```
!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/python3.9/dist-packages (1.5.13)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/dist-packages (from kaggle) (1.26.15
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from kaggle) (2.27.1
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.9/dist-packages (from kaggle) (1.16.
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.9/dist-packages (from kaggle)
Requirement already satisfied: certifi in /usr/local/lib/python3.9/dist-packages (from kaggle) (2022.12
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from kaggle) (4.65.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.9/dist-packages (from kaggle) (
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.9/dist-packages (from pyth
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->k
```

```
!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
Downloading mango-leaf-disease-dataset.zip to /content
 90% 93.0M/103M [00:00<00:00, 213MB/s]
100% 103M/103M [00:00<00:00, 173MB/s]
```

```
from zipfile import ZipFile

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

 #extract in different directory
 f.extractall('images')
```

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import glob
import cv2
import os

data_dir = '../content/images'
print(os.listdir(data_dir))
```

```
['test', 'train', 'Anthracnose', 'Powdery Mildew', 'Cutting Weevil', 'val', 'Healthy', 'Gall Midge', 'B
```

```
import os
import shutil
import random

root_folder = "/content/images"  # Replace with the actual path to your images folder
train_ratio = 0.7  # Percentage of images for the training set
```

```
val_ratio = 0.15  # Percentage of images for the validation set
test_ratio = 0.15  # Percentage of images for the test set

# Get the 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()]

    # Shuffle the images randomly
    random.shuffle(images)

    # Calculate the number of images for each set
    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

    # Split the images into train, validation, and test sets
    train_images = images[:num_train]
    val_images = images[num_train:num_train + num_val]
    test_images = images[num_train + num_val:]

    # Create directories for train, validation, and test sets
    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)

    # Move images to their respective directories
    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)
```

```
import os

subfolder_path = "/content/images"  # Replace with the actual path to your folder
valid_extensions = (".jpg", ".jpeg", ".png", ".gif")  # Add any other valid image extensions

def count_images(folder):
    image_count = 0
    for root, dirs, files in os.walk(folder):
        for file_name in files:
            if file_name.lower().endswith(valid_extensions):
                image_count += 1
    return image_count

total_image_count = count_images(subfolder_path)

print(f"Total number of images in the folder and its subfolders: {total_image_count}")
```

```
Total number of images in the folder and its subfolders: 4000
```

```
import glob
from PIL import Image

# Specify the directory path where the images are located
directory_path = '/content/images/train/Die Back'

# Search for image files in the directory
image_files = glob.glob(directory_path + '/*.jpg')  # Modify the file extension if necessary

if len(image_files) > 0:
    # Open the first image
    first_image = Image.open(image_files[0])

    # Perform operations on the first image
    # For example, you can display the image
    first_image.show()
else:
    print("No image files found in the directory.")
```

```
# Creating the Pathlib PATH objects
train_path = Path("/content/images/train")
valid_path = Path("/content/images/val")
```

```
batch_size = 72
epochs = 45
img_channel = 9
img_width, img_height = (45,45)
train_dataset_main = data_dir + "/train"
valid_dataset_main = data_dir + "/val"
```

```
def create_dataset_df(main_path, dataset_name):
    print(f"{dataset_name} is creating ...")
    df = {"img_path":[],"class_names":[]}
    for class_names in os.listdir(main_path):
            for img_path in glob.glob(f"{main_path}/{class_names}/*"):
                df["img_path"].append(img_path)
                df["class_names"].append(class_names)
    df = pd.DataFrame(df)
    print(f"{dataset_name} is created !")
    return df



train_df = create_dataset_df(train_dataset_main, "Train dataset")

valid_df=create_dataset_df(valid_dataset_main, "Validation dataset")

print(f"train samples: {len(train_df)} \n validation samples: {len(valid_df)}")
def vizualizing_images(df,n_rows,n_cols):
    plt.figure(figsize=(10,10))
    for i in range(n_rows*n_cols):
        index = np.random.randint(0, len(df))
        img = cv2.imread(df.img_path[index])
        class_nm = df.class_names[index]
        plt.subplot(n_rows, n_cols, i+1)
        plt.imshow(img)
        plt.title(class_nm)
    plt.show()
vizualizing_images(train_df, 3, 3)

plt.figure(figsize=(25,5))
# train dataset
plt.subplot(1,2,1)
sns.countplot(data=train_df.sort_values("class_names"),x="class_names")
plt.title("Train dataset")
plt.xticks(rotation = 60)
# validation dataset
plt.subplot(1,2,2)
sns.countplot(data=valid_df.sort_values("class_names"),x="class_names")
plt.title("Validation dataset")
plt.xticks(rotation = 60)

plt.show()

from sklearn.preprocessing import LabelEncoder

Le = LabelEncoder()
train_df["class_names"] = Le.fit_transform(train_df["class_names"])

#train_df["class_names"].value_counts()


valid_df["class_names"] = Le.transform(valid_df["class_names"])
#One Hot encoding
train_labels = tf.keras.utils.to_categorical(train_df["class_names"])
valid_labels = tf.keras.utils.to_categorical(valid_df["class_names"])
train_labels[:10]
train_labels.sum(axis=0)

# Compute class weights

classTotals = train_labels.sum(axis=0)
classWeight = classTotals.max() / classTotals
```

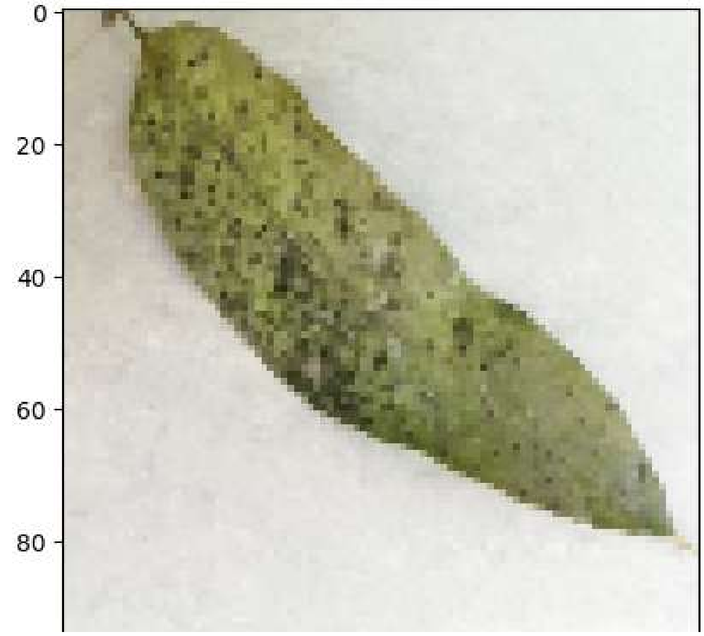
```
class_weight = {e : weight for e , weight in enumerate(classWeight)}
print(class_weight)
input_image = cv2.imread(train_df.img_path[0])

input_image.shape
```

```
Train dataset is creating ...
Train dataset is created !
Validation dataset is creating ...
Validation dataset is created !
train samples: 2800
 validation samples: 600
```

```
# Function used for Transformation

def load(image , label):
    image = tf.io.read_file(image)
    image = tf.io.decode_jpeg(image , channels = 3)
    return image , label

# Define IMAGE SIZE and BATCH SIZE
IMG_SIZE = 96
BATCH_SIZE = 64

# Basic Transformation
resize = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.Resizing(IMG_SIZE, IMG_SIZE)
])

# Data Augmentation
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),
    tf.keras.layers.experimental.preprocessing.RandomZoom(height_factor = (-0.1, -0.05))
])

# Function used to Create a Tensorflow Data Object
AUTOTUNE = tf.data.experimental.AUTOTUNE #to find a good allocation of its CPU budget across all parameters
def get_dataset(paths , labels , train = True):
    image_paths = tf.convert_to_tensor(paths)
    labels = tf.convert_to_tensor(labels)

    image_dataset = tf.data.Dataset.from_tensor_slices(image_paths)
    label_dataset = tf.data.Dataset.from_tensor_slices(labels)

    dataset = tf.data.Dataset.zip((image_dataset , label_dataset))

    dataset = dataset.map(lambda image , label : load(image , label))
    dataset = dataset.map(lambda image, label: (resize(image), label) , num_parallel_calls=AUTOTUNE)
    dataset = dataset.shuffle(1000)
    dataset = dataset.batch(BATCH_SIZE)

    if train:
        dataset = dataset.map(lambda image, label: (data_augmentation(image), label) , num_parallel_calls=AUT
        dataset = dataset.repeat()

    return dataset

# Creating Train Dataset object and Verifying it
%time
train_dataset = get_dataset(train_df["img_path"], train_labels)
```

```
    train_dataset = get_dataset(train_df["img_path"], train_labels)

    #iter() returns an iterator of the given object
    #next() returns the next number in an iterator
    image , label = next(iter(train_dataset))
    print(image.shape)
    print(label.shape)
    # View a sample Training Image
    print(Le.inverse_transform(np.argmax(label , axis = 1))[0])
    plt.imshow((image[0].numpy()/255).reshape(96 , 96 , 3))


    %time
    val_dataset = get_dataset(valid_df["img_path"] , valid_labels , train = False)

    image , label = next(iter(val_dataset))
    print(image.shape)
    print(label.shape)

    # View a sample Validation Image
    print(Le.inverse_transform(np.argmax(label , axis = 1))[0])
    plt.imshow((image[0].numpy()/255).reshape(96 , 96 , 3))


    # Building EfficientNet model
    from tensorflow.keras.applications import EfficientNetB2
    from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPooling2D, Dropout, Dense, Inp


    backbone = EfficientNetB2(
        input_shape=(96, 96, 3),
        include_top=False
    )

    n = 64
    model = tf.keras.Sequential([
        backbone,
        tf.keras.layers.Conv2D(128, 3, padding='same'),
        tf.keras.layers.LeakyReLU(alpha=0.2),
        tf.keras.layers.GlobalAveragePooling2D(),
        tf.keras.layers.Dense(128),
        tf.keras.layers.LeakyReLU(alpha=0.2),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Dense(8, activation='softmax')
    ])

    model.summary()

    # Compiling your model by providing the Optimizer , Loss and Metrics
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07),
        loss = 'categorical_crossentropy',
        metrics=['accuracy' , tf.keras.metrics.Precision(name='precision'),tf.keras.metrics.Recall(name='recall')
    )
    #
    len(train_labels),len(valid_labels)
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.58 µs
(64, 96, 96, 3)
(64, 8)
Anthracnose
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 5.72 µs
(64, 96, 96, 3)
(64, 8)
Sooty Mould
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb2 (Functional)  (None, 3, 3, 1408)       7768569

 conv2d_1 (Conv2D)           (None, 3, 3, 128)         1622144

 leaky_re_lu_2 (LeakyReLU)   (None, 3, 3, 128)         0

 global_average_pooling2d_1   (None, 128)              0
 (GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 128)               16512

 leaky_re_lu_3 (LeakyReLU)   (None, 128)               0

 dropout_1 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 8)                 1032

=================================================================
Total params: 9,408,257
Trainable params: 9,340,682
Non-trainable params: 67,575
_________________________________________________________________
(2800, 600)
```

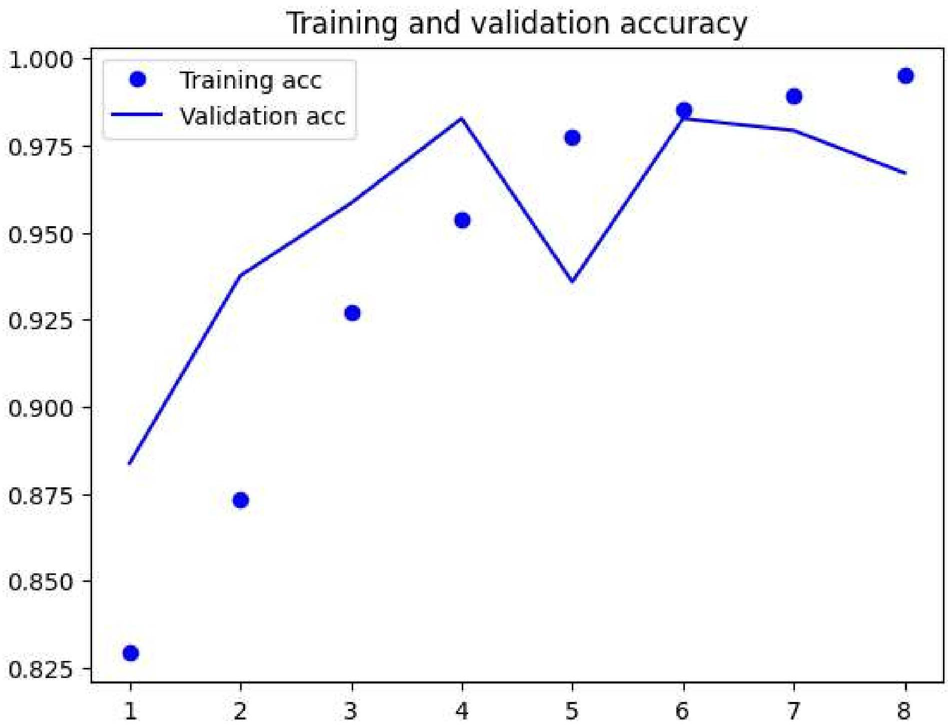


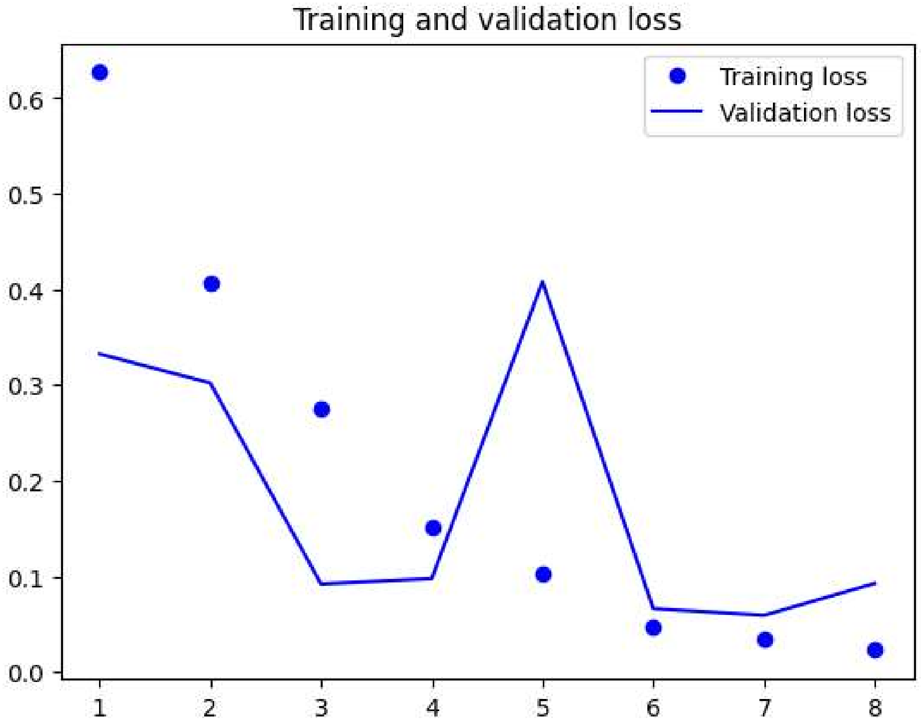
```
early_stopping=tf.keras.callbacks.EarlyStopping(monitor="accuracy",patience=2,mode="auto")
# Train the model
history = model.fit(
    train_dataset,
    steps_per_epoch=len(train_labels)//BATCH_SIZE,
    epochs=12,
        callbacks=[early_stopping],
        validation_data=val_dataset,
        validation_steps = len(valid_labels)//BATCH_SIZE,
        class_weight=class_weight
    )
    model.layers[0].trainable = False
    # Defining our callbacks
    checkpoint = tf.keras.callbacks.ModelCheckpoint("best_weights.h5",verbose=1,save_best_only=True,save_weights_
    early_stop = tf.keras.callbacks.EarlyStopping(monitor="accuracy",patience=2)
    model.summary()


    # 2nd Train the model
    history = model.fit(
        train_dataset,
        steps_per_epoch=len(train_labels)//BATCH_SIZE,
        epochs=8,
        callbacks=[checkpoint , early_stop],
        validation_data=val_dataset,
        validation_steps = len(valid_labels)//BATCH_SIZE,
        class_weight=class_weight
    )
    # Save Model
    #model.save("FacialExpressionModel.h5")
    # Save Label Encoder
    ##def save_object(obj , name):
      ###pickle_obj.close()
    #save_object(Le, "LabelEncoder")
```

```
 _________________________________________________________________
  Layer (type)                Output Shape              Param #
 =================================================================
  efficientnetb2 (Functional)  (None, 3, 3, 1408)       7768569

  conv2d_1 (Conv2D)           (None, 3, 3, 128)         1622144

  leaky_re_lu_2 (LeakyReLU)   (None, 3, 3, 128)         0

  global_average_pooling2d_1   (None, 128)              0
  (GlobalAveragePooling2D)

  dense_2 (Dense)             (None, 128)               16512

  leaky_re_lu_3 (LeakyReLU)   (None, 128)               0

  dropout_1 (Dropout)         (None, 128)               0

  dense_3 (Dense)             (None, 8)                 1032

 =================================================================
 Total params: 9,408,257
 Trainable params: 1,639,688
 Non-trainable params: 7,768,569
```

```
43/43 [==============================] - ETA: 0s - loss: 0.2755 - accuracy: 0.9273 - precision: 0.93
Epoch 3: val_loss improved from 0.30212 to 0.09194, saving model to best_weights.h5
43/43 [==============================] - 302s 7s/step - loss: 0.2755 - accuracy: 0.9273 - precision:
Epoch 4/8
43/43 [==============================] - ETA: 0s - loss: 0.1511 - accuracy: 0.9539 - precision: 0.95
Epoch 4: val_loss did not improve from 0.09194
43/43 [==============================] - 301s 7s/step - loss: 0.1511 - accuracy: 0.9539 - precision:
Epoch 5/8
43/43 [==============================] - ETA: 0s - loss: 0.1036 - accuracy: 0.9773 - precision: 0.97
Epoch 5: val_loss did not improve from 0.09194
43/43 [==============================] - 303s 7s/step - loss: 0.1036 - accuracy: 0.9773 - precision:
Epoch 6/8
43/43 [==============================] - ETA: 0s - loss: 0.0478 - accuracy: 0.9850 - precision: 0.98
Epoch 6: val_loss improved from 0.09194 to 0.06618, saving model to best_weights.h5
43/43 [==============================] - 303s 7s/step - loss: 0.0478 - accuracy: 0.9850 - precision:
Epoch 7/8
43/43 [==============================] - ETA: 0s - loss: 0.0344 - accuracy: 0.9890 - precision: 0.99
Epoch 7: val_loss improved from 0.06618 to 0.05915, saving model to best_weights.h5
43/43 [==============================] - 299s 7s/step - loss: 0.0344 - accuracy: 0.9890 - precision:
Epoch 8/8
43/43 [==============================] - ETA: 0s - loss: 0.0228 - accuracy: 0.9949 - precision: 0.99
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

import matplotlib.pyplot as plt

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





✓ 0s completed at 2:11 AM