face_reco_with_cnn

December 25, 2021

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

0.1 Load Dataset

Dataset Details: ORL face database composed of 400 images of size 112 x 92. The images were taken at different times, lighting and facial expressions. The faces are in an upright position in frontal view, with a slight left-right rotation.

Link to the Dataset: https://www.dropbox.com/s/i7uzp5yxk7wruva/ORL_faces.npz?dl=0

[2]:
```
data = np.load('ORL_faces.npz')
```

[3]:
```
data.files
```

[3]:
```
['testY', 'testX', 'trainX', 'trainY']
```

[4]:
```
x_train = data['trainX']
x_test = data['testX']

y_train = data['trainY']
y_test = data['testY']

print('Train data size ', y_train.shape)
print('Test data size ', y_test.shape)
```

```
Train data size  (240,)
Test data size  (160,)
```

[5]:
```
y_train
```

[5]:
```
array([ 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,  2,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,
        4,  4,  4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
        5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  7,
        7,  7,  7,  7,  7,  7,  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,
 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, 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, 16,
17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18,
18, 18, 18, 18, 18, 18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19,
19, 19], dtype=uint8)
```

[6]:
```
y_test
```

[6]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  2,
        2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,  3,  3,  3,  3,  4,  4,
        4,  4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,
        6,  6,  6,  6,  6,  7,  7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,
        8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10, 10, 10, 10,
       10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12,
       12, 12, 13, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14,
       14, 15, 15, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16,
       17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18, 19,
       19, 19, 19, 19, 19, 19, 19], dtype=uint8)
```

Total 400 images. There are 20 people, 12 images per person in Training set and 8 images per person in the Testing set.

0.2 Normalizing the dataset.

[7]:
```
x_train = np.array(x_train, dtype='float32')/255
x_test = np.array(x_test, dtype='float32')/255
```

0.2.1 Spliting the data into train and validation

[8]:
```
from sklearn.model_selection import train_test_split
```

[9]:
```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
                                                  test_size=0.1,
                                                  random_state=99)
```

Reshaping all the images to size 112x92

[10]:
```
image_size = (112, 92, 1)
batch_size = 512

x_train = x_train.reshape(x_train.shape[0], *image_size)
```

```
x_test = x_test.reshape(x_test.shape[0], *image_size)
x_val = x_val.reshape(x_val.shape[0], *image_size)
```

0.3 Build Deep CNN model

[11]:
```
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from keras.callbacks import EarlyStopping
```

[12]:
```
face_rec_model = Sequential()

face_rec_model.add(Conv2D(filters=36,
                          kernel_size=3,
                          activation='relu',
                          input_shape= image_size))

face_rec_model.add(Conv2D(filters=36,
                          kernel_size=3,
                          activation='relu',
                          input_shape= image_size))

face_rec_model.add(MaxPooling2D(pool_size=2))

face_rec_model.add(Conv2D(filters=64,
                          kernel_size=3,
                          activation='relu',
                          input_shape= image_size))

face_rec_model.add(Conv2D(filters=64,
                          kernel_size=3,
                          activation='relu',
                          input_shape= image_size))

face_rec_model.add(MaxPooling2D(pool_size=2))

face_rec_model.add(Flatten())

face_rec_model.add(Dense(1024, activation='relu'))
face_rec_model.add(Dropout(0.5))
face_rec_model.add(Dense(512, activation='relu'))
face_rec_model.add(Dropout(0.4))

face_rec_model.add(Dense(20, activation='softmax'))
```

[13]:
```
face_rec_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 110, 90, 36)       360

 conv2d_1 (Conv2D)           (None, 108, 88, 36)       11700

 max_pooling2d (MaxPooling2D  (None, 54, 44, 36)       0
 )

 conv2d_2 (Conv2D)           (None, 52, 42, 64)        20800

 conv2d_3 (Conv2D)           (None, 50, 40, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 25, 20, 64)       0
 2D)

 flatten (Flatten)           (None, 32000)             0

 dense (Dense)               (None, 1024)              32769024

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 512)               524800

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 20)                10260

=================================================================
Total params: 33,373,872
Trainable params: 33,373,872
Non-trainable params: 0
_________________________________________________________________
```

[14]:
```
face_rec_model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(learning_rate=0.0001),
    metrics=['accuracy']
)
```

[15]:
```
callback = EarlyStopping(monitor='loss', patience=3)
```

[16]:
```
history = face_rec_model.fit(np.array(x_train), np.array(y_train),
                             batch_size=batch_size,
                             epochs=150,
                             verbose=2,
                             validation_data=(np.array(x_val), np.array(y_val)),
                             callbacks=[callback]
                             )
```

```
Epoch 1/150
1/1 - 21s - loss: 2.9956 - accuracy: 0.0463 - val_loss: 2.9941 - val_accuracy:
0.1250 - 21s/epoch - 21s/step
Epoch 2/150
1/1 - 13s - loss: 2.9941 - accuracy: 0.0926 - val_loss: 3.0013 - val_accuracy:
0.1250 - 13s/epoch - 13s/step
Epoch 3/150
1/1 - 13s - loss: 2.9766 - accuracy: 0.0602 - val_loss: 3.0072 - val_accuracy:
0.0417 - 13s/epoch - 13s/step
Epoch 4/150
1/1 - 11s - loss: 2.9862 - accuracy: 0.1204 - val_loss: 3.0117 - val_accuracy:
0.0417 - 11s/epoch - 11s/step
Epoch 5/150
1/1 - 11s - loss: 2.9758 - accuracy: 0.0556 - val_loss: 3.0155 - val_accuracy:
0.0417 - 11s/epoch - 11s/step
Epoch 6/150
1/1 - 11s - loss: 2.9611 - accuracy: 0.0602 - val_loss: 3.0209 - val_accuracy:
0.0417 - 11s/epoch - 11s/step
Epoch 7/150
1/1 - 11s - loss: 2.9531 - accuracy: 0.0787 - val_loss: 3.0262 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 8/150
1/1 - 11s - loss: 2.9489 - accuracy: 0.0741 - val_loss: 3.0315 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 9/150
1/1 - 11s - loss: 2.9394 - accuracy: 0.1111 - val_loss: 3.0351 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 10/150
1/1 - 12s - loss: 2.9444 - accuracy: 0.0926 - val_loss: 3.0364 - val_accuracy:
0.0000e+00 - 12s/epoch - 12s/step
Epoch 11/150
1/1 - 11s - loss: 2.9304 - accuracy: 0.1111 - val_loss: 3.0352 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 12/150
1/1 - 11s - loss: 2.8983 - accuracy: 0.1250 - val_loss: 3.0329 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 13/150
1/1 - 11s - loss: 2.9047 - accuracy: 0.1204 - val_loss: 3.0306 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
```

```
Epoch 14/150
1/1 - 11s - loss: 2.8910 - accuracy: 0.1204 - val_loss: 3.0264 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 15/150
1/1 - 11s - loss: 2.8818 - accuracy: 0.1343 - val_loss: 3.0167 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 16/150
1/1 - 11s - loss: 2.8754 - accuracy: 0.1528 - val_loss: 3.0047 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 17/150
1/1 - 11s - loss: 2.7970 - accuracy: 0.2315 - val_loss: 2.9887 - val_accuracy:
0.0000e+00 - 11s/epoch - 11s/step
Epoch 18/150
1/1 - 11s - loss: 2.8438 - accuracy: 0.1389 - val_loss: 2.9655 - val_accuracy:
0.0417 - 11s/epoch - 11s/step
Epoch 19/150
1/1 - 11s - loss: 2.8234 - accuracy: 0.1898 - val_loss: 2.9334 - val_accuracy:
0.1250 - 11s/epoch - 11s/step
Epoch 20/150
1/1 - 11s - loss: 2.7688 - accuracy: 0.1852 - val_loss: 2.8975 - val_accuracy:
0.2083 - 11s/epoch - 11s/step
Epoch 21/150
1/1 - 11s - loss: 2.7693 - accuracy: 0.2222 - val_loss: 2.8597 - val_accuracy:
0.2500 - 11s/epoch - 11s/step
Epoch 22/150
1/1 - 11s - loss: 2.7090 - accuracy: 0.2546 - val_loss: 2.8181 - val_accuracy:
0.2500 - 11s/epoch - 11s/step
Epoch 23/150
1/1 - 11s - loss: 2.6626 - accuracy: 0.2870 - val_loss: 2.7802 - val_accuracy:
0.2917 - 11s/epoch - 11s/step
Epoch 24/150
1/1 - 11s - loss: 2.6705 - accuracy: 0.2176 - val_loss: 2.7474 - val_accuracy:
0.3333 - 11s/epoch - 11s/step
Epoch 25/150
1/1 - 11s - loss: 2.5955 - accuracy: 0.3194 - val_loss: 2.7084 - val_accuracy:
0.3333 - 11s/epoch - 11s/step
Epoch 26/150
1/1 - 11s - loss: 2.5803 - accuracy: 0.2454 - val_loss: 2.6613 - val_accuracy:
0.3750 - 11s/epoch - 11s/step
Epoch 27/150
1/1 - 11s - loss: 2.5858 - accuracy: 0.2824 - val_loss: 2.6097 - val_accuracy:
0.3750 - 11s/epoch - 11s/step
Epoch 28/150
1/1 - 11s - loss: 2.4455 - accuracy: 0.3380 - val_loss: 2.5664 - val_accuracy:
0.4167 - 11s/epoch - 11s/step
Epoch 29/150
1/1 - 11s - loss: 2.4835 - accuracy: 0.3102 - val_loss: 2.5235 - val_accuracy:
0.3750 - 11s/epoch - 11s/step
```

```
Epoch 30/150
1/1 - 11s - loss: 2.3771 - accuracy: 0.3519 - val_loss: 2.4721 - val_accuracy:
0.4167 - 11s/epoch - 11s/step
Epoch 31/150
1/1 - 11s - loss: 2.3568 - accuracy: 0.3333 - val_loss: 2.4162 - val_accuracy:
0.4167 - 11s/epoch - 11s/step
Epoch 32/150
1/1 - 11s - loss: 2.3015 - accuracy: 0.3565 - val_loss: 2.3480 - val_accuracy:
0.4167 - 11s/epoch - 11s/step
Epoch 33/150
1/1 - 11s - loss: 2.2027 - accuracy: 0.4213 - val_loss: 2.2836 - val_accuracy:
0.5000 - 11s/epoch - 11s/step
Epoch 34/150
1/1 - 11s - loss: 2.1498 - accuracy: 0.4630 - val_loss: 2.2163 - val_accuracy:
0.5417 - 11s/epoch - 11s/step
Epoch 35/150
1/1 - 11s - loss: 2.0164 - accuracy: 0.4769 - val_loss: 2.1435 - val_accuracy:
0.5833 - 11s/epoch - 11s/step
Epoch 36/150
1/1 - 11s - loss: 2.0111 - accuracy: 0.4954 - val_loss: 2.0767 - val_accuracy:
0.6250 - 11s/epoch - 11s/step
Epoch 37/150
1/1 - 11s - loss: 1.9572 - accuracy: 0.4954 - val_loss: 2.0013 - val_accuracy:
0.6250 - 11s/epoch - 11s/step
Epoch 38/150
1/1 - 11s - loss: 1.8679 - accuracy: 0.4907 - val_loss: 1.9390 - val_accuracy:
0.6250 - 11s/epoch - 11s/step
Epoch 39/150
1/1 - 11s - loss: 1.8664 - accuracy: 0.4676 - val_loss: 1.8742 - val_accuracy:
0.6667 - 11s/epoch - 11s/step
Epoch 40/150
1/1 - 11s - loss: 1.7379 - accuracy: 0.5602 - val_loss: 1.7962 - val_accuracy:
0.6667 - 11s/epoch - 11s/step
Epoch 41/150
1/1 - 11s - loss: 1.6637 - accuracy: 0.5509 - val_loss: 1.7136 - val_accuracy:
0.6667 - 11s/epoch - 11s/step
Epoch 42/150
1/1 - 11s - loss: 1.5786 - accuracy: 0.5880 - val_loss: 1.6277 - val_accuracy:
0.7083 - 11s/epoch - 11s/step
Epoch 43/150
1/1 - 11s - loss: 1.5579 - accuracy: 0.5370 - val_loss: 1.5414 - val_accuracy:
0.6667 - 11s/epoch - 11s/step
Epoch 44/150
1/1 - 11s - loss: 1.4494 - accuracy: 0.5880 - val_loss: 1.4533 - val_accuracy:
0.7917 - 11s/epoch - 11s/step
Epoch 45/150
1/1 - 11s - loss: 1.3812 - accuracy: 0.6389 - val_loss: 1.3793 - val_accuracy:
0.7500 - 11s/epoch - 11s/step
```

```
Epoch 46/150
1/1 - 11s - loss: 1.3272 - accuracy: 0.6435 - val_loss: 1.3121 - val_accuracy:
0.7917 - 11s/epoch - 11s/step
Epoch 47/150
1/1 - 11s - loss: 1.2817 - accuracy: 0.6574 - val_loss: 1.2788 - val_accuracy:
0.7500 - 11s/epoch - 11s/step
Epoch 48/150
1/1 - 11s - loss: 1.2305 - accuracy: 0.6574 - val_loss: 1.2643 - val_accuracy:
0.7500 - 11s/epoch - 11s/step
Epoch 49/150
1/1 - 11s - loss: 1.2048 - accuracy: 0.6574 - val_loss: 1.2333 - val_accuracy:
0.7500 - 11s/epoch - 11s/step
Epoch 50/150
1/1 - 11s - loss: 1.1059 - accuracy: 0.6806 - val_loss: 1.1496 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 51/150
1/1 - 11s - loss: 1.0194 - accuracy: 0.7361 - val_loss: 1.0146 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 52/150
1/1 - 11s - loss: 1.0003 - accuracy: 0.6944 - val_loss: 0.8967 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 53/150
1/1 - 11s - loss: 0.9412 - accuracy: 0.7315 - val_loss: 0.8396 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 54/150
1/1 - 11s - loss: 0.9842 - accuracy: 0.7130 - val_loss: 0.8034 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 55/150
1/1 - 11s - loss: 0.8116 - accuracy: 0.7963 - val_loss: 0.7523 - val_accuracy:
0.7917 - 11s/epoch - 11s/step
Epoch 56/150
1/1 - 11s - loss: 0.7583 - accuracy: 0.8194 - val_loss: 0.7169 - val_accuracy:
0.7917 - 11s/epoch - 11s/step
Epoch 57/150
1/1 - 11s - loss: 0.7403 - accuracy: 0.7917 - val_loss: 0.7016 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 58/150
1/1 - 11s - loss: 0.6814 - accuracy: 0.8472 - val_loss: 0.7119 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 59/150
1/1 - 11s - loss: 0.6330 - accuracy: 0.8194 - val_loss: 0.6636 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 60/150
1/1 - 11s - loss: 0.5846 - accuracy: 0.8611 - val_loss: 0.5764 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 61/150
1/1 - 11s - loss: 0.5758 - accuracy: 0.8519 - val_loss: 0.4738 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
```

```
Epoch 62/150
1/1 - 11s - loss: 0.5438 - accuracy: 0.8380 - val_loss: 0.3854 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 63/150
1/1 - 11s - loss: 0.4716 - accuracy: 0.8796 - val_loss: 0.3308 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 64/150
1/1 - 11s - loss: 0.4932 - accuracy: 0.8426 - val_loss: 0.3334 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 65/150
1/1 - 11s - loss: 0.4577 - accuracy: 0.8657 - val_loss: 0.3506 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 66/150
1/1 - 11s - loss: 0.4077 - accuracy: 0.8981 - val_loss: 0.3613 - val_accuracy:
0.8750 - 11s/epoch - 11s/step
Epoch 67/150
1/1 - 11s - loss: 0.3674 - accuracy: 0.8935 - val_loss: 0.3699 - val_accuracy:
0.8333 - 11s/epoch - 11s/step
Epoch 68/150
1/1 - 11s - loss: 0.3590 - accuracy: 0.9074 - val_loss: 0.3504 - val_accuracy:
0.8750 - 11s/epoch - 11s/step
Epoch 69/150
1/1 - 11s - loss: 0.3590 - accuracy: 0.8981 - val_loss: 0.2954 - val_accuracy:
0.9167 - 11s/epoch - 11s/step
Epoch 70/150
1/1 - 11s - loss: 0.3253 - accuracy: 0.9259 - val_loss: 0.2983 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 71/150
1/1 - 11s - loss: 0.3482 - accuracy: 0.8981 - val_loss: 0.2814 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 72/150
1/1 - 11s - loss: 0.3748 - accuracy: 0.8889 - val_loss: 0.2098 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 73/150
1/1 - 11s - loss: 0.2670 - accuracy: 0.9213 - val_loss: 0.1711 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 74/150
1/1 - 11s - loss: 0.2788 - accuracy: 0.9306 - val_loss: 0.1460 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 75/150
1/1 - 11s - loss: 0.2626 - accuracy: 0.9398 - val_loss: 0.1432 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 76/150
1/1 - 11s - loss: 0.2545 - accuracy: 0.9213 - val_loss: 0.1618 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 77/150
1/1 - 11s - loss: 0.2054 - accuracy: 0.9676 - val_loss: 0.1814 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
```

```
Epoch 78/150
1/1 - 11s - loss: 0.2142 - accuracy: 0.9352 - val_loss: 0.1504 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 79/150
1/1 - 11s - loss: 0.2050 - accuracy: 0.9583 - val_loss: 0.1002 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 80/150
1/1 - 11s - loss: 0.1806 - accuracy: 0.9630 - val_loss: 0.0832 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 81/150
1/1 - 11s - loss: 0.1701 - accuracy: 0.9583 - val_loss: 0.0843 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 82/150
1/1 - 11s - loss: 0.1349 - accuracy: 0.9861 - val_loss: 0.0989 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 83/150
1/1 - 11s - loss: 0.1233 - accuracy: 0.9815 - val_loss: 0.1335 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 84/150
1/1 - 11s - loss: 0.1447 - accuracy: 0.9583 - val_loss: 0.1354 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 85/150
1/1 - 11s - loss: 0.1384 - accuracy: 0.9769 - val_loss: 0.1081 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 86/150
1/1 - 11s - loss: 0.1071 - accuracy: 0.9861 - val_loss: 0.0730 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 87/150
1/1 - 11s - loss: 0.1424 - accuracy: 0.9722 - val_loss: 0.0509 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 88/150
1/1 - 11s - loss: 0.1589 - accuracy: 0.9537 - val_loss: 0.0451 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 89/150
1/1 - 11s - loss: 0.0739 - accuracy: 0.9907 - val_loss: 0.0463 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 90/150
1/1 - 11s - loss: 0.1460 - accuracy: 0.9583 - val_loss: 0.0492 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 91/150
1/1 - 11s - loss: 0.1094 - accuracy: 0.9769 - val_loss: 0.0594 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
Epoch 92/150
1/1 - 11s - loss: 0.0715 - accuracy: 0.9954 - val_loss: 0.0773 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 93/150
1/1 - 11s - loss: 0.0818 - accuracy: 0.9815 - val_loss: 0.0688 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
```

```
Epoch 94/150
1/1 - 11s - loss: 0.0858 - accuracy: 0.9722 - val_loss: 0.0520 - val_accuracy:
0.9583 - 11s/epoch - 11s/step
Epoch 95/150
1/1 - 11s - loss: 0.0926 - accuracy: 0.9769 - val_loss: 0.0354 - val_accuracy:
1.0000 - 11s/epoch - 11s/step
```

[17]:
```
print(history.history.keys())
```

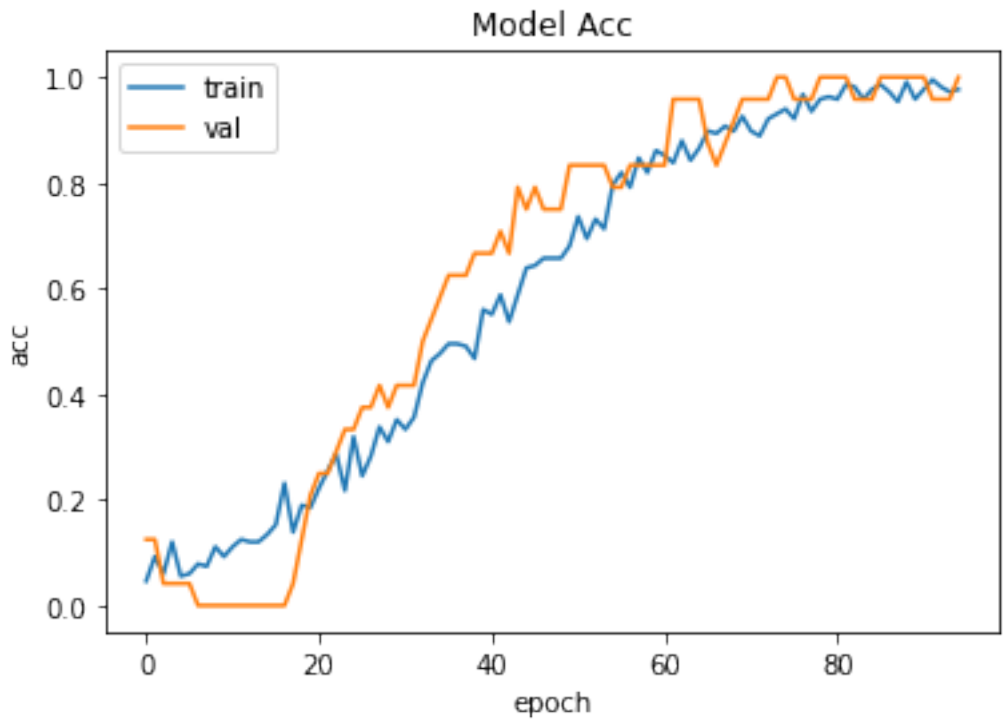
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

0.4 Model Evaluation

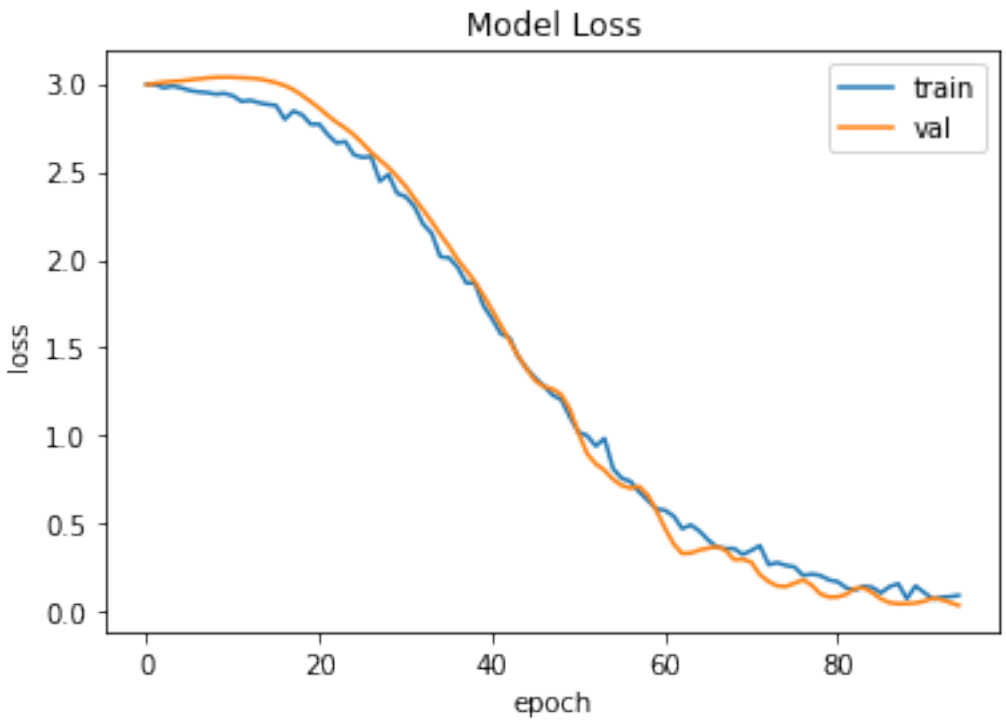
[18]:
```
eval = face_rec_model.evaluate(np.array(x_test), np.array(y_test), verbose=0)

print('Test loss ', eval[0])
print('Test Acc ', eval[1])
```

```
Test loss  0.21071676909923553
Test Acc  0.949999988079071
```

[19]:
```
plt.title('Model Acc')
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('epoch')
plt.ylabel('acc')
plt.legend(['train', 'val'])
plt.show()
```



[20]:
```
plt.title('Model Loss')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'val'])
plt.show()
```



[21]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc

import seaborn as sns

from keras.utils import np_utils
```

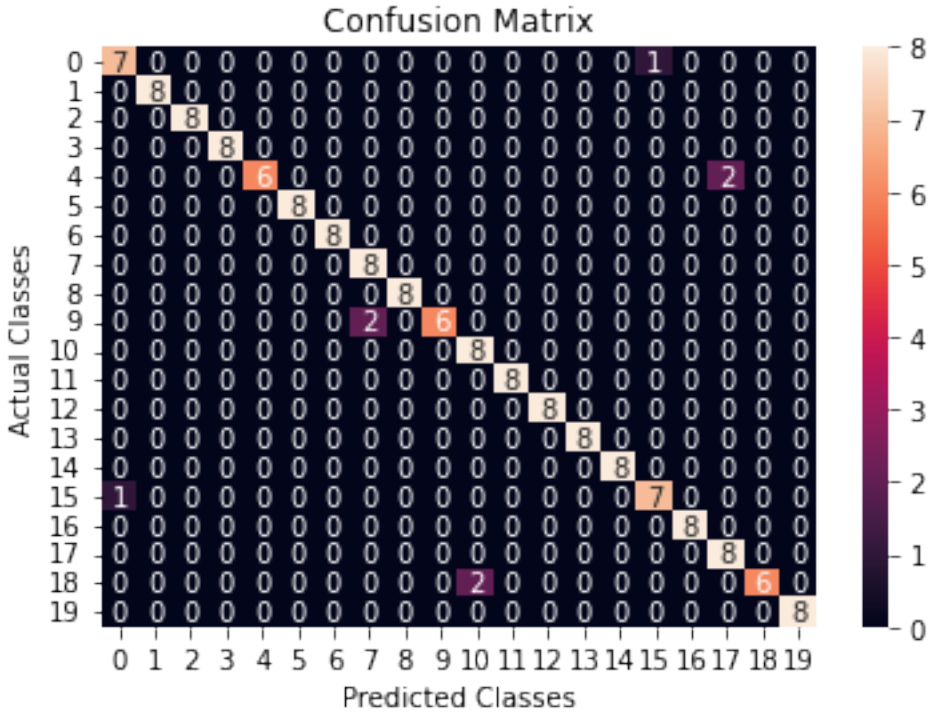
[22]:
```
pred = np.array(face_rec_model.predict(x_test))

y_pred = np.argmax(face_rec_model.predict(x_test), axis=-1)
```

[23]:
```
acc_score = accuracy_score(y_test, y_pred)
print('Acc Score ', acc_score)
```

```
Acc Score  0.95
```

[24]:
```
cmatrix = confusion_matrix(y_test, y_pred)

ax = sns.heatmap(cmatrix, annot=True)
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted Classes')
ax.set_ylabel('Actual Classes')
```

```
plt.show()
```


Confusion Matrix
Actual Classes
Predicted Classes
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
1 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0
5 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0
8 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0 2 0 6 0 0 0 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 0
12 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0
13 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0
14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0
15 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0
16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0
17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0
18 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 6 0
19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8
8
7
6
5
4
3
2
1
0