Applying CNN on fruits dataset

```
import shutil,os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.image import imread
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

```
my_data_dir = '/content/drive/My Drive/fruit-360'
os.listdir(my_data_dir)
```

```
['papers', 'test-multiple_fruits', 'Test', 'Training']
```

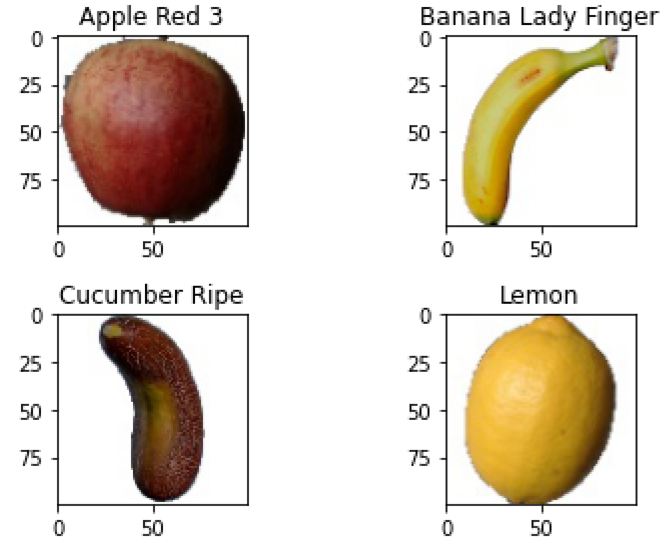
```
train_path = my_data_dir+'/Training/'
test_path = my_data_dir+'/Test/'
```

```
classes = os.listdir(train_path)
print(classes)
```

```
['Apple Red 3', 'Apple Red 2', 'Apple Red Delicious', 'Apple Red Yellow 1', 'Apricot',
```

```
file_name = '0_100.jpg'
width=8
height=8
rows = 2
cols = 2
axes=[]
fig=plt.figure()
i=0
for a in range(rows*cols):
    img = imread(train_path+classes[i]+'/'+file_name)
    axes.append( fig.add_subplot(rows, cols, a+1) )
    subplot_title=classes[i]
```

```
        axes[-1].set_title(subplot_title)
        plt.imshow(img)
        i=i+20
    fig.tight_layout()
    plt.show()
    img_shape=img.shape
    print("Image shape:"+str(img_shape))
```



```
Image shape:(100, 100, 3)
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# help(ImageDataGenerator)
```

```
image_gen = ImageDataGenerator(rotation_range=20, # rotate the image 20 degrees
                               width_shift_range=0.10, # Shift the pic width by a max of 5%
                               height_shift_range=0.10, # Shift the pic height by a max of 5%
                               rescale=1/255, # Rescale the image by normalzing it.
                               shear_range=0.1, # Shear means cutting away part of the image
                               zoom_range=0.1, # Zoom in by 10% max
                               horizontal_flip=True, # Allo horizontal flipping
                               fill_mode='nearest' # Fill in missing pixels with the nearest
                              )
```

```
batch_size=512
```

```
train_image_gen = image_gen.flow_from_directory(train_path,
                                                target_size=img_shape[:2],
                                                 color_mode='rgb',
                                                batch_size=batch_size,
                                                class_mode='categorical')
```

```
Found 67726 images belonging to 131 classes.
```

```
test_image_gen = image_gen.flow_from_directory(test_path,
                                               target_size=img_shape[:2],
                                               color_mode='rgb',
                                               batch_size=batch_size,
                                               class_mode='categorical',shuffle=False)
```

```
Found 22412 images belonging to 129 classes.
```

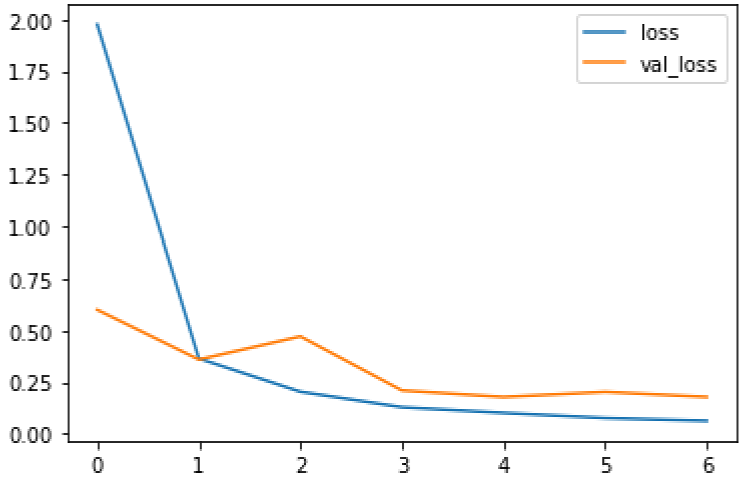
```
import tensorflow as tf


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense,Conv2D,MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping


model = Sequential()

model.add(Conv2D(filters=16, kernel_size=(5,5),input_shape=img_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(filters=32, kernel_size=(5,5),input_shape=img_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(filters=64, kernel_size=(5,5),input_shape=img_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))


model.add(Flatten())


model.add(Dense(1024))
model.add(Activation('relu'))

# Dropouts help reduce overfitting by randomly turning neurons off during training.
# Here we say randomly turn off 50% of neurons.
model.add(Dropout(0.5))

model.add(Dense(131))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])


early_stop = EarlyStopping(monitor='val_loss',verbose=1, patience=2)


#Ignore warnings
with tf.device('/GPU:0'):
    results = model.fit(train_image_gen,validation_data=test_image_gen,callbacks=[early_stop]
                   )
```

```
Epoch 1/12
133/133 [==============================] - 430s 3s/step - loss: 1.9765 - accuracy: 0.47
Epoch 2/12
133/133 [==============================] - 300s 2s/step - loss: 0.3648 - accuracy: 0.87
Epoch 3/12
133/133 [==============================] - 297s 2s/step - loss: 0.2040 - accuracy: 0.93
Epoch 4/12
133/133 [==============================] - 295s 2s/step - loss: 0.1291 - accuracy: 0.95
Epoch 5/12
133/133 [==============================] - 298s 2s/step - loss: 0.1015 - accuracy: 0.96
Epoch 6/12
133/133 [==============================] - 301s 2s/step - loss: 0.0766 - accuracy: 0.97
Epoch 7/12
133/133 [==============================] - 297s 2s/step - loss: 0.0631 - accuracy: 0.97
Epoch 00007: early stopping
```

```
losses = pd.DataFrame(model.history.history)
```

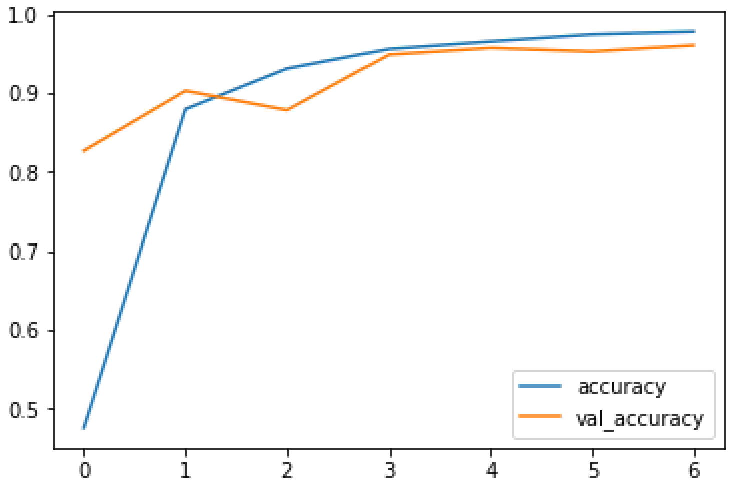
```
losses[['loss','val_loss']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18650977d0>
```



```
losses[['accuracy','val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f189d9dc4d0>
```



```
model.evaluate_generator(test_image_gen)
#[loss,accuracy]
```

```
[0.17831288278102875, 0.9603755474090576]
```

```
model.save('Fruits_Classifier_v1.h5')
```