```
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.optimizers import adam_v2
from keras.callbacks import TensorBoard

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from keras.utils import np_utils
import itertools
```

```
location = r'/ORL_faces.npz'
```

```
import os
path =  os.path.expanduser(rb'~\Downloads\2017.csv')
```

```
data = np.load(location)
# load the "Train Images"
x_train = data['trainX']
#normalize every image
x_train = np.array(x_train,dtype='float32')/255

x_test = data['testX']
x_test = np.array(x_test,dtype='float32')/255

# load the Label of Images
y_train= data['trainY']
y_test= data['testY']

# show the train and test Data format
print('x_train : {}'.format(x_train[:]))
print('Y-train shape: {}'.format(y_train))
print('x_test shape: {}'.format(x_test.shape))
#data =pd.read_csv(r'/Users/dkumar/Downloads/2017.csv')
```

```
x_train : [[0.1882353  0.19215687 0.1764706  0.18431373 ... 0.17254902 0.1843137
 [0.23529412 0.23529412 0.24313726 0.20784314 ... 0.14509805 0.1254902  0.133333
```

```
  [0.15294118 0.17254902 0.20784314 0.14509805 ... 0.12156863 0.11372549 0.101960
  [0.24705882 0.20784314 0.13725491 0.14117648 ... 0.5372549  0.16078432 0.039215
  ...
  [0.44705883 0.43137255 0.44705883 0.43529412 ... 0.38431373 0.40784314 0.352941
  [0.44705883 0.45882353 0.44705883 0.45882353 ... 0.3882353  0.38431373 0.376470
  [0.4117647  0.4117647  0.41960785 0.41568628 ... 0.11372549 0.21176471 0.184313
  [0.45490196 0.44705883 0.45882353 0.45882353 ... 0.39607844 0.37254903 0.392156
 Y-train shape: [ 0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1
   3  3  4  4  4  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5  5  5  5  5  5  5
   7  7  7  7  8  8  8  8  8  8  8  8  8  8  8  8  9  9  9  9  9  9  9  9  9  9
  11 11 11 11 11 11 12 12 12 12 12 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13 1
  15 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 1
  19 19 19 19 19 19 19 19 19 19]
 x_test shape: (160, 10304)
```

```
#split dataset

x_train, x_valid, y_train, y_valid= train_test_split(
    x_train, y_train, test_size=.05, random_state=1234)


#change the sie of images to feed to CNN

im_rows=112
im_cols=92
batch_size=512
im_shape=(im_rows, im_cols, 1)

#change the size of images
x_train = x_train.reshape(x_train.shape[0], *im_shape)
x_test = x_test.reshape(x_test.shape[0], *im_shape)
x_valid = x_valid.reshape(x_valid.shape[0], *im_shape)

print('x_train shape: {}'.format(y_train.shape[0]))
print('x_test shape: {}'.format(y_test.shape))
```

```
 x_train shape: 228
 x_test shape: (160,)
```

```
#Build a CNN model that has 3 main layers
#Convolotional Layer
#Pooling Layer
#Fully Connected Layer

#filters= the depth of output image or kernels

cnn_model= Sequential([
    Conv2D(filters=36, kernel_size=7, activation='relu', input_shape= im_shape),
    MaxPooling2D(pool_size=2),
    Conv2D(filters=54, kernel_size=5, activation='relu', input_shape= im_shape),
    MaxPooling2D(pool_size=2),
    Flatten(),
```

```
    Dense(2024, activation='relu'),
     Dropout(0.5),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(512, activation='relu'),
    Dropout(0.5),
    #20 is the number of outputs
    Dense(20, activation='softmax')
])

cnn_model.compile(
    loss='sparse_categorical_crossentropy',#'categorical_crossentropy',
    optimizer=adam_v2.Adam(learning_rate=0.0001),
    metrics=['accuracy']
)
```

```
cnn_model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_12 (Conv2D)          (None, 106, 86, 36)       1800      
                                                                 
 max_pooling2d_12 (MaxPoolin  (None, 53, 43, 36)       0         
 g2D)                                                            
                                                                 
 conv2d_13 (Conv2D)          (None, 49, 39, 54)        48654     
                                                                 
 max_pooling2d_13 (MaxPoolin  (None, 24, 19, 54)       0         
 g2D)                                                            
                                                                 
 flatten_6 (Flatten)         (None, 24624)             0         
                                                                 
 dense_24 (Dense)            (None, 2024)              49841000  
                                                                 
 dropout_18 (Dropout)        (None, 2024)              0         
                                                                 
 dense_25 (Dense)            (None, 1024)              2073600   
                                                                 
 dropout_19 (Dropout)        (None, 1024)              0         
                                                                 
 dense_26 (Dense)            (None, 512)               524800    
                                                                 
 dropout_20 (Dropout)        (None, 512)               0         
                                                                 
 dense_27 (Dense)            (None, 20)                10260     
                                                                 
=================================================================
Total params: 52,500,114
Trainable params: 52,500,114
```

```
Non-trainable params: 0
```

```
#train the model
history=cnn_model.fit(
    np.array(x_train), np.array(y_train), batch_size=512,
    epochs=150, verbose=2,
    validation_data=(np.array(x_valid),np.array(y_valid)),
)
```

```
Epoch 121/150
1/1 - 8s - loss: 0.1789 - accuracy: 0.9518 - val_loss: 0.0402 - val_accuracy: 1.
Epoch 122/150
1/1 - 8s - loss: 0.1993 - accuracy: 0.9298 - val_loss: 0.0353 - val_accuracy: 1.
Epoch 123/150
1/1 - 8s - loss: 0.2019 - accuracy: 0.9298 - val_loss: 0.0193 - val_accuracy: 1.
Epoch 124/150
1/1 - 8s - loss: 0.1669 - accuracy: 0.9649 - val_loss: 0.0167 - val_accuracy: 1.
Epoch 125/150
1/1 - 8s - loss: 0.1265 - accuracy: 0.9649 - val_loss: 0.0170 - val_accuracy: 1.
Epoch 126/150
1/1 - 8s - loss: 0.1546 - accuracy: 0.9605 - val_loss: 0.0204 - val_accuracy: 1.
Epoch 127/150
1/1 - 8s - loss: 0.1056 - accuracy: 0.9912 - val_loss: 0.0278 - val_accuracy: 1.
Epoch 128/150
1/1 - 8s - loss: 0.1523 - accuracy: 0.9561 - val_loss: 0.0368 - val_accuracy: 1.
Epoch 129/150
1/1 - 8s - loss: 0.1108 - accuracy: 0.9737 - val_loss: 0.0465 - val_accuracy: 1.
Epoch 130/150
1/1 - 8s - loss: 0.1821 - accuracy: 0.9430 - val_loss: 0.0334 - val_accuracy: 1.
Epoch 131/150
1/1 - 8s - loss: 0.1335 - accuracy: 0.9605 - val_loss: 0.0253 - val_accuracy: 1.
Epoch 132/150
1/1 - 8s - loss: 0.0999 - accuracy: 0.9912 - val_loss: 0.0205 - val_accuracy: 1.
Epoch 133/150
1/1 - 8s - loss: 0.1129 - accuracy: 0.9649 - val_loss: 0.0184 - val_accuracy: 1.
Epoch 134/150
1/1 - 8s - loss: 0.1151 - accuracy: 0.9737 - val_loss: 0.0182 - val_accuracy: 1.
Epoch 135/150
1/1 - 8s - loss: 0.0907 - accuracy: 0.9825 - val_loss: 0.0192 - val_accuracy: 1.
Epoch 136/150
1/1 - 8s - loss: 0.0807 - accuracy: 0.9956 - val_loss: 0.0202 - val_accuracy: 1.
Epoch 137/150
1/1 - 8s - loss: 0.1183 - accuracy: 0.9693 - val_loss: 0.0169 - val_accuracy: 1.
Epoch 138/150
1/1 - 8s - loss: 0.0783 - accuracy: 0.9868 - val_loss: 0.0154 - val_accuracy: 1.
Epoch 139/150
1/1 - 8s - loss: 0.1011 - accuracy: 0.9825 - val_loss: 0.0142 - val_accuracy: 1.
Epoch 140/150
1/1 - 8s - loss: 0.0623 - accuracy: 0.9912 - val_loss: 0.0133 - val_accuracy: 1.
Epoch 141/150
1/1 - 8s - loss: 0.0658 - accuracy: 0.9912 - val_loss: 0.0135 - val_accuracy: 1.
Epoch 142/150
1/1 - 8s - loss: 0.0719 - accuracy: 0.9912 - val_loss: 0.0151 - val_accuracy: 1.
Epoch 143/150
1/1 - 8s - loss: 0.0728 - accuracy: 0.9825 - val_loss: 0.0142 - val_accuracy: 1.
```

```
Epoch 144/150
1/1 - 8s - loss: 0.0804 - accuracy: 0.9868 - val_loss: 0.0092 - val_accuracy: 1.
Epoch 145/150
1/1 - 8s - loss: 0.0677 - accuracy: 0.9868 - val_loss: 0.0056 - val_accuracy: 1.
Epoch 146/150
1/1 - 8s - loss: 0.0695 - accuracy: 0.9825 - val_loss: 0.0048 - val_accuracy: 1.
Epoch 147/150
1/1 - 8s - loss: 0.0675 - accuracy: 0.9825 - val_loss: 0.0047 - val_accuracy: 1.
Epoch 148/150
1/1 - 8s - loss: 0.0756 - accuracy: 0.9825 - val_loss: 0.0051 - val_accuracy: 1.
Epoch 149/150
1/1 - 8s - loss: 0.0607 - accuracy: 0.9868 - val_loss: 0.0076 - val_accuracy: 1.
```

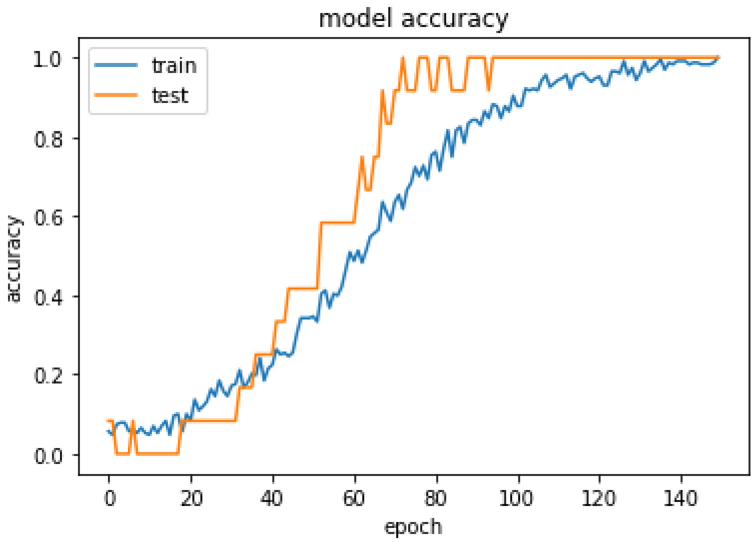
```
#Evaluate Test data
score = cnn_model.evaluate( np.array(x_test),  np.array(y_test), verbose=0)

print('test los {:.4f}'.format(score[0]))
print('test acc {:.4f}'.format(score[1]))
```

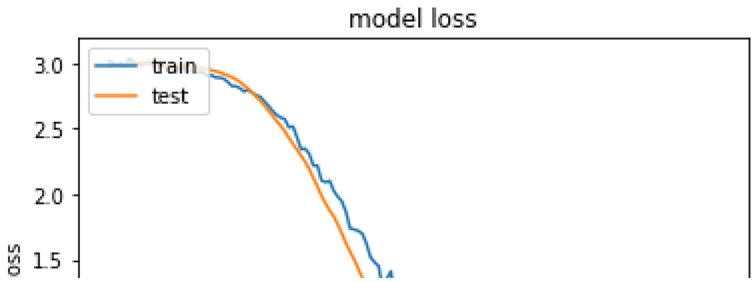
```
test los 0.2777
test acc 0.9438
```

```
# list all data in history
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

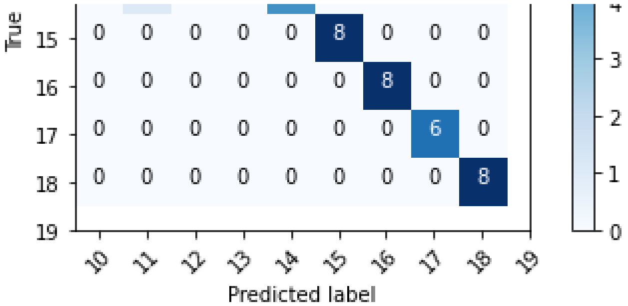
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```






```
predicted =np.array( cnn_model.predict(x_test))
#print(predicted)
#print(y_test)
ynew = cnn_model.predict(x_test)

#predict_x=model.predict(X_test)
classes_x=np.argmax(ynew,axis=1)
Acc=accuracy_score(y_test,classes_x)
print("accuracy : ")
print(Acc)
#/tn, fp, fn, tp = confusion_matrix(np.array(y_test), ynew).ravel()
cnf_matrix=confusion_matrix(np.array(y_test), classes_x)

y_test1 = np_utils.to_categorical(y_test, 20)



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        #print("Normalized confusion matrix")
    else:
```

```
    else:
        print('Confusion matrix, without normalization')

    #print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()


print('Confusion matrix, without normalization')
print(cnf_matrix)

plt.figure()
plot_confusion_matrix(cnf_matrix[1:10,1:10], classes=[0,1,2,3,4,5,6,7,8,9],
                      title='Confusion matrix, without normalization')

plt.figure()
plot_confusion_matrix(cnf_matrix[11:20,11:20], classes=[10,11,12,13,14,15,16,17,18,19
                      title='Confusion matrix, without normalization')

print("Confusion matrix:\n%s" % confusion_matrix(np.array(y_test), classes_x))
print(classification_report(np.array(y_test), classes_x))
```




```
Confusion matrix:
[[8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0]
 [0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 2 0 6 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0]
 [2 0 0 0 0 0 0 0 0 0 0 0 1 0 0 5 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0]
 [0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 6 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8]]
              precision    recall  f1-score   support

           0       0.80      1.00      0.89         8
           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         8
           3       1.00      1.00      1.00         8
           4       1.00      0.75      0.86         8
           5       1.00      1.00      1.00         8
           6       1.00      1.00      1.00         8
           7       0.80      1.00      0.89         8
           8       1.00      1.00      1.00         8
           9       1.00      0.75      0.86         8
          10       0.80      1.00      0.89         8
          11       1.00      1.00      1.00         8
          12       0.89      1.00      0.94         8
          13       1.00      1.00      1.00         8
          14       1.00      1.00      1.00         8
          15       1.00      0.62      0.77         8
          16       1.00      1.00      1.00         8
          17       0.80      1.00      0.89         8
          18       1.00      0.75      0.86         8
          19       1.00      1.00      1.00         8

    accuracy                           0.94       160
   macro avg       0.95      0.94      0.94       160
weighted avg       0.95      0.94      0.94       160
```