```
import os
import plotly.express as px
```

```
import os
from pathlib import PosixPath
import random
import matplotlib.pyplot as plt
%matplotlib inline

import numpy as np
from PIL import Image

import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras import preprocessing
from tensorflow.keras import callbacks
from tensorflow.keras import regularizers
from tensorflow.keras import optimizers
```

```
root_path = '/content/drive/MyDrive/Data/'
data_dir = PosixPath(root_path)
```

```
image_count = len(list(data_dir.glob("*/*")))
print(f"Image count: {image_count}")
```

```
Image count: 11739
```

```
class_names = os.listdir(root_path)
class_names
```

```
['Crocodile',
 'Chopper',
 'Kurohige',
 'Luffy',
 'Franky',
 'Law',
 'Jinbei',
 'Brook',
 'Ace',
 'Akainu',
 'Rayleigh',
 'Shanks',
 'Zoro',
 'Robin',
 'Sanji',
 'Usopp',
 'Nami',
 'Mihawk',
 'SplitImages']
```

```
class_names.remove('SplitImages')
```

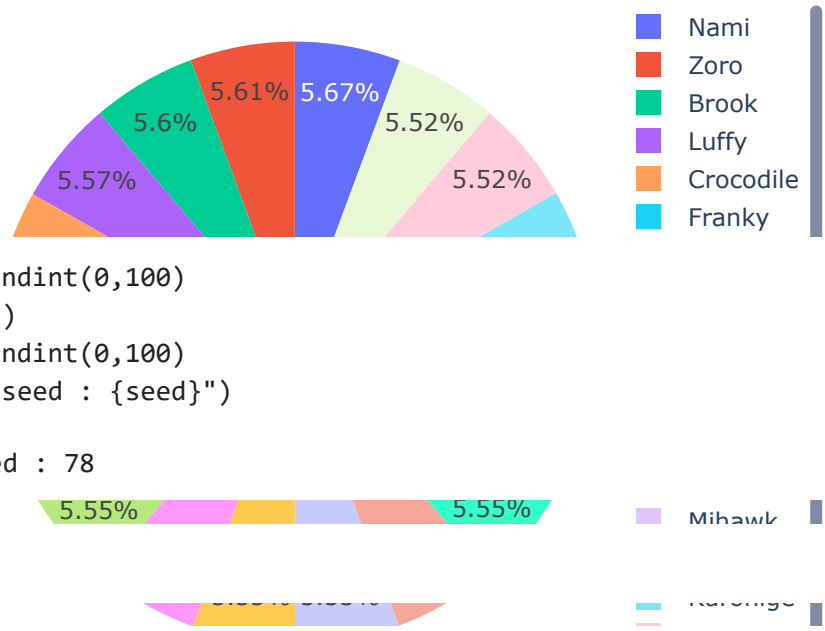
```
len(class_names)
```

```
18
```

```
class_distribution = [len(os.listdir(root_path + name)) for name in class_names]
class_distribution
```

```
[651,
 648,
 648,
 654,
 651,
 651,
 651,
 657,
 648,
 651,
 651,
 651,
 659,
 651,
 651,
 648,
 665,
 651]
```

```
fig = px.pie(
    names=class_names,
    values=class_distribution,
    width=500,
    hole=0.2,
    title="Class Distribution"
)
fig.update_layout({'title':{'x':0.5}})
fig.show()
```



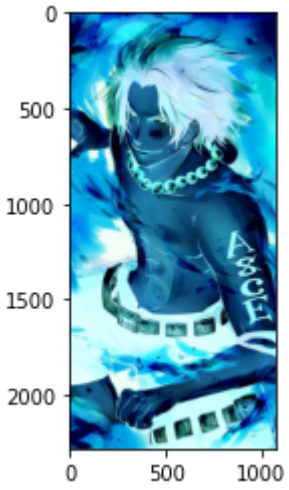

```
seed = random.randint(0,100)
random.seed(seed)
seed = random.randint(0,100)
print(f"Current seed : {seed}")
```

```
Current seed : 78
```



```
import cv2
```

```
img_array = cv2.imread('/content/drive/MyDrive/Data/Ace/1.jpg_inverted.png')
img_array = cv2.cvtColor(img_array,cv2.COLOR_BGR2RGB)
plt.imshow(img_array)
plt.show()
```




```
#!pip install split-folders


#import splitfolders

#splitfolders.ratio(root_path, output="/content/drive/MyDrive/Data/SplitImages",
                   #  seed=seed, ratio=(.7, .3),
                   #  group_prefix=None)


train_data_path = '/content/drive/MyDrive/Data/SplitImages/train'
test_data_path = '/content/drive/MyDrive/Data/SplitImages/val'
```

```
training_data = []
test_data = []

def create_data(my_data_path,my_data):
    for ct in class_names:
      path = os.path.join(my_data_path,ct)
      class_num = class_names.index(ct)
      for img in os.listdir(path):
        try:
          img_array = cv2.imread(os.path.join(path,img))
          img_array = cv2.cvtColor(img_array,cv2.COLOR_BGR2RGB)
          img_array = cv2.resize(img_array, (100, 100))
          my_data.append([img_array,class_num])
        except Exception as e:
          pass


batch_size = 32
img_size = 100


from keras.preprocessing.image import ImageDataGenerator


datagen = ImageDataGenerator(rescale=1/255.,
                             zoom_range=0.2,
                             horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1/255.)

train_generator = datagen.flow_from_directory(train_data_path,
                                              target_size=(img_size, img_size),
                                              batch_size=batch_size,
                                              shuffle=True,
                                              subset='training',
                                              class_mode='categorical')

test_generator = test_datagen.flow_from_directory(test_data_path,
                                                  target_size=(img_size, img_size),
                                                  batch_size=batch_size,
                                                  shuffle=False,
                                                  class_mode='categorical')
```

```
Found 8204 images belonging to 18 classes.
Found 3533 images belonging to 18 classes.
```

```
create_data(train_data_path,training_data)
create_data(test_data_path,test_data)


random.shuffle(training_data)


len(training_data)
```

```
8204
```

```
# def list_splitter(list_to_split, ratio):
#     elements = len(list_to_split)
#     middle = int(elements * ratio)
#     return [list_to_split[:middle], list_to_split[middle:]]

# train_data, val_data = list_splitter(training_data, .9)


# len(train_data)


# len(val_data)


x = []
y = []

for features, label in training_data:
  x.append(features)
  y.append(label)

x_test = []
y_test = []

for features, label in test_data:
  x_test.append(features)
  y_test.append(label)

# x_val = []
# y_val = []

# for features, label in val_data:
#   x_test.append(features)
#   y_test.append(label)


y = np.array(y)
y_test = np.array(y_test)
# y_val = np.array(y_val)

x = np.array(x)
x_test = np.array(x_test)
# x_val = np.array(x_val)

x = np.array(x).reshape(-1, img_size, img_size, 3)
x_test = np.array(x_test).reshape(-1, img_size, img_size, 3)
# x_val = np.array(x_val).reshape(-1, img_size, img_size, 3)


from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

early_stopping_monitor = EarlyStopping(monitor='val_accuracy', patience=25, restore_best_w

learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=25,
```

```
                                    verbose=1,
                                    factor=0.5,
                                    min_lr=0.00001)

best_model = ModelCheckpoint('/content/drive/MyDrive/bestmodel.hdf5', monitor='accuracy',
best_val_acc = ModelCheckpoint('/content/drive/MyDrive/best_val_acc.hdf5', monitor='val_ac

data_augmentation = Sequential(
    layers=[
        tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical", i
        tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
        tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),
        ],
    name="data_augmentation"
    )

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchN
from tensorflow.keras.utils import to_categorical

model = Sequential()

model.add(data_augmentation)

model.add(Conv2D(16, (3, 3), kernel_regularizer=regularizers.l2(0.001), activation='relu',
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(16, (3, 3), kernel_regularizer=regularizers.l2(0.001), activation='relu')
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(18, activation='softmax'))

opt = tf.keras.optimizers.Adam(learning_rate=0.01, decay=1e-6)

model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])


# from keras.utils import to_categorical
y_one_hot=to_categorical(y)
# y_val_one_hot=to_categorical(y_val)
```

Do not use shuffle and validation split same time or else val_acc: 0.0000e+00

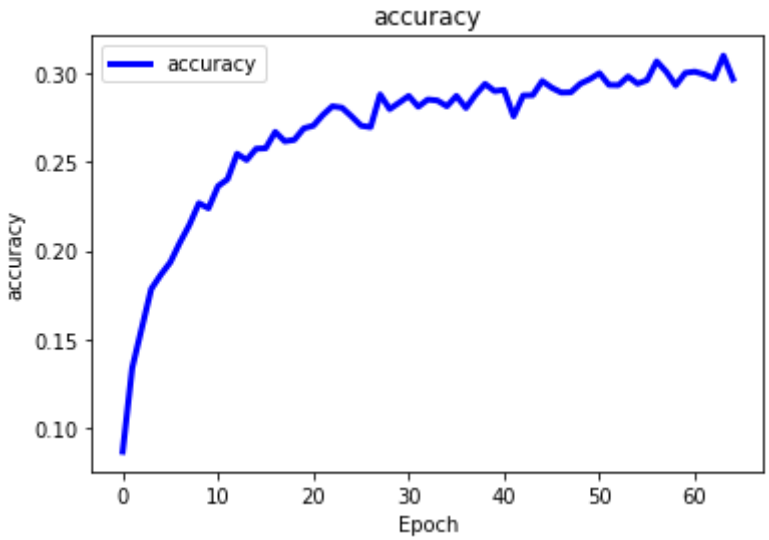
```
hist = model.fit(x, y_one_hot, validation_split=.1, verbose=2, epochs=125, batch_size=128,
    early_stopping_monitor,
    learning_rate_reduction,
    best_model, best_val_acc])
```

```
58/58 - 44s - loss: 2.3132 - accuracy: 0.2880 - val_loss: 2.3026 - val_accuracy:
Epoch 39/125
58/58 - 43s - loss: 2.3138 - accuracy: 0.2941 - val_loss: 2.1281 - val_accuracy:
Epoch 40/125
58/58 - 45s - loss: 2.3037 - accuracy: 0.2900 - val_loss: 2.1641 - val_accuracy:
Epoch 41/125
58/58 - 44s - loss: 2.3143 - accuracy: 0.2907 - val_loss: 2.2931 - val_accuracy:
Epoch 42/125
58/58 - 43s - loss: 2.3257 - accuracy: 0.2758 - val_loss: 2.2769 - val_accuracy:
Epoch 43/125
58/58 - 43s - loss: 2.3061 - accuracy: 0.2876 - val_loss: 2.2524 - val_accuracy:
Epoch 44/125
58/58 - 44s - loss: 2.3131 - accuracy: 0.2877 - val_loss: 2.2232 - val_accuracy:
Epoch 45/125
58/58 - 43s - loss: 2.2925 - accuracy: 0.2957 - val_loss: 2.1780 - val_accuracy:
Epoch 46/125
58/58 - 43s - loss: 2.3108 - accuracy: 0.2919 - val_loss: 2.5959 - val_accuracy:
Epoch 47/125
58/58 - 43s - loss: 2.3023 - accuracy: 0.2892 - val_loss: 2.5688 - val_accuracy:
Epoch 48/125
58/58 - 44s - loss: 2.3063 - accuracy: 0.2893 - val_loss: 2.2905 - val_accuracy:
Epoch 49/125
58/58 - 43s - loss: 2.2995 - accuracy: 0.2942 - val_loss: 2.1660 - val_accuracy:
Epoch 50/125
58/58 - 43s - loss: 2.2830 - accuracy: 0.2968 - val_loss: 2.4173 - val_accuracy:
Epoch 51/125
58/58 - 45s - loss: 2.2985 - accuracy: 0.3001 - val_loss: 2.1484 - val_accuracy:
Epoch 52/125
58/58 - 43s - loss: 2.2981 - accuracy: 0.2935 - val_loss: 2.3737 - val_accuracy:
Epoch 53/125
58/58 - 43s - loss: 2.2845 - accuracy: 0.2934 - val_loss: 2.2298 - val_accuracy:
Epoch 54/125
58/58 - 44s - loss: 2.2868 - accuracy: 0.2983 - val_loss: 2.2689 - val_accuracy:
Epoch 55/125
58/58 - 43s - loss: 2.2742 - accuracy: 0.2942 - val_loss: 2.3342 - val_accuracy:
Epoch 56/125
58/58 - 43s - loss: 2.2974 - accuracy: 0.2962 - val_loss: 2.7239 - val_accuracy:
Epoch 57/125
58/58 - 45s - loss: 2.2750 - accuracy: 0.3068 - val_loss: 2.7788 - val_accuracy:
Epoch 58/125
58/58 - 43s - loss: 2.2853 - accuracy: 0.3010 - val_loss: 2.4319 - val_accuracy:
Epoch 59/125
58/58 - 43s - loss: 2.2887 - accuracy: 0.2932 - val_loss: 2.2954 - val_accuracy:
Epoch 60/125
58/58 - 44s - loss: 2.2813 - accuracy: 0.3003 - val_loss: 2.2429 - val_accuracy:
Epoch 61/125
58/58 - 43s - loss: 2.2738 - accuracy: 0.3010 - val_loss: 2.1272 - val_accuracy:
Epoch 62/125
58/58 - 43s - loss: 2.2904 - accuracy: 0.2995 - val_loss: 2.3526 - val_accuracy:
Epoch 63/125
58/58 - 44s - loss: 2.2714 - accuracy: 0.2972 - val_loss: 2.2467 - val_accuracy:
Epoch 64/125
58/58 - 43s - loss: 2.2632 - accuracy: 0.3100 - val_loss: 2.5228 - val_accuracy:
Epoch 65/125
```

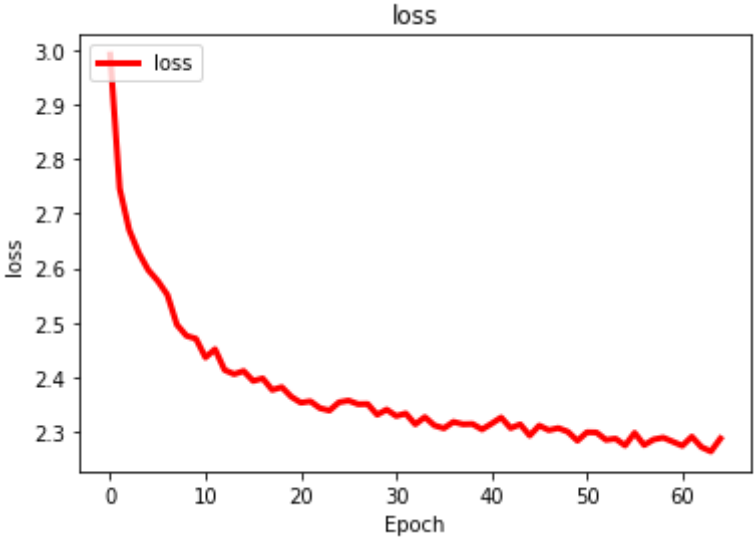
```
Epoch 65: ReduceLROnPlateau reducing learning rate to 0.004999999888241291.
58/58 - 43s - loss: 2.2871 - accuracy: 0.2968 - val_loss: 2.2735 - val_accuracy:
```

```
def visualization(name,h,color):
  t = h.history[name]
  my_max = max(t)
  my_min = min(t)
  print(f'Name : {name} max : {my_max} min : {my_min}')
  plt.plot(t,color=color,linewidth=3.0)
  plt.title(name)
  plt.ylabel(name)
  plt.xlabel('Epoch')
  plt.legend([name],loc='upper left')
  plt.show()


visualization('accuracy',hist,'Blue')
visualization('loss',hist,'Red')
visualization('val_accuracy',hist,'Green')
visualization('val_loss',hist,'Black')
```

```
Name : accuracy max : 0.3100365698337555 min : 0.08655018359422684
```




```
Name : loss max : 2.992206335067749 min : 2.2632436752319336
```




```
import seaborn as sns
from sklearn.metrics import f1_score, confusion_matrix, ConfusionMatrixDisplay
```

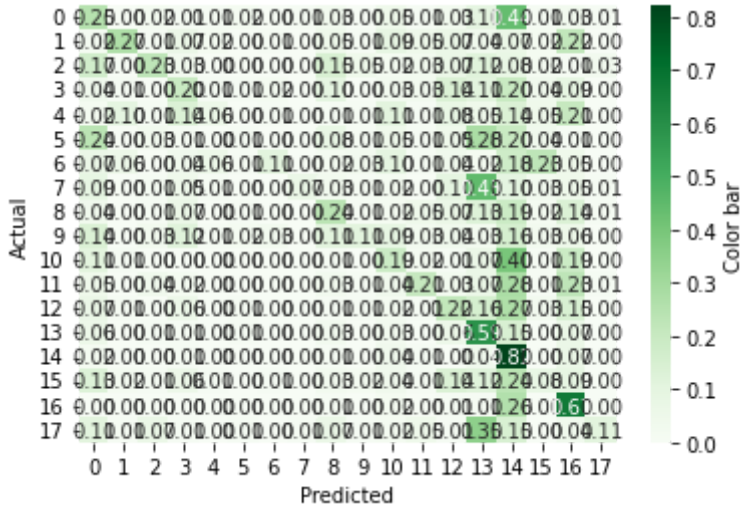
```
model.load_weights('/content/drive/MyDrive/bestmodel.hdf5')
yt_one_hot=to_categorical(y_test)
res = model.evaluate(x_test, yt_one_hot)
print("test loss, test acc:", res)
```

```
111/111 [==============================] - 5s 47ms/step - loss: 2.4531 - accuracy: 0
test loss, test acc: [2.453094482421875, 0.2459665983915329]
```

```
def my_predict(my_model,my_x_test):
  y_pred = my_model.predict(my_x_test)
  return y_pred

def my_f1_score(my_y_test,my_y_pred):
  f1 = f1_score(my_y_test, my_y_pred, average="micro")
  return f1

def my_conf_matrix(my_y_test,my_y_pred):
```

```
  cm = confusion_matrix(my_y_test, my_y_pred)
  cm_norm = np.round(cm/np.sum(cm,axis=1).reshape(-1,1),2)
  sns.heatmap(cm_norm,cmap='Greens',annot=True,
              cbar_kws={'orientation' : 'vertical','label' : 'Color bar'},
              fmt='.2f'
              )
  plt.xlabel('Predicted')
  plt.ylabel('Actual')
  plt.show()

y_pred_res = my_predict(model,x_test)
y_pred_res = np.argmax(y_pred_res, axis=-1)
print(my_f1_score(y_test,y_pred_res))
my_conf_matrix(y_test,y_pred_res)
```

```
0.24596660062270026
```




```
model.load_weights('/content/drive/MyDrive/best_val_acc.hdf5')
res = model.evaluate(x_test, yt_one_hot)
print("test loss, test acc:", res)
```

```
111/111 [==============================] - 5s 47ms/step - loss: 2.1625 - accuracy: 0
test loss, test acc: [2.162519931793213, 0.3523917496204376]
```

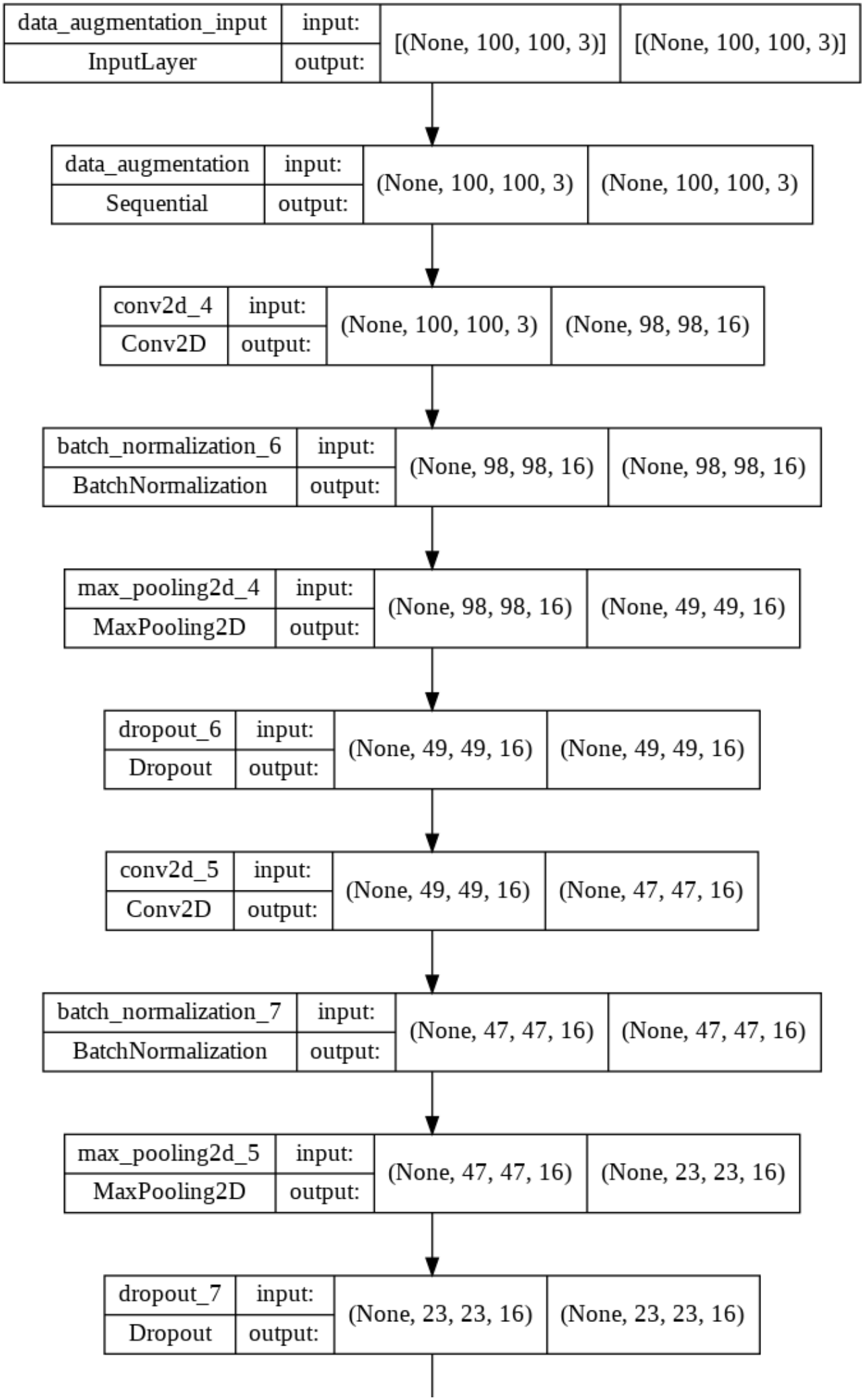
```
y_pred_res = my_predict(model,x_test)
y_pred_res = np.argmax(y_pred_res, axis=-1)
print(my_f1_score(y_test,y_pred_res))
my_conf_matrix(y_test,y_pred_res)
```

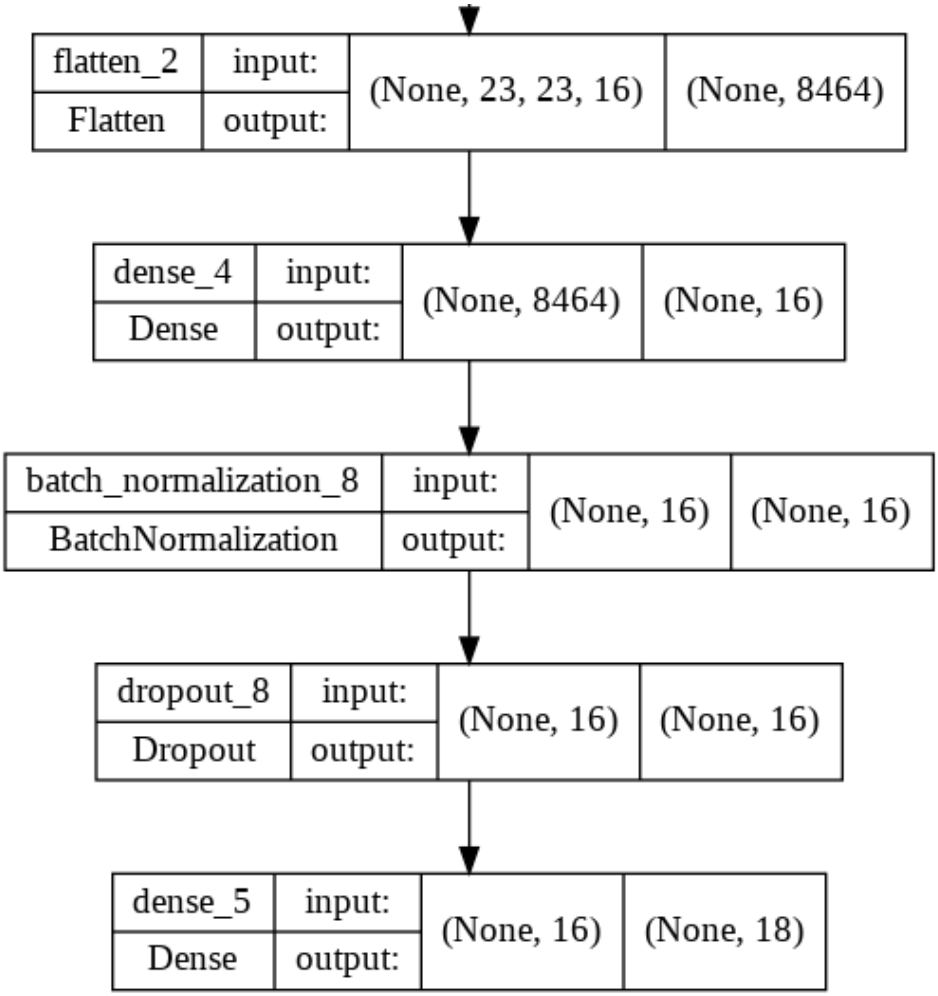
```
0.3523917350693462
```


```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 data_augmentation (Sequenti  (None, 100, 100, 3)      0
 al)

 conv2d_4 (Conv2D)           (None, 98, 98, 16)        448

 batch_normalization_6 (Batc  (None, 98, 98, 16)       64
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 49, 49, 16)       0
 2D)

 dropout_6 (Dropout)         (None, 49, 49, 16)        0

 conv2d_5 (Conv2D)           (None, 47, 47, 16)        2320

 batch_normalization_7 (Batc  (None, 47, 47, 16)       64
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 23, 23, 16)       0
 2D)

 dropout_7 (Dropout)         (None, 23, 23, 16)        0

 flatten_2 (Flatten)         (None, 8464)              0

 dense_4 (Dense)             (None, 16)                135440

 batch_normalization_8 (Batc  (None, 16)               64
 hNormalization)

 dropout_8 (Dropout)         (None, 16)                0

 dense_5 (Dense)             (None, 18)                306

=================================================================
Total params: 138,706
Trainable params: 138,610
Non-trainable params: 96
_________________________________________________________________
```

```
tf.keras.utils.plot_model(model,show_shapes=True)
```


data_augmentation_input
InputLayer
input:
output:
[(None, 100, 100, 3)]
[(None, 100, 100, 3)]
data_augmentation
Sequential
input:
output:
(None, 100, 100, 3)
(None, 100, 100, 3)
conv2d_4
Conv2D
input:
output:
(None, 100, 100, 3)
(None, 98, 98, 16)
batch_normalization_6
BatchNormalization
input:
output:
(None, 98, 98, 16)
(None, 98, 98, 16)
max_pooling2d_4
MaxPooling2D
input:
output:
(None, 98, 98, 16)
(None, 49, 49, 16)
dropout_6
Dropout
input:
output:
(None, 49, 49, 16)
(None, 49, 49, 16)
conv2d_5
Conv2D
input:
output:
(None, 49, 49, 16)
(None, 47, 47, 16)
batch_normalization_7
BatchNormalization
input:
output:
(None, 47, 47, 16)
(None, 47, 47, 16)
max_pooling2d_5
MaxPooling2D
input:
output:
(None, 47, 47, 16)
(None, 23, 23, 16)
dropout_7
Dropout
input:
output:
(None, 23, 23, 16)
(None, 23, 23, 16)




```
model.evaluate(x=x_test, return_dict=True)
```

```
111/111 [==============================] - 0s 1ms/step - loss: 0.0474 - accuracy: 0.
{'loss': 0.04742131754755974, 'accuracy': 0.0}
```