COS30019 INTRODUCTION TO ARTIFICIAL INTELLIGENT SEMESTER 1 2020

ASSIGNMENT 2

Prepared by:

Group 7

Lai Kok Wui (101211447)

Yeat Nai Cheng (100078269)

Didier Luther Ho Chih Yuan (101214093)

Prepare for:

Joel Than

Contents

1 Introduction

In order to help Mr. Joel to differentiate whether today Suzy is in a good day or a bad day. The team had created an image classification using two different classifiers which are machine learning and deep learning to classify the images sent by Suzy. We choose Random Forest Classifier for the machine learning and convolutional neural network for the deep learning. We are using 3 sets of the training data set which are FEI (Brazil), JAFFE(Japanese) and combination of FEI and JAFFE for training model.

The source code can be found in the folder “Assignment2_CODEANDDATA”

- Assignment2_ML_RFC.ipynb

- Assignment2_DL_CNN.ipynb

The link of the demonstration video was shown below:

https://www.youtube.com/watch?v=z_-QyAeyKz4

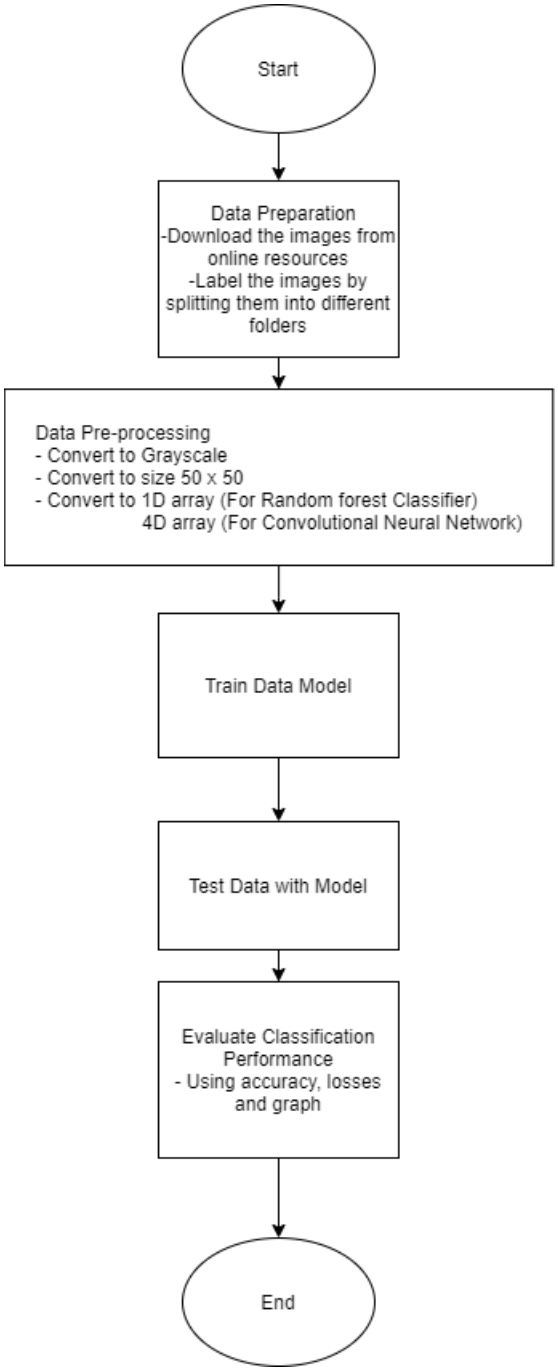

Figure 1 Flow Chart of Machine Learning Process

The flow chart above shows the general process of the machine learning which needs to be done to classify the images.

2 Contribution

Team Member	Tasks Contributed	Signature
Lai Kok Wui	Trained Data Set Preparation and Pre-processed	
Didier Luther Ho Chih-Yuan	Train the Classifier Model	
Yeat Nai Cheng	Test Data and Evaluate Data	

Requirements	Maximum Mark (20)	Actual
Program – correct implementation and demonstration	6	
Report – data preparation and pre-processing	2	
Report – training	5	
Report – evaluation	4	
Report – discussion & conclusion	3	

3 Data preparation and pre-processing

There are three datasets chosen by the team for creating image classification, namely JAFFE dataset and FEI dataset where the former contains of the facial expressions posted by 10 Japanese female models and the latter is a dataset containing facial expression posted by Brazilian. As for the third dataset is the combination of the JAFFE dataset and FEI datasets mentioned above.

After the selection of dataset, the classification process is been proceeded to split the images from the dataset into bad and good classes. This is done manually and the filter condition being the facial expression, whether it is happy, sad, angry or neutral. If the image shows a “happy” expression, then the said image will be included into the “good” class. The same principle is applied to “neutral” and “angry” facial expression where the class would be “bad” instead of “good”.

The filtering process has proven that the number of bad facial expression is more than the number of good expressions. Hence, a further filtering is performed to balance the number of images of the 2 classes in order to obtain a better result when performing the randomization during the training process. Hence, a further filtering is performed

The images are then being converted to gray scale with the size of 50x50. The conversion standard is the same for the Machine Learning model and the Deep Learning model. However, the images are converted to 1D array in the Machine Learning model, whereas the images are converted into 4D array in the Deep Learning model. The differences in the conversion standard is due to the requirements of the modelling functions to execute.

The diagram below shows the codes used in both machine learning model and the deep learning model for the image conversion based on the standard stated above.

```
IMG_SIZE = 50 #set image size
training_data = []

def create_training_data():
    for category in CATEGORIES:  # do bad and good

        path = os.path.join(DATADIR,category)  # create path to bad and good
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1). 0=bad 1=good

        for img in tqdm(os.listdir(path)):  # iterate over each image per bad and good
            try:
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to normalize data size
                training_data.append([new_array, class_num])  # add this to our training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
```

Figure 2: Image pre-processing

One of the examples on the application of facial expression recognition would be crime prevention. According to asmag (2017), the process of person identification in search of fugitive criminal or terrorist and crowd monitoring to spot suspicious behaviors via their emotional state can potentially reduce the crime rate.

4 Training

There are a few machine learning algorithms that can be used for image classification.

Firstly, Random Forest is a machine learning algorithm developed by Leo Breiman back from 2001 (Breiman 2001, p. 5). It has the advantage of evaluating the contribution of used features to the model compared to neural networks and support vector machines. This algorithm creates many subsets by bootstrap sampling which is known as restoration extraction of supervised data, and various decision trees that are constructed for each subset. Since the predicted values of each decision tree are different, the final predicted value will be obtained by using the majority decision for the discrimination problem and the average value for the regression problem.

Secondly, Support Vector Machines also known as SVM is a linear model for classification and regression problems. According to Pupale (2018) mentioned that SVM finds a hyperplane between data points of two classes by taking the input data and then outputs a line that separates those classes. He also stated that SVM find points or support vectors that are closest to the line from both the classes. Next, it will compute margin by calculating the distance between the line and the support vectors. SVM goals is to find the optimal hyperplane by maximizing the margin (Pupale 2018).

Thirdly, Naïve Bayes is a classifier that is used for classification task (Gandhi 2018). The crux of the classifier is based on the Bayes theorem. Ganndhi (2018) mentioned that there are a few types of Naïve Bayes classifiers such as Multinomial Naïve Bayes, Bernoulli Naïve Bayes and Gaussian Naïve Bayes. He explained that Multinomial Naïve Bayes is to classify the category of the document which are mostly use for classifying the frequency of the words presented in the documents. Bernoulli Naïve Bayes is almost like Multinomial Naïve Bayes except it output everything in Boolean values. Gaussian Naïve Bayes on the other hand is used to sample the values from a gaussian distribution.

Next, K-Nearest Neighbors is a supervised machine learning algorithm also known as KNN (Harrison 2018). KNN relies on label data to help make predictions to the given

new unlabeled data. After the image data is loaded, the algorithm will initialize the K to the selected number of “neighbors” to be grouped. Next, the distance between the query example and the current example from the data, and the sum of the distance between the example’s index to the ordered collection is calculated. The 2 calculation processes above is repeated for the number of examples in the data feed into the algorithm. The first K entries from the sorted collections are selected to obtain the label of the chosen K entries and perform the following based on the method used (regression/classification).

- Regression: return the mean of K labels
- Classification: return the mode of K labels

One of the most commonly used Deep Learning algorithm used for image classification is the Convolutional Neural Network (CNN). According to Saha (2018), the CNN algorithm can receive input images and differentiate the aspects within the image by assigning the weights and biases to numerous aspects that can be found within the said image. He continues writes that compared to other algorithm, the preprocessing needed for CNN to perform the similar tasks is lower as it can learn the filtering conditions needed in contrast to that in other algorithm where the conditions are hard-coded.

Finally, the team used random forest algorithm for the machine learning model and the CNN algorithm for the deep learning model.

4.1 Random Forest

As explained above, the Random Forest will create several decision trees for the given training dataset where the training process is performed with the help of the class ‘RandomForestClassifier’ of sklearn. The codes used for the training process of Random Forest is show below:

```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, BaggingRegressor, RandomForestRegressor, GradientBoostingRegresso
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import plot_confusion_matrix
from sklearn import metrics


import sklearn
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=10)
rfc.fit(X_train, y_train)
```

Figure 3: Random Forest - Library Import

As shown in the figure above, after importing the libraries needed for the training process, the code containing RandomForestClassifier() and .fit() functions from the imported libraries are used, where the n_jobs represents the number of jobs run in parallel and by assigning the "-1" means securing and using all the processors available; whereas n_ estimators represent the number of decision trees in the Random Forest algorithm (データ科学便覧 2017).

4.2 CNN

When the CNN algorithm receives an image, it will process the image while ensuring the features are not lost during the process. The image will go through a process called convolution. The is a convolution is a process that takes 2 inputs, namely the gird-shaped numerical data called kernel (filter) and part of the input image that have the same size as the kernel called window, where the window is shifted little by little to convert the numerical data called tensor (TensorFlow 2018). The tensor obtained from the convolution process is called as the Feature Map which is the characteristic quantity extracted by the kernel and can produce various information by the kernel from the Feature Map. Through this process, the algorithm can learn and identify the features/characteristics of the image.

```
import pickle
TRAINED_DATASET_X="X_FEI_4D.pickle"
TRAINED_DATASET_y="y_FEI_4D.pickle"
pickle_in = open(TRAINED_DATASET_X,"rb")
X = pickle.load(pickle_in)

pickle_in = open(TRAINED_DATASET_y,"rb")
y = pickle.load(pickle_in)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

Figure 4: CNN - Dataset Loading & Library Importing

The diagram above shows the codes that are used to load the processed datasets and imports the libraries needed for the image classification.

```
y=np.array(y)
X = X/255.0

model = Sequential()
model.add(Conv2D(256, (3, 3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))
```

Figure 5: CNN - Feature Maps

The diagram above shows the first block of codes are used to convert the y dataset into a NumPy array and divide the X dataset by 255. The second and third block however, functions to create a sequential model (Feature Map) and pass the data with the set

parameters into it. The program continues with a line of code of to convert the Feature Map into 1D feature vectors which is followed with the codes to pass in further data into the model, which is now a 1D feature vector.

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(X, y, batch_size=32, epochs=8, validation_split=0.3)
# Classification performance
score = model.evaluate(X, y)
print('loss=', score[0])
print('accuracy=', score[1])
```

Figure 6: CNN - Model Training

This diagram shows the block of codes that are responsible to perform the training process by compiling from the 1D feature vectors model with the parameters as shown above. The “batch_size” means the number of samples per gradient update; the epochs are defined as the number of iterations that the algorithm is going to run with the entire dataset; lastly the validation_split is to select a fraction of the dataset that will be used to evaluate the model metrics at the of each epoch (TensorFlow n.d.).

```
#Save model into a hdf5 file
hdf5_file = "CNN_TRAINEDMODEL_FEI.hdf5"
model.save_weights(hdf5_file)
```

Figure 7: CNN - Save Output

```
Epoch 1/8
9/9 [==============================] - 3s 342ms/step - loss: 0.8249 - accuracy: 0.4821 - val_loss: 0.6915 - val_accuracy: 0.5583
Epoch 2/8
9/9 [==============================] - 3s 344ms/step - loss: 0.6930 - accuracy: 0.4750 - val_loss: 0.6933 - val_accuracy: 0.4417
Epoch 3/8
9/9 [==============================] - 3s 332ms/step - loss: 0.6931 - accuracy: 0.5250 - val_loss: 0.6946 - val_accuracy: 0.4417
Epoch 4/8
9/9 [==============================] - 3s 325ms/step - loss: 0.6928 - accuracy: 0.5250 - val_loss: 0.6948 - val_accuracy: 0.4417
Epoch 5/8
9/9 [==============================] - 3s 324ms/step - loss: 0.6918 - accuracy: 0.5250 - val_loss: 0.7023 - val_accuracy: 0.4417
Epoch 6/8
9/9 [==============================] - 3s 320ms/step - loss: 0.6947 - accuracy: 0.5250 - val_loss: 0.6966 - val_accuracy: 0.4417
Epoch 7/8
9/9 [==============================] - 3s 330ms/step - loss: 0.6896 - accuracy: 0.5250 - val_loss: 0.6947 - val_accuracy: 0.4417
Epoch 8/8
9/9 [==============================] - 3s 338ms/step - loss: 0.6843 - accuracy: 0.5571 - val_loss: 0.6922 - val_accuracy: 0.4750
13/13 [==============================] - 1s 83ms/step - loss: 0.6807 - accuracy: 0.5300
loss= 0.6807345747947693
accuracy= 0.5299999713897705
```

Figure 8: CNN - Epoch Output

Figure 7 shows the codes to save the trained model after all the epochs are done into a hdf5 file whereas the second diagram illustrates the result/output of each epoch together with the number of loss and the accuracy.

5 Testing and Evaluation

After pre-processed the training data and trained the model, the next step is just to load in the test data and test the result. We will also evaluate the tested result and see the accuracy of the model.

5.1 Random Forest Classifier

5.1.1 Load Test Data Set

In order to test the test data set (Suzy's pictures), we need to preprocess the test data set same as the trained data set.

Step of Processing data:

1. Convert to grayscale.
2. Convert to pixel size of 50x50.
3. Convert into 1D array.

```
import os
from tqdm import tqdm

X_test_data = []
X_test_data_2D=[]
y_test_data=[]
files = []
files_path=[]
def loadall_test_data():

    DATADIR = "Assignment_Image"
    CATEGORIES = ["bad", "good"]
    for category in CATEGORIES:  # do bad and good
        path = os.path.join(DATADIR,category)  # create path to bad and good
        for img in tqdm(os.listdir(path)):  # iterate over each image per bad and good
            try:
                files_path.append(os.path.join(path,img))
                files.append(img) # Add file name
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to normalize data size
                X_test_data.append(new_array)  # add this to our test_data
                if(category==CATEGORIES[0]):
                    y_test_data.append(0)
                elif(category==CATEGORIES[1]):
                    y_test_data.append(1)

            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

loadall_test_data()
for b in X_test_data:
    X_test_data_2D.append(b.reshape(-1))
```

Figure 9 RFC_Load Test Data Set

The figure above shows that I tried to preprocess and load the test data set into a list.

5.1.2 Test Suzy Pictures

```
#Test the data
print("Accuracy")
rfc.score(X_test_data_2D, y_test_data)
y_pred=rfc.predict(X_test_data_2D)
print(rfc.score(X_test_data_2D, y_test_data))
print(y_pred)

for i, p in enumerate(y_pred):
    print("File Name:",files[i])
    print("File Path:",files_path[i])
    print("Label: ",CATEGORIES[p])
    print(i)
```

```
Accuracy
0.4
[0 0 1 0 1 1 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 0 0
 1 1 0]
File Name: 1.JPG
File Path: Assignment_Image\bad\1.JPG
Label:  bad
0
File Name: 10.JPG
File Path: Assignment_Image\bad\10.JPG
Label:  bad
1
File Name: 11.JPG
File Path: Assignment_Image\bad\11.JPG
Label:  good
2
File Name: 12.JPG
File Path: Assignment_Image\bad\12.JPG
Label:  bad
```

Figure 10 RFC Test Data

The figure above shows that Suzy's pictures are being put into the classifier and being classified. However, the classified result only has 40% of the accuracy. There are 3 different models which are produced by 3 different trained data set. The figure above shows the result for FEI and JAFFE trained data set. File path was being printed for

reference so that we know the actual label of the test picture. The predicted label was printed below the file path.

```
import matplotlib
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
f,ax=plt.subplots(1,1,figsize=(12,12))
# print("Classification report for classifier %s:\n%s\n"
#       % (rfc, metrics.classification_report(y_test, y_pred)))
disp = metrics.plot_confusion_matrix(rfc, X_test_data_2D, y_test_data,normalize='true',ax=ax)

# disp.figure_.suptitle("Confusion Matrix")
# print("Confusion matrix:\n%s" % disp.confusion_matrix)
```

Figure 11 RFC_Plot Confusion Matrix

The figure above shows the code for showing the confusion matrix of the random forest classifier result.

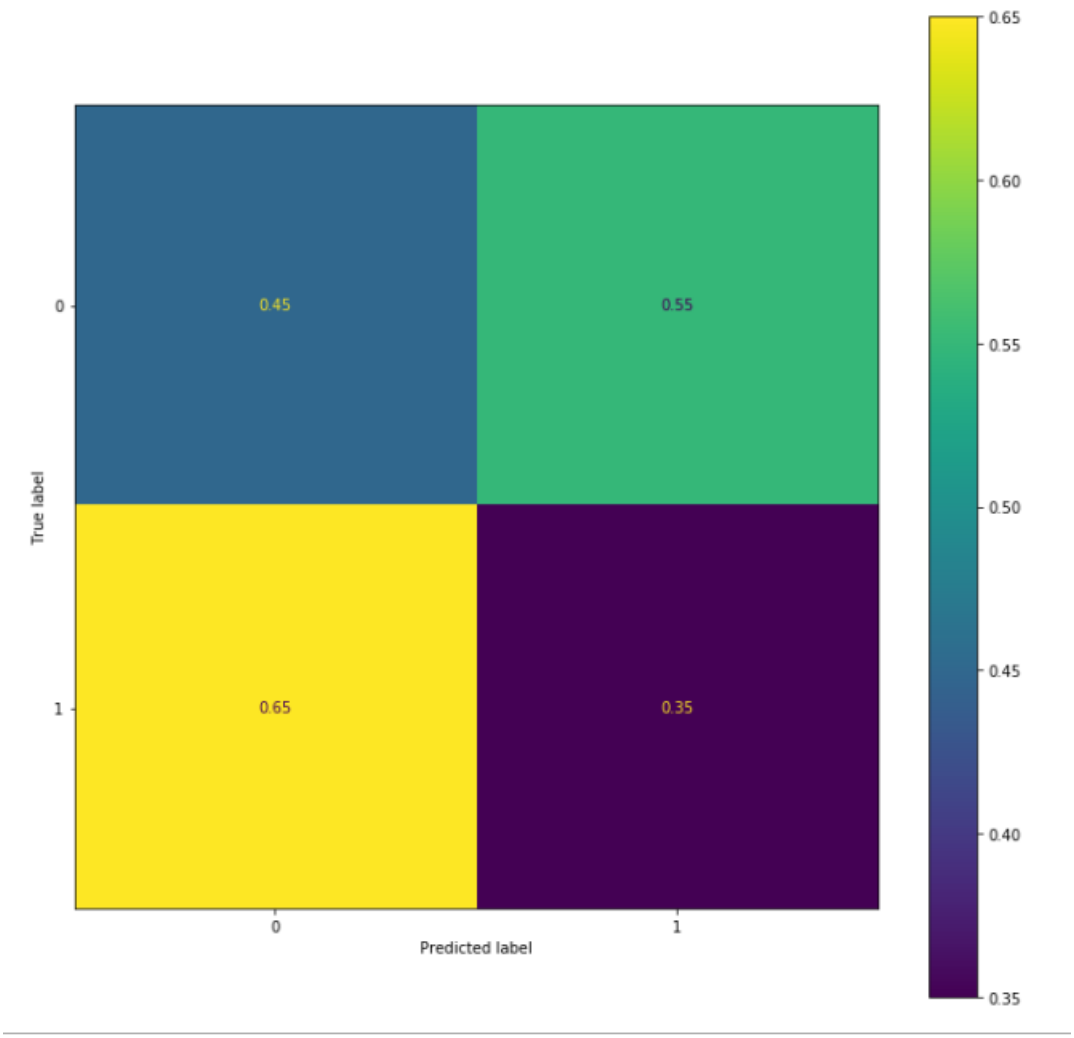

Figure 12 RFC Confusion Matrix Example

The figure above shows the confusion matrix of the FEI and JAFFE trained models with the Suzy's Test Images. The index is representing different labels for the images. The index 0 means "bad" while index 1 means "good". There are 45% of the true "bad" images being labeled as bad which means 45% of the "bad" images were being labeled correctly. On the other hand, there are 55% of the "bad" images being predicted wrongly which in this case is being predicted as "good". There are 35% of the true "good" images being predicted as "good" which means being predicted correctly. In the other hand, there are 65% of "good" images being predicted wrongly as "bad".

5.1.3 Random Forest Classifier Summary

Trained Data Set	Accuracy	Accuracy (%)
FEI (Brazil)	0.55	55
JAFFE (Japanese)	0.575	57.5
FEI and JAFFE	0.4	40

Table 1 RFC Result Summary

The table above shows the summary of the accuracy by different data sets. JAFFE(Japanese) dataset gives us the highest accuracy.

5.2 Convolutional Neural Network

The training model will have 6 variation with 3 different data sets and 2 different epochs number of configurations. The summary of the tested result will be shown in the section below.

5.2.1 Load Test Data Set

In order to test the test data set (Suzy's pictures), we need to preprocess the test data set same as the trained data set.

Step of Processing data

1. Convert to grayscale.
2. Convert to pixel size of 50x50.
3. Convert into 4D array.

```
X_test_data = []
X_test_data_toshow=[]
y_test_data=[]
files = []
files_path=[]
IMG_SIZE = 50

categories = ["bad","good"]

######## Check All

def loadall_test_data():

    DATADIR = "Assignment_Image"
    CATEGORIES = ["bad", "good"]
    for category in CATEGORIES:  # do bad and good
        path = os.path.join(DATADIR,category)  # create path to bad and good
        for img in tqdm(os.listdir(path)):  # iterate over each image per bad and good

            try:
                files_path.append(os.path.join(path,img))
                files.append(img) # Add file name
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to normalize data size
                X_test_data.append(new_array)  # add this to our test_data
                if(category==CATEGORIES[0]):
                    y_test_data.append(0)
                elif(category==CATEGORIES[1]):
                    y_test_data.append(1)
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

loadall_test_data()
X_test_data_toshow=X_test_data
X_test_data = np.array(X_test_data).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

Figure 13 CNN Load Test Data Set

The figure above shows that I tried to preprocess and load the test data set into a list.

5.2.2 Load Trained Model

Since the trained model was saved to hdf5 files previously, now we can just select the required model from it and load the trained model for testing.

```
model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X_test_data.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))
# model.add(Activation('softmax'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
# model.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])


#
# model.load_weights("CNN_TRAINEDMODEL_FEI.hdf5")
# model.load_weights("CNN_TRAINEDMODEL_JAFFE.hdf5")
model.load_weights("CNN_TRAINEDMODEL_FEIANDJAFFE.hdf5")
```

Figure 14 Load The Trained Model

Figure above shows that the trained model with trained data set of FEI and JAFFE was being loaded.

5.2.3 Test Suzy Pictures

```
#Predict the model
pre = model.predict(X_test_data)
print(pre)
for i, p in enumerate(pre):
    y = p.argmax()
    print("File Name:", files[i])
    print("File Path:",files_path[i])
    print("Label:", categories[int(round(pre[i][0]))])
    plt.imshow(X_test_data_toshow[i], cmap='gray')  # graph it
    plt.show()  # display!
```

```
100%|██████████████████████████████████████████████████████████| 20/20 [00:00<00:00, 1205.40it/s]
100%|██████████████████████████████████████████████████████████| 20/20 [00:00<00:00, 1280.78it/s]
[[0.11759073]
 [0.53582424]
 [0.50697905]
 [0.4174719 ]
 [0.23890063]
 [0.11954236]
 [0.28859478]
 [0.14200732]
 [0.7453358 ]
```

Figure 15 CNN_Test Data_1

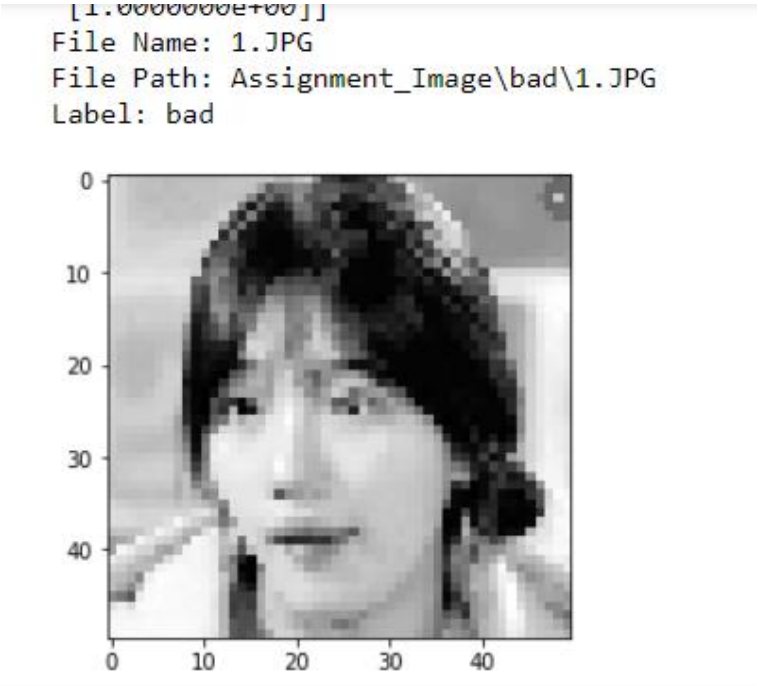

Figure 16 CNN_Test Data_2

The figure above shows that Suzy's pictures are loaded into the model and do the prediction by using the predict function. The returned value is in decimals instead of integer label of 0 and 1. This is due to the conversion of the data into floating point during the CNN process. If the returned value nears to 0 meaning it is more likely to be label of index 0 which is "bad" while if it is near the value of 1 meaning it is more likely to be label of index 1 which is "good". The program was code in such a way that it will print out the file name, file path and predicted label. We can see that the tested image above supposed to be bad image and it was being predicted as bad as well which is correct.

```
: print(model.evaluate(X_test_data,y_test_data))
```

```
40/40 [==============================] - 0s 3ms/sample - loss: 0.7344 - acc: 0.5500
[0.7343917965888977, 0.55]
```

Figure 17 Evaluate Test Result

Figure above shows the evaluation result of the test data on the trained model. The result shows the loss of 0.7344 while having accuracy of 55%.

5.2.4 Returned Predict Value

```
#Predict the model
pre = model.predict(X_test_data)
print(pre)
for i, p in enumerate(pre):
    y = p.argmax()
    print("File Name:", files[i])
    print("File Path:",files_path[i])
    print("Label:", categories[int(round(pre[i][0]))])
    plt.imshow(X_test_data_toshow[i], cmap='gray')  # graph it
    plt.show()  # display!
```

```
100%|
100%|
[[0.48487803]
 [0.4874173 ]
 [0.49240375]
 [0.4908185 ]
 [0.48415503]
 [0.47952914]
 [0.4702235 ]
 [0.47800842]
 [0.48581073]
 [0.49335253]
 [0.48659724]
 [0.4781204 ]
```

Figure 18 CNN Return Prediction Value_Epochs 3

```
#Predict the model
pre = model.predict(X_test_data)
print(pre)
for i, p in enumerate(pre):
    y = p.argmax()
    print("File Name:", files[i])
    print("File Path:",files_path[i])
    print("Label:", categories[int(round(pre[i][0]))])
    plt.imshow(X_test_data_toshow[i], cmap='gray')  # graph it
    plt.show()  # display!
```

```
100%|
100%|
[[7.4207783e-05]
 [9.4302368e-01]
 [9.8513985e-01]
 [4.9934089e-03]
 [2.0778179e-04]
 [9.9016809e-01]
 [9.9249482e-01]
 [6.1593652e-03]
 [5.2863532e-01]
 [9.9982756e-01]
 [3.0285120e-04]
 [9.9885178e-01]
```

Figure 19 CNN Return Prediction Value_Epochs 20

The figure above shows the return value of predict function of CNN with the same training data set of FEI and JAFFE data set. Both are using same training data set but different in number of epochs where one of it using epochs of 3 while the other one using epochs of 20. The one using epochs=3 is having returned value of nearly same to each other with around 0.5 of the return values. However, the one using ‘epochs=20’ is having the predicted return value of nearly 0 and nearly 1. This will be discussed more in the discussion section.

5.2.5 Check Model Summary

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 48, 48, 256)       2560
_________________________________________________________________
activation_6 (Activation)    (None, 48, 48, 256)       0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 24, 24, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 22, 22, 256)       590080
_________________________________________________________________
activation_7 (Activation)    (None, 22, 22, 256)       0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 11, 11, 256)       0
_________________________________________________________________
flatten_2 (Flatten)          (None, 30976)             0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                1982528
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_8 (Activation)    (None, 1)                 0
=================================================================
Total params: 2,575,233
Trainable params: 2,575,233
Non-trainable params: 0
```

Figure 20 CNN Model Summary

We can use the summary function to check the layer inside the CNN model.

5.2.6 Convolutional Neural Network Summary

Table 2 CNN Result Sumary

Trained Data Set	Epochs	Loss	Accuracy
FEI (Brazil)	3	0.692	0.55
JAFFE (Japanese)		0.690	0.525
FEI and JAFFE		0.678	0.725
FEI (Brazil)	8	0.692	0.425
JAFFE (Japanese)		0.712	0.525
FEI and JAFFE		0.734	0.55

The result of the test data with different trained data set and epochs configuration was shown in the table above. The training data set with 3 numbers of epochs is having the highest accuracy which is 72.5%.

6 Discussion

6.1 Random Forest Classifier

The random forest classifier was done successful as it can at least tried to differentiate the test data in different labels.

Trained Data Set	Accuracy	Accuracy (%)
FEI (Brazil)	0.55	55
JAFFE (Japanese)	0.575	57.5
FEI and JAFFE	0.4	40

The table above shows that the random forest classifier with JAFFE trained data set having the highest accuracy which is 57.5%. Since the test data set, Suzy's picture is also Asian, it should have the higher similarities between them, and this should be the reason why it has higher accuracy for this trained data set.

6.2 Convolutional Neural Network

Trained Data Set	Epochs	Loss	Accuracy
FEI (Brazil)	3	0.692	0.55
JAFFE (Japanese)		0.690	0.525
FEI and JAFFE		0.678	0.725
FEI (Brazil)	8	0.692	0.425
JAFFE (Japanese)		0.712	0.525
FEI and JAFFE		0.734	0.55

The test result with trained data set of FEI and JAFFE and epochs of 3 is having the highest accuracy which is 72.5% among all the trained data set. The result shows that it does not means that the higher the number of epochs, the higher the accuracy.

As mentioned in the section of 5.2.4 Returned Predict Value, the high value of epochs will result in the returned value of either extremely near to 0 or 1. However, the one with low epochs value of configuration (epochs=3) will result in the return value of nearly similar to 0.5 which means it is quite hard to be differentiate as 0(bad) or 1(good). In another word, the result summary shown above is only more meaningful as for the one with 8 epochs. The reason why is it the resulted accuracy is so high for the one with 3 epochs is that the training data set and the test data set are quite low. Hence, it is not enough to show the correct accuracy.

Considering the returned value of the prediction function, it can be said that the resultant accuracy value is more reliable for the one using epochs=8 which results in higher difference in the returned predict value. The test data set using the FEI and JAFFE training data set with the epochs=8 configuration is having the highest accuracy of 55% among all the training data set result with configuration of epochs=8.

6.3 Improvement Suggestion

The suggestion of improvement for both the random forest classifier and the convolutional neural network is by adding more training data set to achieve higher accuracy. The current training data set for FEI is having a total number of training data of 400, JAFFE is having a total number of training data of 79 and the combination of both FEI and JAFFE is having a total number of training data of 479. This number of datasets is considered very low as compared to those professional data set which have more than 10,000 of training data.

As for the convolutional neural network (CNN), we can find tune the model by changing the layer configuration and the epochs value. For example, we can increase the number of epochs to decrease the losses and increase the accuracy. However, it cannot be increased too much as it will result in overfitting. For example, if we want to classify a shoe and a shirt. If most of the training data set of shoes are sport shoes and it was being trained too much on it, even we give a test data of slipper to it, the program will not recognize it as a shoes as it only accept the sport shoes as shoes.

The pre-processed of the trained and test data set was done by convert it into grayscale and size of 50 x 50. This was being done to minimize the training time and get the enough image details. For different kind of application, it might be having a case that it might need a higher image size so that there is enough information captured from the image. However, it will decrease in the training process time. Hence, it is a give and take. It will be mostly depending on the experienced. We must choose the data size so that it can be optimally processed as in small size for lower processing time and big size for enough data to be processed.

7 Conclusion

General summary of findings of the Suzy's data results and best approach.

In conclusion, we have tried out the machine learning by using random forest classifier and deep learning by using convolutional neural network for classified the test data set. Both methods achieved the accuracy result of higher than 50% which are considered as low accuracy but enough for the first experienced. We have observed and analyze the confusion matrix of the random forest classifier and understand how to read the confusion matrix.

There are still a lot of improvement can be done on the training model such as increase training data set, increase epochs value for decreasing loss and increase the accuracy, and change the training data and test data pre-processed image size.

Finally, the current classifier is not ready yet to help Mr. Joel to really differentiate whether Suzy is in good or bad mood. However, if the model could be improved according to the suggestion listed in the discussion section, we believe that the accuracy of the classifier could be increased until a standard that it could help Mr. Joel to solve the problem.

8 References

asmag 2017, *The many applications of emotion recognition*, asmag, viewed 2 June 2020, <https://www.asmag.com/showpost/23883.aspx>.

Breiman, L 2001, 'Random forests. machine learning', *SpringerLink*, no 45, pp. 5–32, viewed 6 June 2020, <https://doi.org/10.1023/A:1010933404324>.

Gandhi R 2018, *Naive bayes classifier*, towards data science, viewed 2 June 2020, <https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c>.

Harrison O 2018, *Machine learning basics with the k-nearest neighbors algorithm*, towards data science, viewed 2 June 2020, <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>.

Pupale R 2018, *Support vector machines(svm) — an overview*, towards data science, viewed 2 June 2020, <https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989>.

Saha, S 2018, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, towards data science, viewed 2 June 2020, <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.

TensorFlow 2018, *第 4 回　CNN（Convolutional Neural Network）を理解しよう（TensorFlow 編）*, TensorFlow, viewed 2 June 2020, <https://www.atmarkit.co.jp/ait/articles/1804/23/news138.html>.

TensorFlow n.d., *tf.keras.Model*, TensorFlow, viewed 6 June 2020, <https://www.tensorflow.org/api_docs/python/tf/keras/Model>.

データ科学便覧 2017, *Scikit-learn によるランダムフォレスト*, データ科学便覧, viewed 6 June 2020, <https://data-science.gr.jp/implementation/iml_sklearn_random_forest.html>.

9 Appendix

Source code for Random Forest Classifier

Import necessary library

```
import time
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import math

from sklearn.datasets import fetch_openml
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
import pickle
```

Load 1D_TrainingData_FEI

```
DATADIR = "AI_Assignment2_DataSet_V1\FEI_DATASET_Sorted_V3"
CATEGORIES = ["bad", "good"]

IMG_SIZE = 50 #set image size
training_data = []
```

```
def create_training_data():
    for category in CATEGORIES:  # do bad and good

        path = os.path.join(DATADIR,category)  # create path to bad and good
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1).
0=bad 1=good

        for img in tqdm(os.listdir(path)):  # iterate over each image per dogs and cats
            try:
                img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
normalize data size
                training_data.append([new_array, class_num])  # add this to our
training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()
print('Length of training Data')
print(len(training_data))

# Shuffle the train data
import random
random.shuffle(training_data)

# Check the train data were shuffled successful
```

```
print("Check Shuffled Data")
for sample in training_data[:10]:
    print(sample[1])

#Add the trained data into a list
X = []
y = []

for features,label in training_data:
    X.append(features)
    y.append(label)

#Change the data to 2D array
X_2D=[]
for a in X:
    X_2D.append(a.reshape(-1))



#Save the processed Training Data into a pickle file
pickle_out = open("X_FEI_2D.pickle","wb")
pickle.dump(X_2D, pickle_out)
pickle_out.close()

pickle_out = open("y_FEI_2D.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

Load 1D_TrainingData_JAFFE

```
DATADIR = "AI_Assignment2_DataSet_V1\JAFFE DATASET_Sorted_V3"
CATEGORIES = ["bad", "good"]
```

```
IMG_SIZE = 50 #set image size
training_data = []

def create_training_data():
    for category in CATEGORIES:  # do bad and good

        path = os.path.join(DATADIR,category)  # create path to bad and good
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1).
0=bad 1=good

        for img in tqdm(os.listdir(path)):  # iterate over each image per dogs and cats
            try:
                img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
normalize data size
                training_data.append([new_array, class_num])  # add this to our
training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()
print('Length of training Data')
print(len(training_data))

# Shuffle the train data
```

```
import random
random.shuffle(training_data)

# Check the train data were shuffled successful
print("Check Shuffled Data")
for sample in training_data[:10]:
    print(sample[1])

#Add the trained data into a list
X = []
y = []

for features,label in training_data:
    X.append(features)
    y.append(label)

#Change the data to 2D array
X_2D=[]
for a in X:
    X_2D.append(a.reshape(-1))



#Save the processed Training Data into a pickle file
pickle_out = open("X_JAFFE_2D.pickle","wb")
pickle.dump(X_2D, pickle_out)
pickle_out.close()

pickle_out = open("y_JAFFE_2D.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

Load 1D_TrainingData_FEIANDJAFFE

```
DATADIR = "AI_Assignment2_DataSet_V1\FEI_AND_JAFFE_DATASET_Sorted_V3"
CATEGORIES = ["bad", "good"]

IMG_SIZE = 50 #set image size
training_data = []

def create_training_data():
    for category in CATEGORIES:  # do bad and good

        path = os.path.join(DATADIR,category)  # create path to bad and good
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1).
0=bad 1=good

        for img in tqdm(os.listdir(path)):  # iterate over each image per dogs and cats
            try:
                img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)  # convert to array
#                 new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
normalize data size
                training_data.append([new_array, class_num])  # add this to our
training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()
```

```
print('Length of training Data')
print(len(training_data))

# Shuffle the train data
import random
random.shuffle(training_data)

# Check the train data were shuffled successful
print("Check Shuffled Data")
for sample in training_data[:10]:
    print(sample[1])

#Add the trained data into a list
X = []
y = []

for features,label in training_data:
    X.append(features)
    y.append(label)

#Change the data to 2D array
X_2D=[]
for a in X:
    X_2D.append(a.reshape(-1))



#Save the processed Training Data into a pickle file
pickle_out = open("X_FEIANDJAFFE_2D.pickle","wb")
pickle.dump(X_2D, pickle_out)
pickle_out.close()
```

```
pickle_out = open("y_FEIANDJAFFE_2D.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

Random Forest Classifier

```
import pickle
# TRAINED_DATASET_X="X_FEI_2D.pickle"
# TRAINED_DATASET_y="y_FEI_2D.pickle"
TRAINED_DATASET_X="X_JAFFE_2D.pickle"
TRAINED_DATASET_y="y_JAFFE_2D.pickle"
# TRAINED_DATASET_X="X_FEIANDJAFFE_2D.pickle"
# TRAINED_DATASET_y="y_FEIANDJAFFE_2D.pickle"



pickle_in = open(TRAINED_DATASET_X,"rb")
X_train = pickle.load(pickle_in)

pickle_in = open(TRAINED_DATASET_y,"rb")
y_train = pickle.load(pickle_in)

# Double check the length of data and labels
print(len(X_train))
print(len(y_train))

from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,
BaggingRegressor, RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import plot_confusion_matrix
```

```
from sklearn import metrics



import sklearn
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=10)
rfc.fit(X_train, y_train)
```

Test Data Set

```
import os
from tqdm import tqdm
import cv2

X_test_data = []
X_test_data_2D=[]
y_test_data=[]
files = []
files_path=[]
IMG_SIZE=50
CATEGORIES = ["bad", "good"]
def loadall_test_data():

    DATADIR = "Assignment_Image"
    CATEGORIES = ["bad", "good"]
    for category in CATEGORIES:  # do bad and good
        path = os.path.join(DATADIR,category)  # create path to bad and good

        for img in tqdm(os.listdir(path)):  # iterate over each image per bad and good
            try:

                files_path.append(os.path.join(path,img))
```

```
            files.append(img) # Add file name
            img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)  # convert to array
            new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
normalize data size
            X_test_data.append(new_array)  # add this to our test_data
            if(category==CATEGORIES[0]):
                y_test_data.append(0)
            elif(category==CATEGORIES[1]):
                y_test_data.append(1)
        except Exception as e:  # in the interest in keeping the output clean...
            pass
        #except OSError as e:
        #    print("OSErrroBad img most likely", e, os.path.join(path,img))
        #except Exception as e:
        #    print("general exception", e, os.path.join(path,img))

loadall_test_data()
for b in X_test_data:
    X_test_data_2D.append(b.reshape(-1))

#Test the data
print("Accuracy")
rfc.score(X_test_data_2D, y_test_data)
y_pred=rfc.predict(X_test_data_2D)
print(rfc.score(X_test_data_2D, y_test_data))
print(y_pred)

for i, p in enumerate(y_pred):
    print("File Name:",files[i])
    print("File Path:",files_path[i])
```

```
    print("Label: ",CATEGORIES[p])
    print(i)

import matplotlib
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
f,ax=plt.subplots(1,1,figsize=(12,12))
# print("Classification report for classifier %s:\n%s\n"
#       % (rfc, metrics.classification_report(y_test, y_pred)))
disp = metrics.plot_confusion_matrix(rfc, X_test_data_2D,
y_test_data,normalize='true',ax=ax)

# disp.figure_.suptitle("Confusion Matrix")
# print("Confusion matrix:\n%s" % disp.confusion_matrix)
```

Source code for Convolutional Neural Network

Import necessary library

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
```

Test Print the Image after the process

```
DATADIR = "AI_Assignment2_DataSet_V1\JAFFE DATASET_Sorted_V3"
CATEGORIES = ["bad", "good"]

for category in CATEGORIES:  # do bad and good
    path = os.path.join(DATADIR,category)  # create path to bad and good
    print(path) #Check path
    for img in os.listdir(path):  # iterate over each image per dogs and cats
        img_array = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE )  #
convert to array
        print(os.path.join(path,img)) #Check Image path
        print(type(img_array))   # Check Image converted type
        plt.imshow(img_array, cmap='gray')  # graph it
        plt.show()  # display!

        break  # we just want one for now so break
#     break  #...and one more!
```

Train 4D_TrainingData_FEI

```
IMG_SIZE = 50 #set image size
training_data = []

DATADIR = "AI_Assignment2_DataSet_V1\FEI_DATASET_Sorted_V3"
CATEGORIES = ["bad", "good"]
def create_training_data():
    for category in CATEGORIES:  # do bad and good

        path = os.path.join(DATADIR,category)  # create path to bad and good
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1).
0=bad 1=good

        for img in tqdm(os.listdir(path)):  # iterate over each image per bad and good
            try:
                img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
normalize data size
                training_data.append([new_array, class_num])  # add this to our
training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()

print(len(training_data))

# Shuffle the train data
import random
random.shuffle(training_data)

# Check the train data were shuffled successful
for sample in training_data[:20]:
    print(sample[1])

#Add the trained data into a list
X = []
y = []

for features,label in training_data:
```

```
    X.append(features)
    y.append(label)

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1) # Convert all the picture array
into 4 dimension with size of 50

import pickle

pickle_out = open("X_FEI_4D.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("y_FEI_4D.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

Train 4D_TrainingData_JAFFE

```
IMG_SIZE = 50 #set image size
training_data = []

DATADIR = "AI_Assignment2_DataSet_V1\JAFFE DATASET_Sorted_V3"
CATEGORIES = ["bad", "good"]
def create_training_data():
    for category in CATEGORIES:  # do bad and good

        path = os.path.join(DATADIR,category)  # create path to bad and good
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1).
0=bad 1=good

        for img in tqdm(os.listdir(path)):  # iterate over each image bad and good
            try:
                img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
normalize data size
                training_data.append([new_array, class_num])  # add this to our
training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))
```

```
create_training_data()

print(len(training_data))

# Shuffle the train data
import random
random.shuffle(training_data)

# Check the train data were shuffled successful
for sample in training_data[:20]:
    print(sample[1])

#Add the trained data into a list
X = []
y = []

for features,label in training_data:
    X.append(features)
    y.append(label)

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1) # Convert all the picture array
into 4 dimension with size of 50

import pickle

pickle_out = open("X_JAFFE_4D.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("y_JAFFE_4D.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

Train 4D_TrainingData_FEIANDJAFFE

```
IMG_SIZE = 50 #set image size
training_data = []

DATADIR = "AI_Assignment2_DataSet_V1\FEI_AND_JAFFE_DATASET_Sorted_V3"
CATEGORIES = ["bad", "good"]
def create_training_data():
    for category in CATEGORIES:  # do bad and good

        path = os.path.join(DATADIR,category)  # create path to bad and good
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1).
0=bad 1=good
```

```
        for img in tqdm(os.listdir(path)):  # iterate over each image per bad and good
            try:
                img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
normalize data size
                training_data.append([new_array, class_num])  # add this to our
training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))

create_training_data()

print(len(training_data))

# Shuffle the train data
import random
random.shuffle(training_data)

# Check the train data were shuffled successful
for sample in training_data[:20]:
    print(sample[1])

#Add the trained data into a list
X = []
y = []

for features,label in training_data:
    X.append(features)
    y.append(label)

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1) # Convert all the picture array
into 4 dimension with size of 50

import pickle

pickle_out = open("X_FEIANDJAFFE_4D.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("y_FEIANDJAFFE_4D.pickle","wb")
```

```
pickle.dump(y, pickle_out)
pickle_out.close()
```

Save CNN_Model_FEI hdf5 file

```
import pickle
TRAINED_DATASET_X="X_FEI_4D.pickle"
TRAINED_DATASET_y="y_FEI_4D.pickle"
pickle_in = open(TRAINED_DATASET_X,"rb")
X = pickle.load(pickle_in)

pickle_in = open(TRAINED_DATASET_y,"rb")
y = pickle.load(pickle_in)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

X = X/255.0

model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

model.fit(X, y, batch_size=32, epochs=8, validation_split=0.3)



#Save model into a hdf5 file
hdf5_file = "CNN_TRAINEDMODEL_FEI.hdf5"
```

```
model.save_weights(hdf5_file)
```

Save CNN_Model_JAFFE hdf5 file

```
import pickle
TRAINED_DATASET_X="X_JAFFE_4D.pickle"
TRAINED_DATASET_y="y_JAFFE_4D.pickle"
pickle_in = open(TRAINED_DATASET_X,"rb")
X = pickle.load(pickle_in)

pickle_in = open(TRAINED_DATASET_y,"rb")
y = pickle.load(pickle_in)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

X = X/255.0

model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

model.fit(X, y, batch_size=32, epochs=8, validation_split=0.3)


#Save model into a hdf5 file
hdf5_file = "CNN_TRAINEDMODEL_JAFFE.hdf5"
model.save_weights(hdf5_file)
```

Save CNN_Model_FEIANDJAFFE hdf5 file

```
import pickle
TRAINED_DATASET_X="X_FEIANDJAFFE_4D.pickle"
TRAINED_DATASET_y="y_FEIANDJAFFE_4D.pickle"
pickle_in = open(TRAINED_DATASET_X,"rb")
X = pickle.load(pickle_in)

pickle_in = open(TRAINED_DATASET_y,"rb")
y = pickle.load(pickle_in)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

X = X/255.0
model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

model.fit(X, y, batch_size=32, epochs=20, validation_split=0.3)


#Save model into a hdf5 file
hdf5_file = "CNN_TRAINEDMODEL_FEIANDJAFFE.hdf5"
model.save_weights(hdf5_file)
```

Load and test the test data

```
import sys, os
```

```
from PIL import Image
import numpy as np
import cv2
import matplotlib.pyplot as plt
from tqdm import tqdm
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

X_test_data = []
X_test_data_toshow=[]
y_test_data=[]
files = []
files_path=[]
IMG_SIZE = 50

categories = ["bad","good"]

######## Check All

def loadall_test_data():

    DATADIR = "Assignment_Image"
    CATEGORIES = ["bad", "good"]
    for category in CATEGORIES:  # do bad and good
        path = os.path.join(DATADIR,category)  # create path to bad and good
        for img in tqdm(os.listdir(path)):  # iterate over each image per bad and good

            try:
                files_path.append(os.path.join(path,img))
                files.append(img) # Add file name
                img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to
normalize data size
                X_test_data.append(new_array)  # add this to our test_data
                if(category==CATEGORIES[0]):
                    y_test_data.append(0)
                elif(category==CATEGORIES[1]):
                    y_test_data.append(1)
            except Exception as e:  # in the interest in keeping the output clean...
                pass
            #except OSError as e:
            #    print("OSErrroBad img most likely", e, os.path.join(path,img))
            #except Exception as e:
            #    print("general exception", e, os.path.join(path,img))
```

```
loadall_test_data()
X_test_data_toshow=X_test_data
X_test_data = np.array(X_test_data).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
#########Check All
X_test_data=X_test_data/255.0

model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X_test_data.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))
# model.add(Activation('softmax'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
# model.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])


#
# model.load_weights("CNN_TRAINEDMODEL_FEI.hdf5")
# model.load_weights("CNN_TRAINEDMODEL_JAFFE.hdf5")
model.load_weights("CNN_TRAINEDMODEL_FEIANDJAFFE.hdf5")



#Predict the model
pre = model.predict(X_test_data)
print(pre)
for i, p in enumerate(pre):
    y = p.argmax()
    print("File Name:", files[i])
    print("File Path:",files_path[i])
    print("Label:", categories[int(round(pre[i][0]))])
    plt.imshow(X_test_data_toshow[i], cmap='gray')  # graph it
    plt.show()  # display!
```

```
model.summary()
print(model.evaluate(X_test_data,y_test_data))
```