changed. We got the below result and plot it. From the graph, we saw at the beginning (from 2 to 8), when the number of folds increased, the score increased as well. However, when n_splits changed from 8 to 10, it decreased. This may be because of the overfitting of train dataset. There were more folds than needed. The program trained too much data so the test result became less accurate. Thus, we found n_splits = 8 was best for K-Fold cross validation.

Kfold					
n_slipts	10	8	6	4	2
Score	0.8732	0.8863	0.8632	0.8342	0.8148



SVM without K-Fold

In this experiment, we split the original train dataset into two parts, 75% was training part and 25% was testing part. Then we ran the function SVM with no cross validation function and got a score 0.8465 with the best_estimator_:

SVC(C=1000.0, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

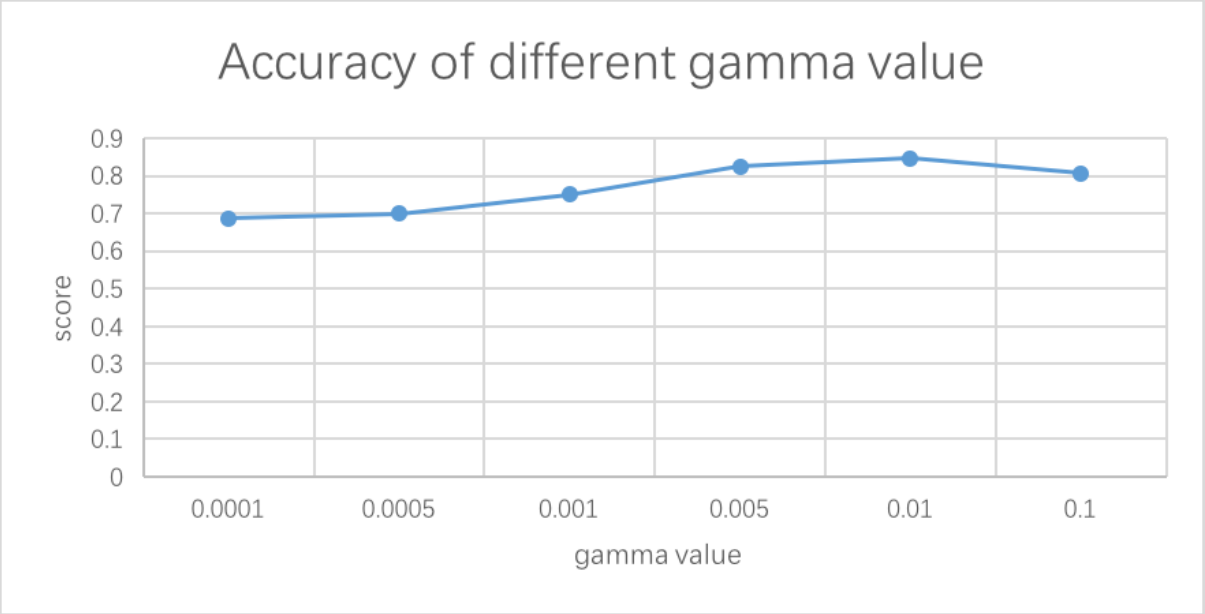
Although, it was less accurate than K-Fold cross validation, it saved a lot of time since there was no iteration needed. So we used svm without k-fold in the following experiments (PAC part) because of the advantage of efficiency. In the current experiment, we set C to different numbers: 1000, 5000, 10000, 50000, 100000 ([1e3, 5e3, 1e4, 5e4, 1e5]) and kept gamma the same value 0.01 (which was the best).

We got the below result. From the table, we concluded that C value did not affect the final outcome in our project. This mainly because the eigenfaces for the same person was very similar and for different people were very different.

SVM without Kfold					
C	1000	5000	10000	50000	100000
Score	0.8465	0.8465	0.8465	0.8465	0.8465

Then we set gamma to different numbers: [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1] and kept C the same value 1e3. We got the below result. From the graph, we could conclude that gamma = 0.01 performed best. The range of score between different gamma values was extremely huge. Thus, in the real environment, test different gamma value was significant. Gamma value affected the accuracy a lot.

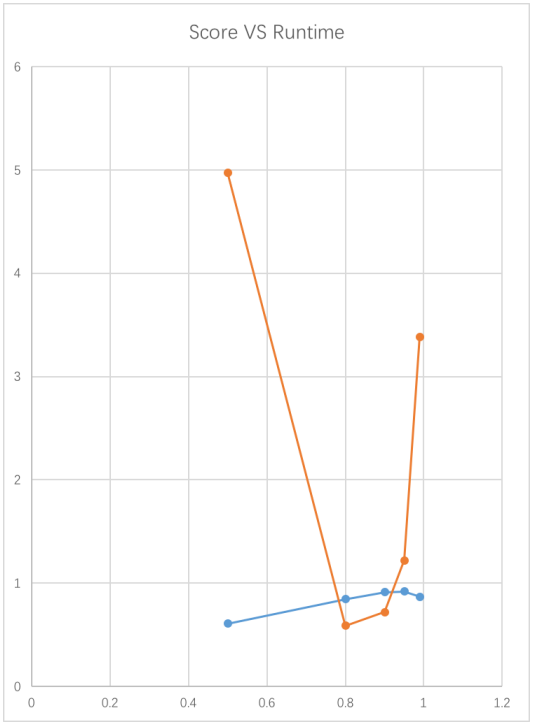
gamma	0.0001	0.0005	0.001	0.005	0.01	0.1
Score	0.6875	0.6988	0.75	0.8238	0.8465	0.8068



PCA

We tested keeping different percentage of components in the PCA step and observed the outcome. If we set the percent value from 50% to 99%, we got the below result table. We can see one interesting thing that the runtime decreased dramatically from 49.7s to 5.9s then increased dramatically from 12.2s to 33.8s. From this table, we could see keep 80% - 95% components was the best for our situation while the number of components was inversely proportional to runtime.

PCA					
n_comp	0.5	0.8	0.9	0.95	0.99
Score	0.6079	0.8465	0.9147	0.9204	0.8693
Runtime	49.7s	5.9s	7.2s	12.2s	33.8s



Reference

Facenet Paper: https://arxiv.org/pdf/1503.03832v3.pdf
Yolo Paper: https://pjreddie.com/media/files/papers/YOLOv3.pdf
Stanford Yolonet: http://cs230.stanford.edu/files_winter_2018/projects/6930066.pdf

https://www.independent.co.uk/news/world/asia/china-police-facial-recognition-technology-ai-jaywalkers-fines-text-wechat-weibo-cctv-a8279531.html