Project: Facial Recognition with Deep Learning in Keras Using CNN

by SHAKTI NATH SAINI

Step 1

In [22]:

```
import keras
import tensorflow as tf
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
from PIL import Image

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
```

Step 2

In [23]:

```
#Load dataset
data= np.load(r'C:\Users\Shakti\Documents\Python Scripts\Projects\Facial Recognition\ORL_fa
```

In [24]:

```
# 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))
```

```
x_train : [[0.1882353  0.19215687 0.1764706  ... 0.18431373 0.18039216 0.180
39216]
 [0.23529412 0.23529412 0.24313726 ... 0.1254902  0.13333334 0.13333334]
 [0.15294118 0.17254902 0.20784314 ... 0.11372549 0.10196079 0.11372549]
 ...
 [0.44705883 0.45882353 0.44705883 ... 0.38431373 0.3764706  0.38431373]
 [0.4117647  0.4117647  0.41960785 ... 0.21176471 0.18431373 0.16078432]
 [0.45490196 0.44705883 0.45882353 ... 0.37254903 0.39215687 0.39607844]]
Y-train shape: [ 0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
1  1  1  1
  2  2  2  2  2  2  2  2  2  2  2  3  3  3  3  3  3  3  3  3  3  3  3
  4  4  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5  5  5  5  5  5  5
  6  6  6  6  6  6  6  6  6  6  6  7  7  7  7  7  7  7  7  7  7  7  7
  8  8  8  8  8  8  8  8  8  8  8  9  9  9  9  9  9  9  9  9  9  9  9
 10 10 10 10 10 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11 11 11
 12 12 12 12 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 13 13
 14 14 14 14 14 14 14 14 14 14 14 15 15 15 15 15 15 15 15 15 15 15 15
 16 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 17 17 17 17
 18 18 18 18 18 18 18 18 18 18 18 19 19 19 19 19 19 19 19 19 19 19 19]
x_test shape: (160, 10304)
```

Step 3

In [25]:

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

Step 4

In [26]:

```
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,)
```

In [27]:

```
#x_train=np.reshape(x_train,(x_train.shape[0],112,-1))
#x_test=np.reshape(x_test,(x_test.shape[0],112,-1))
```

In [28]:

```
#plt.imshow(x_train[2])
```

Step 5

In [29]:

```
#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']
)
```

Step 5

Viewing Model parameters

In [30]:

```
cnn_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 106, 86, 36)       1800
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 53, 43, 36)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 49, 39, 54)        48654
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 24, 19, 54)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 24624)             0
_________________________________________________________________
dense_8 (Dense)              (None, 2024)              49841000
_________________________________________________________________
dropout_6 (Dropout)          (None, 2024)              0
_________________________________________________________________
dense_9 (Dense)              (None, 1024)              2073600
_________________________________________________________________
dropout_7 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_11 (Dense)             (None, 20)                10260
=================================================================
Total params: 52,500,114
Trainable params: 52,500,114
Non-trainable params: 0
_________________________________________________________________
```

Step 6

Train the model

Note: You can change the number of epochs

In [ ]:

```

```

In [31]:

```
history=cnn_model.fit(
    np.array(x_train), np.array(y_train), batch_size=512,
    epochs=100, verbose=2,
    validation_data=(np.array(x_valid),np.array(y_valid)),
)
```

```
Epoch 1/100
1/1 - 8s - loss: 3.0146 - accuracy: 0.0395 - val_loss: 3.0305 - val_accura
cy: 0.0000e+00
Epoch 2/100
1/1 - 5s - loss: 3.0074 - accuracy: 0.0526 - val_loss: 3.0434 - val_accura
cy: 0.0000e+00
Epoch 3/100
1/1 - 5s - loss: 3.0058 - accuracy: 0.0526 - val_loss: 3.0398 - val_accura
cy: 0.0000e+00
Epoch 4/100
1/1 - 5s - loss: 3.0357 - accuracy: 0.0614 - val_loss: 3.0302 - val_accura
cy: 0.0000e+00
Epoch 5/100
1/1 - 5s - loss: 3.0186 - accuracy: 0.0702 - val_loss: 3.0274 - val_accura
cy: 0.0000e+00
Epoch 6/100
1/1 - 5s - loss: 3.0008 - accuracy: 0.0439 - val_loss: 3.0180 - val_accura
cy: 0.0000e+00
Epoch 7/100
```

In [32]:

```
# Evaluate the test data
```

In [33]:

```
scor = cnn_model.evaluate( np.array(x_test),  np.array(y_test), verbose=0)

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

```
test los 0.3364
test acc 0.9375
```

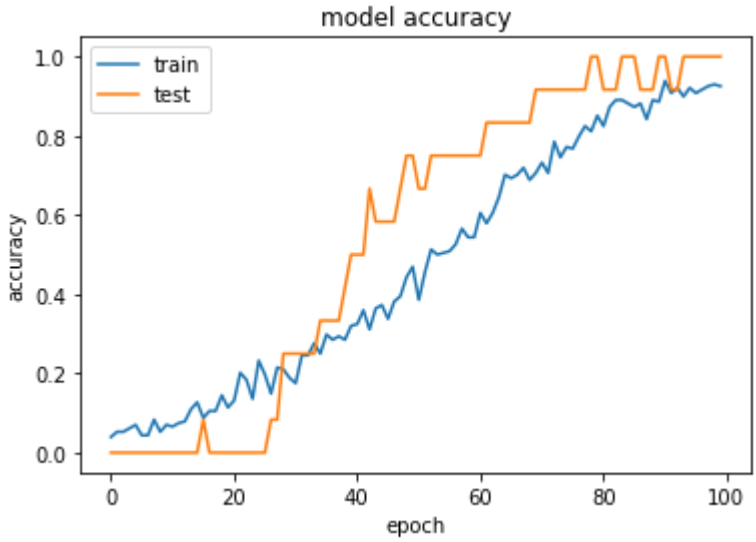
In [34]:

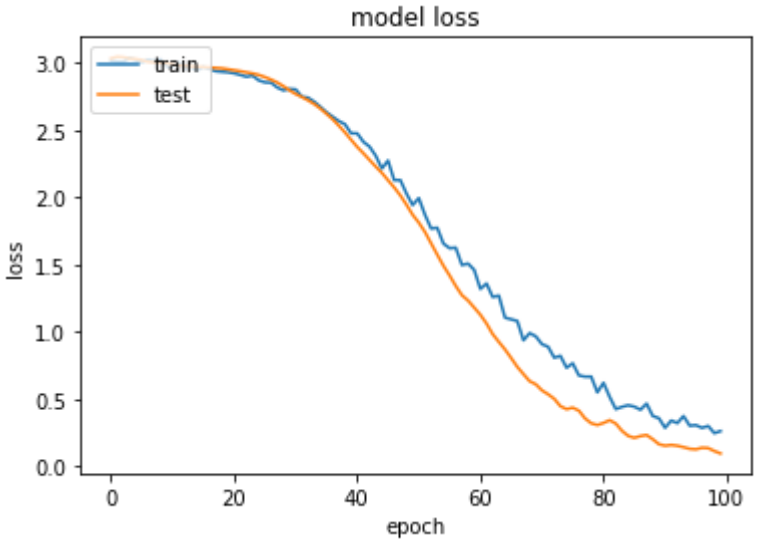
```
## Step 7
### plot the result
```

In [35]:

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

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





Step 8

Plot Confusion Matrix

In [ ]:

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

Acc=accuracy_score(y_test, ynew)
print("accuracy : ")
print(Acc)


#/tn, fp, fn, tp = confusion_matrix(np.array(y_test), ynew).ravel()
cnf_matrix=confusion_matrix(np.array(y_test), ynew)

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:
        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), ynew))
print(classification_report(np.array(y_test), ynew))
```

In [ ]: