Project 1: Perform Facial Recognition with Deep Learning in Keras Using CNN

DESCRIPTION

Problem Statement: Facial recognition is a biometric alternative that measures unique characteristics of a human face. Applications available today include flight check in, tagging friends and family members in photos, and "tailored" advertising. You are a computer vision engineer who needs to develop a face recognition programme with deep convolutional neural networks. Objective: Use a deep convolutional neural network to perform facial recognition using Keras. Dataset Details: ORL face database composed of 400 images of size 112 x 92. There are 40 people, 10 images per person. 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 Prerequisites: Keras Scikit Learn Steps to be followed:

1. Input the required libraries
2. Load the dataset after loading the dataset, you have to normalize every image.
3. Split the dataset
4. Transform the images to equal sizes to feed in CNN
5. Build a CNN model that has 3 main layers:

i. Convolutional Layer ii. Pooling Layer iii. Fully Connected Layer

1. Train the model
2. Plot the result
3. Iterate the model until the accuracy is above 90%

In [1]:
```
## Import the Operating System and Set the Working Directory

# Import the operating system. The OS module in Python provides functions for creating
# fetching its contents, changing and identifying the current directory, etc.

import os
```

In [2]:
```
os.getcwd()
```

Out[2]:
```
'C:\\Users\\Byju'
```

In [4]:
```
#Lets create a course specific subdirectory under the student directory "Byju" to save
if not os.path.exists('C:/Users/Byju/Dropbox/1. BNGPersonal/Purdue-SimpliLearn/3. AI &
    #os.makedirs('/home/labsuser/Byju/Course3-PG-AI-ML')
    os.mkdir('C:/Users/Byju/Dropbox/1. BNGPersonal/Purdue-SimpliLearn/3. AI & ML/Assesm

#Lets set directory "AssesmentProjects" as working diectory
os.chdir("..")
```

```
#Let's explicitly reset the directory "Module4" as the desired working directory
os.chdir('C:/Users/Byju/Dropbox/1. BNGPersonal/Purdue-SimpliLearn/3. AI & ML/AssesmentP

#display the current working directory for confirmation
os.getcwd()
```

Out[4]:
```
'C:\\Users\\Byju\\Dropbox\\1. BNGPersonal\\Purdue-SimpliLearn\\3. AI & ML\\AssesmentProj
ects\\Module4'
```

Step1: Input the required libraries

In [6]:
```
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 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

from livelossplot import PlotLossesKeras

#import numpy as np
#import matplotlib.pyplot as plt
#import keras
#from keras.layers import Input, Dense, Reshape, Flatten, Dropout
#from keras.layers import BatchNormalization, Activation, ZeroPadding2D
#from keras.layers.advanced_activations import LeakyReLU
#from keras.layers.convolutional import UpSampling2D, Conv2D
#from keras.models import Sequential, Model
#from tensorflow.keras.optimizers import Adam, SGD
#import keras
```

Step2: Load the dataset and then 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 [7]:
```
#check the current directory
os.getcwd()
```

Out[7]:
```
'C:\\Users\\Byju\\Dropbox\\1. BNGPersonal\\Purdue-SimpliLearn\\3. AI & ML\\AssesmentProj
ects\\Module4'
```

In [13]:
```
#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('x_train shape: {}'.format(x_train.shape))
print('')
print('Y-train shape: {}'.format(y_train))
print('Y_train shape: {}'.format(y_train.shape))
print('')
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]]
x_train shape: (240, 10304)

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]
Y_train shape: (240,)

x_test shape: (160, 10304)
```

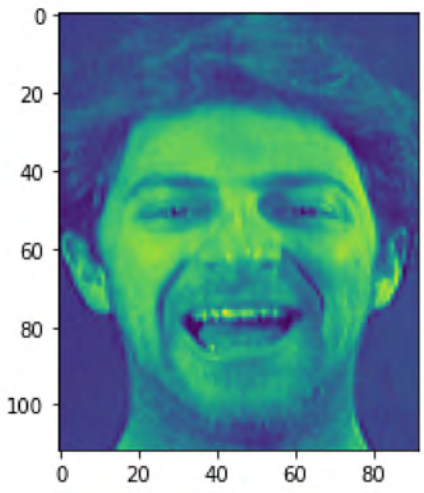
Step 3: Split DataSet : Validation data and Train

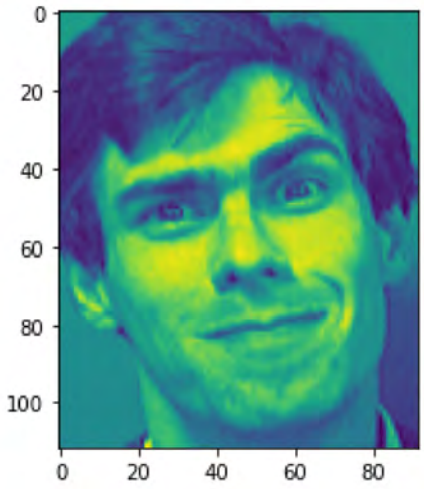
We split the train dataset into Train and validation datasets. Validation dataSet is used to minimize overfitting. If the accuracy over the train data set increases, but the accuracy over then validation data set stays the same or decreases, then it is a case of overfitted neural network and we should stop training. Note: Recommendation is to use 30 percent of every dataset as the validation data but only 5 percent is used because the number of images in this dataset is very low.

In [14]:
```
x_train, x_valid, y_train, y_valid= train_test_split(x_train, y_train, test_size=.05, r
```

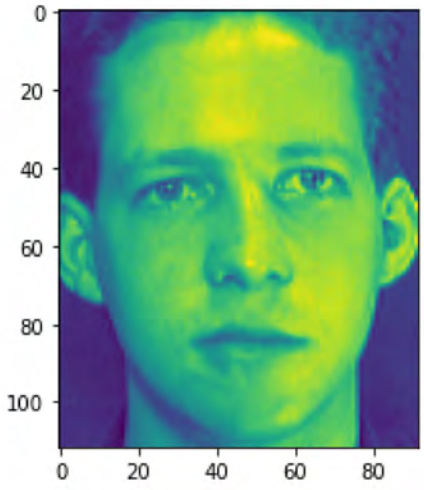
Let us visualise the Images in the Train, Validation and Test Datasets

In [19]:
```
# Images in the Train, Validation and Test data sets

t1=x_train[1].reshape(112,92)
plt.imshow(t1)
plt.show()
v1=x_valid[1].reshape(112,92)
plt.imshow(v1)
plt.show()
t2=x_test[1].reshape(112,92)
plt.imshow(t2)
```





Out[19]: <matplotlib.image.AxesImage at 0x2168b76e250>



Step 4: for using the CNN, we need to change the size of images (The size of images must be the same)

In [20]:
```
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 layers:

1) Convolutional layer 2) pooling layer 3) fully connected layer

We shall increase or decrease the convolution, max pooling, and hidden ANN layers and the number of neurons in them.

But more the layers/neurons added, the model becomes slower

Explicitly, we create a CNN model with

2 hidden layers of convolution 2 hidden layers of max pooling 1 layer of flattening 3 Hidden ANN layer 1 output layer with 20-neurons

In [24]:
```
#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=64, 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(learning_rate=0.0001),
```

```
    metrics=['accuracy']
)
```

In [25]:
```
cnn_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 106, 86, 36)       1800

 max_pooling2d_6 (MaxPooling  (None, 53, 43, 36)       0
 2D)

 conv2d_7 (Conv2D)           (None, 49, 39, 64)        57664

 max_pooling2d_7 (MaxPooling  (None, 24, 19, 64)       0
 2D)

 flatten_3 (Flatten)         (None, 29184)             0

 dense_12 (Dense)            (None, 2024)              59070440

 dropout_9 (Dropout)         (None, 2024)              0

 dense_13 (Dense)            (None, 1024)              2073600

 dropout_10 (Dropout)        (None, 1024)              0

 dense_14 (Dense)            (None, 512)               524800

 dropout_11 (Dropout)        (None, 512)               0

 dense_15 (Dense)            (None, 20)                10260

=================================================================
Total params: 61,738,564
Trainable params: 61,738,564
Non-trainable params: 0
_________________________________________________________________
```

Step 6: Train the Model

In [26]:
```
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)),
)
```

```
Epoch 1/250
1/1 - 4s - loss: 2.9984 - accuracy: 0.0658 - val_loss: 2.9929 - val_accuracy: 0.1667 - 4
s/epoch - 4s/step
Epoch 2/250
1/1 - 1s - loss: 2.9998 - accuracy: 0.0439 - val_loss: 2.9916 - val_accuracy: 0.1667 - 1
s/epoch - 1s/step
Epoch 3/250
1/1 - 1s - loss: 3.0013 - accuracy: 0.0526 - val_loss: 2.9957 - val_accuracy: 0.2500 - 1
s/epoch - 1s/step
Epoch 4/250
```

```
1/1 - 1s - loss: 2.9887 - accuracy: 0.0614 - val_loss: 2.9910 - val_accuracy: 0.1667 - 1
s/epoch - 1s/step
Epoch 5/250
1/1 - 1s - loss: 3.0086 - accuracy: 0.0482 - val_loss: 2.9853 - val_accuracy: 0.1667 - 1
s/epoch - 1s/step
Epoch 6/250
1/1 - 2s - loss: 2.9844 - accuracy: 0.0833 - val_loss: 2.9791 - val_accuracy: 0.1667 - 2
s/epoch - 2s/step
Epoch 7/250
1/1 - 2s - loss: 2.9897 - accuracy: 0.0746 - val_loss: 2.9764 - val_accuracy: 0.1667 - 2
s/epoch - 2s/step
Epoch 8/250
1/1 - 2s - loss: 3.0164 - accuracy: 0.0570 - val_loss: 2.9753 - val_accuracy: 0.2500 - 2
s/epoch - 2s/step
Epoch 9/250
1/1 - 2s - loss: 2.9925 - accuracy: 0.0702 - val_loss: 2.9728 - val_accuracy: 0.2500 - 2
s/epoch - 2s/step
Epoch 10/250
1/1 - 2s - loss: 2.9743 - accuracy: 0.0702 - val_loss: 2.9722 - val_accuracy: 0.2500 - 2
s/epoch - 2s/step
Epoch 11/250
1/1 - 2s - loss: 2.9877 - accuracy: 0.0395 - val_loss: 2.9727 - val_accuracy: 0.2500 - 2
s/epoch - 2s/step
Epoch 12/250
1/1 - 1s - loss: 2.9489 - accuracy: 0.1360 - val_loss: 2.9715 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 13/250
1/1 - 1s - loss: 2.9466 - accuracy: 0.0833 - val_loss: 2.9687 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 14/250
1/1 - 2s - loss: 2.9750 - accuracy: 0.0658 - val_loss: 2.9647 - val_accuracy: 0.0833 - 2
s/epoch - 2s/step
Epoch 15/250
1/1 - 1s - loss: 2.9548 - accuracy: 0.0746 - val_loss: 2.9605 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 16/250
1/1 - 1s - loss: 2.9490 - accuracy: 0.1053 - val_loss: 2.9577 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 17/250
1/1 - 1s - loss: 2.9340 - accuracy: 0.1404 - val_loss: 2.9577 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 18/250
1/1 - 1s - loss: 2.9444 - accuracy: 0.1009 - val_loss: 2.9555 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 19/250
1/1 - 1s - loss: 2.9133 - accuracy: 0.1404 - val_loss: 2.9513 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 20/250
1/1 - 2s - loss: 2.9289 - accuracy: 0.1184 - val_loss: 2.9462 - val_accuracy: 0.0833 - 2
s/epoch - 2s/step
Epoch 21/250
1/1 - 1s - loss: 2.9066 - accuracy: 0.1623 - val_loss: 2.9409 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 22/250
1/1 - 1s - loss: 2.9015 - accuracy: 0.1316 - val_loss: 2.9372 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 23/250
1/1 - 1s - loss: 2.8938 - accuracy: 0.1404 - val_loss: 2.9312 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 24/250
1/1 - 1s - loss: 2.8990 - accuracy: 0.1184 - val_loss: 2.9229 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 25/250
1/1 - 1s - loss: 2.8736 - accuracy: 0.1623 - val_loss: 2.9118 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
```

```
Epoch 26/250
1/1 - 1s - loss: 2.8269 - accuracy: 0.1930 - val_loss: 2.8981 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 27/250
1/1 - 1s - loss: 2.7981 - accuracy: 0.2061 - val_loss: 2.8796 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 28/250
1/1 - 1s - loss: 2.8005 - accuracy: 0.2325 - val_loss: 2.8586 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 29/250
1/1 - 2s - loss: 2.8062 - accuracy: 0.1886 - val_loss: 2.8351 - val_accuracy: 0.0833 - 2
s/epoch - 2s/step
Epoch 30/250
1/1 - 2s - loss: 2.7621 - accuracy: 0.2061 - val_loss: 2.8125 - val_accuracy: 0.0833 - 2
s/epoch - 2s/step
Epoch 31/250
1/1 - 2s - loss: 2.7482 - accuracy: 0.2632 - val_loss: 2.7852 - val_accuracy: 0.0833 - 2
s/epoch - 2s/step
Epoch 32/250
1/1 - 1s - loss: 2.7090 - accuracy: 0.2281 - val_loss: 2.7549 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 33/250
1/1 - 2s - loss: 2.7226 - accuracy: 0.2368 - val_loss: 2.7216 - val_accuracy: 0.1667 - 2
s/epoch - 2s/step
Epoch 34/250
1/1 - 1s - loss: 2.6011 - accuracy: 0.3158 - val_loss: 2.6889 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 35/250
1/1 - 1s - loss: 2.6245 - accuracy: 0.2588 - val_loss: 2.6532 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 36/250
1/1 - 1s - loss: 2.6251 - accuracy: 0.2807 - val_loss: 2.6106 - val_accuracy: 0.0833 - 1
s/epoch - 1s/step
Epoch 37/250
1/1 - 1s - loss: 2.5462 - accuracy: 0.2982 - val_loss: 2.5655 - val_accuracy: 0.1667 - 1
s/epoch - 1s/step
Epoch 38/250
1/1 - 1s - loss: 2.4851 - accuracy: 0.3333 - val_loss: 2.5222 - val_accuracy: 0.2500 - 1
s/epoch - 1s/step
Epoch 39/250
1/1 - 2s - loss: 2.5139 - accuracy: 0.2675 - val_loss: 2.4787 - val_accuracy: 0.3333 - 2
s/epoch - 2s/step
Epoch 40/250
1/1 - 2s - loss: 2.4017 - accuracy: 0.3553 - val_loss: 2.4271 - val_accuracy: 0.3333 - 2
s/epoch - 2s/step
Epoch 41/250
1/1 - 2s - loss: 2.3870 - accuracy: 0.3684 - val_loss: 2.3689 - val_accuracy: 0.4167 - 2
s/epoch - 2s/step
Epoch 42/250
1/1 - 1s - loss: 2.3205 - accuracy: 0.3684 - val_loss: 2.3039 - val_accuracy: 0.5000 - 1
s/epoch - 1s/step
Epoch 43/250
1/1 - 1s - loss: 2.2453 - accuracy: 0.3947 - val_loss: 2.2342 - val_accuracy: 0.5000 - 1
s/epoch - 1s/step
Epoch 44/250
1/1 - 1s - loss: 2.1790 - accuracy: 0.3991 - val_loss: 2.1577 - val_accuracy: 0.5833 - 1
s/epoch - 1s/step
Epoch 45/250
1/1 - 1s - loss: 2.1366 - accuracy: 0.3947 - val_loss: 2.0789 - val_accuracy: 0.5833 - 1
s/epoch - 1s/step
Epoch 46/250
1/1 - 1s - loss: 2.0604 - accuracy: 0.4386 - val_loss: 1.9971 - val_accuracy: 0.5833 - 1
s/epoch - 1s/step
Epoch 47/250
1/1 - 1s - loss: 1.9906 - accuracy: 0.4561 - val_loss: 1.9092 - val_accuracy: 0.5833 - 1
```

```
s/epoch - 1s/step
Epoch 48/250
1/1 - 1s - loss: 1.9375 - accuracy: 0.4693 - val_loss: 1.8214 - val_accuracy: 0.5833 - 1
s/epoch - 1s/step
Epoch 49/250
1/1 - 1s - loss: 1.8830 - accuracy: 0.4737 - val_loss: 1.7291 - val_accuracy: 0.5833 - 1
s/epoch - 1s/step
Epoch 50/250
1/1 - 1s - loss: 1.8435 - accuracy: 0.4737 - val_loss: 1.6437 - val_accuracy: 0.5833 - 1
s/epoch - 1s/step
Epoch 51/250
1/1 - 1s - loss: 1.7836 - accuracy: 0.5088 - val_loss: 1.5705 - val_accuracy: 0.6667 - 1
s/epoch - 1s/step
Epoch 52/250
1/1 - 2s - loss: 1.6954 - accuracy: 0.5395 - val_loss: 1.4998 - val_accuracy: 0.6667 - 2
s/epoch - 2s/step
Epoch 53/250
1/1 - 2s - loss: 1.5842 - accuracy: 0.5570 - val_loss: 1.4215 - val_accuracy: 0.7500 - 2
s/epoch - 2s/step
Epoch 54/250
1/1 - 1s - loss: 1.5969 - accuracy: 0.5307 - val_loss: 1.3458 - val_accuracy: 0.7500 - 1
s/epoch - 1s/step
Epoch 55/250
1/1 - 1s - loss: 1.5750 - accuracy: 0.5658 - val_loss: 1.2753 - val_accuracy: 0.9167 - 1
s/epoch - 1s/step
Epoch 56/250
1/1 - 1s - loss: 1.3915 - accuracy: 0.6228 - val_loss: 1.1925 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 57/250
1/1 - 1s - loss: 1.3105 - accuracy: 0.6316 - val_loss: 1.1236 - val_accuracy: 0.9167 - 1
s/epoch - 1s/step
Epoch 58/250
1/1 - 1s - loss: 1.3819 - accuracy: 0.5965 - val_loss: 1.0326 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 59/250
1/1 - 1s - loss: 1.2809 - accuracy: 0.5965 - val_loss: 0.9639 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 60/250
1/1 - 1s - loss: 1.4460 - accuracy: 0.5570 - val_loss: 0.9183 - val_accuracy: 0.9167 - 1
s/epoch - 1s/step
Epoch 61/250
1/1 - 1s - loss: 1.1518 - accuracy: 0.7149 - val_loss: 0.8746 - val_accuracy: 0.9167 - 1
s/epoch - 1s/step
Epoch 62/250
1/1 - 2s - loss: 1.1467 - accuracy: 0.6404 - val_loss: 0.8467 - val_accuracy: 0.8333 - 2
s/epoch - 2s/step
Epoch 63/250
1/1 - 2s - loss: 1.0617 - accuracy: 0.6798 - val_loss: 0.7926 - val_accuracy: 0.9167 - 2
s/epoch - 2s/step
Epoch 64/250
1/1 - 2s - loss: 0.9796 - accuracy: 0.7237 - val_loss: 0.7382 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 65/250
1/1 - 1s - loss: 0.9929 - accuracy: 0.7193 - val_loss: 0.6777 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 66/250
1/1 - 1s - loss: 0.9124 - accuracy: 0.7412 - val_loss: 0.6073 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 67/250
1/1 - 3s - loss: 0.8771 - accuracy: 0.7368 - val_loss: 0.5563 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 68/250
1/1 - 4s - loss: 0.8539 - accuracy: 0.7237 - val_loss: 0.5034 - val_accuracy: 1.0000 - 4
s/epoch - 4s/step
Epoch 69/250
```

```
1/1 - 3s - loss: 0.8400 - accuracy: 0.7412 - val_loss: 0.4669 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 70/250
1/1 - 3s - loss: 0.7941 - accuracy: 0.7675 - val_loss: 0.4386 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 71/250
1/1 - 3s - loss: 0.7937 - accuracy: 0.7939 - val_loss: 0.4035 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 72/250
1/1 - 3s - loss: 0.6661 - accuracy: 0.8640 - val_loss: 0.3817 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 73/250
1/1 - 3s - loss: 0.6009 - accuracy: 0.8158 - val_loss: 0.3597 - val_accuracy: 0.9167 - 3
s/epoch - 3s/step
Epoch 74/250
1/1 - 2s - loss: 0.5785 - accuracy: 0.8289 - val_loss: 0.3130 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 75/250
1/1 - 3s - loss: 0.5689 - accuracy: 0.8421 - val_loss: 0.2714 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 76/250
1/1 - 3s - loss: 0.5732 - accuracy: 0.8596 - val_loss: 0.2439 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 77/250
1/1 - 3s - loss: 0.5776 - accuracy: 0.8465 - val_loss: 0.2184 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 78/250
1/1 - 2s - loss: 0.5592 - accuracy: 0.8465 - val_loss: 0.2005 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 79/250
1/1 - 2s - loss: 0.4915 - accuracy: 0.8816 - val_loss: 0.1867 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 80/250
1/1 - 2s - loss: 0.4206 - accuracy: 0.8904 - val_loss: 0.1744 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 81/250
1/1 - 2s - loss: 0.4766 - accuracy: 0.8684 - val_loss: 0.1702 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 82/250
1/1 - 2s - loss: 0.3750 - accuracy: 0.8947 - val_loss: 0.1618 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 83/250
1/1 - 1s - loss: 0.3884 - accuracy: 0.9123 - val_loss: 0.1478 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 84/250
1/1 - 2s - loss: 0.3599 - accuracy: 0.9386 - val_loss: 0.1258 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 85/250
1/1 - 1s - loss: 0.3266 - accuracy: 0.9211 - val_loss: 0.1146 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 86/250
1/1 - 1s - loss: 0.3055 - accuracy: 0.9167 - val_loss: 0.1088 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 87/250
1/1 - 1s - loss: 0.2825 - accuracy: 0.9254 - val_loss: 0.1017 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 88/250
1/1 - 1s - loss: 0.3159 - accuracy: 0.9342 - val_loss: 0.0883 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 89/250
1/1 - 1s - loss: 0.2387 - accuracy: 0.9605 - val_loss: 0.0714 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 90/250
1/1 - 1s - loss: 0.3189 - accuracy: 0.9079 - val_loss: 0.0580 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
```

```
Epoch 91/250
1/1 - 1s - loss: 0.2580 - accuracy: 0.9298 - val_loss: 0.0546 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 92/250
1/1 - 2s - loss: 0.2558 - accuracy: 0.9474 - val_loss: 0.0621 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 93/250
1/1 - 1s - loss: 0.2564 - accuracy: 0.9167 - val_loss: 0.0668 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 94/250
1/1 - 2s - loss: 0.2890 - accuracy: 0.9167 - val_loss: 0.0452 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 95/250
1/1 - 1s - loss: 0.2069 - accuracy: 0.9342 - val_loss: 0.0397 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 96/250
1/1 - 1s - loss: 0.2019 - accuracy: 0.9518 - val_loss: 0.0382 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 97/250
1/1 - 1s - loss: 0.1929 - accuracy: 0.9561 - val_loss: 0.0402 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 98/250
1/1 - 1s - loss: 0.1663 - accuracy: 0.9649 - val_loss: 0.0600 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 99/250
1/1 - 1s - loss: 0.2363 - accuracy: 0.9474 - val_loss: 0.0775 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 100/250
1/1 - 1s - loss: 0.1639 - accuracy: 0.9737 - val_loss: 0.0571 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 101/250
1/1 - 1s - loss: 0.1539 - accuracy: 0.9649 - val_loss: 0.0320 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 102/250
1/1 - 1s - loss: 0.1590 - accuracy: 0.9561 - val_loss: 0.0208 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 103/250
1/1 - 1s - loss: 0.2020 - accuracy: 0.9254 - val_loss: 0.0183 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 104/250
1/1 - 2s - loss: 0.1668 - accuracy: 0.9561 - val_loss: 0.0202 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 105/250
1/1 - 2s - loss: 0.1421 - accuracy: 0.9649 - val_loss: 0.0326 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 106/250
1/1 - 1s - loss: 0.1171 - accuracy: 0.9781 - val_loss: 0.0397 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 107/250
1/1 - 1s - loss: 0.1405 - accuracy: 0.9605 - val_loss: 0.0229 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 108/250
1/1 - 1s - loss: 0.1454 - accuracy: 0.9693 - val_loss: 0.0162 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 109/250
1/1 - 1s - loss: 0.1125 - accuracy: 0.9693 - val_loss: 0.0119 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 110/250
1/1 - 1s - loss: 0.1025 - accuracy: 0.9825 - val_loss: 0.0112 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 111/250
1/1 - 1s - loss: 0.1137 - accuracy: 0.9737 - val_loss: 0.0111 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 112/250
1/1 - 1s - loss: 0.1129 - accuracy: 0.9781 - val_loss: 0.0120 - val_accuracy: 1.0000 - 1
```

```
s/epoch - 1s/step
Epoch 113/250
1/1 - 1s - loss: 0.1022 - accuracy: 0.9868 - val_loss: 0.0150 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 114/250
1/1 - 2s - loss: 0.1211 - accuracy: 0.9737 - val_loss: 0.0161 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 115/250
1/1 - 2s - loss: 0.0820 - accuracy: 0.9868 - val_loss: 0.0168 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 116/250
1/1 - 2s - loss: 0.1396 - accuracy: 0.9605 - val_loss: 0.0130 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 117/250
1/1 - 1s - loss: 0.0968 - accuracy: 0.9781 - val_loss: 0.0100 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 118/250
1/1 - 1s - loss: 0.0628 - accuracy: 0.9912 - val_loss: 0.0084 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 119/250
1/1 - 2s - loss: 0.0925 - accuracy: 0.9781 - val_loss: 0.0076 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 120/250
1/1 - 2s - loss: 0.0873 - accuracy: 0.9781 - val_loss: 0.0077 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 121/250
1/1 - 1s - loss: 0.1224 - accuracy: 0.9737 - val_loss: 0.0107 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 122/250
1/1 - 1s - loss: 0.1430 - accuracy: 0.9561 - val_loss: 0.0133 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 123/250
1/1 - 1s - loss: 0.0929 - accuracy: 0.9781 - val_loss: 0.0125 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 124/250
1/1 - 1s - loss: 0.0762 - accuracy: 0.9825 - val_loss: 0.0093 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 125/250
1/1 - 2s - loss: 0.0690 - accuracy: 0.9912 - val_loss: 0.0074 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 126/250
1/1 - 2s - loss: 0.0845 - accuracy: 0.9825 - val_loss: 0.0065 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 127/250
1/1 - 2s - loss: 0.0761 - accuracy: 0.9868 - val_loss: 0.0071 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 128/250
1/1 - 1s - loss: 0.0915 - accuracy: 0.9912 - val_loss: 0.0083 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 129/250
1/1 - 1s - loss: 0.0714 - accuracy: 0.9912 - val_loss: 0.0099 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 130/250
1/1 - 1s - loss: 0.0548 - accuracy: 0.9912 - val_loss: 0.0101 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 131/250
1/1 - 1s - loss: 0.0670 - accuracy: 0.9825 - val_loss: 0.0088 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 132/250
1/1 - 1s - loss: 0.0567 - accuracy: 0.9825 - val_loss: 0.0064 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 133/250
1/1 - 1s - loss: 0.0377 - accuracy: 0.9956 - val_loss: 0.0049 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 134/250
```

```
1/1 - 1s - loss: 0.0551 - accuracy: 0.9868 - val_loss: 0.0041 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 135/250
1/1 - 1s - loss: 0.0522 - accuracy: 0.9868 - val_loss: 0.0042 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 136/250
1/1 - 2s - loss: 0.0504 - accuracy: 0.9868 - val_loss: 0.0047 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 137/250
1/1 - 2s - loss: 0.0457 - accuracy: 0.9868 - val_loss: 0.0054 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 138/250
1/1 - 2s - loss: 0.0673 - accuracy: 0.9912 - val_loss: 0.0055 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 139/250
1/1 - 1s - loss: 0.0448 - accuracy: 0.9912 - val_loss: 0.0051 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 140/250
1/1 - 1s - loss: 0.0388 - accuracy: 1.0000 - val_loss: 0.0055 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 141/250
1/1 - 1s - loss: 0.0730 - accuracy: 0.9781 - val_loss: 0.0060 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 142/250
1/1 - 1s - loss: 0.0431 - accuracy: 1.0000 - val_loss: 0.0050 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 143/250
1/1 - 1s - loss: 0.0306 - accuracy: 0.9912 - val_loss: 0.0044 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 144/250
1/1 - 1s - loss: 0.0490 - accuracy: 0.9912 - val_loss: 0.0055 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 145/250
1/1 - 1s - loss: 0.0377 - accuracy: 0.9912 - val_loss: 0.0063 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 146/250
1/1 - 1s - loss: 0.0487 - accuracy: 0.9825 - val_loss: 0.0060 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 147/250
1/1 - 2s - loss: 0.0482 - accuracy: 0.9868 - val_loss: 0.0045 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 148/250
1/1 - 1s - loss: 0.0476 - accuracy: 0.9868 - val_loss: 0.0035 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 149/250
1/1 - 2s - loss: 0.0480 - accuracy: 0.9912 - val_loss: 0.0025 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 150/250
1/1 - 1s - loss: 0.0687 - accuracy: 0.9737 - val_loss: 0.0025 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 151/250
1/1 - 1s - loss: 0.0245 - accuracy: 1.0000 - val_loss: 0.0027 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 152/250
1/1 - 1s - loss: 0.0373 - accuracy: 0.9956 - val_loss: 0.0028 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 153/250
1/1 - 1s - loss: 0.0409 - accuracy: 0.9912 - val_loss: 0.0032 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 154/250
1/1 - 1s - loss: 0.0256 - accuracy: 1.0000 - val_loss: 0.0036 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 155/250
1/1 - 1s - loss: 0.0417 - accuracy: 0.9868 - val_loss: 0.0034 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
```

```
Epoch 156/250
1/1 - 1s - loss: 0.0338 - accuracy: 0.9956 - val_loss: 0.0029 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 157/250
1/1 - 1s - loss: 0.0411 - accuracy: 0.9956 - val_loss: 0.0025 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 158/250
1/1 - 2s - loss: 0.0291 - accuracy: 1.0000 - val_loss: 0.0023 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 159/250
1/1 - 2s - loss: 0.0330 - accuracy: 0.9912 - val_loss: 0.0026 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 160/250
1/1 - 2s - loss: 0.0295 - accuracy: 0.9956 - val_loss: 0.0025 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 161/250
1/1 - 1s - loss: 0.0520 - accuracy: 0.9868 - val_loss: 0.0021 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 162/250
1/1 - 1s - loss: 0.0385 - accuracy: 0.9956 - val_loss: 0.0015 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 163/250
1/1 - 1s - loss: 0.0374 - accuracy: 0.9912 - val_loss: 0.0012 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 164/250
1/1 - 1s - loss: 0.0198 - accuracy: 1.0000 - val_loss: 0.0010 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 165/250
1/1 - 1s - loss: 0.0282 - accuracy: 0.9956 - val_loss: 9.2945e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 166/250
1/1 - 1s - loss: 0.0288 - accuracy: 0.9956 - val_loss: 8.7733e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 167/250
1/1 - 1s - loss: 0.0297 - accuracy: 0.9912 - val_loss: 9.1149e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 168/250
1/1 - 1s - loss: 0.0369 - accuracy: 0.9912 - val_loss: 0.0010 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 169/250
1/1 - 2s - loss: 0.0266 - accuracy: 0.9956 - val_loss: 0.0012 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 170/250
1/1 - 2s - loss: 0.0253 - accuracy: 1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 171/250
1/1 - 2s - loss: 0.0237 - accuracy: 1.0000 - val_loss: 0.0021 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 172/250
1/1 - 1s - loss: 0.0405 - accuracy: 0.9868 - val_loss: 0.0024 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 173/250
1/1 - 1s - loss: 0.0204 - accuracy: 1.0000 - val_loss: 0.0022 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 174/250
1/1 - 1s - loss: 0.0374 - accuracy: 0.9956 - val_loss: 0.0013 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 175/250
1/1 - 1s - loss: 0.0205 - accuracy: 1.0000 - val_loss: 8.5288e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 176/250
1/1 - 1s - loss: 0.0267 - accuracy: 0.9912 - val_loss: 7.2261e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 177/250
1/1 - 1s - loss: 0.0246 - accuracy: 0.9956 - val_loss: 7.1006e-04 - val_accuracy: 1.0000
```

```
- 1s/epoch - 1s/step
Epoch 178/250
1/1 - 1s - loss: 0.0409 - accuracy: 0.9912 - val_loss: 7.2268e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 179/250
1/1 - 1s - loss: 0.0460 - accuracy: 0.9868 - val_loss: 8.6032e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 180/250
1/1 - 2s - loss: 0.0234 - accuracy: 1.0000 - val_loss: 0.0011 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 181/250
1/1 - 2s - loss: 0.0143 - accuracy: 1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 182/250
1/1 - 2s - loss: 0.0205 - accuracy: 0.9912 - val_loss: 0.0016 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 183/250
1/1 - 1s - loss: 0.0385 - accuracy: 0.9956 - val_loss: 0.0016 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 184/250
1/1 - 1s - loss: 0.0212 - accuracy: 1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 185/250
1/1 - 1s - loss: 0.0104 - accuracy: 1.0000 - val_loss: 0.0012 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 186/250
1/1 - 1s - loss: 0.0203 - accuracy: 0.9956 - val_loss: 0.0010 - val_accuracy: 1.0000 - 1
s/epoch - 1s/step
Epoch 187/250
1/1 - 1s - loss: 0.0112 - accuracy: 1.0000 - val_loss: 8.9101e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 188/250
1/1 - 1s - loss: 0.0260 - accuracy: 0.9912 - val_loss: 6.6995e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 189/250
1/1 - 1s - loss: 0.0274 - accuracy: 0.9956 - val_loss: 5.1627e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 190/250
1/1 - 2s - loss: 0.0192 - accuracy: 0.9956 - val_loss: 4.3276e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 191/250
1/1 - 2s - loss: 0.0280 - accuracy: 0.9956 - val_loss: 4.4664e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 192/250
1/1 - 2s - loss: 0.0260 - accuracy: 0.9956 - val_loss: 6.3213e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 193/250
1/1 - 1s - loss: 0.0201 - accuracy: 0.9956 - val_loss: 8.6757e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 194/250
1/1 - 1s - loss: 0.0135 - accuracy: 1.0000 - val_loss: 9.6351e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 195/250
1/1 - 1s - loss: 0.0120 - accuracy: 1.0000 - val_loss: 9.5519e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 196/250
1/1 - 1s - loss: 0.0199 - accuracy: 0.9956 - val_loss: 7.4627e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 197/250
1/1 - 1s - loss: 0.0088 - accuracy: 1.0000 - val_loss: 6.0598e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 198/250
1/1 - 1s - loss: 0.0146 - accuracy: 0.9956 - val_loss: 5.2755e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 199/250
```

```
1/1 - 1s - loss: 0.0215 - accuracy: 0.9912 - val_loss: 4.7892e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 200/250
1/1 - 1s - loss: 0.0187 - accuracy: 1.0000 - val_loss: 4.3748e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 201/250
1/1 - 2s - loss: 0.0163 - accuracy: 1.0000 - val_loss: 4.2037e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 202/250
1/1 - 2s - loss: 0.0109 - accuracy: 1.0000 - val_loss: 4.2921e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 203/250
1/1 - 2s - loss: 0.0260 - accuracy: 0.9912 - val_loss: 4.8049e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 204/250
1/1 - 1s - loss: 0.0211 - accuracy: 0.9956 - val_loss: 5.2580e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 205/250
1/1 - 1s - loss: 0.0119 - accuracy: 1.0000 - val_loss: 5.6667e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 206/250
1/1 - 1s - loss: 0.0070 - accuracy: 1.0000 - val_loss: 5.9990e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 207/250
1/1 - 1s - loss: 0.0157 - accuracy: 1.0000 - val_loss: 5.8876e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 208/250
1/1 - 1s - loss: 0.0140 - accuracy: 1.0000 - val_loss: 5.4167e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 209/250
1/1 - 2s - loss: 0.0202 - accuracy: 0.9956 - val_loss: 5.2149e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 210/250
1/1 - 1s - loss: 0.0214 - accuracy: 0.9956 - val_loss: 4.6519e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 211/250
1/1 - 1s - loss: 0.0222 - accuracy: 0.9912 - val_loss: 4.2931e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 212/250
1/1 - 2s - loss: 0.0216 - accuracy: 0.9956 - val_loss: 4.1386e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 213/250
1/1 - 2s - loss: 0.0138 - accuracy: 1.0000 - val_loss: 4.1641e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 214/250
1/1 - 2s - loss: 0.0069 - accuracy: 1.0000 - val_loss: 4.3968e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 215/250
1/1 - 1s - loss: 0.0150 - accuracy: 1.0000 - val_loss: 5.0133e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 216/250
1/1 - 2s - loss: 0.0115 - accuracy: 1.0000 - val_loss: 6.4609e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 217/250
1/1 - 1s - loss: 0.0077 - accuracy: 1.0000 - val_loss: 8.5208e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 218/250
1/1 - 2s - loss: 0.0130 - accuracy: 1.0000 - val_loss: 0.0010 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 219/250
1/1 - 2s - loss: 0.0136 - accuracy: 0.9956 - val_loss: 9.4804e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 220/250
1/1 - 2s - loss: 0.0240 - accuracy: 0.9912 - val_loss: 0.0014 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
```

```
Epoch 221/250
1/1 - 2s - loss: 0.0139 - accuracy: 1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 222/250
1/1 - 2s - loss: 0.0132 - accuracy: 1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 223/250
1/1 - 2s - loss: 0.0101 - accuracy: 1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 224/250
1/1 - 2s - loss: 0.0139 - accuracy: 1.0000 - val_loss: 9.9845e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 225/250
1/1 - 2s - loss: 0.0057 - accuracy: 1.0000 - val_loss: 7.0240e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 226/250
1/1 - 1s - loss: 0.0088 - accuracy: 1.0000 - val_loss: 5.2101e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 227/250
1/1 - 2s - loss: 0.0185 - accuracy: 1.0000 - val_loss: 4.9448e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 228/250
1/1 - 2s - loss: 0.0218 - accuracy: 1.0000 - val_loss: 5.1698e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 229/250
1/1 - 2s - loss: 0.0146 - accuracy: 0.9956 - val_loss: 4.5547e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 230/250
1/1 - 2s - loss: 0.0074 - accuracy: 1.0000 - val_loss: 4.1559e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 231/250
1/1 - 1s - loss: 0.0081 - accuracy: 1.0000 - val_loss: 3.8654e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
Epoch 232/250
1/1 - 2s - loss: 0.0132 - accuracy: 1.0000 - val_loss: 3.9357e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 233/250
1/1 - 2s - loss: 0.0312 - accuracy: 0.9825 - val_loss: 6.0651e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 234/250
1/1 - 4s - loss: 0.0093 - accuracy: 1.0000 - val_loss: 9.8317e-04 - val_accuracy: 1.0000
- 4s/epoch - 4s/step
Epoch 235/250
1/1 - 3s - loss: 0.0085 - accuracy: 1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 236/250
1/1 - 3s - loss: 0.0118 - accuracy: 0.9956 - val_loss: 0.0029 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 237/250
1/1 - 3s - loss: 0.0080 - accuracy: 1.0000 - val_loss: 0.0037 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 238/250
1/1 - 3s - loss: 0.0122 - accuracy: 1.0000 - val_loss: 0.0033 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 239/250
1/1 - 3s - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0028 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 240/250
1/1 - 3s - loss: 0.0094 - accuracy: 1.0000 - val_loss: 0.0021 - val_accuracy: 1.0000 - 3
s/epoch - 3s/step
Epoch 241/250
1/1 - 2s - loss: 0.0052 - accuracy: 1.0000 - val_loss: 0.0015 - val_accuracy: 1.0000 - 2
s/epoch - 2s/step
Epoch 242/250
1/1 - 2s - loss: 0.0065 - accuracy: 1.0000 - val_loss: 0.0011 - val_accuracy: 1.0000 - 2
```

```
s/epoch - 2s/step
Epoch 243/250
1/1 - 2s - loss: 0.0178 - accuracy: 0.9956 - val_loss: 5.9796e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 244/250
1/1 - 2s - loss: 0.0146 - accuracy: 0.9956 - val_loss: 3.9081e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 245/250
1/1 - 3s - loss: 0.0138 - accuracy: 0.9956 - val_loss: 3.4157e-04 - val_accuracy: 1.0000
- 3s/epoch - 3s/step
Epoch 246/250
1/1 - 3s - loss: 0.0133 - accuracy: 1.0000 - val_loss: 3.4440e-04 - val_accuracy: 1.0000
- 3s/epoch - 3s/step
Epoch 247/250
1/1 - 2s - loss: 0.0076 - accuracy: 1.0000 - val_loss: 3.3158e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 248/250
1/1 - 2s - loss: 0.0162 - accuracy: 1.0000 - val_loss: 2.9143e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 249/250
1/1 - 2s - loss: 0.0112 - accuracy: 1.0000 - val_loss: 4.1702e-04 - val_accuracy: 1.0000
- 2s/epoch - 2s/step
Epoch 250/250
1/1 - 1s - loss: 0.0094 - accuracy: 1.0000 - val_loss: 8.3418e-04 - val_accuracy: 1.0000
- 1s/epoch - 1s/step
```

In [27]:

```
#Evaluate the test data

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 0.2801
test acc 0.9563
```

Step 7: Plot the result

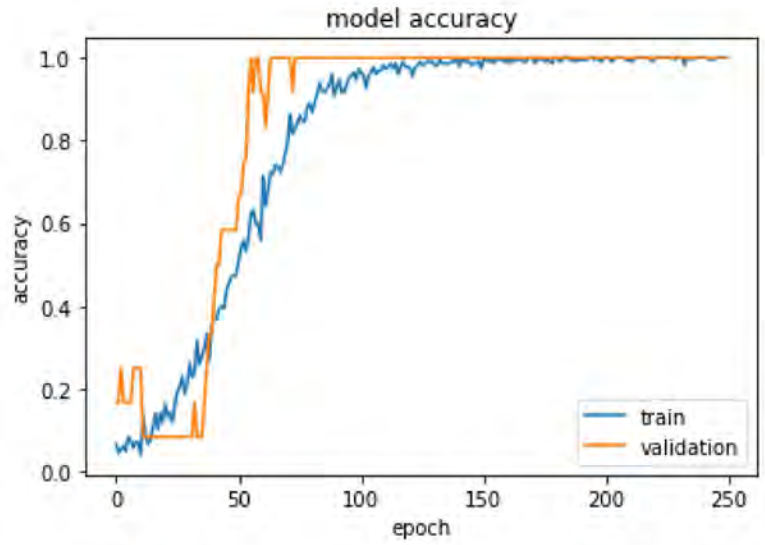
In [31]:

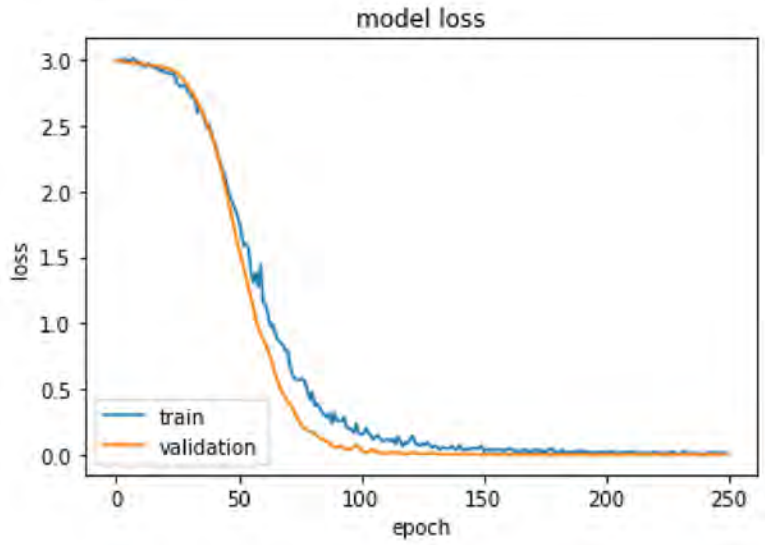
```
# list all data in history
print(history.history.keys())
```

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

In [36]:

```
# 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', 'validation'], loc='lower right')
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', 'validation'], loc='lower left')
plt.show()
```





Step 8: Plot Confusion Matrix

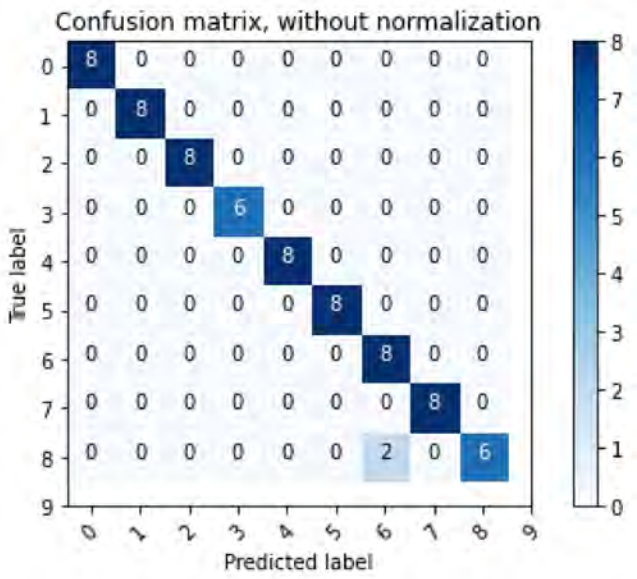
In [44]:
```
predicted =np.array(cnn_model.predict(x_test))
#print(predicted)
#print(y_test)
ynew =np.argmax(cnn_model.predict(x_test), axis=-1)
```

In [45]:
```
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):
    """
```

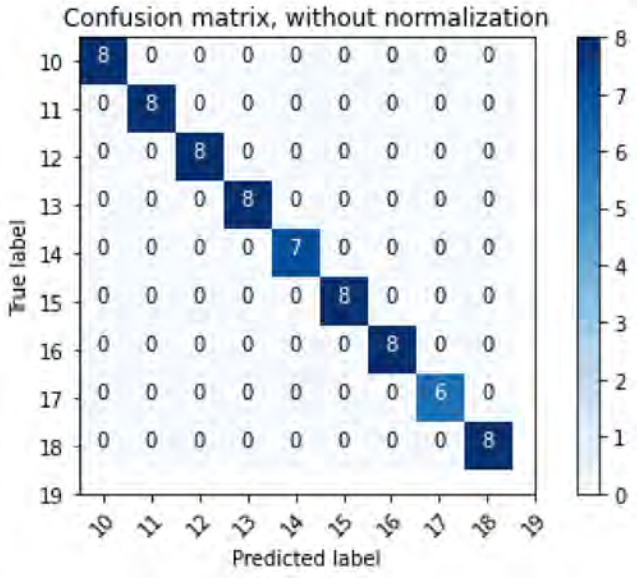
```
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        #print("Normalized confusion matrix")
    else:
        print('Confusion matrix, 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 :
0.95625
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 2 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 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 8 0 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]
 [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 8 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 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

In [ ]: