Perform Facial Recognition with Deep Learning in Keras Using CNN :

Importing required libraries :

In [1]:
```
import tensorflow as tf
import keras
from keras.models import Model, Sequential
from keras.layers import Conv2D, MaxPooling2D, Input, Dense, Flatten, Dropout

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
```

Loading the dataset :

In [2]:
```
dataset = np.load(r'C:\Users\hp\OneDrive\Desktop\AI\Advanced deep learning and Computer vision\Datasets\Project datasets\ORL_faces.npz')
print(type(dataset))
print('=' * 50)

for i in dataset.iterkeys():
    print(i)
```

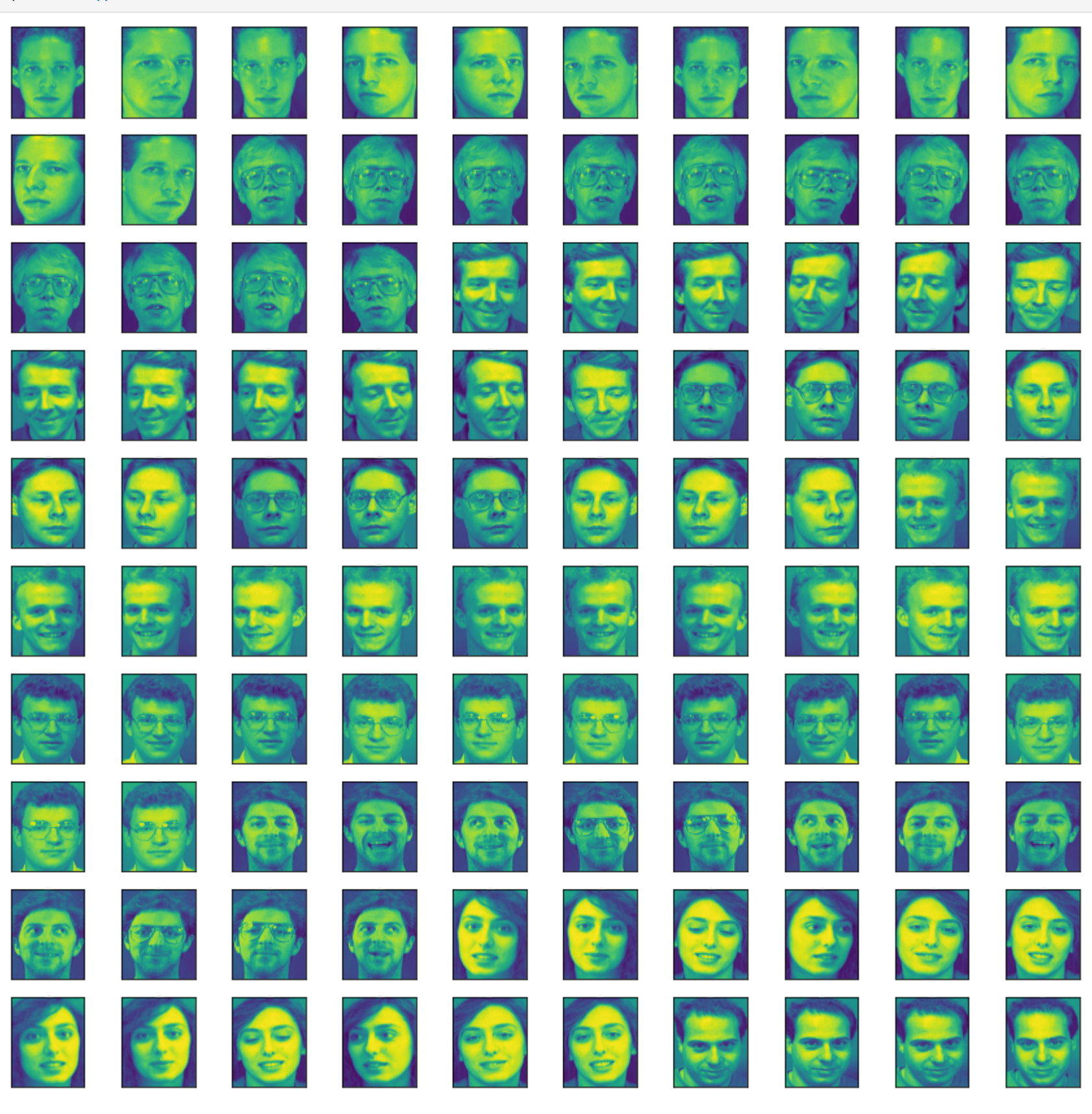
```
<class 'numpy.lib.npyio.NpzFile'>
==================================================
testY
testX
trainX
trainY
```

```
<ipython-input-2-e444dab27a27>:5: DeprecationWarning: NpzFile.iterkeys is deprecated in python 3, to match the removal of dict.iterkeys. Use .keys() instead.
  for i in dataset.iterkeys():
```

In [3]:
```
### Splitting the data :
X_train = dataset['trainX']
X_test = dataset['testX']
y_train = dataset['trainY']
y_test = dataset['testY']

## Printing shapes :
print('Shape of X_train is ', X_train.shape)
print('Shape of X_test is ', X_test.shape)
print('Shape of y_train is ', y_train.shape)
print('Shape of y_test is ', y_test.shape)
```

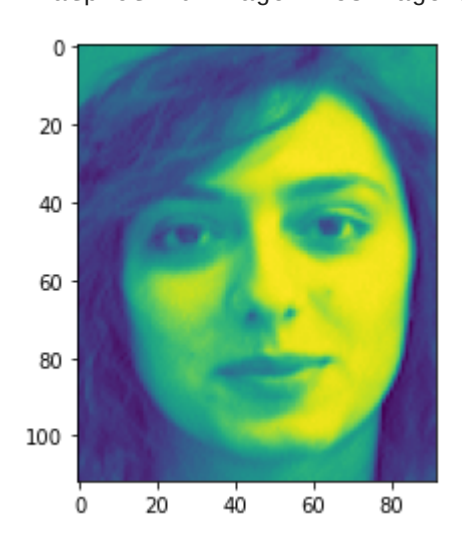
```
Shape of X_train is  (240, 10304)
Shape of X_test is  (160, 10304)
Shape of y_train is  (240,)
Shape of y_test is  (160,)
```

In [4]:
```
### Visualising some images :
plt.figure(figsize = (15, 15))

for i in range(100):
    plt.subplot(10, 10, i + 1)
    plt.imshow(X_train[i].reshape(112, 92))
    plt.xlabel(y_train[i], color = 'white')
    #plt.axis('off')
    plt.xticks([])
    plt.yticks([])

plt.show()
```



In [5]:
```
plt.imshow(X_train[91].reshape(112, 92))
```

Out[5]: `<matplotlib.image.AxesImage at 0x2030f355bb0>`



In [6]:
```
X_train.shape, X_test.shape
```

Out[6]: `((240, 10304), (160, 10304))`

In [7]:
```
### Reshaping to images :
X_train = X_train.reshape(240, 112, 92)
X_test = X_test.reshape(160, 112, 92)

print('X_train shape : ', X_train.shape)
print('X_test shape : ', X_test.shape)
```

```
X_train shape :  (240, 112, 92)
X_test shape :  (160, 112, 92)
```

Normalize the images :

In [8]:
```
### Scaling all the images :
X_train_scaled = X_train.astype('float32') / 255.0
X_test_scaled = X_test.astype('float32') / 255.0
```

In [9]:
```
### Checking whether all the images are in same size :
for img in X_train_scaled:
    if img.shape != (112, 92):
        print(img.shape)
else:
    print('all images are in the same size')
```

```
all images are in the same size
```

Splitting the train data into train and validation sets :

In [10]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train_scaled, y_train, test_size = 0.05, random_state = 21, shuffle=True)

print('X_train shape :', X_train.shape)
print('X_val shape :', X_val.shape)
print('=' * 50)
print('y_train shape :', y_train.shape)
print('y_val shape :', y_val.shape)
```

```
X_train shape : (228, 112, 92)
X_val shape : (12, 112, 92)
==================================================
y_train shape : (228,)
y_val shape : (12,)
```

In [11]:
```
X_train = np.expand_dims(X_train, axis = 3)
X_val = np.expand_dims(X_val, axis = 3)
X_test = np.expand_dims(X_test, axis = 3)
y_train = np.expand_dims(y_train, axis = 1)
y_val = np.expand_dims(y_val, axis = 1)
y_test = np.expand_dims(y_test, axis = 1)

print('X_train shape :', X_train.shape)
print('X_val shape :', X_val.shape)
print('X_test shape :', X_test.shape)
print('=' * 50)
print('y_train shape :', y_train.shape)
print('y_test shape :', y_test.shape)
print('y_val shape :', y_val.shape)
```

```
X_train shape : (228, 112, 92, 1)
X_val shape : (12, 112, 92, 1)
X_test shape : (160, 112, 92, 1)
==================================================
y_train shape : (228, 1)
y_test shape : (160, 1)
y_val shape : (12, 1)
```

CNN Model :

In [12]:
```
### Creating the CNN Model :
input_shape = Input(shape = (112, 92, 1))

x = Conv2D(36, (7, 7), activation = 'relu')(input_shape)
x = MaxPooling2D(2, 2)(x)
x = Conv2D(54, (5, 5), activation = 'relu')(x)
x = MaxPooling2D(2, 2)(x)

x = Flatten()(x)
x = Dense(2024, activation = 'relu')(x)
x = Dropout(0.5)(x)
x = Dense(1024, activation = 'relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation = 'relu')(x)
x = Dropout(0.5)(x)

output_class = Dense(20, activation = 'softmax')(x)

CNN_model = Model(input_shape, output_class)
CNN_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 112, 92, 1)]      0
_________________________________________________________________
conv2d (Conv2D)              (None, 106, 86, 36)       1800
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 53, 43, 36)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 49, 39, 54)        48654
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 19, 54)        0
_________________________________________________________________
flatten (Flatten)            (None, 24624)             0
_________________________________________________________________
dense (Dense)                (None, 2024)              49841000
_________________________________________________________________
dropout (Dropout)            (None, 2024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2073600
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 20)                10260
=================================================================
Total params: 52,500,114
Trainable params: 52,500,114
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```
CNN_model.compile(loss = 'sparse_categorical_crossentropy',
                  optimizer = 'adam',
             metrics = ['accuracy'])
```

In [14]:
```
his = CNN_model.fit(X_train, y_train, epochs = 50, validation_data = (X_val, y_val), verbose = 2)
```

```
Epoch 1/50
8/8 - 47s - loss: 3.0716 - accuracy: 0.0482 - val_loss: 2.9952 - val_accuracy: 0.0833
Epoch 2/50
8/8 - 11s - loss: 3.0044 - accuracy: 0.0351 - val_loss: 3.0059 - val_accuracy: 0.0000e+00
Epoch 3/50
8/8 - 11s - loss: 3.0110 - accuracy: 0.0789 - val_loss: 3.0248 - val_accuracy: 0.0000e+00
Epoch 4/50
8/8 - 11s - loss: 2.9866 - accuracy: 0.0614 - val_loss: 3.0194 - val_accuracy: 0.0000e+00
Epoch 5/50
8/8 - 12s - loss: 2.9813 - accuracy: 0.0658 - val_loss: 3.0077 - val_accuracy: 0.0000e+00
Epoch 6/50
8/8 - 9s - loss: 2.9556 - accuracy: 0.0789 - val_loss: 2.9572 - val_accuracy: 0.0000e+00
Epoch 7/50
8/8 - 9s - loss: 2.9203 - accuracy: 0.1140 - val_loss: 2.7607 - val_accuracy: 0.2500
Epoch 8/50
8/8 - 9s - loss: 2.7042 - accuracy: 0.1623 - val_loss: 2.2292 - val_accuracy: 0.4167
Epoch 9/50
8/8 - 12s - loss: 2.1566 - accuracy: 0.3246 - val_loss: 1.2216 - val_accuracy: 0.6667
Epoch 10/50
8/8 - 9s - loss: 1.7271 - accuracy: 0.4386 - val_loss: 0.6439 - val_accuracy: 0.8333
Epoch 11/50
8/8 - 9s - loss: 1.2808 - accuracy: 0.5921 - val_loss: 0.7013 - val_accuracy: 0.7500
Epoch 12/50
8/8 - 9s - loss: 0.9359 - accuracy: 0.7061 - val_loss: 0.4677 - val_accuracy: 0.8333
Epoch 13/50
8/8 - 13s - loss: 0.6916 - accuracy: 0.7500 - val_loss: 0.0749 - val_accuracy: 1.0000
Epoch 14/50
8/8 - 10s - loss: 0.3872 - accuracy: 0.8640 - val_loss: 0.1182 - val_accuracy: 1.0000
Epoch 15/50
8/8 - 9s - loss: 0.2808 - accuracy: 0.9035 - val_loss: 0.1095 - val_accuracy: 0.9167
Epoch 16/50
8/8 - 9s - loss: 0.2115 - accuracy: 0.9430 - val_loss: 0.0861 - val_accuracy: 1.0000
Epoch 17/50
8/8 - 9s - loss: 0.1330 - accuracy: 0.9649 - val_loss: 0.0067 - val_accuracy: 1.0000
Epoch 18/50
8/8 - 9s - loss: 0.1009 - accuracy: 0.9737 - val_loss: 0.0019 - val_accuracy: 1.0000
Epoch 19/50
8/8 - 9s - loss: 0.0246 - accuracy: 0.9912 - val_loss: 2.1750e-04 - val_accuracy: 1.0000
Epoch 20/50
8/8 - 9s - loss: 0.0507 - accuracy: 0.9912 - val_loss: 0.0315 - val_accuracy: 1.0000
Epoch 21/50
8/8 - 10s - loss: 0.0574 - accuracy: 0.9781 - val_loss: 9.6428e-04 - val_accuracy: 1.0000
Epoch 22/50
8/8 - 9s - loss: 0.0481 - accuracy: 0.9781 - val_loss: 2.0160e-04 - val_accuracy: 1.0000
Epoch 23/50
8/8 - 9s - loss: 0.0570 - accuracy: 0.9868 - val_loss: 0.0066 - val_accuracy: 1.0000
Epoch 24/50
8/8 - 9s - loss: 0.0714 - accuracy: 0.9781 - val_loss: 0.0252 - val_accuracy: 1.0000
Epoch 25/50
8/8 - 9s - loss: 0.0268 - accuracy: 0.9912 - val_loss: 1.9802e-04 - val_accuracy: 1.0000
Epoch 26/50
8/8 - 9s - loss: 0.0343 - accuracy: 0.9868 - val_loss: 0.0024 - val_accuracy: 1.0000
Epoch 27/50
8/8 - 9s - loss: 0.0135 - accuracy: 1.0000 - val_loss: 0.0095 - val_accuracy: 1.0000
Epoch 28/50
8/8 - 9s - loss: 0.0214 - accuracy: 0.9956 - val_loss: 0.0030 - val_accuracy: 1.0000
Epoch 29/50
8/8 - 10s - loss: 0.0628 - accuracy: 0.9737 - val_loss: 0.0155 - val_accuracy: 1.0000
Epoch 30/50
8/8 - 13s - loss: 0.0470 - accuracy: 0.9956 - val_loss: 0.0410 - val_accuracy: 1.0000
Epoch 31/50
8/8 - 11s - loss: 0.0213 - accuracy: 0.9912 - val_loss: 3.2062e-04 - val_accuracy: 1.0000
Epoch 32/50
8/8 - 11s - loss: 0.0187 - accuracy: 0.9912 - val_loss: 9.2657e-04 - val_accuracy: 1.0000
Epoch 33/50
8/8 - 11s - loss: 0.0241 - accuracy: 0.9912 - val_loss: 0.0028 - val_accuracy: 1.0000
Epoch 34/50
8/8 - 11s - loss: 0.0093 - accuracy: 1.0000 - val_loss: 3.8603e-04 - val_accuracy: 1.0000
Epoch 35/50
8/8 - 11s - loss: 0.0171 - accuracy: 0.9956 - val_loss: 1.1963e-04 - val_accuracy: 1.0000
Epoch 36/50
8/8 - 11s - loss: 0.0197 - accuracy: 0.9912 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 37/50
8/8 - 12s - loss: 0.0296 - accuracy: 0.9912 - val_loss: 0.0010 - val_accuracy: 1.0000
Epoch 38/50
8/8 - 11s - loss: 0.0393 - accuracy: 0.9912 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 39/50
8/8 - 11s - loss: 0.0292 - accuracy: 0.9956 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 40/50
8/8 - 11s - loss: 0.0515 - accuracy: 0.9825 - val_loss: 0.0025 - val_accuracy: 1.0000
Epoch 41/50
8/8 - 11s - loss: 0.0100 - accuracy: 0.9956 - val_loss: 5.2778e-04 - val_accuracy: 1.0000
Epoch 42/50
8/8 - 12s - loss: 0.0051 - accuracy: 1.0000 - val_loss: 3.8075e-05 - val_accuracy: 1.0000
Epoch 43/50
8/8 - 11s - loss: 0.0256 - accuracy: 0.9956 - val_loss: 0.0048 - val_accuracy: 1.0000
Epoch 44/50
8/8 - 11s - loss: 0.0142 - accuracy: 0.9912 - val_loss: 0.0024 - val_accuracy: 1.0000
Epoch 45/50
8/8 - 12s - loss: 0.0233 - accuracy: 0.9912 - val_loss: 1.5100e-06 - val_accuracy: 1.0000
Epoch 46/50
8/8 - 11s - loss: 0.0019 - accuracy: 1.0000 - val_loss: 1.4901e-07 - val_accuracy: 1.0000
Epoch 47/50
8/8 - 12s - loss: 0.0165 - accuracy: 0.9912 - val_loss: 4.2266e-04 - val_accuracy: 1.0000
Epoch 48/50
8/8 - 11s - loss: 0.0364 - accuracy: 0.9868 - val_loss: 2.6122e-04 - val_accuracy: 1.0000
Epoch 49/50
8/8 - 9s - loss: 0.0216 - accuracy: 0.9956 - val_loss: 2.7963e-05 - val_accuracy: 1.0000
Epoch 50/50
8/8 - 10s - loss: 0.0179 - accuracy: 0.9912 - val_loss: 7.1660e-04 - val_accuracy: 1.0000
```

In [ ]:

Model evaluation :

In [15]:
```
loss, accuarcy = CNN_model.evaluate(X_test, y_test)
print('Loss of the test data :', loss)
print('Accuarcy of the test data :', accuarcy)
```

```
5/5 [==============================] - 2s 300ms/step - loss: 80.0032 - accuracy: 0.9250
Loss of the test data : 80.003173828125
Accuarcy of the test data : 0.925000011920929
```

In [ ]:

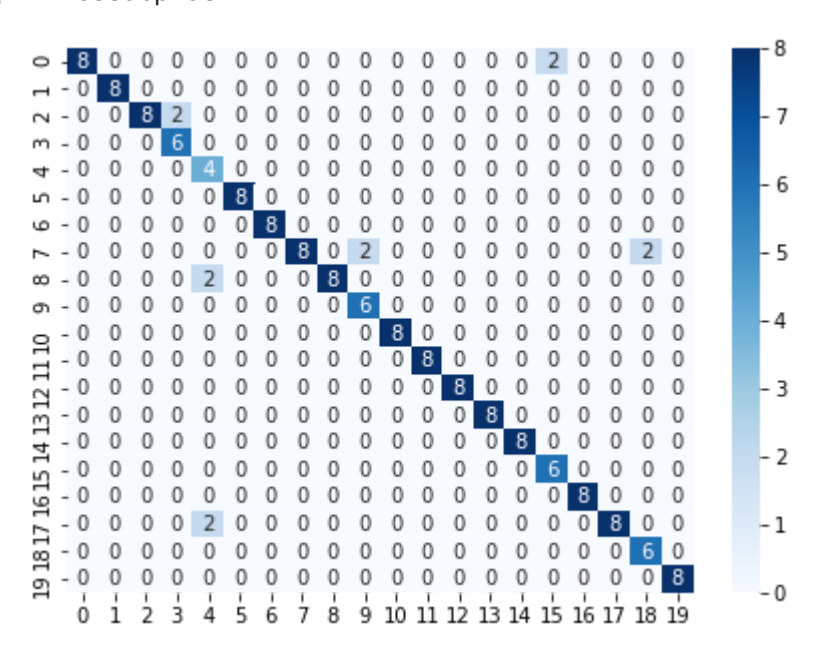
Predictions :

In [16]:
```
predictions = CNN_model.predict(X_test)
prediction_list = []

for prediction in predictions:
    temp = np.argmax(prediction)
    prediction_list.append(temp)

print('Accuracy score of the test data :', accuracy_score(prediction_list, y_test))
print('=' * 100)
print('Confusion matrix of the test data :', confusion_matrix(prediction_list, y_test))
print('=' * 100)
print('Classification report of the test data :\n', classification_report(prediction_list, y_test))
```

```
Accuracy score of the test data : 0.925
====================================================================================================
Confusion matrix of the test data : [[8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 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 2 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 0 0 0 0]
 [0 0 0 0 4 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 2 0 0 0 0 0 0 0 0 2 0]
 [0 0 0 0 2 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 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]
 [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 0 0 0 0 0 0 0 0 8 0 0 0]
 [0 0 0 0 2 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 0 0 0 0 0 0 0 0 8]]
====================================================================================================
Classification report of the test data :
               precision    recall  f1-score   support

           0       1.00      0.80      0.89        10
           1       1.00      1.00      1.00         8
           2       1.00      0.80      0.89        10
           3       0.75      1.00      0.86         6
           4       0.50      1.00      0.67         4
           5       1.00      1.00      1.00         8
           6       1.00      1.00      1.00         8
           7       1.00      0.67      0.80        12
           8       1.00      0.80      0.89        10
           9       0.75      1.00      0.86         6
          10       1.00      1.00      1.00         8
          11       1.00      1.00      1.00         8
          12       1.00      1.00      1.00         8
          13       1.00      1.00      1.00         8
          14       1.00      1.00      1.00         8
          15       0.75      1.00      0.86         6
          16       1.00      1.00      1.00         8
          17       1.00      0.80      0.89        10
          18       0.75      1.00      0.86         6
          19       1.00      1.00      1.00         8

    accuracy                           0.93       160
   macro avg       0.93      0.94      0.92       160
weighted avg       0.95      0.93      0.93       160
```

In [17]:
```
plt.figure(figsize = (7, 5))
sns.heatmap(confusion_matrix(prediction_list, y_test), annot = True, cmap = plt.cm.Blues)
```

Out[17]: `<AxesSubplot:>`



Saving the trained model :

In [18]:
```
CNN_model.save('FaceRecognition_model')
loaded_model = tf.keras.models.load_model('FaceRecognition_model')
loaded_model
```

```
INFO:tensorflow:Assets written to: FaceRecognition_model\assets
```

Out[18]: `<tensorflow.python.keras.engine.functional.Functional at 0x2030ca55640>`

In [19]:
```
loaded_model.predict(X_test)
```

Out[19]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

In [20]:
```
X_test_10 = np.expand_dims(X_test[10], axis = 0)
X_test_10.shape
```

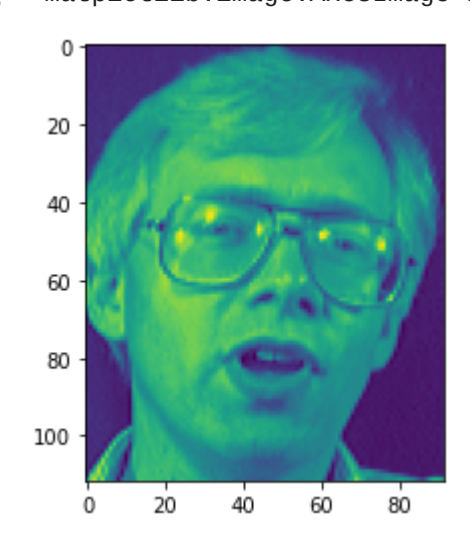
Out[20]: `(1, 112, 92, 1)`

In [21]:
```
X_test.shape, X_test[0].shape
```

Out[21]: `((160, 112, 92, 1), (112, 92, 1))`

In [22]:
```
plt.imshow(X_test[10])
```

Out[22]: `<matplotlib.image.AxesImage at 0x2030e548460>`



In [23]:
```
y_test[10]
```

Out[23]: `array([1], dtype=uint8)`

In [24]:
```
pre = loaded_model.predict(X_test_10)
np.argmax(pre)
```

Out[24]: `1`

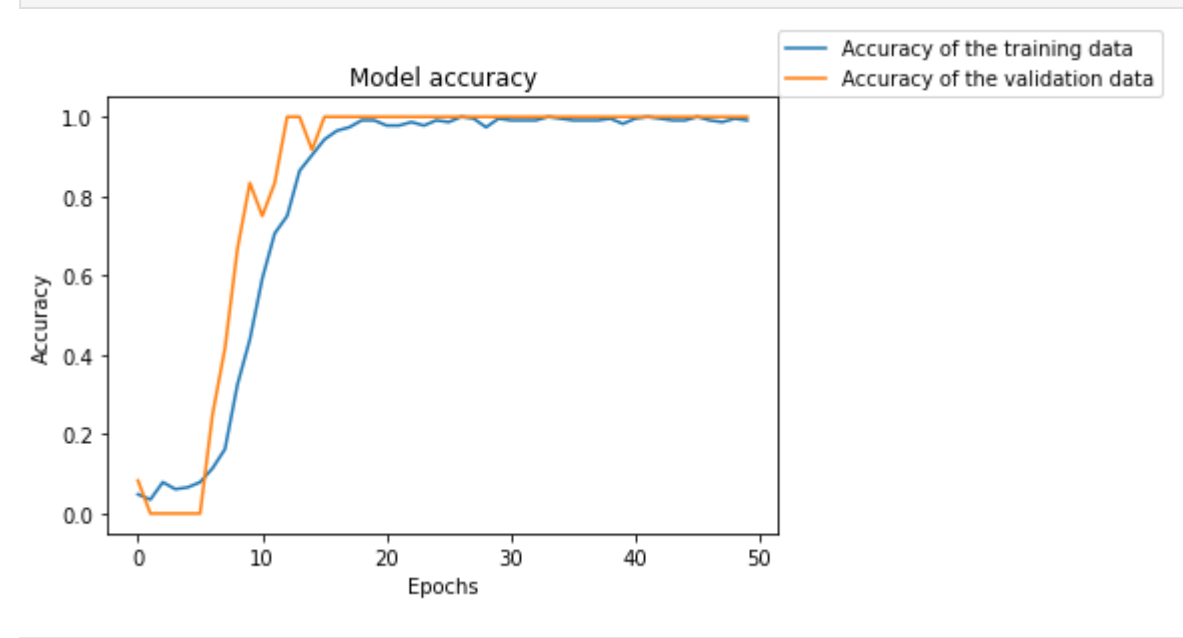
In [ ]:

Plot the result :

In [25]:
```
print(his.history.keys())
```

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

In [26]:
```
### Plotting the accuracy graphs of both train validation data :
plt.plot(his.history['accuracy'], label = 'Accuracy of the training data')
plt.plot(his.history['val_accuracy'], label = 'Accuracy of the validation data')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model accuracy')
plt.legend(loc = (1, 1))
plt.show()
```



In [27]:
```
### Plotting the accuracy graphs of both train validation data :
plt.plot(his.history['loss'], label = 'loss of the training data')
plt.plot(his.history['val_loss'], label = 'loss of the validation data')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.title('Model loss')
plt.legend(loc = (1, 1))
plt.show()
```