Face Recognition using CNN

Step1:

At the first, you should input the required libraries:

In [16]:

```
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
```

Step2:

- Load Dataset :

After loading the Dataset you have to normalize every image.

Note: an image is a Uint8 matrix of pixels and for calculation, you need to convert the format of the image to float or double

In [17]:

```
#load dataset
data = np.load('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)
```

Step 3

Split 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.

- Note: we 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 [18]:

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

Step 4

for using the CNN, we need to change The size of images (The size of images must be the same)

In [19]:

```
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,)
```

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 [20]:

```
#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 [21]:

```
cnn_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 106, 86, 36)       1800
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 53, 43, 36)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 49, 39, 54)        48654
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 24, 19, 54)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 24624)             0
_________________________________________________________________
dense_9 (Dense)              (None, 2024)              49841000
```

```
dropout_7 (Dropout)          (None, 2024)              0
_________________________________________________________________
dense_10 (Dense)             (None, 1024)              2073600
_________________________________________________________________
dropout_8 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_11 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_12 (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 [22]:

```
history=cnn_model.fit(
    np.array(x_train), np.array(y_train), batch_size=512,
    epochs=250, verbose=2,
    validation_data=(np.array(x_valid),np.array(y_valid)),
)
```

```
Train on 228 samples, validate on 12 samples
Epoch 1/250
 - 12s - loss: 3.0241 - acc: 0.0351 - val_loss: 2.9856 - val_acc: 0.0000e+00
Epoch 2/250
 - 9s - loss: 2.9859 - acc: 0.0702 - val_loss: 2.9945 - val_acc: 0.1667
Epoch 3/250
 - 9s - loss: 2.9899 - acc: 0.0570 - val_loss: 2.9919 - val_acc: 0.0833
Epoch 4/250
 - 9s - loss: 2.9841 - acc: 0.1009 - val_loss: 2.9984 - val_acc: 0.0833
Epoch 5/250
 - 9s - loss: 2.9962 - acc: 0.0789 - val_loss: 3.0046 - val_acc: 0.0833
Epoch 6/250
 - 9s - loss: 3.0058 - acc: 0.0614 - val_loss: 3.0101 - val_acc: 0.0833
Epoch 7/250
 - 9s - loss: 2.9962 - acc: 0.0702 - val_loss: 3.0122 - val_acc: 0.0000e+00
Epoch 8/250
 - 9s - loss: 2.9871 - acc: 0.0614 - val_loss: 3.0126 - val_acc: 0.0000e+00
Epoch 9/250
 - 8s - loss: 2.9781 - acc: 0.0526 - val_loss: 3.0112 - val_acc: 0.0833
Epoch 10/250
 - 9s - loss: 2.9795 - acc: 0.0658 - val_loss: 3.0063 - val_acc: 0.1667
Epoch 11/250
 - 8s - loss: 2.9598 - acc: 0.0921 - val_loss: 3.0008 - val_acc: 0.1667
Epoch 12/250
 - 9s - loss: 2.9651 - acc: 0.0658 - val_loss: 2.9972 - val_acc: 0.0833
Epoch 13/250
 - 9s - loss: 2.9695 - acc: 0.0746 - val_loss: 2.9925 - val_acc: 0.0833
Epoch 14/250
 - 9s - loss: 2.9467 - acc: 0.0921 - val_loss: 2.9882 - val_acc: 0.0833
Epoch 15/250
 - 9s - loss: 2.9243 - acc: 0.1096 - val_loss: 2.9834 - val_acc: 0.0833
Epoch 16/250
 - 9s - loss: 2.9266 - acc: 0.1272 - val_loss: 2.9799 - val_acc: 0.0833
Epoch 17/250
```

```
 - 9s - loss: 2.9306 - acc: 0.1009 - val_loss: 2.9777 - val_acc: 0.0833
Epoch 18/250
 - 9s - loss: 2.9270 - acc: 0.1316 - val_loss: 2.9729 - val_acc: 0.0833
Epoch 19/250
 - 9s - loss: 2.9075 - acc: 0.1228 - val_loss: 2.9656 - val_acc: 0.0000e+00
Epoch 20/250
 - 9s - loss: 2.9092 - acc: 0.1272 - val_loss: 2.9574 - val_acc: 0.0000e+00
Epoch 21/250
 - 8s - loss: 2.9160 - acc: 0.1316 - val_loss: 2.9457 - val_acc: 0.0000e+00
Epoch 22/250
 - 9s - loss: 2.8786 - acc: 0.1842 - val_loss: 2.9315 - val_acc: 0.0000e+00
Epoch 23/250
 - 9s - loss: 2.8699 - acc: 0.1447 - val_loss: 2.9181 - val_acc: 0.0000e+00
Epoch 24/250
 - 9s - loss: 2.8586 - acc: 0.1886 - val_loss: 2.9045 - val_acc: 0.0000e+00
Epoch 25/250
 - 8s - loss: 2.8381 - acc: 0.1842 - val_loss: 2.8914 - val_acc: 0.0000e+00
Epoch 26/250
 - 9s - loss: 2.8381 - acc: 0.1974 - val_loss: 2.8779 - val_acc: 0.0000e+00
Epoch 27/250
 - 8s - loss: 2.7878 - acc: 0.1930 - val_loss: 2.8586 - val_acc: 0.1667
Epoch 28/250
 - 9s - loss: 2.8155 - acc: 0.1886 - val_loss: 2.8369 - val_acc: 0.2500
Epoch 29/250
 - 9s - loss: 2.8191 - acc: 0.1623 - val_loss: 2.8123 - val_acc: 0.2500
Epoch 30/250
 - 9s - loss: 2.7367 - acc: 0.2368 - val_loss: 2.7827 - val_acc: 0.2500
Epoch 31/250
 - 8s - loss: 2.6911 - acc: 0.2588 - val_loss: 2.7515 - val_acc: 0.2500
Epoch 32/250
 - 9s - loss: 2.6601 - acc: 0.2895 - val_loss: 2.7199 - val_acc: 0.2500
Epoch 33/250
 - 9s - loss: 2.6383 - acc: 0.2456 - val_loss: 2.6830 - val_acc: 0.3333
Epoch 34/250
 - 9s - loss: 2.5997 - acc: 0.2412 - val_loss: 2.6406 - val_acc: 0.2500
Epoch 35/250
 - 9s - loss: 2.5982 - acc: 0.2851 - val_loss: 2.5921 - val_acc: 0.2500
Epoch 36/250
 - 9s - loss: 2.5720 - acc: 0.2632 - val_loss: 2.5389 - val_acc: 0.4167
Epoch 37/250
 - 9s - loss: 2.5450 - acc: 0.2895 - val_loss: 2.4815 - val_acc: 0.4167
Epoch 38/250
 - 9s - loss: 2.4674 - acc: 0.3246 - val_loss: 2.4141 - val_acc: 0.5000
Epoch 39/250
 - 9s - loss: 2.4370 - acc: 0.2719 - val_loss: 2.3507 - val_acc: 0.6667
Epoch 40/250
 - 9s - loss: 2.4006 - acc: 0.3202 - val_loss: 2.2899 - val_acc: 0.6667
Epoch 41/250
 - 9s - loss: 2.3303 - acc: 0.3158 - val_loss: 2.2297 - val_acc: 0.6667
Epoch 42/250
 - 9s - loss: 2.2922 - acc: 0.3509 - val_loss: 2.1744 - val_acc: 0.6667
Epoch 43/250
 - 9s - loss: 2.1786 - acc: 0.3904 - val_loss: 2.1217 - val_acc: 0.5833
Epoch 44/250
 - 8s - loss: 2.1274 - acc: 0.4035 - val_loss: 2.0688 - val_acc: 0.5833
Epoch 45/250
 - 9s - loss: 2.0503 - acc: 0.4430 - val_loss: 2.0049 - val_acc: 0.5833
Epoch 46/250
 - 9s - loss: 2.0954 - acc: 0.3904 - val_loss: 1.9244 - val_acc: 0.5833
Epoch 47/250
 - 9s - loss: 2.0250 - acc: 0.4211 - val_loss: 1.8520 - val_acc: 0.5833
Epoch 48/250
 - 9s - loss: 1.9488 - acc: 0.4123 - val_loss: 1.7859 - val_acc: 0.5833
Epoch 49/250
 - 9s - loss: 1.8484 - acc: 0.4737 - val_loss: 1.7199 - val_acc: 0.6667
Epoch 50/250
 - 9s - loss: 1.7988 - acc: 0.5000 - val_loss: 1.6530 - val_acc: 0.6667
Epoch 51/250
 - 9s - loss: 1.7378 - acc: 0.5263 - val_loss: 1.5927 - val_acc: 0.6667
```

```
Epoch 52/250
 - 8s - loss: 1.7034 - acc: 0.5482 - val_loss: 1.5389 - val_acc: 0.6667
Epoch 53/250
 - 9s - loss: 1.6067 - acc: 0.5395 - val_loss: 1.4540 - val_acc: 0.6667
Epoch 54/250
 - 9s - loss: 1.5123 - acc: 0.5965 - val_loss: 1.3733 - val_acc: 0.6667
Epoch 55/250
 - 9s - loss: 1.5860 - acc: 0.5307 - val_loss: 1.3199 - val_acc: 0.7500
Epoch 56/250
 - 9s - loss: 1.4928 - acc: 0.5351 - val_loss: 1.2782 - val_acc: 0.6667
Epoch 57/250
 - 9s - loss: 1.4437 - acc: 0.5877 - val_loss: 1.2578 - val_acc: 0.6667
Epoch 58/250
 - 9s - loss: 1.3763 - acc: 0.6096 - val_loss: 1.2308 - val_acc: 0.7500
Epoch 59/250
 - 9s - loss: 1.3220 - acc: 0.6140 - val_loss: 1.1715 - val_acc: 0.7500
Epoch 60/250
 - 9s - loss: 1.2354 - acc: 0.6447 - val_loss: 1.0687 - val_acc: 0.8333
Epoch 61/250
 - 10s - loss: 1.1840 - acc: 0.6579 - val_loss: 0.9655 - val_acc: 0.8333
Epoch 62/250
 - 10s - loss: 1.0620 - acc: 0.7061 - val_loss: 0.8766 - val_acc: 0.9167
Epoch 63/250
 - 10s - loss: 1.0752 - acc: 0.7105 - val_loss: 0.8144 - val_acc: 0.9167
Epoch 64/250
 - 10s - loss: 1.1705 - acc: 0.6096 - val_loss: 0.7638 - val_acc: 0.9167
Epoch 65/250
 - 10s - loss: 1.0069 - acc: 0.7061 - val_loss: 0.7105 - val_acc: 0.9167
Epoch 66/250
 - 10s - loss: 0.9794 - acc: 0.7105 - val_loss: 0.6614 - val_acc: 0.8333
Epoch 67/250
 - 10s - loss: 0.9386 - acc: 0.7368 - val_loss: 0.6268 - val_acc: 0.9167
Epoch 68/250
 - 10s - loss: 0.8271 - acc: 0.7368 - val_loss: 0.6041 - val_acc: 0.8333
Epoch 69/250
 - 10s - loss: 0.9126 - acc: 0.7368 - val_loss: 0.5821 - val_acc: 0.9167
Epoch 70/250
 - 10s - loss: 0.7961 - acc: 0.7456 - val_loss: 0.5488 - val_acc: 0.9167
Epoch 71/250
 - 10s - loss: 0.8478 - acc: 0.7500 - val_loss: 0.5236 - val_acc: 0.9167
Epoch 72/250
 - 10s - loss: 0.6960 - acc: 0.7982 - val_loss: 0.4989 - val_acc: 0.9167
Epoch 73/250
 - 10s - loss: 0.6640 - acc: 0.8202 - val_loss: 0.4646 - val_acc: 0.9167
Epoch 74/250
 - 11s - loss: 0.6572 - acc: 0.8246 - val_loss: 0.4159 - val_acc: 0.9167
Epoch 75/250
 - 10s - loss: 0.6347 - acc: 0.8158 - val_loss: 0.3640 - val_acc: 1.0000
Epoch 76/250
 - 10s - loss: 0.6157 - acc: 0.8158 - val_loss: 0.3361 - val_acc: 0.9167
Epoch 77/250
 - 10s - loss: 0.5418 - acc: 0.8728 - val_loss: 0.3183 - val_acc: 0.9167
Epoch 78/250
 - 10s - loss: 0.5267 - acc: 0.8465 - val_loss: 0.2801 - val_acc: 0.9167
Epoch 79/250
 - 10s - loss: 0.4929 - acc: 0.8465 - val_loss: 0.2455 - val_acc: 1.0000
Epoch 80/250
 - 10s - loss: 0.4589 - acc: 0.8816 - val_loss: 0.2345 - val_acc: 0.9167
Epoch 81/250
 - 11s - loss: 0.4170 - acc: 0.8816 - val_loss: 0.2175 - val_acc: 0.9167
Epoch 82/250
 - 10s - loss: 0.4080 - acc: 0.8947 - val_loss: 0.2006 - val_acc: 0.9167
Epoch 83/250
 - 10s - loss: 0.4301 - acc: 0.8684 - val_loss: 0.1748 - val_acc: 0.9167
Epoch 84/250
 - 10s - loss: 0.4580 - acc: 0.8772 - val_loss: 0.1749 - val_acc: 0.9167
Epoch 85/250
 - 10s - loss: 0.4363 - acc: 0.8816 - val_loss: 0.1857 - val_acc: 0.9167
Epoch 86/250
```

```
 - 10s - loss: 0.4055 - acc: 0.8640 - val_loss: 0.1820 - val_acc: 0.9167
Epoch 87/250
 - 10s - loss: 0.3093 - acc: 0.9123 - val_loss: 0.1757 - val_acc: 1.0000
Epoch 88/250
 - 9s - loss: 0.3045 - acc: 0.9298 - val_loss: 0.1525 - val_acc: 1.0000
Epoch 89/250
 - 10s - loss: 0.3038 - acc: 0.9123 - val_loss: 0.1478 - val_acc: 1.0000
Epoch 90/250
 - 10s - loss: 0.2781 - acc: 0.9342 - val_loss: 0.1583 - val_acc: 1.0000
Epoch 91/250
 - 10s - loss: 0.3201 - acc: 0.8991 - val_loss: 0.1476 - val_acc: 1.0000
Epoch 92/250
 - 10s - loss: 0.2351 - acc: 0.9474 - val_loss: 0.1184 - val_acc: 1.0000
Epoch 93/250
 - 10s - loss: 0.2505 - acc: 0.9386 - val_loss: 0.1005 - val_acc: 1.0000
Epoch 94/250
 - 10s - loss: 0.2680 - acc: 0.9386 - val_loss: 0.0889 - val_acc: 1.0000
Epoch 95/250
 - 10s - loss: 0.2424 - acc: 0.9342 - val_loss: 0.0914 - val_acc: 1.0000
Epoch 96/250
 - 9s - loss: 0.2124 - acc: 0.9430 - val_loss: 0.1030 - val_acc: 1.0000
Epoch 97/250
 - 9s - loss: 0.2608 - acc: 0.9254 - val_loss: 0.0951 - val_acc: 1.0000
Epoch 98/250
 - 9s - loss: 0.2182 - acc: 0.9386 - val_loss: 0.0663 - val_acc: 1.0000
Epoch 99/250
 - 9s - loss: 0.2048 - acc: 0.9386 - val_loss: 0.0552 - val_acc: 1.0000
Epoch 100/250
 - 9s - loss: 0.1738 - acc: 0.9649 - val_loss: 0.0561 - val_acc: 1.0000
Epoch 101/250
 - 9s - loss: 0.1816 - acc: 0.9561 - val_loss: 0.0694 - val_acc: 1.0000
Epoch 102/250
 - 11s - loss: 0.1444 - acc: 0.9737 - val_loss: 0.0790 - val_acc: 1.0000
Epoch 103/250
 - 11s - loss: 0.1792 - acc: 0.9605 - val_loss: 0.0884 - val_acc: 1.0000
Epoch 104/250
 - 11s - loss: 0.1870 - acc: 0.9430 - val_loss: 0.0710 - val_acc: 1.0000
Epoch 105/250
 - 11s - loss: 0.1325 - acc: 0.9693 - val_loss: 0.0460 - val_acc: 1.0000
Epoch 106/250
 - 11s - loss: 0.1634 - acc: 0.9518 - val_loss: 0.0357 - val_acc: 1.0000
Epoch 107/250
 - 11s - loss: 0.1144 - acc: 0.9781 - val_loss: 0.0355 - val_acc: 1.0000
Epoch 108/250
 - 10s - loss: 0.1537 - acc: 0.9649 - val_loss: 0.0438 - val_acc: 1.0000
Epoch 109/250
 - 11s - loss: 0.1460 - acc: 0.9605 - val_loss: 0.0575 - val_acc: 1.0000
Epoch 110/250
 - 11s - loss: 0.1546 - acc: 0.9561 - val_loss: 0.0616 - val_acc: 1.0000
Epoch 111/250
 - 10s - loss: 0.1199 - acc: 0.9737 - val_loss: 0.0534 - val_acc: 1.0000
Epoch 112/250
 - 11s - loss: 0.1172 - acc: 0.9737 - val_loss: 0.0310 - val_acc: 1.0000
Epoch 113/250
 - 13s - loss: 0.1183 - acc: 0.9825 - val_loss: 0.0203 - val_acc: 1.0000
Epoch 114/250
 - 11s - loss: 0.0972 - acc: 0.9825 - val_loss: 0.0192 - val_acc: 1.0000
Epoch 115/250
 - 11s - loss: 0.1189 - acc: 0.9781 - val_loss: 0.0263 - val_acc: 1.0000
Epoch 116/250
 - 11s - loss: 0.0832 - acc: 0.9781 - val_loss: 0.0304 - val_acc: 1.0000
Epoch 117/250
 - 10s - loss: 0.0866 - acc: 0.9868 - val_loss: 0.0249 - val_acc: 1.0000
Epoch 118/250
 - 9s - loss: 0.0953 - acc: 0.9649 - val_loss: 0.0141 - val_acc: 1.0000
Epoch 119/250
 - 9s - loss: 0.1017 - acc: 0.9825 - val_loss: 0.0122 - val_acc: 1.0000
Epoch 120/250
 - 9s - loss: 0.0850 - acc: 0.9693 - val_loss: 0.0139 - val_acc: 1.0000
```

```
Epoch 121/250
 - 9s - loss: 0.1045 - acc: 0.9737 - val_loss: 0.0256 - val_acc: 1.0000
Epoch 122/250
 - 9s - loss: 0.0894 - acc: 0.9781 - val_loss: 0.0480 - val_acc: 1.0000
Epoch 123/250
 - 9s - loss: 0.0677 - acc: 0.9912 - val_loss: 0.0654 - val_acc: 1.0000
Epoch 124/250
 - 9s - loss: 0.0845 - acc: 0.9781 - val_loss: 0.0562 - val_acc: 1.0000
Epoch 125/250
 - 9s - loss: 0.0769 - acc: 0.9868 - val_loss: 0.0289 - val_acc: 1.0000
Epoch 126/250
 - 9s - loss: 0.0763 - acc: 0.9825 - val_loss: 0.0113 - val_acc: 1.0000
Epoch 127/250
 - 9s - loss: 0.0834 - acc: 0.9868 - val_loss: 0.0074 - val_acc: 1.0000
Epoch 128/250
 - 9s - loss: 0.0804 - acc: 0.9781 - val_loss: 0.0068 - val_acc: 1.0000
Epoch 129/250
 - 9s - loss: 0.0711 - acc: 0.9868 - val_loss: 0.0092 - val_acc: 1.0000
Epoch 130/250
 - 9s - loss: 0.0476 - acc: 0.9956 - val_loss: 0.0168 - val_acc: 1.0000
Epoch 131/250
 - 10s - loss: 0.0575 - acc: 0.9912 - val_loss: 0.0330 - val_acc: 1.0000
Epoch 132/250
 - 10s - loss: 0.0626 - acc: 0.9868 - val_loss: 0.0445 - val_acc: 1.0000
Epoch 133/250
 - 9s - loss: 0.0753 - acc: 0.9649 - val_loss: 0.0270 - val_acc: 1.0000
Epoch 134/250
 - 9s - loss: 0.0759 - acc: 0.9781 - val_loss: 0.0131 - val_acc: 1.0000
Epoch 135/250
 - 9s - loss: 0.0709 - acc: 0.9868 - val_loss: 0.0085 - val_acc: 1.0000
Epoch 136/250
 - 9s - loss: 0.0794 - acc: 0.9825 - val_loss: 0.0073 - val_acc: 1.0000
Epoch 137/250
 - 9s - loss: 0.0564 - acc: 0.9956 - val_loss: 0.0080 - val_acc: 1.0000
Epoch 138/250
 - 9s - loss: 0.0571 - acc: 0.9912 - val_loss: 0.0115 - val_acc: 1.0000
Epoch 139/250
 - 9s - loss: 0.0645 - acc: 0.9868 - val_loss: 0.0203 - val_acc: 1.0000
Epoch 140/250
 - 9s - loss: 0.0477 - acc: 0.9912 - val_loss: 0.0327 - val_acc: 1.0000
Epoch 141/250
 - 9s - loss: 0.0574 - acc: 0.9912 - val_loss: 0.0401 - val_acc: 1.0000
Epoch 142/250
 - 9s - loss: 0.0640 - acc: 0.9868 - val_loss: 0.0178 - val_acc: 1.0000
Epoch 143/250
 - 9s - loss: 0.0597 - acc: 0.9825 - val_loss: 0.0074 - val_acc: 1.0000
Epoch 144/250
 - 9s - loss: 0.0593 - acc: 0.9956 - val_loss: 0.0042 - val_acc: 1.0000
Epoch 145/250
 - 9s - loss: 0.0422 - acc: 0.9912 - val_loss: 0.0035 - val_acc: 1.0000
Epoch 146/250
 - 9s - loss: 0.0398 - acc: 0.9956 - val_loss: 0.0034 - val_acc: 1.0000
Epoch 147/250
 - 9s - loss: 0.0444 - acc: 0.9912 - val_loss: 0.0038 - val_acc: 1.0000
Epoch 148/250
 - 9s - loss: 0.0367 - acc: 0.9912 - val_loss: 0.0052 - val_acc: 1.0000
Epoch 149/250
 - 9s - loss: 0.0383 - acc: 0.9912 - val_loss: 0.0085 - val_acc: 1.0000
Epoch 150/250
 - 9s - loss: 0.0376 - acc: 0.9956 - val_loss: 0.0131 - val_acc: 1.0000
Epoch 151/250
 - 9s - loss: 0.0381 - acc: 0.9956 - val_loss: 0.0180 - val_acc: 1.0000
Epoch 152/250
 - 9s - loss: 0.0350 - acc: 1.0000 - val_loss: 0.0173 - val_acc: 1.0000
Epoch 153/250
 - 9s - loss: 0.0480 - acc: 0.9912 - val_loss: 0.0132 - val_acc: 1.0000
Epoch 154/250
 - 9s - loss: 0.0371 - acc: 0.9956 - val_loss: 0.0097 - val_acc: 1.0000
Epoch 155/250
```

```
 - 9s - loss: 0.0304 - acc: 1.0000 - val_loss: 0.0071 - val_acc: 1.0000
Epoch 156/250
 - 9s - loss: 0.0259 - acc: 1.0000 - val_loss: 0.0057 - val_acc: 1.0000
Epoch 157/250
 - 9s - loss: 0.0328 - acc: 1.0000 - val_loss: 0.0049 - val_acc: 1.0000
Epoch 158/250
 - 9s - loss: 0.0383 - acc: 0.9912 - val_loss: 0.0047 - val_acc: 1.0000
Epoch 159/250
 - 9s - loss: 0.0495 - acc: 0.9825 - val_loss: 0.0057 - val_acc: 1.0000
Epoch 160/250
 - 9s - loss: 0.0295 - acc: 0.9956 - val_loss: 0.0079 - val_acc: 1.0000
Epoch 161/250
 - 9s - loss: 0.0335 - acc: 0.9912 - val_loss: 0.0111 - val_acc: 1.0000
Epoch 162/250
 - 9s - loss: 0.0331 - acc: 0.9956 - val_loss: 0.0150 - val_acc: 1.0000
Epoch 163/250
 - 9s - loss: 0.0396 - acc: 0.9912 - val_loss: 0.0174 - val_acc: 1.0000
Epoch 164/250
 - 9s - loss: 0.0268 - acc: 1.0000 - val_loss: 0.0168 - val_acc: 1.0000
Epoch 165/250
 - 9s - loss: 0.0253 - acc: 1.0000 - val_loss: 0.0125 - val_acc: 1.0000
Epoch 166/250
 - 10s - loss: 0.0271 - acc: 0.9956 - val_loss: 0.0080 - val_acc: 1.0000
Epoch 167/250
 - 9s - loss: 0.0321 - acc: 0.9956 - val_loss: 0.0043 - val_acc: 1.0000
Epoch 168/250
 - 9s - loss: 0.0221 - acc: 0.9956 - val_loss: 0.0027 - val_acc: 1.0000
Epoch 169/250
 - 9s - loss: 0.0150 - acc: 1.0000 - val_loss: 0.0019 - val_acc: 1.0000
Epoch 170/250
 - 9s - loss: 0.0358 - acc: 0.9956 - val_loss: 0.0015 - val_acc: 1.0000
Epoch 171/250
 - 9s - loss: 0.0225 - acc: 0.9956 - val_loss: 0.0016 - val_acc: 1.0000
Epoch 172/250
 - 9s - loss: 0.0340 - acc: 1.0000 - val_loss: 0.0023 - val_acc: 1.0000
Epoch 173/250
 - 9s - loss: 0.0233 - acc: 1.0000 - val_loss: 0.0041 - val_acc: 1.0000
Epoch 174/250
 - 9s - loss: 0.0363 - acc: 0.9825 - val_loss: 0.0062 - val_acc: 1.0000
Epoch 175/250
 - 9s - loss: 0.0216 - acc: 0.9956 - val_loss: 0.0092 - val_acc: 1.0000
Epoch 176/250
 - 8s - loss: 0.0316 - acc: 0.9956 - val_loss: 0.0123 - val_acc: 1.0000
Epoch 177/250
 - 9s - loss: 0.0250 - acc: 0.9956 - val_loss: 0.0129 - val_acc: 1.0000
Epoch 178/250
 - 9s - loss: 0.0309 - acc: 0.9868 - val_loss: 0.0105 - val_acc: 1.0000
Epoch 179/250
 - 8s - loss: 0.0153 - acc: 1.0000 - val_loss: 0.0063 - val_acc: 1.0000
Epoch 180/250
 - 9s - loss: 0.0278 - acc: 0.9956 - val_loss: 0.0038 - val_acc: 1.0000
Epoch 181/250
 - 8s - loss: 0.0457 - acc: 0.9868 - val_loss: 0.0022 - val_acc: 1.0000
Epoch 182/250
 - 9s - loss: 0.0316 - acc: 0.9912 - val_loss: 0.0013 - val_acc: 1.0000
Epoch 183/250
 - 9s - loss: 0.0215 - acc: 0.9956 - val_loss: 9.2073e-04 - val_acc: 1.0000
Epoch 184/250
 - 9s - loss: 0.0236 - acc: 1.0000 - val_loss: 7.9891e-04 - val_acc: 1.0000
Epoch 185/250
 - 9s - loss: 0.0364 - acc: 0.9825 - val_loss: 9.2675e-04 - val_acc: 1.0000
Epoch 186/250
 - 8s - loss: 0.0212 - acc: 1.0000 - val_loss: 0.0012 - val_acc: 1.0000
Epoch 187/250
 - 9s - loss: 0.0205 - acc: 1.0000 - val_loss: 0.0015 - val_acc: 1.0000
Epoch 188/250
 - 8s - loss: 0.0197 - acc: 1.0000 - val_loss: 0.0019 - val_acc: 1.0000
Epoch 189/250
 - 9s - loss: 0.0266 - acc: 0.9956 - val_loss: 0.0020 - val_acc: 1.0000
```

```
Epoch 190/250
 - 9s - loss: 0.0254 - acc: 0.9956 - val_loss: 0.0017 - val_acc: 1.0000
Epoch 191/250
 - 9s - loss: 0.0230 - acc: 0.9956 - val_loss: 0.0013 - val_acc: 1.0000
Epoch 192/250
 - 8s - loss: 0.0194 - acc: 0.9956 - val_loss: 0.0012 - val_acc: 1.0000
Epoch 193/250
 - 9s - loss: 0.0218 - acc: 0.9912 - val_loss: 0.0011 - val_acc: 1.0000
Epoch 194/250
 - 9s - loss: 0.0389 - acc: 0.9868 - val_loss: 0.0012 - val_acc: 1.0000
Epoch 195/250
 - 9s - loss: 0.0197 - acc: 0.9956 - val_loss: 0.0017 - val_acc: 1.0000
Epoch 196/250
 - 9s - loss: 0.0211 - acc: 0.9956 - val_loss: 0.0030 - val_acc: 1.0000
Epoch 197/250
 - 8s - loss: 0.0087 - acc: 1.0000 - val_loss: 0.0050 - val_acc: 1.0000
Epoch 198/250
 - 9s - loss: 0.0194 - acc: 1.0000 - val_loss: 0.0091 - val_acc: 1.0000
Epoch 199/250
 - 9s - loss: 0.0183 - acc: 0.9956 - val_loss: 0.0144 - val_acc: 1.0000
Epoch 200/250
 - 9s - loss: 0.0196 - acc: 0.9956 - val_loss: 0.0153 - val_acc: 1.0000
Epoch 201/250
 - 8s - loss: 0.0162 - acc: 1.0000 - val_loss: 0.0139 - val_acc: 1.0000
Epoch 202/250
 - 9s - loss: 0.0195 - acc: 1.0000 - val_loss: 0.0089 - val_acc: 1.0000
Epoch 203/250
 - 9s - loss: 0.0162 - acc: 1.0000 - val_loss: 0.0050 - val_acc: 1.0000
Epoch 204/250
 - 9s - loss: 0.0246 - acc: 0.9912 - val_loss: 0.0025 - val_acc: 1.0000
Epoch 205/250
 - 9s - loss: 0.0121 - acc: 1.0000 - val_loss: 0.0015 - val_acc: 1.0000
Epoch 206/250
 - 9s - loss: 0.0311 - acc: 0.9912 - val_loss: 0.0010 - val_acc: 1.0000
Epoch 207/250
 - 8s - loss: 0.0158 - acc: 0.9956 - val_loss: 7.7045e-04 - val_acc: 1.0000
Epoch 208/250
 - 9s - loss: 0.0117 - acc: 1.0000 - val_loss: 6.6625e-04 - val_acc: 1.0000
Epoch 209/250
 - 9s - loss: 0.0111 - acc: 1.0000 - val_loss: 6.3390e-04 - val_acc: 1.0000
Epoch 210/250
 - 9s - loss: 0.0175 - acc: 1.0000 - val_loss: 6.7840e-04 - val_acc: 1.0000
Epoch 211/250
 - 9s - loss: 0.0114 - acc: 1.0000 - val_loss: 7.5318e-04 - val_acc: 1.0000
Epoch 212/250
 - 9s - loss: 0.0176 - acc: 0.9956 - val_loss: 9.1340e-04 - val_acc: 1.0000
Epoch 213/250
 - 9s - loss: 0.0143 - acc: 1.0000 - val_loss: 9.4964e-04 - val_acc: 1.0000
Epoch 214/250
 - 9s - loss: 0.0167 - acc: 0.9956 - val_loss: 8.8168e-04 - val_acc: 1.0000
Epoch 215/250
 - 9s - loss: 0.0130 - acc: 0.9956 - val_loss: 9.1121e-04 - val_acc: 1.0000
Epoch 216/250
 - 9s - loss: 0.0135 - acc: 0.9956 - val_loss: 8.7252e-04 - val_acc: 1.0000
Epoch 217/250
 - 9s - loss: 0.0071 - acc: 1.0000 - val_loss: 8.4030e-04 - val_acc: 1.0000
Epoch 218/250
 - 8s - loss: 0.0152 - acc: 0.9956 - val_loss: 8.5319e-04 - val_acc: 1.0000
Epoch 219/250
 - 9s - loss: 0.0122 - acc: 0.9956 - val_loss: 0.0011 - val_acc: 1.0000
Epoch 220/250
 - 9s - loss: 0.0078 - acc: 1.0000 - val_loss: 0.0014 - val_acc: 1.0000
Epoch 221/250
 - 8s - loss: 0.0138 - acc: 1.0000 - val_loss: 0.0016 - val_acc: 1.0000
Epoch 222/250
 - 9s - loss: 0.0101 - acc: 1.0000 - val_loss: 0.0017 - val_acc: 1.0000
Epoch 223/250
 - 8s - loss: 0.0083 - acc: 1.0000 - val_loss: 0.0018 - val_acc: 1.0000
Epoch 224/250
```

```
 - 9s - loss: 0.0065 - acc: 1.0000 - val_loss: 0.0018 - val_acc: 1.0000
Epoch 225/250
 - 8s - loss: 0.0100 - acc: 1.0000 - val_loss: 0.0019 - val_acc: 1.0000
Epoch 226/250
 - 9s - loss: 0.0122 - acc: 1.0000 - val_loss: 0.0021 - val_acc: 1.0000
Epoch 227/250
 - 9s - loss: 0.0214 - acc: 0.9912 - val_loss: 0.0020 - val_acc: 1.0000
Epoch 228/250
 - 9s - loss: 0.0157 - acc: 1.0000 - val_loss: 0.0017 - val_acc: 1.0000
Epoch 229/250
 - 9s - loss: 0.0106 - acc: 0.9956 - val_loss: 0.0014 - val_acc: 1.0000
Epoch 230/250
 - 9s - loss: 0.0164 - acc: 0.9956 - val_loss: 0.0012 - val_acc: 1.0000
Epoch 231/250
 - 9s - loss: 0.0061 - acc: 1.0000 - val_loss: 0.0011 - val_acc: 1.0000
Epoch 232/250
 - 9s - loss: 0.0245 - acc: 0.9868 - val_loss: 0.0014 - val_acc: 1.0000
Epoch 233/250
 - 9s - loss: 0.0102 - acc: 1.0000 - val_loss: 0.0017 - val_acc: 1.0000
Epoch 234/250
 - 8s - loss: 0.0328 - acc: 0.9912 - val_loss: 0.0021 - val_acc: 1.0000
Epoch 235/250
 - 9s - loss: 0.0105 - acc: 1.0000 - val_loss: 0.0019 - val_acc: 1.0000
Epoch 236/250
 - 10s - loss: 0.0125 - acc: 1.0000 - val_loss: 0.0016 - val_acc: 1.0000
Epoch 237/250
 - 9s - loss: 0.0103 - acc: 1.0000 - val_loss: 0.0013 - val_acc: 1.0000
Epoch 238/250
 - 9s - loss: 0.0110 - acc: 0.9956 - val_loss: 9.5821e-04 - val_acc: 1.0000
Epoch 239/250
 - 9s - loss: 0.0187 - acc: 0.9956 - val_loss: 8.3563e-04 - val_acc: 1.0000
Epoch 240/250
 - 9s - loss: 0.0053 - acc: 1.0000 - val_loss: 7.7206e-04 - val_acc: 1.0000
Epoch 241/250
 - 9s - loss: 0.0104 - acc: 1.0000 - val_loss: 8.3726e-04 - val_acc: 1.0000
Epoch 242/250
 - 9s - loss: 0.0071 - acc: 1.0000 - val_loss: 9.1068e-04 - val_acc: 1.0000
Epoch 243/250
 - 10s - loss: 0.0155 - acc: 0.9956 - val_loss: 0.0011 - val_acc: 1.0000
Epoch 244/250
 - 10s - loss: 0.0062 - acc: 1.0000 - val_loss: 0.0014 - val_acc: 1.0000
Epoch 245/250
 - 9s - loss: 0.0119 - acc: 0.9956 - val_loss: 0.0016 - val_acc: 1.0000
Epoch 246/250
 - 9s - loss: 0.0087 - acc: 1.0000 - val_loss: 0.0017 - val_acc: 1.0000
Epoch 247/250
 - 10s - loss: 0.0120 - acc: 0.9956 - val_loss: 0.0015 - val_acc: 1.0000
Epoch 248/250
 - 9s - loss: 0.0154 - acc: 0.9956 - val_loss: 0.0013 - val_acc: 1.0000
Epoch 249/250
 - 9s - loss: 0.0092 - acc: 1.0000 - val_loss: 0.0011 - val_acc: 1.0000
Epoch 250/250
 - 9s - loss: 0.0264 - acc: 0.9956 - val_loss: 6.4789e-04 - val_acc: 1.0000
```

Evaluate the test data

In [26]:

```
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 0.3612
test acc 0.9375
```

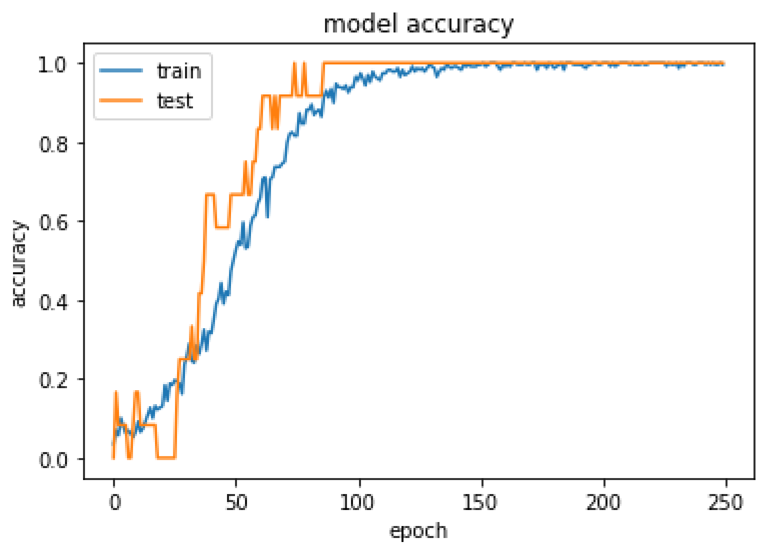
Step 7

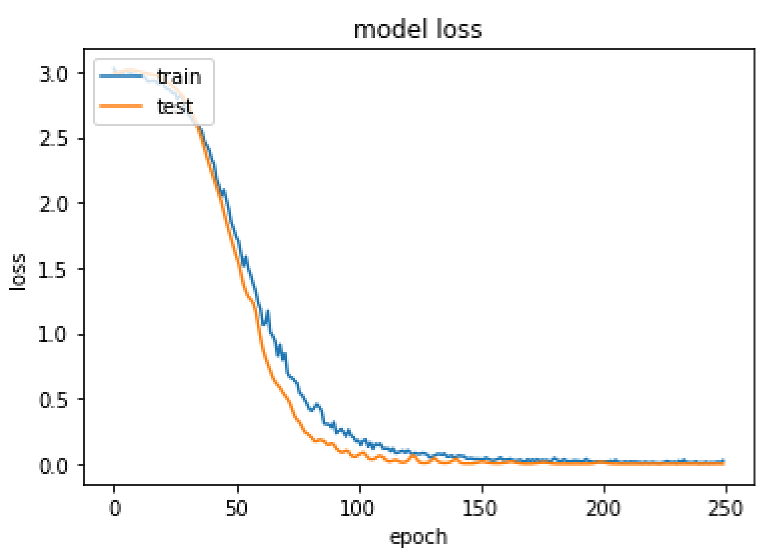
plot the result

In [27]:

```
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
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(['val loss', 'val acc', 'loss', 'acc'])
```





step 8

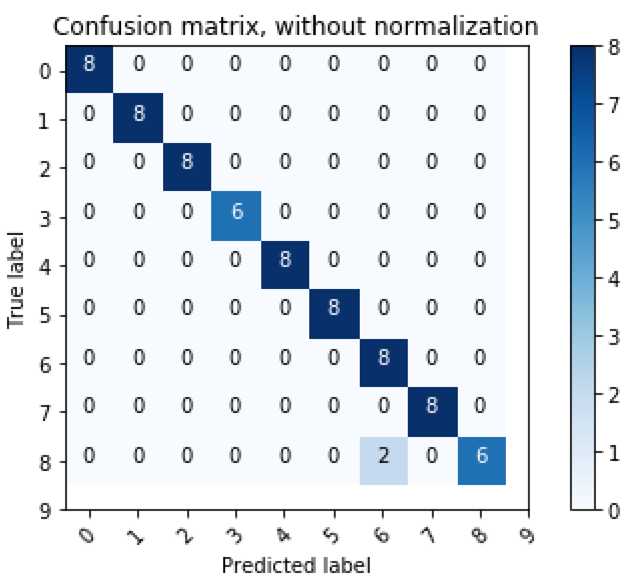
Plot Confusion Matrix

In [28]:

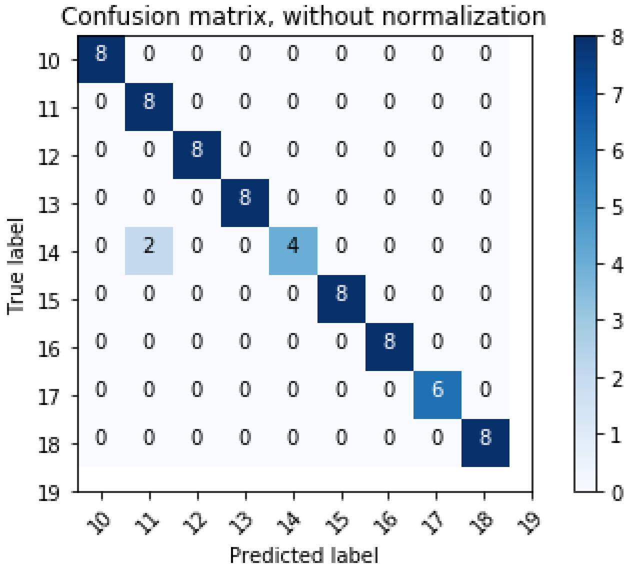
```
predicted =np.array( cnn_model.predict(x_test))
#print(predicted)
#print(y_test)
ynew = cnn_model.predict_classes(x_test)


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))
```

```
accuracy :
0.9375
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 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 2 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 2 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 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]
 [2 0 0 0 0 0 0 0 0 0 0 0 2 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 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       0.80      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       0.67      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       0.80      1.00      0.89         8
         13       1.00      1.00      1.00         8
         14       1.00      1.00      1.00         8
         15       1.00      0.50      0.67         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

avg / total       0.95      0.94      0.94       160
```

By:Abdullah Alwabel