

```
!pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.13)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2022.12.7)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.65.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.26.15)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.27.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.4)
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
!kaggle datasets download -d aryashah2k/mango-leaf-disease-dataset
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading mango-leaf-disease-dataset.zip to /content
 99% 102M/103M [00:04<00:00, 28.8MB/s]
100% 103M/103M [00:04<00:00, 23.8MB/s]
```

```
from zipfile import ZipFile
```

```
with ZipFile('mango-leaf-disease-dataset.zip', 'r') as f:
```

```
    #extracting in directory 'images'
    f.extractall('images')
```

```
import os
import shutil
import random
```

```
root_folder = "/content/images"
train_ratio = 0.7
val_ratio = 0.15
test_ratio = 0.15
```

```
#list of subfolders (labels)
subfolders = [f.name for f in os.scandir(root_folder) if f.is_dir()]
```

```
for subfolder in subfolders:
    subfolder_path = os.path.join(root_folder, subfolder)
    images = [f.name for f in os.scandir(subfolder_path) if f.is_file()]
```

```
    random.shuffle(images)
```

```
    num_images = len(images)
    num_train = int(num_images * train_ratio)
    num_val = int(num_images * val_ratio)
    num_test = num_images - num_train - num_val
```

```
    train_images = images[:num_train]
    val_images = images[num_train:num_train + num_val]
    test_images = images[num_train + num_val:]
```

```
    train_dir = os.path.join(root_folder, 'train', subfolder)
    val_dir = os.path.join(root_folder, 'val', subfolder)
    test_dir = os.path.join(root_folder, 'test', subfolder)
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)
```

```
    for image in train_images:
        src = os.path.join(subfolder_path, image)
        dst = os.path.join(train_dir, image)
        shutil.move(src, dst)
```

```
    for image in val_images:
        src = os.path.join(subfolder_path, image)
        dst = os.path.join(val_dir, image)
```

```

shutil.move(src, dst)

for image in test_images:
    src = os.path.join(subfolder_path, image)
    dst = os.path.join(test_dir, image)
    shutil.move(src, dst)

os.rmdir(subfolder_path)

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Preprocessing the Training dataset
train_datagen = ImageDataGenerator(rescale=1./255)
training_set = train_datagen.flow_from_directory('/content/images/train',
                                                target_size = (64, 64),
                                                batch_size = 32,
                                                class_mode = 'binary')

Found 2800 images belonging to 8 classes.

# Preprocessing the Test set
test_datagen = ImageDataGenerator(rescale=1./255)
test_set = test_datagen.flow_from_directory('/content/images/test',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')

Found 600 images belonging to 8 classes.

from tensorflow.keras.layers import Flatten
from keras.layers.pooling.max_pooling2d import MaxPool2D
from tensorflow.keras.layers import LeakyReLU
import tensorflow as tf
from keras.api._v2.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.regularizers import l2

cnn = tf.keras.models.Sequential()

# Convolution
cnn.add(tf.keras.layers.Conv2D(filters=32,padding="same",kernel_size=3, activation='relu', strides=2, input_shape=[64, 64, 3]))
cnn.add(LeakyReLU(alpha=0.2))

cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

cnn.add(Conv2D(filters=64, kernel_size=3, padding='same', strides=1))
cnn.add(LeakyReLU(alpha=0.2))
cnn.add(MaxPool2D(pool_size=2, strides=2))

cnn.add(Conv2D(filters=128, kernel_size=3, padding='same', strides=1))
cnn.add(LeakyReLU(alpha=0.2))
cnn.add(MaxPool2D(pool_size=2, strides=2))

cnn.add(Flatten())

cnn.add(Dense(units=256))
cnn.add(LeakyReLU(alpha=0.2))
cnn.add(BatchNormalization())

cnn.add(Dense(units=1, activation='linear', kernel_regularizer=l2(0.01)))

cnn.add(Dense(4, kernel_regularizer=tf.keras.regularizers.l2(0.01),activation
            ='softmax'))
cnn.compile(optimizer = 'adam', loss = 'squared_hinge', metrics = ['accuracy'])
cnn.summary()

```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 32, 32, 32)	896

leaky_re_lu_2 (LeakyReLU)	(None, 32, 32, 32)	0
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_5 (Conv2D)	(None, 16, 16, 64)	18496
leaky_re_lu_3 (LeakyReLU)	(None, 16, 16, 64)	0
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_6 (Conv2D)	(None, 8, 8, 128)	73856
leaky_re_lu_4 (LeakyReLU)	(None, 8, 8, 128)	0
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_2 (Dense)	(None, 256)	524544
leaky_re_lu_5 (LeakyReLU)	(None, 256)	0
batch_normalization (BatchNormalization)	(None, 256)	1024
dense_3 (Dense)	(None, 1)	257
dense_4 (Dense)	(None, 4)	8
=====		
Total params: 619,081		
Trainable params: 618,569		
Non-trainable params: 512		

```
from tensorflow.keras.optimizers import Adam
learning_rate = 0.1
optimizer = Adam(learning_rate=learning_rate)
# Compiling the CNN
cnn.compile(optimizer = optimizer, loss = 'hinge', metrics = ['accuracy'])

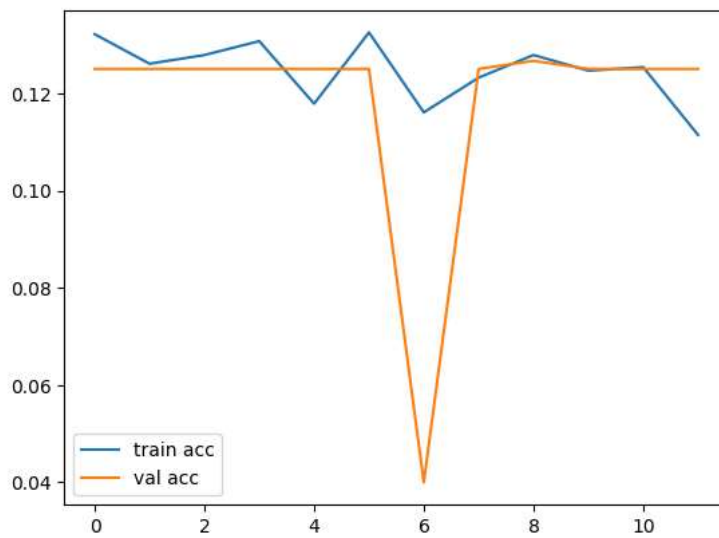
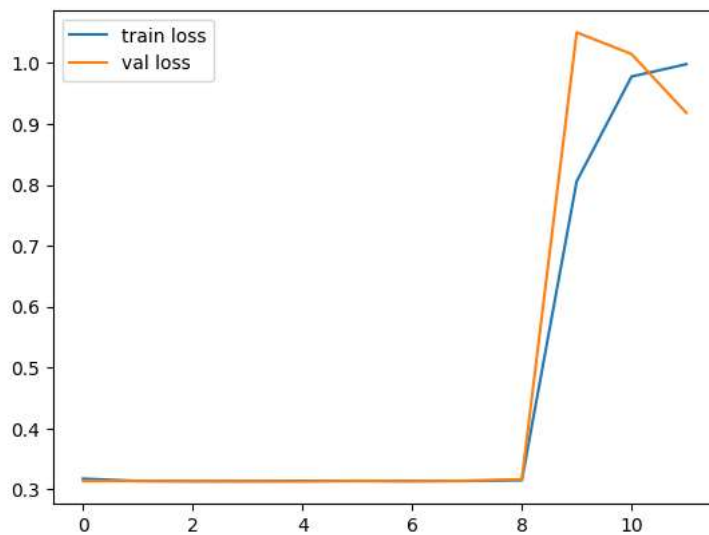
# Training the CNN on the Training set and evaluating it on the Test set
r=cnn.fit(x = training_set, validation_data = test_set, epochs = 12)

Epoch 1/12
88/88 [=====] - 17s 166ms/step - loss: 0.3174 - accuracy: 0.1321 - val_loss: 0.3135 - val_accuracy: 0.1250
Epoch 2/12
88/88 [=====] - 14s 162ms/step - loss: 0.3136 - accuracy: 0.1261 - val_loss: 0.3137 - val_accuracy: 0.1250
Epoch 3/12
88/88 [=====] - 18s 200ms/step - loss: 0.3134 - accuracy: 0.1279 - val_loss: 0.3135 - val_accuracy: 0.1250
Epoch 4/12
88/88 [=====] - 16s 176ms/step - loss: 0.3134 - accuracy: 0.1307 - val_loss: 0.3133 - val_accuracy: 0.1250
Epoch 5/12
88/88 [=====] - 15s 175ms/step - loss: 0.3138 - accuracy: 0.1179 - val_loss: 0.3128 - val_accuracy: 0.1250
Epoch 6/12
88/88 [=====] - 15s 166ms/step - loss: 0.3136 - accuracy: 0.1325 - val_loss: 0.3140 - val_accuracy: 0.1250
Epoch 7/12
88/88 [=====] - 14s 161ms/step - loss: 0.3135 - accuracy: 0.1161 - val_loss: 0.3133 - val_accuracy: 0.0400
Epoch 8/12
88/88 [=====] - 15s 175ms/step - loss: 0.3137 - accuracy: 0.1232 - val_loss: 0.3140 - val_accuracy: 0.1250
Epoch 9/12
88/88 [=====] - 14s 161ms/step - loss: 0.3148 - accuracy: 0.1279 - val_loss: 0.3164 - val_accuracy: 0.1267
Epoch 10/12
88/88 [=====] - 15s 175ms/step - loss: 0.8059 - accuracy: 0.1246 - val_loss: 1.0505 - val_accuracy: 0.1250
Epoch 11/12
88/88 [=====] - 15s 175ms/step - loss: 0.9780 - accuracy: 0.1254 - val_loss: 1.0148 - val_accuracy: 0.1250
Epoch 12/12
88/88 [=====] - 17s 186ms/step - loss: 0.9982 - accuracy: 0.1114 - val_loss: 0.9182 - val_accuracy: 0.1250

import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plotting the model accuracy
```

```
plt.plot(r.history['accuracy'], label='train acc')  
plt.plot(r.history['val_accuracy'], label='val acc')  
plt.legend()  
plt.show()  
plt.savefig('AccVal_acc')
```



<Figure size 640x480 with 0 Axes>

✓ 0s completed at 3:58 PM

● ✕