Project 2: Perform Facial Recognition with Deep Learning in Keras Using CNN

Step1: Input the required libraries

In [1]:

```
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.optimizers import Adam
from keras.callbacks import TensorBoard

import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.optimizers import Adam
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
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

Step2: Load the dataset after loading the dataset and have to normalize every image.

- **Note: you need to convert the format of the image to float or double.**

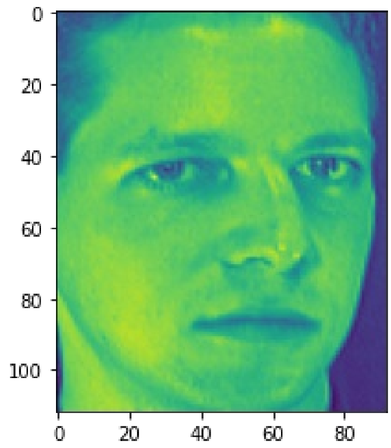
In [2]:

```
#load dataset
data = np.load('../input/orlfaces/ORL_faces.npz')

# 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 0.18039216]
 [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  2  3  3  3  3  3  3  3  3  3  3  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
  6  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  8  9  9  9  9  9  9  9  9  9  9  9  9
 10 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 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 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]
x_test shape: (160, 10304)
```

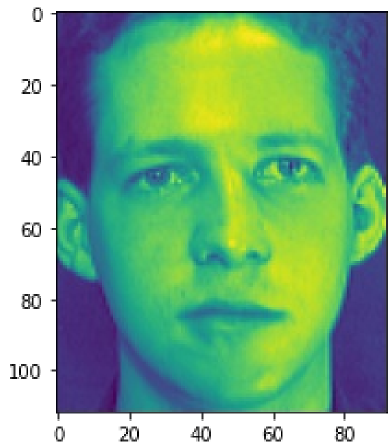
In [3]:

```
# To See the images in Train and Test data.

c=x_train[1].reshape(112,92)
plt.imshow(c)
plt.show()
d=x_test[1].reshape(112,92)
plt.imshow(d)
```



Out[3]:

```
<matplotlib.image.AxesImage at 0x7f8c69dcb2d0>
```



Step 3: Split the dataset

Validation data and Train

Validation dataset: this data set is used to minimize overfitting.If the accuracy over the training data set increases, but the accuracy over then validation data set stays the same or decreases, then you're overfitting

your neural network and you should stop training.

- **Usually use 30 percent of every dataset as the validation data but Here we only used 5 percent because the number of images in this dataset is very low.**

In [4]:

```
x_train, x_valid, y_train, y_valid= train_test_split(
    x_train, y_train, test_size=.05, random_state=42,)
```

Step 4: Transform the images to equal sizes to feed in CNN

In [5]:

```
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[0]))
```

```
x_train shape: 228
x_test shape: 160
```

Step 5: Build CNN model:

CNN have 3 main layer:

- **1-Convolotional layer**
- **2- pooling layer**
- **3- fully connected layer**

  we could build a new architecture of CNN by changing the number and position of layers.

In [6]:

```
#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(lr=0.0001),
    metrics=['accuracy']
)
```

Show the model's parameters.

In [7]:

```
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
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
_________________________________________________________________
```

Step 6: Train the Model

- **Note: You can change the number of epochs**

In [8]:

```
history=cnn_model.fit(
    np.array(x_train), np.array(y_train), batch_size=512,
    epochs=180, verbose=2,
    validation_data=(np.array(x_valid),np.array(y_valid)),
)
```

```
Epoch 1/180
1/1 - 5s - loss: 3.0067 - accuracy: 0.0439 - val_loss: 3.0210 - val_accuracy: 0.0000e+00
Epoch 2/180
1/1 - 4s - loss: 3.0148 - accuracy: 0.0570 - val_loss: 3.0287 - val_accuracy: 0.0000e+00
Epoch 3/180
1/1 - 4s - loss: 2.9961 - accuracy: 0.0439 - val_loss: 3.0427 - val_accuracy: 0.0000e+00
Epoch 4/180
1/1 - 4s - loss: 3.0120 - accuracy: 0.0439 - val_loss: 3.0462 - val_accuracy: 0.0000e+00
Epoch 5/180
1/1 - 4s - loss: 2.9659 - accuracy: 0.0833 - val_loss: 3.0475 - val_accuracy: 0.0000e+00
Epoch 6/180
1/1 - 4s - loss: 2.9954 - accuracy: 0.0789 - val_loss: 3.0443 - val_accuracy: 0.0000e+00
Epoch 7/180
1/1 - 4s - loss: 2.9953 - accuracy: 0.0307 - val_loss: 3.0396 - val_accuracy: 0.0000e+00
Epoch 8/180
1/1 - 4s - loss: 2.9690 - accuracy: 0.0482 - val_loss: 3.0332 - val_accuracy: 0.0833
Epoch 9/180
1/1 - 4s - loss: 2.9843 - accuracy: 0.0877 - val_loss: 3.0285 - val_accuracy: 0.0833
```

```
Epoch 10/180
1/1 - 4s - loss: 2.9669 - accuracy: 0.0702 - val_loss: 3.0216 - val_accuracy: 0.0833
Epoch 11/180
1/1 - 4s - loss: 2.9524 - accuracy: 0.0789 - val_loss: 3.0136 - val_accuracy: 0.0833
Epoch 12/180
1/1 - 4s - loss: 2.9800 - accuracy: 0.0658 - val_loss: 3.0052 - val_accuracy: 0.0833
Epoch 13/180
1/1 - 4s - loss: 2.9610 - accuracy: 0.1009 - val_loss: 2.9972 - val_accuracy: 0.1667
Epoch 14/180
1/1 - 4s - loss: 2.9426 - accuracy: 0.1009 - val_loss: 2.9901 - val_accuracy: 0.1667
Epoch 15/180
1/1 - 4s - loss: 2.9341 - accuracy: 0.1184 - val_loss: 2.9842 - val_accuracy: 0.1667
Epoch 16/180
1/1 - 4s - loss: 2.9345 - accuracy: 0.1140 - val_loss: 2.9785 - val_accuracy: 0.1667
Epoch 17/180
1/1 - 4s - loss: 2.9230 - accuracy: 0.1228 - val_loss: 2.9738 - val_accuracy: 0.1667
Epoch 18/180
1/1 - 4s - loss: 2.9161 - accuracy: 0.1535 - val_loss: 2.9684 - val_accuracy: 0.1667
Epoch 19/180
1/1 - 4s - loss: 2.9082 - accuracy: 0.1272 - val_loss: 2.9619 - val_accuracy: 0.1667
Epoch 20/180
1/1 - 4s - loss: 2.8876 - accuracy: 0.1667 - val_loss: 2.9547 - val_accuracy: 0.1667
Epoch 21/180
1/1 - 4s - loss: 2.8782 - accuracy: 0.1579 - val_loss: 2.9469 - val_accuracy: 0.2500
Epoch 22/180
1/1 - 4s - loss: 2.8638 - accuracy: 0.1798 - val_loss: 2.9390 - val_accuracy: 0.2500
Epoch 23/180
1/1 - 4s - loss: 2.8458 - accuracy: 0.1667 - val_loss: 2.9282 - val_accuracy: 0.2500
Epoch 24/180
1/1 - 4s - loss: 2.8436 - accuracy: 0.1886 - val_loss: 2.9147 - val_accuracy: 0.3333
Epoch 25/180
1/1 - 4s - loss: 2.8149 - accuracy: 0.1667 - val_loss: 2.8993 - val_accuracy: 0.3333
Epoch 26/180
1/1 - 4s - loss: 2.8036 - accuracy: 0.2018 - val_loss: 2.8830 - val_accuracy: 0.3333
Epoch 27/180
1/1 - 4s - loss: 2.8016 - accuracy: 0.1798 - val_loss: 2.8683 - val_accuracy: 0.3333
Epoch 28/180
1/1 - 4s - loss: 2.7664 - accuracy: 0.2061 - val_loss: 2.8497 - val_accuracy: 0.3333
Epoch 29/180
1/1 - 4s - loss: 2.7659 - accuracy: 0.2061 - val_loss: 2.8251 - val_accuracy: 0.4167
Epoch 30/180
1/1 - 4s - loss: 2.7579 - accuracy: 0.1667 - val_loss: 2.7970 - val_accuracy: 0.3333
Epoch 31/180
1/1 - 4s - loss: 2.6914 - accuracy: 0.2281 - val_loss: 2.7687 - val_accuracy: 0.3333
Epoch 32/180
1/1 - 4s - loss: 2.6726 - accuracy: 0.2544 - val_loss: 2.7407 - val_accuracy: 0.3333
Epoch 33/180
1/1 - 4s - loss: 2.6381 - accuracy: 0.2544 - val_loss: 2.7091 - val_accuracy: 0.3333
Epoch 34/180
1/1 - 4s - loss: 2.6081 - accuracy: 0.2544 - val_loss: 2.6687 - val_accuracy: 0.4167
Epoch 35/180
1/1 - 4s - loss: 2.5779 - accuracy: 0.3333 - val_loss: 2.6256 - val_accuracy: 0.5000
Epoch 36/180
1/1 - 4s - loss: 2.5572 - accuracy: 0.2675 - val_loss: 2.5798 - val_accuracy: 0.5000
Epoch 37/180
1/1 - 4s - loss: 2.4751 - accuracy: 0.3246 - val_loss: 2.5301 - val_accuracy: 0.5833
Epoch 38/180
1/1 - 4s - loss: 2.4563 - accuracy: 0.3114 - val_loss: 2.4794 - val_accuracy: 0.5000
Epoch 39/180
1/1 - 4s - loss: 2.3741 - accuracy: 0.3202 - val_loss: 2.4233 - val_accuracy: 0.5000
Epoch 40/180
1/1 - 4s - loss: 2.3543 - accuracy: 0.3289 - val_loss: 2.3616 - val_accuracy: 0.5000
Epoch 41/180
1/1 - 4s - loss: 2.2143 - accuracy: 0.3816 - val_loss: 2.3001 - val_accuracy: 0.5833
Epoch 42/180
1/1 - 4s - loss: 2.1558 - accuracy: 0.3991 - val_loss: 2.2373 - val_accuracy: 0.5833
Epoch 43/180
1/1 - 4s - loss: 2.2395 - accuracy: 0.3246 - val_loss: 2.1826 - val_accuracy: 0.5833
Epoch 44/180
1/1 - 4s - loss: 2.0898 - accuracy: 0.3904 - val_loss: 2.1254 - val_accuracy: 0.6667
Epoch 45/180
1/1 - 4s - loss: 2.0329 - accuracy: 0.4211 - val_loss: 2.0627 - val_accuracy: 0.6667
```

```
Epoch 46/180
1/1 - 4s - loss: 1.9870 - accuracy: 0.4561 - val_loss: 2.0014 - val_accuracy: 0.5833
Epoch 47/180
1/1 - 4s - loss: 1.8667 - accuracy: 0.4781 - val_loss: 1.9381 - val_accuracy: 0.5833
Epoch 48/180
1/1 - 4s - loss: 1.9186 - accuracy: 0.4123 - val_loss: 1.8609 - val_accuracy: 0.5833
Epoch 49/180
1/1 - 4s - loss: 1.8044 - accuracy: 0.4561 - val_loss: 1.7807 - val_accuracy: 0.5833
Epoch 50/180
1/1 - 4s - loss: 1.7259 - accuracy: 0.4956 - val_loss: 1.6968 - val_accuracy: 0.7500
Epoch 51/180
1/1 - 4s - loss: 1.7180 - accuracy: 0.4868 - val_loss: 1.6183 - val_accuracy: 0.7500
Epoch 52/180
1/1 - 4s - loss: 1.6958 - accuracy: 0.4737 - val_loss: 1.5497 - val_accuracy: 0.7500
Epoch 53/180
1/1 - 4s - loss: 1.6157 - accuracy: 0.5132 - val_loss: 1.4895 - val_accuracy: 0.7500
Epoch 54/180
1/1 - 4s - loss: 1.6315 - accuracy: 0.5439 - val_loss: 1.4374 - val_accuracy: 0.8333
Epoch 55/180
1/1 - 4s - loss: 1.4955 - accuracy: 0.5526 - val_loss: 1.3799 - val_accuracy: 0.8333
Epoch 56/180
1/1 - 4s - loss: 1.4983 - accuracy: 0.5482 - val_loss: 1.3233 - val_accuracy: 0.8333
Epoch 57/180
1/1 - 4s - loss: 1.4125 - accuracy: 0.5965 - val_loss: 1.2732 - val_accuracy: 0.8333
Epoch 58/180
1/1 - 4s - loss: 1.3845 - accuracy: 0.6228 - val_loss: 1.2286 - val_accuracy: 0.8333
Epoch 59/180
1/1 - 4s - loss: 1.3377 - accuracy: 0.6272 - val_loss: 1.1837 - val_accuracy: 0.8333
Epoch 60/180
1/1 - 4s - loss: 1.2796 - accuracy: 0.6272 - val_loss: 1.1379 - val_accuracy: 0.8333
Epoch 61/180
1/1 - 4s - loss: 1.1339 - accuracy: 0.7018 - val_loss: 1.0853 - val_accuracy: 0.8333
Epoch 62/180
1/1 - 4s - loss: 1.0792 - accuracy: 0.7237 - val_loss: 1.0256 - val_accuracy: 0.8333
Epoch 63/180
1/1 - 4s - loss: 1.0503 - accuracy: 0.7237 - val_loss: 0.9634 - val_accuracy: 0.8333
Epoch 64/180
1/1 - 4s - loss: 0.9939 - accuracy: 0.7281 - val_loss: 0.9071 - val_accuracy: 0.8333
Epoch 65/180
1/1 - 4s - loss: 0.9732 - accuracy: 0.7193 - val_loss: 0.8552 - val_accuracy: 0.9167
Epoch 66/180
1/1 - 4s - loss: 0.9713 - accuracy: 0.6798 - val_loss: 0.8238 - val_accuracy: 0.9167
Epoch 67/180
1/1 - 4s - loss: 0.8425 - accuracy: 0.7412 - val_loss: 0.8172 - val_accuracy: 0.8333
Epoch 68/180
1/1 - 4s - loss: 0.8331 - accuracy: 0.7500 - val_loss: 0.7797 - val_accuracy: 0.8333
Epoch 69/180
1/1 - 4s - loss: 0.7623 - accuracy: 0.7632 - val_loss: 0.7375 - val_accuracy: 0.9167
Epoch 70/180
1/1 - 4s - loss: 0.7559 - accuracy: 0.7895 - val_loss: 0.6813 - val_accuracy: 0.9167
Epoch 71/180
1/1 - 4s - loss: 0.7281 - accuracy: 0.7939 - val_loss: 0.6417 - val_accuracy: 0.8333
Epoch 72/180
1/1 - 4s - loss: 0.6102 - accuracy: 0.8289 - val_loss: 0.6405 - val_accuracy: 0.8333
Epoch 73/180
1/1 - 4s - loss: 0.6511 - accuracy: 0.8333 - val_loss: 0.6131 - val_accuracy: 0.8333
Epoch 74/180
1/1 - 4s - loss: 0.6381 - accuracy: 0.8114 - val_loss: 0.5629 - val_accuracy: 0.8333
Epoch 75/180
1/1 - 4s - loss: 0.5005 - accuracy: 0.8728 - val_loss: 0.5376 - val_accuracy: 0.9167
Epoch 76/180
1/1 - 4s - loss: 0.5836 - accuracy: 0.8070 - val_loss: 0.5133 - val_accuracy: 0.9167
Epoch 77/180
1/1 - 4s - loss: 0.5838 - accuracy: 0.8465 - val_loss: 0.5208 - val_accuracy: 0.8333
Epoch 78/180
1/1 - 4s - loss: 0.4885 - accuracy: 0.8421 - val_loss: 0.5781 - val_accuracy: 0.8333
Epoch 79/180
1/1 - 4s - loss: 0.5442 - accuracy: 0.8421 - val_loss: 0.5031 - val_accuracy: 0.8333
Epoch 80/180
1/1 - 4s - loss: 0.4433 - accuracy: 0.8816 - val_loss: 0.4199 - val_accuracy: 0.9167
Epoch 81/180
1/1 - 4s - loss: 0.4375 - accuracy: 0.8684 - val_loss: 0.3917 - val_accuracy: 0.9167
```

```
Epoch 82/180
1/1 - 4s - loss: 0.4324 - accuracy: 0.8991 - val_loss: 0.3760 - val_accuracy: 0.9167
Epoch 83/180
1/1 - 4s - loss: 0.4279 - accuracy: 0.8553 - val_loss: 0.3861 - val_accuracy: 0.9167
Epoch 84/180
1/1 - 4s - loss: 0.3495 - accuracy: 0.9211 - val_loss: 0.4127 - val_accuracy: 0.8333
Epoch 85/180
1/1 - 4s - loss: 0.3128 - accuracy: 0.9123 - val_loss: 0.4071 - val_accuracy: 0.8333
Epoch 86/180
1/1 - 4s - loss: 0.3856 - accuracy: 0.8860 - val_loss: 0.3767 - val_accuracy: 0.9167
Epoch 87/180
1/1 - 4s - loss: 0.3788 - accuracy: 0.8816 - val_loss: 0.3385 - val_accuracy: 0.9167
Epoch 88/180
1/1 - 4s - loss: 0.3292 - accuracy: 0.9167 - val_loss: 0.3120 - val_accuracy: 0.9167
Epoch 89/180
1/1 - 4s - loss: 0.2871 - accuracy: 0.9518 - val_loss: 0.2984 - val_accuracy: 0.9167
Epoch 90/180
1/1 - 4s - loss: 0.3049 - accuracy: 0.9167 - val_loss: 0.2914 - val_accuracy: 0.9167
Epoch 91/180
1/1 - 4s - loss: 0.2345 - accuracy: 0.9342 - val_loss: 0.2839 - val_accuracy: 0.9167
Epoch 92/180
1/1 - 4s - loss: 0.2303 - accuracy: 0.9474 - val_loss: 0.2855 - val_accuracy: 0.9167
Epoch 93/180
1/1 - 4s - loss: 0.2215 - accuracy: 0.9430 - val_loss: 0.2664 - val_accuracy: 0.9167
Epoch 94/180
1/1 - 4s - loss: 0.2471 - accuracy: 0.9254 - val_loss: 0.2319 - val_accuracy: 0.9167
Epoch 95/180
1/1 - 4s - loss: 0.2255 - accuracy: 0.9430 - val_loss: 0.2180 - val_accuracy: 0.9167
Epoch 96/180
1/1 - 4s - loss: 0.2045 - accuracy: 0.9561 - val_loss: 0.2164 - val_accuracy: 0.9167
Epoch 97/180
1/1 - 4s - loss: 0.2504 - accuracy: 0.9386 - val_loss: 0.2260 - val_accuracy: 0.9167
Epoch 98/180
1/1 - 4s - loss: 0.2082 - accuracy: 0.9518 - val_loss: 0.2392 - val_accuracy: 0.9167
Epoch 99/180
1/1 - 4s - loss: 0.1900 - accuracy: 0.9518 - val_loss: 0.2291 - val_accuracy: 0.9167
Epoch 100/180
1/1 - 4s - loss: 0.1712 - accuracy: 0.9605 - val_loss: 0.2216 - val_accuracy: 0.9167
Epoch 101/180
1/1 - 4s - loss: 0.1904 - accuracy: 0.9518 - val_loss: 0.2190 - val_accuracy: 0.9167
Epoch 102/180
1/1 - 4s - loss: 0.1850 - accuracy: 0.9561 - val_loss: 0.2220 - val_accuracy: 0.9167
Epoch 103/180
1/1 - 4s - loss: 0.1543 - accuracy: 0.9605 - val_loss: 0.2063 - val_accuracy: 0.9167
Epoch 104/180
1/1 - 4s - loss: 0.1451 - accuracy: 0.9737 - val_loss: 0.2099 - val_accuracy: 0.9167
Epoch 105/180
1/1 - 4s - loss: 0.2052 - accuracy: 0.9474 - val_loss: 0.2001 - val_accuracy: 0.9167
Epoch 106/180
1/1 - 4s - loss: 0.1243 - accuracy: 0.9868 - val_loss: 0.1914 - val_accuracy: 0.9167
Epoch 107/180
1/1 - 4s - loss: 0.1314 - accuracy: 0.9825 - val_loss: 0.1794 - val_accuracy: 0.9167
Epoch 108/180
1/1 - 4s - loss: 0.1241 - accuracy: 0.9605 - val_loss: 0.1647 - val_accuracy: 0.9167
Epoch 109/180
1/1 - 4s - loss: 0.1413 - accuracy: 0.9561 - val_loss: 0.1635 - val_accuracy: 0.9167
Epoch 110/180
1/1 - 4s - loss: 0.1635 - accuracy: 0.9561 - val_loss: 0.1597 - val_accuracy: 0.9167
Epoch 111/180
1/1 - 4s - loss: 0.1360 - accuracy: 0.9781 - val_loss: 0.1422 - val_accuracy: 0.9167
Epoch 112/180
1/1 - 4s - loss: 0.1139 - accuracy: 0.9781 - val_loss: 0.1314 - val_accuracy: 1.0000
Epoch 113/180
1/1 - 4s - loss: 0.1272 - accuracy: 0.9693 - val_loss: 0.1270 - val_accuracy: 1.0000
Epoch 114/180
1/1 - 4s - loss: 0.0833 - accuracy: 0.9781 - val_loss: 0.1207 - val_accuracy: 1.0000
Epoch 115/180
1/1 - 4s - loss: 0.1255 - accuracy: 0.9781 - val_loss: 0.1208 - val_accuracy: 1.0000
Epoch 116/180
1/1 - 4s - loss: 0.0835 - accuracy: 0.9781 - val_loss: 0.1252 - val_accuracy: 0.9167
Epoch 117/180
1/1 - 4s - loss: 0.1024 - accuracy: 0.9868 - val_loss: 0.1244 - val_accuracy: 0.9167
```

```
Epoch 118/180
1/1 - 4s - loss: 0.0915 - accuracy: 0.9825 - val_loss: 0.1151 - val_accuracy: 1.0000
Epoch 119/180
1/1 - 4s - loss: 0.1199 - accuracy: 0.9605 - val_loss: 0.0989 - val_accuracy: 1.0000
Epoch 120/180
1/1 - 4s - loss: 0.1108 - accuracy: 0.9649 - val_loss: 0.0894 - val_accuracy: 1.0000
Epoch 121/180
1/1 - 4s - loss: 0.0770 - accuracy: 0.9868 - val_loss: 0.0801 - val_accuracy: 1.0000
Epoch 122/180
1/1 - 4s - loss: 0.1125 - accuracy: 0.9781 - val_loss: 0.0670 - val_accuracy: 1.0000
Epoch 123/180
1/1 - 4s - loss: 0.0889 - accuracy: 0.9737 - val_loss: 0.0564 - val_accuracy: 1.0000
Epoch 124/180
1/1 - 4s - loss: 0.0781 - accuracy: 0.9825 - val_loss: 0.0487 - val_accuracy: 1.0000
Epoch 125/180
1/1 - 4s - loss: 0.0765 - accuracy: 0.9912 - val_loss: 0.0463 - val_accuracy: 1.0000
Epoch 126/180
1/1 - 4s - loss: 0.0668 - accuracy: 0.9868 - val_loss: 0.0468 - val_accuracy: 1.0000
Epoch 127/180
1/1 - 4s - loss: 0.0681 - accuracy: 0.9912 - val_loss: 0.0473 - val_accuracy: 1.0000
Epoch 128/180
1/1 - 4s - loss: 0.0657 - accuracy: 0.9956 - val_loss: 0.0479 - val_accuracy: 1.0000
Epoch 129/180
1/1 - 4s - loss: 0.0602 - accuracy: 0.9912 - val_loss: 0.0435 - val_accuracy: 1.0000
Epoch 130/180
1/1 - 4s - loss: 0.0702 - accuracy: 0.9825 - val_loss: 0.0415 - val_accuracy: 1.0000
Epoch 131/180
1/1 - 4s - loss: 0.0500 - accuracy: 0.9956 - val_loss: 0.0401 - val_accuracy: 1.0000
Epoch 132/180
1/1 - 4s - loss: 0.0665 - accuracy: 0.9825 - val_loss: 0.0432 - val_accuracy: 1.0000
Epoch 133/180
1/1 - 4s - loss: 0.0457 - accuracy: 0.9912 - val_loss: 0.0454 - val_accuracy: 1.0000
Epoch 134/180
1/1 - 4s - loss: 0.0603 - accuracy: 0.9868 - val_loss: 0.0473 - val_accuracy: 1.0000
Epoch 135/180
1/1 - 4s - loss: 0.0588 - accuracy: 0.9868 - val_loss: 0.0473 - val_accuracy: 1.0000
Epoch 136/180
1/1 - 4s - loss: 0.0822 - accuracy: 0.9825 - val_loss: 0.0466 - val_accuracy: 1.0000
Epoch 137/180
1/1 - 4s - loss: 0.0601 - accuracy: 0.9825 - val_loss: 0.0431 - val_accuracy: 1.0000
Epoch 138/180
1/1 - 4s - loss: 0.0707 - accuracy: 0.9825 - val_loss: 0.0403 - val_accuracy: 1.0000
Epoch 139/180
1/1 - 4s - loss: 0.0483 - accuracy: 0.9912 - val_loss: 0.0398 - val_accuracy: 1.0000
Epoch 140/180
1/1 - 4s - loss: 0.0485 - accuracy: 0.9868 - val_loss: 0.0402 - val_accuracy: 1.0000
Epoch 141/180
1/1 - 4s - loss: 0.0400 - accuracy: 0.9912 - val_loss: 0.0385 - val_accuracy: 1.0000
Epoch 142/180
1/1 - 4s - loss: 0.0455 - accuracy: 0.9912 - val_loss: 0.0337 - val_accuracy: 1.0000
Epoch 143/180
1/1 - 4s - loss: 0.0411 - accuracy: 0.9912 - val_loss: 0.0298 - val_accuracy: 1.0000
Epoch 144/180
1/1 - 4s - loss: 0.0463 - accuracy: 0.9912 - val_loss: 0.0263 - val_accuracy: 1.0000
Epoch 145/180
1/1 - 4s - loss: 0.0572 - accuracy: 0.9825 - val_loss: 0.0224 - val_accuracy: 1.0000
Epoch 146/180
1/1 - 4s - loss: 0.0633 - accuracy: 0.9825 - val_loss: 0.0205 - val_accuracy: 1.0000
Epoch 147/180
1/1 - 4s - loss: 0.0459 - accuracy: 0.9956 - val_loss: 0.0190 - val_accuracy: 1.0000
Epoch 148/180
1/1 - 4s - loss: 0.0385 - accuracy: 0.9956 - val_loss: 0.0177 - val_accuracy: 1.0000
Epoch 149/180
1/1 - 4s - loss: 0.0613 - accuracy: 0.9912 - val_loss: 0.0179 - val_accuracy: 1.0000
Epoch 150/180
1/1 - 4s - loss: 0.0319 - accuracy: 0.9956 - val_loss: 0.0177 - val_accuracy: 1.0000
Epoch 151/180
1/1 - 4s - loss: 0.0369 - accuracy: 0.9912 - val_loss: 0.0158 - val_accuracy: 1.0000
Epoch 152/180
1/1 - 4s - loss: 0.0567 - accuracy: 0.9825 - val_loss: 0.0158 - val_accuracy: 1.0000
Epoch 153/180
1/1 - 4s - loss: 0.0317 - accuracy: 1.0000 - val_loss: 0.0162 - val_accuracy: 1.0000
```

```
Epoch 154/180
1/1 - 4s - loss: 0.0257 - accuracy: 1.0000 - val_loss: 0.0174 - val_accuracy: 1.0000
Epoch 155/180
1/1 - 4s - loss: 0.0579 - accuracy: 0.9825 - val_loss: 0.0196 - val_accuracy: 1.0000
Epoch 156/180
1/1 - 4s - loss: 0.0399 - accuracy: 0.9912 - val_loss: 0.0238 - val_accuracy: 1.0000
Epoch 157/180
1/1 - 4s - loss: 0.0331 - accuracy: 0.9912 - val_loss: 0.0258 - val_accuracy: 1.0000
Epoch 158/180
1/1 - 4s - loss: 0.0276 - accuracy: 0.9956 - val_loss: 0.0246 - val_accuracy: 1.0000
Epoch 159/180
1/1 - 4s - loss: 0.0238 - accuracy: 1.0000 - val_loss: 0.0232 - val_accuracy: 1.0000
Epoch 160/180
1/1 - 4s - loss: 0.0259 - accuracy: 1.0000 - val_loss: 0.0217 - val_accuracy: 1.0000
Epoch 161/180
1/1 - 4s - loss: 0.0534 - accuracy: 0.9825 - val_loss: 0.0174 - val_accuracy: 1.0000
Epoch 162/180
1/1 - 4s - loss: 0.0359 - accuracy: 0.9912 - val_loss: 0.0142 - val_accuracy: 1.0000
Epoch 163/180
1/1 - 4s - loss: 0.0301 - accuracy: 0.9912 - val_loss: 0.0121 - val_accuracy: 1.0000
Epoch 164/180
1/1 - 4s - loss: 0.0241 - accuracy: 1.0000 - val_loss: 0.0115 - val_accuracy: 1.0000
Epoch 165/180
1/1 - 4s - loss: 0.0361 - accuracy: 0.9912 - val_loss: 0.0106 - val_accuracy: 1.0000
Epoch 166/180
1/1 - 4s - loss: 0.0267 - accuracy: 1.0000 - val_loss: 0.0096 - val_accuracy: 1.0000
Epoch 167/180
1/1 - 4s - loss: 0.0369 - accuracy: 0.9912 - val_loss: 0.0090 - val_accuracy: 1.0000
Epoch 168/180
1/1 - 4s - loss: 0.0203 - accuracy: 0.9912 - val_loss: 0.0095 - val_accuracy: 1.0000
Epoch 169/180
1/1 - 4s - loss: 0.0488 - accuracy: 0.9825 - val_loss: 0.0133 - val_accuracy: 1.0000
Epoch 170/180
1/1 - 4s - loss: 0.0162 - accuracy: 1.0000 - val_loss: 0.0201 - val_accuracy: 1.0000
Epoch 171/180
1/1 - 4s - loss: 0.0392 - accuracy: 0.9912 - val_loss: 0.0201 - val_accuracy: 1.0000
Epoch 172/180
1/1 - 4s - loss: 0.0221 - accuracy: 1.0000 - val_loss: 0.0158 - val_accuracy: 1.0000
Epoch 173/180
1/1 - 4s - loss: 0.0262 - accuracy: 0.9956 - val_loss: 0.0104 - val_accuracy: 1.0000
Epoch 174/180
1/1 - 4s - loss: 0.0249 - accuracy: 0.9912 - val_loss: 0.0084 - val_accuracy: 1.0000
Epoch 175/180
1/1 - 4s - loss: 0.0230 - accuracy: 1.0000 - val_loss: 0.0089 - val_accuracy: 1.0000
Epoch 176/180
1/1 - 4s - loss: 0.0245 - accuracy: 0.9956 - val_loss: 0.0100 - val_accuracy: 1.0000
Epoch 177/180
1/1 - 4s - loss: 0.0484 - accuracy: 0.9912 - val_loss: 0.0096 - val_accuracy: 1.0000
Epoch 178/180
1/1 - 4s - loss: 0.0285 - accuracy: 0.9956 - val_loss: 0.0100 - val_accuracy: 1.0000
Epoch 179/180
1/1 - 4s - loss: 0.0332 - accuracy: 0.9912 - val_loss: 0.0122 - val_accuracy: 1.0000
Epoch 180/180
1/1 - 4s - loss: 0.0176 - accuracy: 1.0000 - val_loss: 0.0177 - val_accuracy: 1.0000
```

Evaluate the test data

In [9]:

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

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

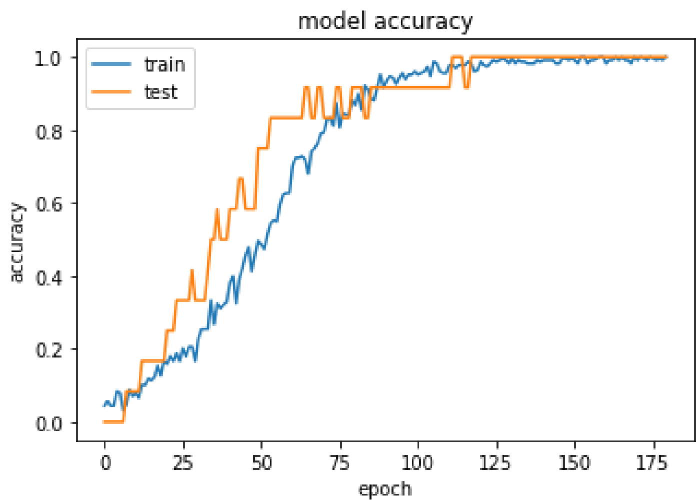
```
test loss 0.2373
test accuracy 0.9563
```

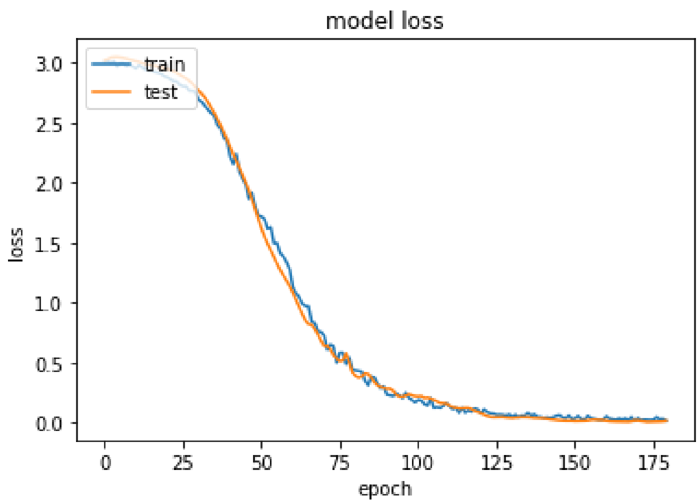
Step 7: Plot the result

In [10]:

```
# 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 [11]:

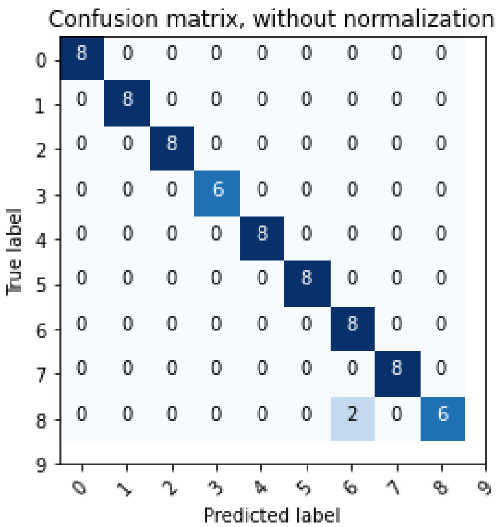
```
predicted =np.array( cnn_model.predict(x_test))
#print(predicted)
#print(y_test)
ynew = cnn_model.predict_classes(x_test)


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))
```

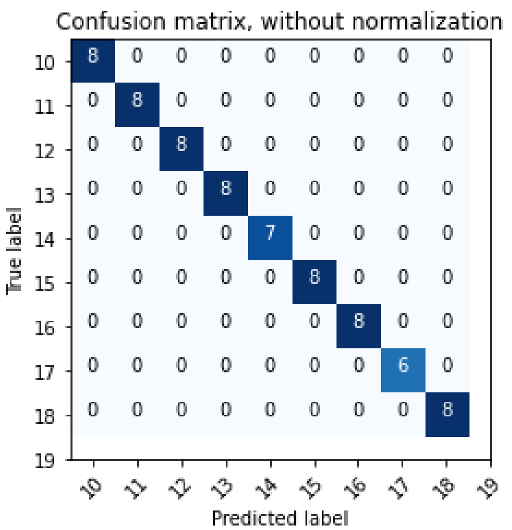
```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/sequential.py:450:
UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01
. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi
-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.pre
dict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it
uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
accuracy :
0.95625
Confusion matrix, without normalization
[[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 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]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 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 2 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]]
Confusion matrix, without normalization
```




```
Confusion matrix, without normalization
```




```
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]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 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 2 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]]
              precision    recall  f1-score   support

           0       0.89      1.00      0.94         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       0.67      1.00      0.80         8
           8       1.00      1.00      1.00         8
           9       1.00      0.75      0.86         8
          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       1.00      0.88      0.93         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                           0.96       160
   macro avg       0.97      0.96      0.96       160
weighted avg       0.97      0.96      0.96       160
```