MLND Nanodegree – Face Recognition for My Family Members

I. Definition

A. Project Overview

The iCloud photos from iOS or Mac OS provided a comprehensive way to tag faces. All photos in the library will be scanned, thereafter faces will be identified and tagged accordingly based on the earlier defined tags/names.

I am a photography enthusiast and I have started to take photos from year 2004. As of today, it's over 200k shots and I've kept 40k of them. There are many desktop applications can do the job of face recognition, but it's going to be super fun if I can build the solution end to end.

Many other interesting use cases can be further built/extended by this, for example, auto greeting to the person sitting in front of the computer by calling his/her name. However, this won't be covered in this project.

B. Problem Statement

There are many great online blogs/projects discussed about the detailed mathematics and implementation of face recognition. For example, this one. I don't have plan to approach my problem in that way which may be too difficult and time consuming for me.

The first two questions came to my mind about this project were how to minimize the distraction factors in the photos and how to extract features understood by computers. All my photos are about something or somebody and I bet none of them is about a single face only. This project is about face recognition and all the other factors other than faces will be 'noise' and should be avoided before any machine learning kicking in. A 300 * 300 pixel color photo is quite good for human eye to distinguish faces. But it's a vector of size 300 * 300 * 3 = 270,000 which sounds too big to be machine learnt by today's computer. After some research and experiments I decided to use OpenCV to detect/extract faces and then used Google Inception V3 to extract features from the face photos.

C. Metrics

In this project both accuracy and speed will be looked into when evaluating the performance of the models. A model is useless if it performs only slightly better than a random guess. Speed is another important metric especially when we deal with large scale of data.

The metric of time is quite intuitive, both the training and testing time will be considered. The accuracy will be slightly more complicated. In the classification problem, we can look at precision, recall and f-beta score. The *precision_recall_fscore_support* from *sklearn* can do all this in one go. The detailed calculations are as follow. I used to have great difficulties to distinguish precision to recall until I found a way to describe them in two sentences under the context that God is the ground truth. Precision measures how much God agrees with me when I say it's positive. Recall measures how good I can find it's positive for whatever God says it's positive.

precision = Number of True Positive / (Number of True Positive + Number of False Positive)

recall = Number of True Positive / (Number of True Positive + Number of False Negative)

*f1 = 2 * (precision * recall) / (precision + recall)*

Training Time = End time of training – start time of training

Testing Time = End time of testing – start time of testing

D. Workflow of the Approach

I. Analysis and data preparation. A brief experiment to display sample image, explanation on the techniques and algorithms will be used.

II. Implementation.

  i. Scan all photos, detect faces and save them as new images with dimension of 299 * 299 pixels.

  ii. Hand pick eligible photos and save them into respective folders.

  iii. Apply Google Inception V3 model to extract the feature vectors of each face image.

  iv. Initial try on applying the different machine models.

III. Refinement.

  i. Use image generator to generate more images.

  ii. Apply K-fold cross validation.

IV. Performance metrics and benchmark. Linear classifier as the baseline. Evaluate the accuracy (F1 score) and time spent for both training and testing.

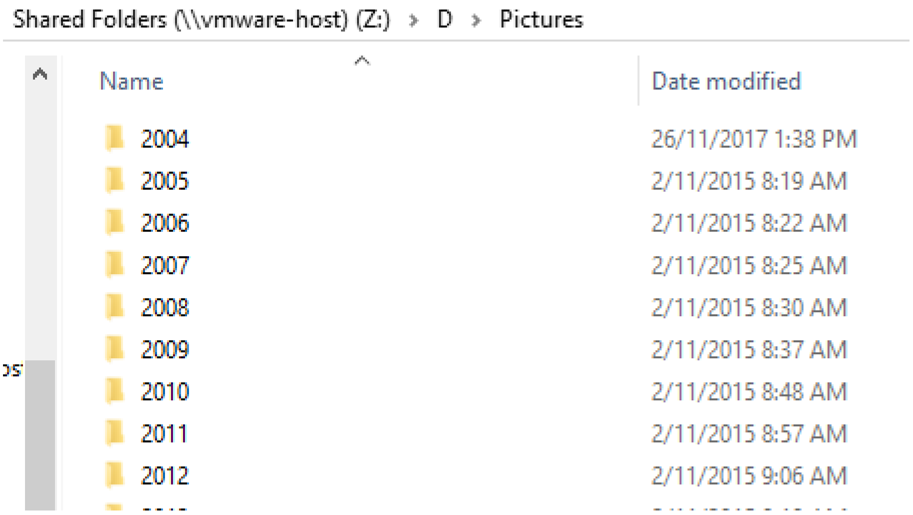
V. Conclusion.

II. Analysis and Data Preparation

A. Data Exploration

All my photos are in D:\Pictures, majority of them are in both .jpg and .nef format. The .nef is a raw image format for Nikon cameras and .jpg is the copy after image post-processing of raw file.

The output of below code chunk shows the root directory of my photo library. For each photo its full address always follows D:\Pictures\[yyyy]\[yyyy.mm.dd] - [event name]\[yyyymmdd-hhmm][-index].jpg

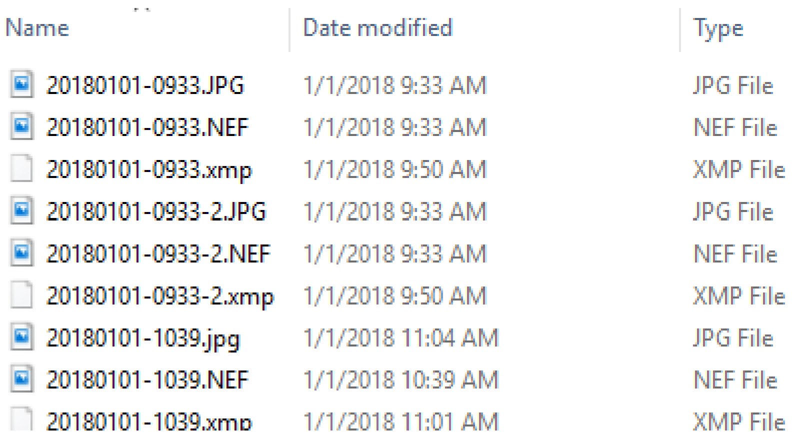


Shared Folders (\\vmware-host) (Z:) › D › Pictures

Name	Date modified
2004	26/11/2017 1:38 PM
2005	2/11/2015 8:19 AM
2006	2/11/2015 8:22 AM
2007	2/11/2015 8:25 AM
2008	2/11/2015 8:30 AM
2009	2/11/2015 8:37 AM
2010	2/11/2015 8:48 AM
2011	2/11/2015 8:57 AM
2012	2/11/2015 9:06 AM

Figure 1. A glance of the folder structure

Each and every file use the time stamp as its file name. .nef is the raw image file, .xmp is generated by Adobe Lightroom, both are not applicable to this project. Only .jpg files are applicable to this project.

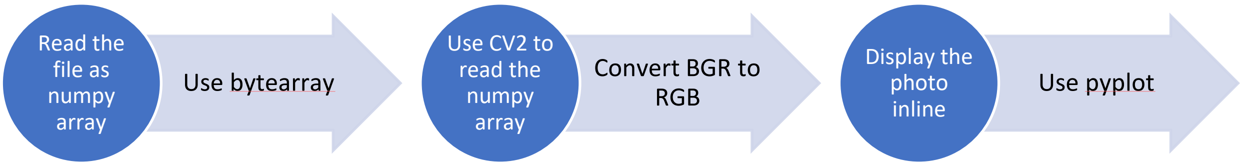


Name	Date modified	Type
20180101-0933.JPG	1/1/2018 9:33 AM	JPG File
20180101-0933.NEF	1/1/2018 9:33 AM	NEF File
20180101-0933.xmp	1/1/2018 9:50 AM	XMP File
20180101-0933-2.JPG	1/1/2018 9:33 AM	JPG File
20180101-0933-2.NEF	1/1/2018 9:33 AM	NEF File
20180101-0933-2.xmp	1/1/2018 9:50 AM	XMP File
20180101-1039.jpg	1/1/2018 11:04 AM	JPG File
20180101-1039.NEF	1/1/2018 10:39 AM	NEF File
20180101-1039.xmp	1/1/2018 11:01 AM	XMP File

Figure 2. A glance of photo files

B. Sample Photo Visualization and Pre-processing

The photo file will be read from the disk as numpy array and then displayed. In this project and my particular case, some special cases were met during the workflow as I have a lot of phots whose full paths contain Chinese characters. It’s not a big challenge, just slightly different when dealing with all English characters. Below figure illustrates the process.



Read the file as numpy array

Use bytearray

Use CV2 to read the numpy array

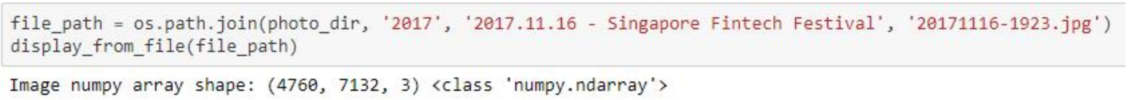
Convert BGR to RGB

Display the photo inline

Use pyplot

Figure 3. Process of reading and displaying a photo

And below is the result of a sample photo.



```
file_path = os.path.join(photo_dir, '2017', '2017.11.16 - Singapore Fintech Festival', '20171116-1923.jpg')
display_from_file(file_path)
```

```
Image numpy array shape: (4760, 7132, 3) <class 'numpy.ndarray'>
```

C. Algorithms and Techniques.

a) Face Detector

In this project, I used Haar Cascades Classifier from OpenCV for face detection. This classifier extracts thousands of features for each possible sub-window in the image. That will result in a huge amount of computation to determine whether this specific sub-windows is a face or not, not even mentioning this will be done for all the possible sub-windows. So the classification for each sub-windows will be gone through by stages. If at an earlier stage 10 features are used to determine whether the sub-window is a face or not, then classification will only continue if the result at this stage is positive. In this way, time won't be wasted to check the remaining thousands of features.

Another important technique used is the 'cascade' way. Instead of having one powerful classifier which could be too difficult to train, a bunch of weak classifiers are used. As long as all the weak classifiers are doing better than random guess, then majority vote can be used for the final decision/classification result.

The Haar Cascades Classifier is fast, but both the precision and recall rates are not perfect. Luckily I have a large raw dataset and I don't think it's a concern in this project.

b) Extract Feature Vectors

Below is the entire structure of Google Inception V3 model. The target layer I want to extract is the average pooling layer as indicated.

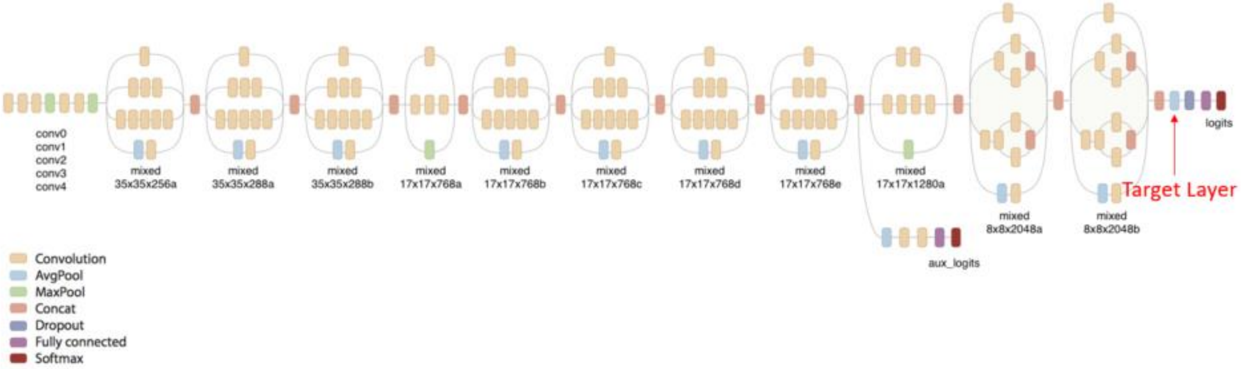
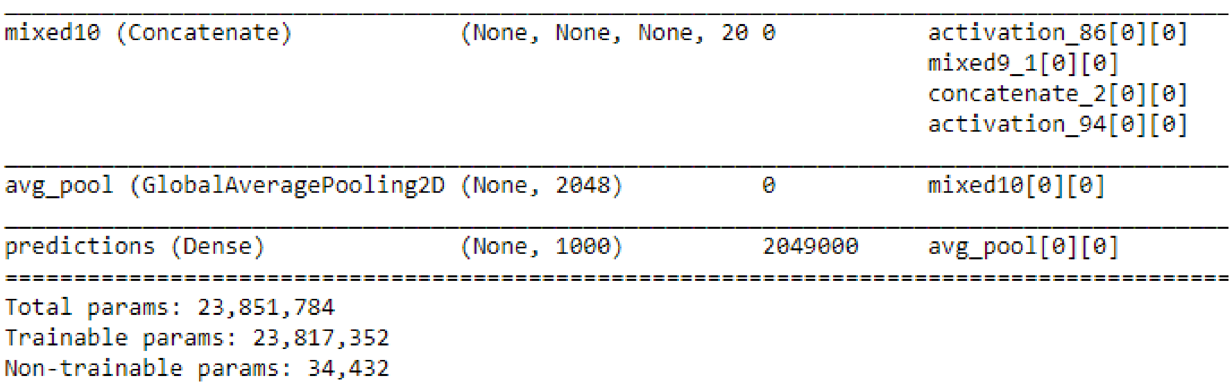


Figure 4. Google Inception V3

Based on the below model summary screenshot, the output of this layer is a vector with size 2048. And that vector is going to be the feature input of the machine learning algorithms in this project.



```
_______________________________________________________________________________________
mixed10 (Concatenate)            (None, None, None, 20 0           activation_86[0][0]
                                                                    mixed9_1[0][0]
                                                                    concatenate_2[0][0]
                                                                    activation_94[0][0]
_______________________________________________________________________________________
avg_pool (GlobalAveragePooling2D (None, 2048)          0           mixed10[0][0]
_______________________________________________________________________________________
predictions (Dense)              (None, 1000)          2049000     avg_pool[0][0]
=======================================================================================
Total params: 23,851,784
Trainable params: 23,817,352
Non-trainable params: 34,432
```

Figure 5. Google Inception V3 summary

The Inception model has some requirements of input data feed into it. The data has to be in 4D tensor shape, value need to be normalized from [0, 255] to [0, 1.0] and below shows the workflow. Eventually a 4D numpy array with shape (x, 299, 299, 3) is obtained.

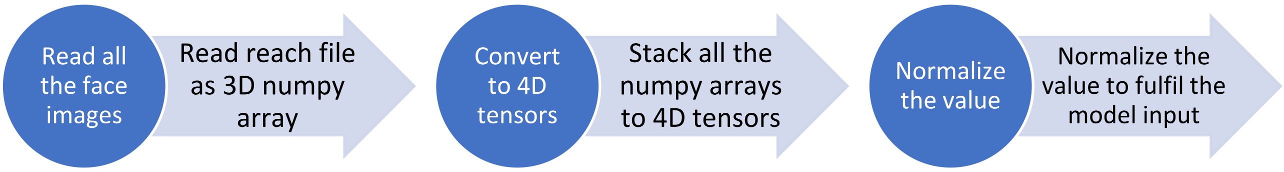

Read all the face images
Read reach file as 3D numpy array
Convert to 4D tensors
Stack all the numpy arrays to 4D tensors
Normalize the value
Normalize the value to fulfil the model input

Figure 6. From images to 4D tensors

c) Machine Learning Models

Four models will be evaluated. SGD Classifier, K-nearest Neighbour, Logistic Regression and Deep Learning model.

SGD Classifier

Both SGD Classifier and Deep Learning using Stochastic Gradient Descent optimizer work quite similar as the 'SGD' way. When one or a batch of samples feeding into the model, a loss (the distance between output and expected output) will be computed. Then this loss value will be used to update the parameters/weights of the model to let it lean towards the expected output.

The SGD classifier prefers data been preprocessed to be normalized with zero mean and unit variance. In this report, the preprocessed data will be normalized from [0, 255] to [0, 1]. Not exactly as what's favored by SGD Classifier, but I standardize it to [0, 1] for all the models here. Default hyper parameters will be used.

K-nearest Neighbour

Instead of constructing a generalized model, KNN is storing all the training data. When performing the prediction, distance (eg, euclidean distance) from the testing data point to all the training data point will be computed. Then based on the defined K value, using majority vote to decide the class of the testing data point. After a few tries, I set K=7.

KNN is simple but it can be very time consuming during testing as distance to all training data are needed. In addition, due to the curse of high dimensionality, it may not be effective always.

Logistic Regression

Despite the 'regression' word in the name, it's indeed a classification model. Logistic regression takes the features into a logit function with output value in the range of [0, 1], which quantifies the probability of correctly predicting the class. When all the training points fed into the model, we get the average of the probability and that's the likelihood the training data is correctly predicted. So the objective in logistic regression training is to maximize this likelihood. There are many different algorithms, e.g. Newton-Raphson, Iteratively re-weighted least squares and etcs.

Logistic regression is widely used industrial widely. But it doesn't perform well when the feature space is large and I have no idea whether it'll do a great job in this project. Default hyper parameters will be used.

Deep Learning

Without a doubt, deep learning has gained great exposure and evolvement over the past few years, especially in the domains of computer vision, natural language processing. In this project, the feature extractor is taking almost 95% of the Google Inception V3 layers. Instead of taking the remaining 5% layers, which will do a job to classify objects into 1000 classes, I will add another few dense layers to do the customization to suit my purpose of classifying objects into 7 classes. Essentially, this is just a transfer learning approach.

Deep learning takes a lot of computation resources and time to do the gradient descent to find the best weights of the neurons. And in this project, I don't need to worry about the convolutional layers except the last few dense layers added by me.

d) Creating More Data

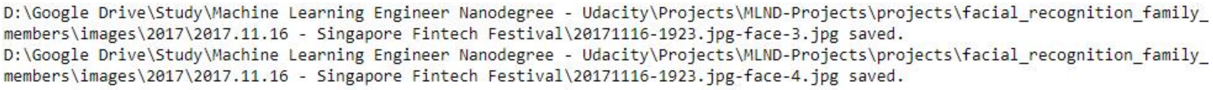
Despite I have a large photo library, there is still a concern on the quantity of the images with good quality. Image generator will be explored if needed

III. Implementation

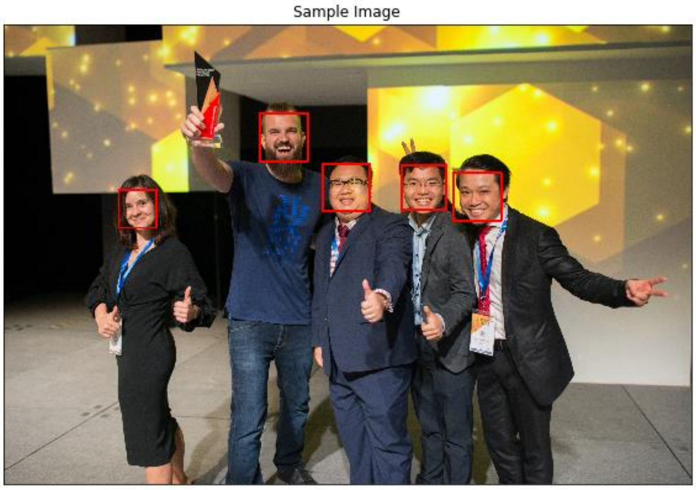
A. Detect Faces, Save as New Files

The objective of this project is for face recognition, it'll be time-consuming, or even non-sense if feeding the above entire photo to the machine learning models. OpenCV will be used in this project for face detection. And the detected faces will be resized to 299 * 299 pixels and saved as new files in a different directory for later machine learning pipeline.

Below figures illustrate that 5 faces detected and saved as new files.

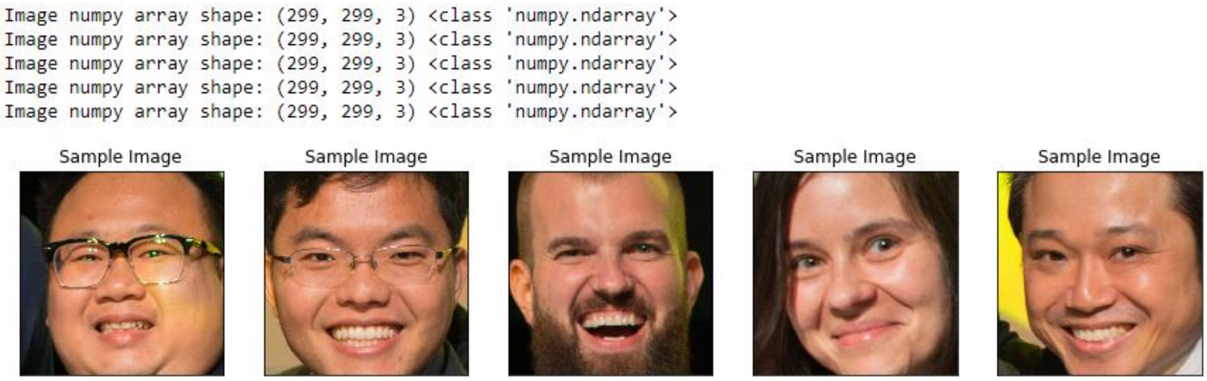


```
D:\Google Drive\Study\Machine Learning Engineer Nanodegree - Udacity\Projects\MLND-Projects\projects\facial_recognition_family_
members\images\2017\2017.11.16 - Singapore Fintech Festival\20171116-1923.jpg-face-3.jpg saved.
D:\Google Drive\Study\Machine Learning Engineer Nanodegree - Udacity\Projects\MLND-Projects\projects\facial_recognition_family_
members\images\2017\2017.11.16 - Singapore Fintech Festival\20171116-1923.jpg-face-4.jpg saved.
```



Sample Image

Figure 7. Faces detected and saved as new files



```
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
```

Figure 8. Files can be read and with expected size

B. Batch Process All Photos and Manual Labelling

Once one photo can be processed like what's been done above, it's just a matter of time to batch process all the 40k photos, indeed it took 13 hours.

After that, I got 100k face photos. Due to the low accuracy of OpenCV face detection. Many of them are not faces.

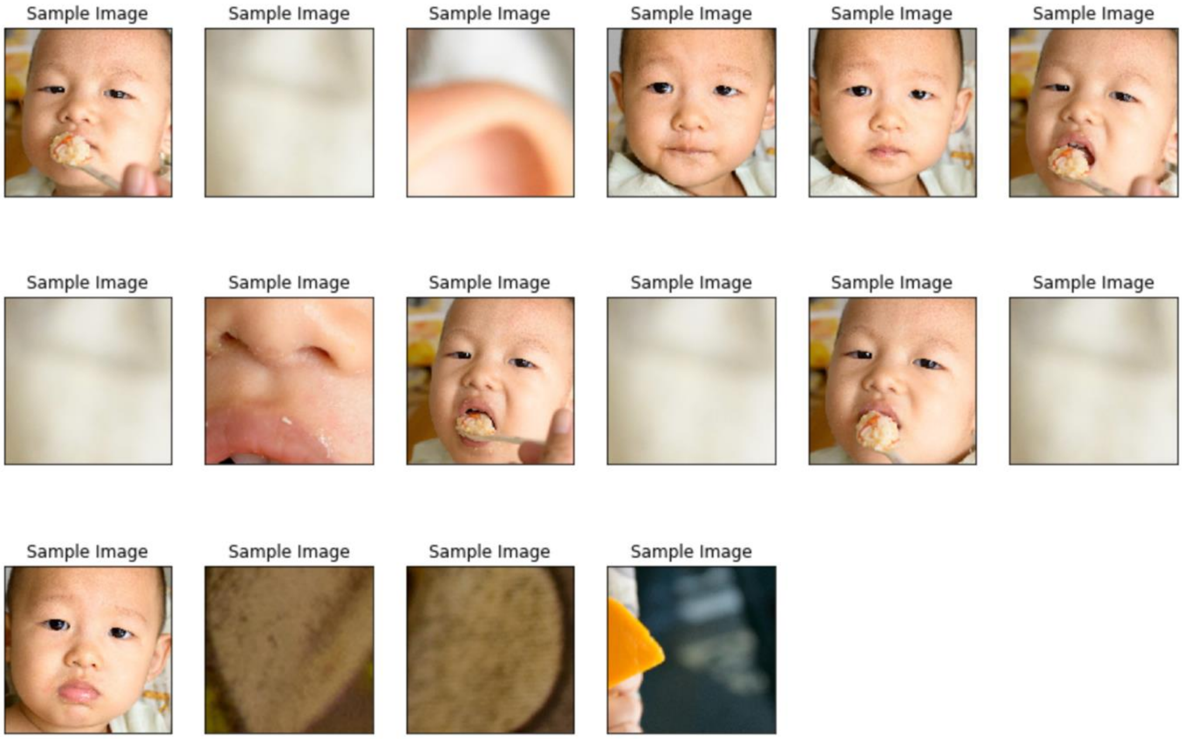


Figure 9. Sample photos after face extraction

I spent about 2 hours to hand pick about 500 photos and below are the distributions for the 7 categories.

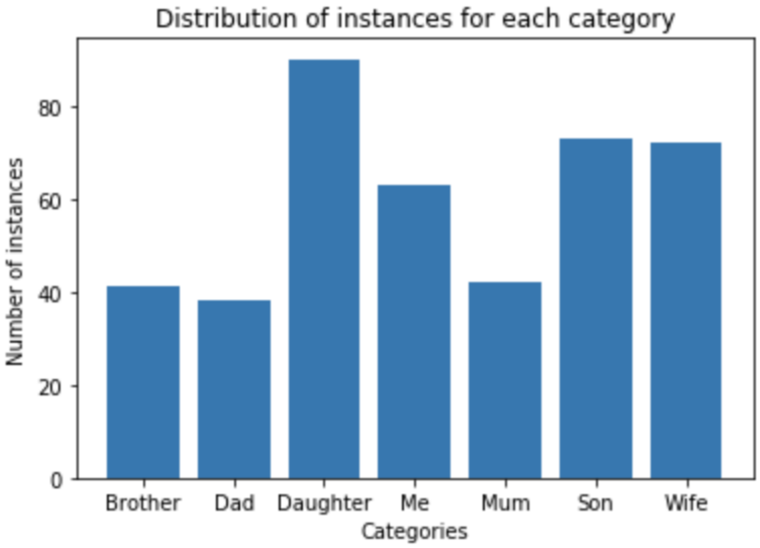


Figure 10. Distribution of the hand-picked photos

The above bar plot shows the distribution of samples of each category. “Daughter” category has the most images, 92. “Son” has the second most, 73, so on and so forth. The distribution of samples are not well balanced. This observation will lead to the stratified split in the later sections.

C. Split the Dataset

Due to the slightly imbalance of the data for each category, I used stratified split in this project. 70% for testing, 15% for validation and 15% for testing.

D. First Attempt for Each Model

Four models were explored, they are SGD Classifier, KNN, Logistic Regression and DNN. The first 3 models I used default parameters and I used 3 dense layers for the DNN model.

The number of epochs was set to 50, based on the loss and accuracy of the validation dataset in the DNN model.

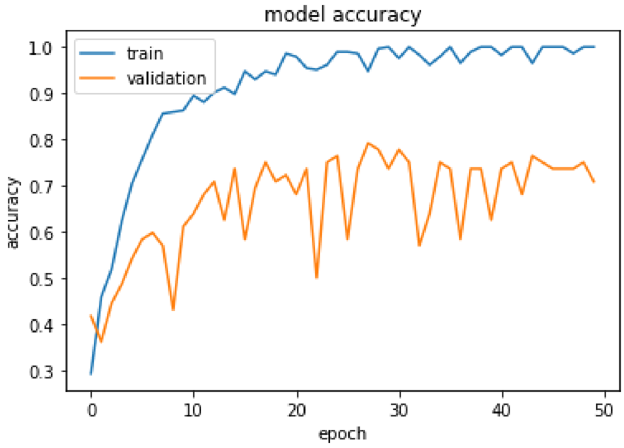


Figure 11. DNN model accuracy

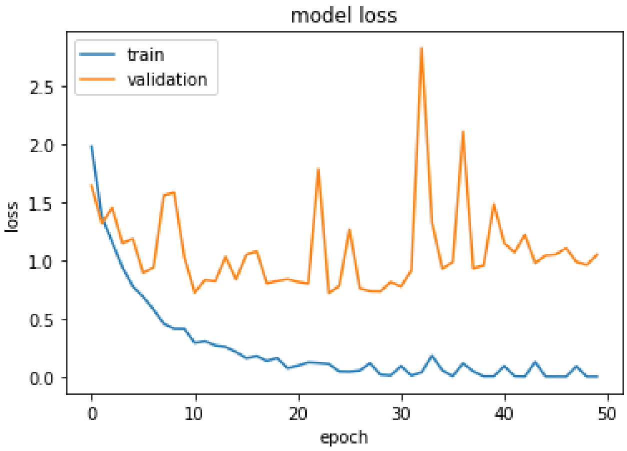


Figure 12. DNN model loss

Below table is the summary of accuracy and time taken.

Model	**Training time (s)**	**Testing time (s)**	**Accuracy (F1 score)**
SGD Classifier	0.063	0.006	78%
KNN	0.017	0.048	68%
Logistic Regression	0.47	0.001	79%
DNN	6.16	0.217	75%

Based on the current split of the dataset, logistic regression performed the best. KNN reached merely 70% while SGD classifier, logistic regression and DNN reached accuracy of 75% to 80%. DNN took significant longer time on training.

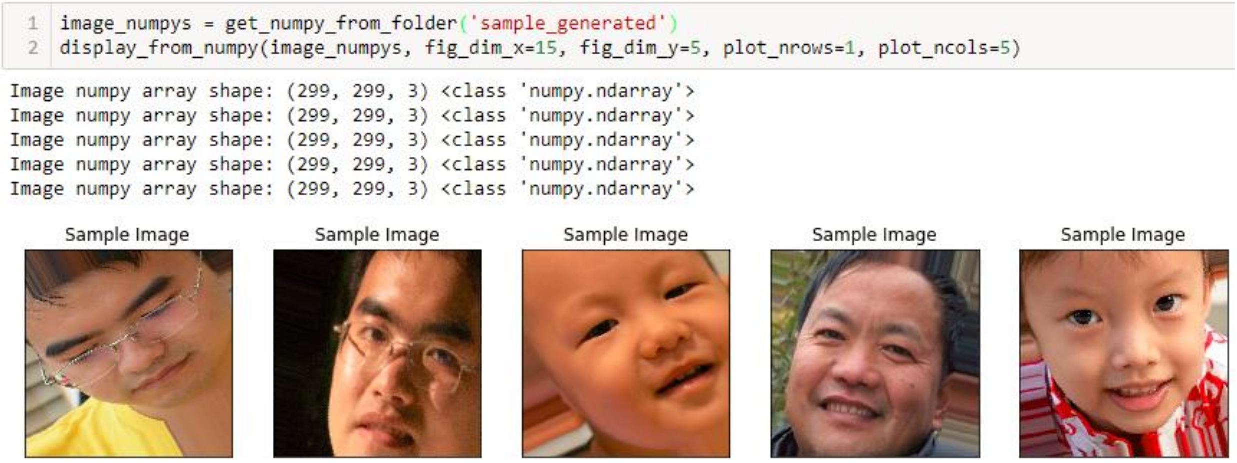
But what if it's just a coincidence, what if I had more images? The next section will look into the refinement of the models from two ways. The first one is to use image generator to have a bigger dataset. The other one will focus on using cross validation to determine the best model.

IV. Refinement

A. Create More Images

Use image generator to create more images. This is a very useful technique when having little data. In this project, I put the new images in a new folder and 10 new images will be generated for each original image. Generating new images is basically to do some distortion on the images. I think it's reasonable to narrow, widen, rotate, zoom and flip the images a little, in a reasonable range. Hence, values of the most parameters I set to 20%.

Below is a peak on what's been generated. The images are distorted, rotated, filled with near pixels and etcs.



```
image_numpys = get_numpy_from_folder('sample_generated')
display_from_numpy(image_numpys, fig_dim_x=15, fig_dim_y=5, plot_nrows=1, plot_ncols=5)
```

```
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
Image numpy array shape: (299, 299, 3) <class 'numpy.ndarray'>
```

Figure 13. Generated images

B. K-fold Cross Validation

Even though splitting the dataset into training and testing is random, there is still a chance that this specific split favoured one model over the other. So, k-fold cross validation will be employed in this project.

I will choose k = 10. The original dataset of 419 image will be split into 10 folds. Each of the training and evaluation hold 1 fold as the testing dataset and use the other 9 folds to generate the new training images. The generated new image from the 9 testing folds will then be used to train the 4 models mentioned earlier. Performance on accuracy and testing time will be measure against the 1 original untouched testing fold.

V. Results and Benchmark

A. Summary of Performance

After the refinement section. Below are the summary of performance in terms of precision, recall, F1 scores and time spent.

Model	**Precision**	**Recall**	**F1**
SGD Classifier	79.5%	79.2%	79.2%
KNN	73.2%	72.3%	72.6%
Logistic Regression	83.5%	82.8%	82.6%
DNN	80.1%	79.5%	79.3%

Model	**Mean Training Time (seconds)**	**Mean Testing Time (seconds)**
SGD Classifier	0.31	0.0004
KNN	0.7751	0.5107
Logistic Regression	13.1209	0.0003
DNN	46.7640	0.3856

The linear model (SGDClassifier) is doing quite ok in the first try and refinement section. It is almost the fastest one on both training and testing. And its performance is the baseline of this project's benchmark.

KNN doesn't perform well in terms of accuracy (F1 score). I guess it's due to the curse of dimensionality. It also takes a significant longer time on testing.

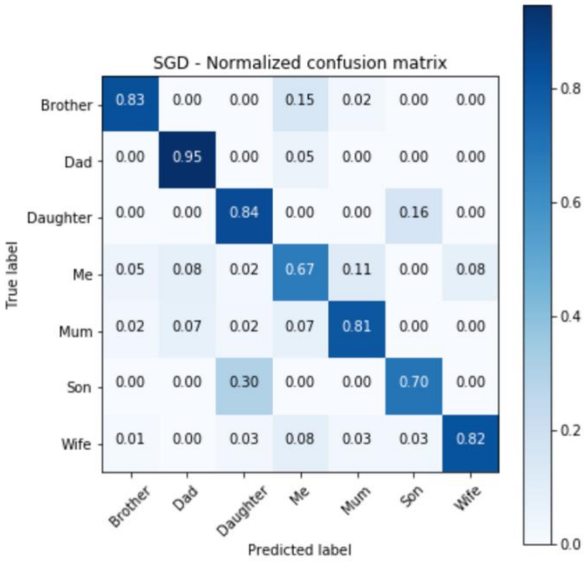
Logistic regression did a better job on accuracy (F1 score) but took a longer time on training.

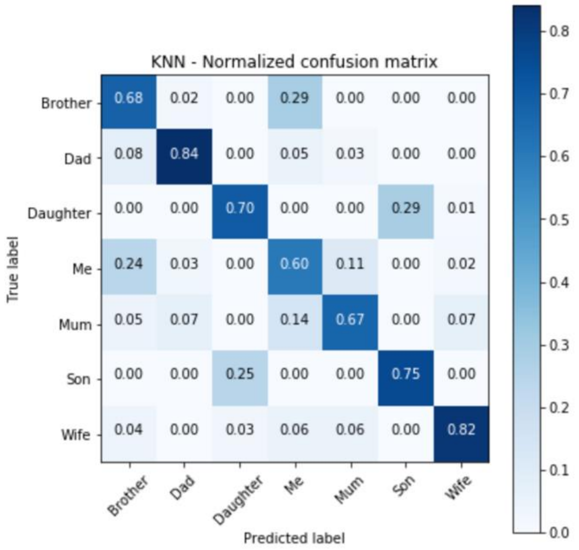
DNN performed almost the same in terms of F1 score. But it took significant longer time on both training and testing.

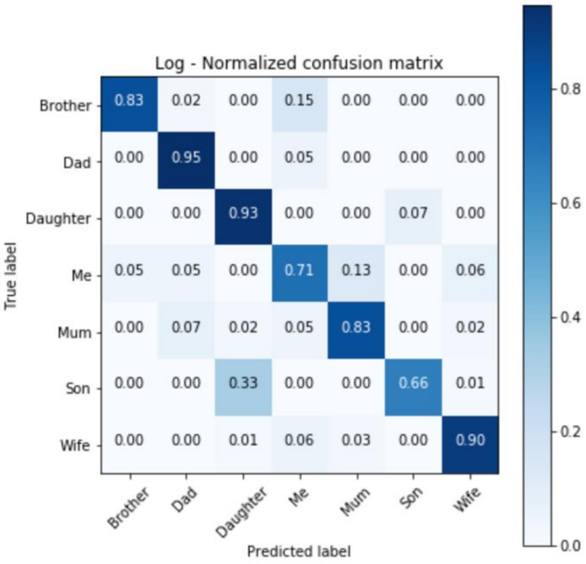
B. Detailed Look into Score

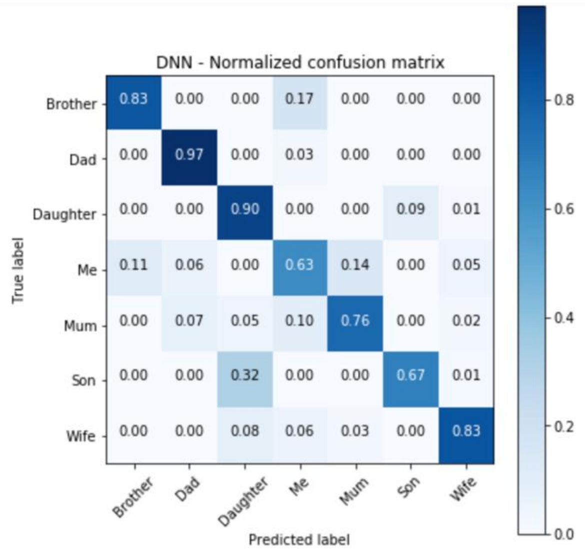
Despite the summary of the scores was shown in the previous section, but the distribution of accuracy for each true and predicted label is still unclear. Plotting the confusion matrix into color bar makes it more intuitive.

The ideal case will be all the squares along the diagonal line are dark blue while the rest are white.









C. Detailed Look into Time

The time spent for training and testing are as below boxplot. As mentioned, SGD is the fastest, KNN takes the longest time on testing, DNN is significant slow on both training and testing, while logistic regression has longer time on training.

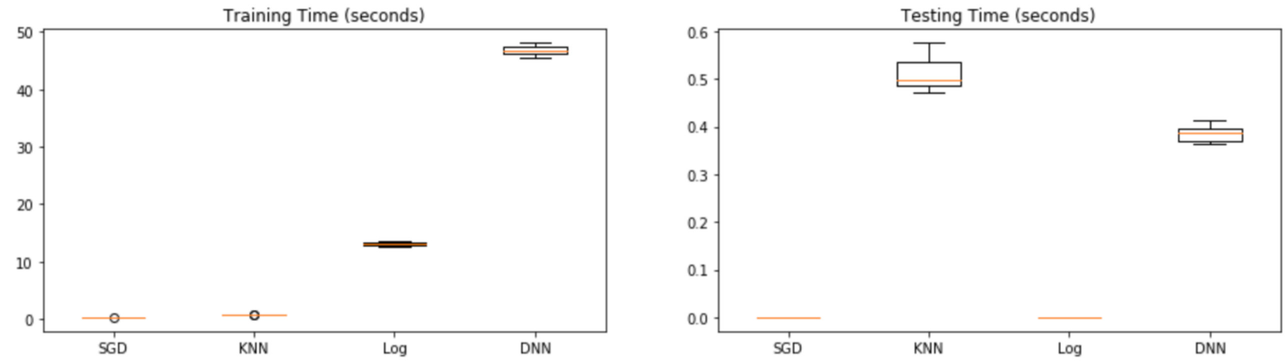


Figure 14. Time spent

VI. Conclusion

Among all the 4 models, logistic regression is the one giving highest accuracy and also acceptable training and testing time.

The accuracy of logistic regression model is 5% to 10% higher than the other 3 models. The training time is longer than SGD classifier and KNN, but not as much longer as DNN. The testing time is very fast, and actually it's the fastest one.

KNN doesn't perform so well in terms of accuracy. I guess it's due to the curse of high dimensionality.

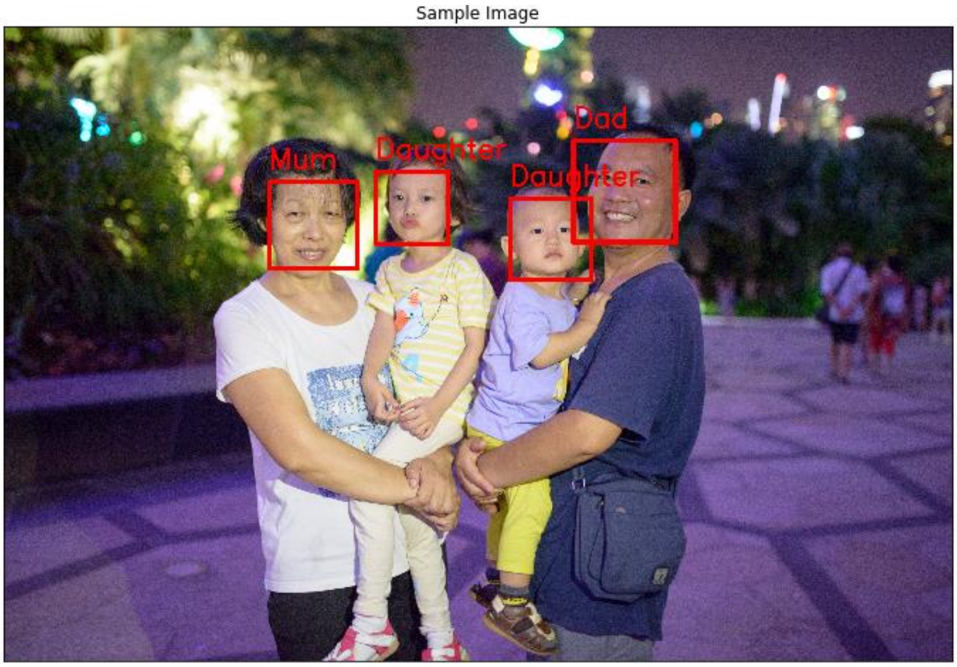
The DNN model in this project is exactly a transfer learning approach. Its accuracy is quite good, but it takes a long time to train and it's also costly. In this project I'm using GTX 1070 Ti, a GPU costs around USD 500. If the training is under the same CPU environment, I won't doubt it'll take at least 10 more times of current time.

The accuracy of 79% from SGD classifier is the base of the benchmark. Logistic regression model has an average accuracy of almost 83% which pushed the boundary by few percent. The only cost is that logistic regression model takes longer time to train. If time is a big concern, probably SGD classifier is the best in terms of both accuracy and time. Otherwise, logistic regression still remains the best accuracy.

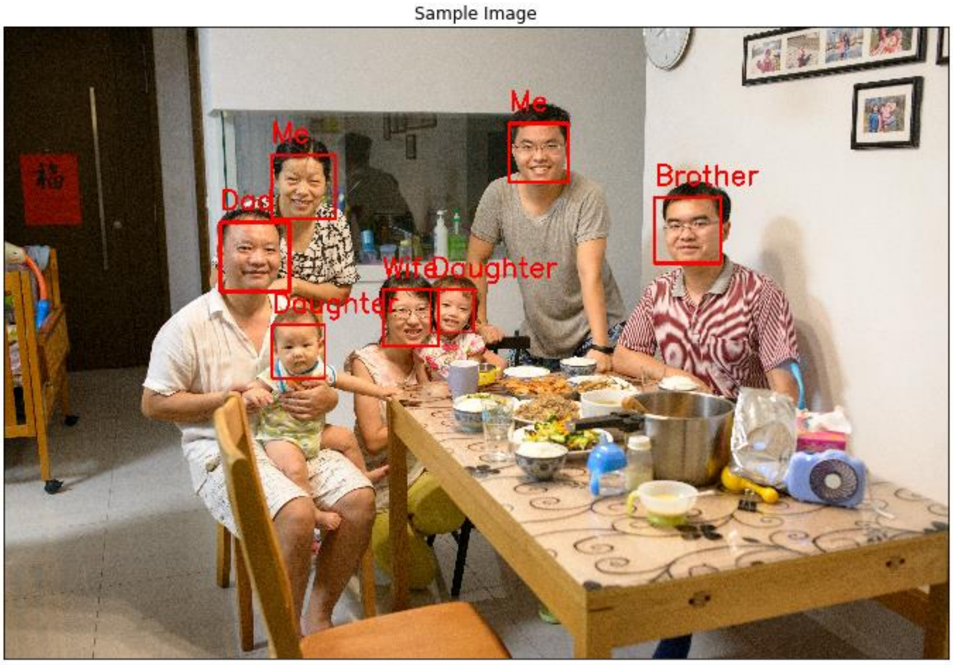
I will choose logistic regression as my final winning model in this project. It doesn't show a perfect result on the accuracy, especially there are only 7 categories in this project. I doubt it'll work well in the real-world scenario that people need to perform face recognition among thousands to millions of faces. However, it still shows the effectiveness of feature engineering by leveraging transfer learning and applying 'traditional' machine learning models.

One possible way to improve the model may be using transfer learning to detect the facial key points (centre and corners of eyes, mouth, nose and etc.) first. Then build mathematic models to represent the relative positions of the key points. Normalization may be needed to make the face centred and 'starring' at the camera. Then the recognition problem will become to compute similarity of the key points map.

VII. Fun Part



Sample Image



Sample Image

The logistic regression model showed almost 85% accuracy in the earlier section. I have chosen 2 photos that never used for training or testing to evaluate the performance of the model in the real-world scenario. The accuracy is not as good as expected, especially the 2nd image has 2 misclassified faces. Well, perfect accuracy is not the purpose for this project. The most important thing is, it's fun and it works!