```
# Mount to Google Drive

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

```
Mounted at /content/drive
```

```
# Navigate to the necessary folder

%cd '/content/drive/My Drive/Data 255 Spring 2024/Google Colab/Homework3/Grapevine_Leaves_Image_Dataset/'
```

```
/content/drive/My Drive/Data 255 Spring 2024/Google Colab/Homework3/Grapevine_Leaves_Image_Dataset
```

```
ls
```

```
Ak/  Ala_Idris/  Buzgulu/  Dimnit/  Grapevine_Leaves_Image_Dataset_Citation_Request.txt  Nazli/
```

Import TensorFlow and other libraries

```
# Import necessary libraries
import matplotlib.pyplot as plt
import numpy as np
import os # import os for various files and directory related operations
import PIL # import python imaging library
import tensorflow as tf
import pathlib # interact with file paths and file systems
# keras is a high-level neural networks API for
# training, building and deploying deep learning models
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Download and explore the dataset

About the dataset: The grapevine leaf dataset consists of healthy leaves and unhealthy leaves affected by the Esca disease. There are 500 images within dataset. The dataset has the following folders:

/Ak

/Nazli

/Dimnit

/Buzgulu

/Ala_Idris

The dataset is downloaded from the https://www.muratkoklu.com/en/publications/ website.

```
data_dir = pathlib.Path("/content/drive/My Drive/Data 255 Spring 2024/Google Colab/Homework3/Grapevine_Leaves_Image_Dataset/")
# Get a list of all items (files and directories) in the directory
all_items = os.listdir(data_dir)

# Filter out only the directories
folders = [item for item in all_items if os.path.isdir(os.path.join(data_dir, item))]

# Print the list of folders
print("Folders in the directory:")
for folder in folders:
    print(folder)
```

```
Folders in the directory:
Ak
Nazli
Dimnit
Buzgulu
Ala_Idris
```

After downloading the dataset there are 500 images within dataset.

```
image_count = len(list(data_dir.glob('*/*.png')))
print(image_count)
```

```
500
```

Here are some leaves.

```
# View one leaf
ak_grape = list(data_dir.glob('Ak/*'))
PIL.Image.open(str(ak_grape[0]))
```

```
# View second leaf
ak_grape = list(data_dir.glob('Ak/*'))
PIL.Image.open(str(ak_grape[1]))
```

Load using keras.preprocessing

Load the images using off the disk using "image_dataset_from_directory" utility.

Create a dataset

Define some parameters for the loader:

```
batch_size = 32
img_height = 511
img_width = 511
```

Let split dataset. 80% for training and 20% for validation.

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

```
Found 500 files belonging to 5 classes.
Using 400 files for training.
```

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

```
Found 500 files belonging to 5 classes.
Using 100 files for validation.
```

You can find the class names in the class_names attributes in dataset. These correspond to the directory names in alphabetical order.

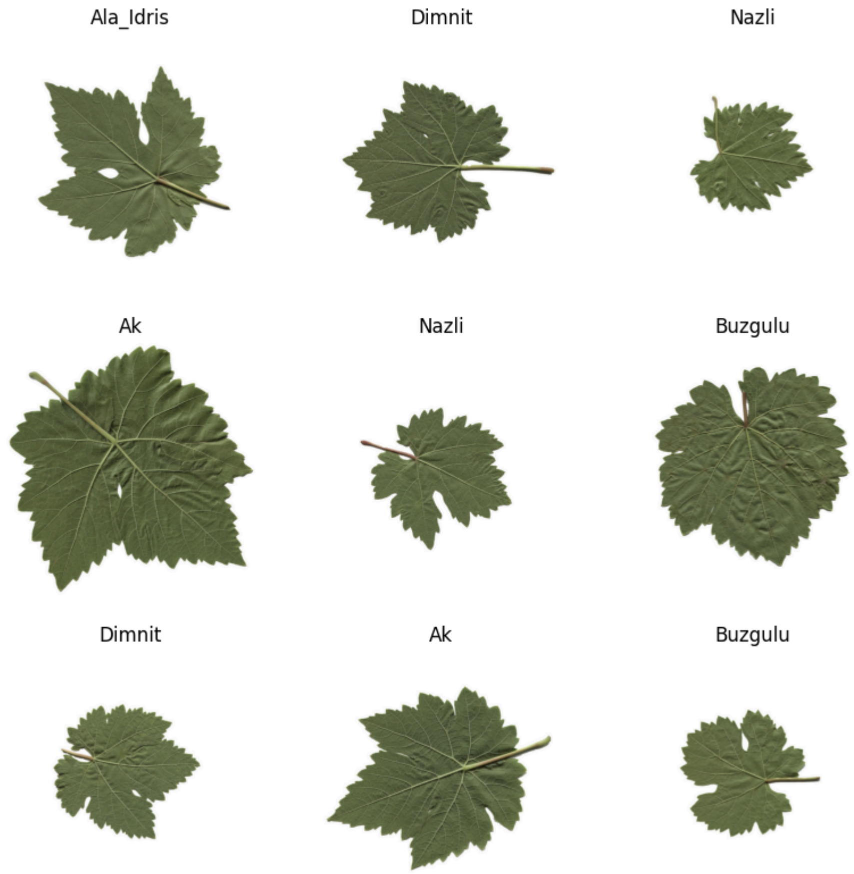
```
class_names = train_ds.class_names
print(class_names)
```

```
['Ak', 'Ala_Idris', 'Buzgulu', 'Dimnit', 'Nazli']
```

Visualize the data

Here are first 9 images from the training dataset.

```
plt.figure(figsize=(10,10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i+1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



Manually iterate over the dataset.

```
for image_batch, label_batch in train_ds:
  print(image_batch.shape)
  print(label_batch.shape)
  break
```

```
(32, 511, 511, 3)
(32,)
```

The image_batch is a tensor of shape (32, 180, 180, 3). This is a batch of 32 images of shape (180, 180, 3) (the last dimension refers to the color channels RGB). The label_batch is a tensor of shape (32,) that corresponds to the 32 images. You call the .numpy() on image_batch and label_batch tensors to conver them to numpy.narray.

Configure the dataset for performance

Use the buffered prefetching so you can yield data from disk with having I/O becoming blocking/issues.Two important methods to use:

Dataset.cache(): method keeps the training data in memory after loaded off disk after the first epoch. This will ensure dataset doesnot become bottleneck while training model.

Dataset.prefetch(): overlaps data preprocessing and model execution while training.

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Standardize the data

The RGB channels values are from [0, 255] range. This is not ideal for neural network, should have small input. Standarzie values to be in [0, 1] range using Rescaling layer.

```
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.078431375 1.0
```

Create the model

The model consists of three convolution layers with a max pool layer in each of them. There is a fully connected layer with 128 neurons on top of it that is activated by the relu activation function.

```
num_classes = 5

model = Sequential([
    # Rescale pixel values between 0 and 1
    layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    # Define a convolution with 16 filters and 3X3 kernel
    #layers.Conv2D(16, 3, padding='same', activation='relu'),
    # Max pooling layer to reduce spatial dimension
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    # Flatten layer to convert the #D output to 1D output for fully connected layer
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])
```

Compile model

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Model Summary

View all the layers of the network using model's summary

```
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 rescaling_10 (Rescaling)    (None, 511, 511, 3)       0         
                                                                 
 conv2d_21 (Conv2D)          (None, 511, 511, 16)      448       
                                                                 
 max_pooling2d_19 (MaxPooli  (None, 255, 255, 16)      0         
 ng2D)                                                           
                                                                 
 conv2d_22 (Conv2D)          (None, 255, 255, 32)      4640      
                                                                 
 max_pooling2d_20 (MaxPooli  (None, 127, 127, 32)      0         
 ng2D)                                                           
                                                                 
 conv2d_23 (Conv2D)          (None, 127, 127, 64)      18496     
                                                                 
 max_pooling2d_21 (MaxPooli  (None, 63, 63, 64)        0         
 ng2D)                                                           
```

```
 flatten_7 (Flatten)         (None, 254016)            0

 dense_18 (Dense)            (None, 128)               32514176

 dense_19 (Dense)            (None, 5)                 645

=================================================================
Total params: 32538405 (124.12 MB)
Trainable params: 32538405 (124.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Train the model

```
epochs = 6

history = model.fit(
    train_ds,
    validation_data = val_ds,
    epochs = epochs
)
```

```
Epoch 1/6
13/13 [==============================] - 140s 11s/step - loss: 6.0466 - accuracy: 0.1725 - val_loss: 1.6031 - val_accuracy:
Epoch 2/6
13/13 [==============================] - 119s 9s/step - loss: 1.6063 - accuracy: 0.2600 - val_loss: 1.5971 - val_accuracy: 0
Epoch 3/6
13/13 [==============================] - 120s 9s/step - loss: 1.5297 - accuracy: 0.3675 - val_loss: 1.5316 - val_accuracy: 0
Epoch 4/6
13/13 [==============================] - 118s 9s/step - loss: 1.3336 - accuracy: 0.4625 - val_loss: 1.5693 - val_accuracy: 0
Epoch 5/6
13/13 [==============================] - 118s 9s/step - loss: 0.9596 - accuracy: 0.7050 - val_loss: 1.5566 - val_accuracy: 0
Epoch 6/6
13/13 [==============================] - 120s 9s/step - loss: 0.6087 - accuracy: 0.8125 - val_loss: 1.9500 - val_accuracy: 0
```

Visualize the results

Create plots for accraucy on training and validation sets

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

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```


```
predictions = model.predict(val_ds)

predicted_categories = [np.argmax(p) for p in predictions]

print(predicted_categories)

test_loss, test_acc = model.evaluate(val_ds)
print('Test accuracy:', test_acc)
```

As you can see from the plots, the training accuracy and validation accuracy are off by a large margin, and the model has achieved only around 40% accuracy on the validation set.

This appears to be a case of overfitting, wherein the data is able to predict with good accuracy on the training data, but not with good accuracy on unseen or validation data.

Let us try to increase the overall performance. Use "data augmentation" and "dropout" techniques.

Overfitting

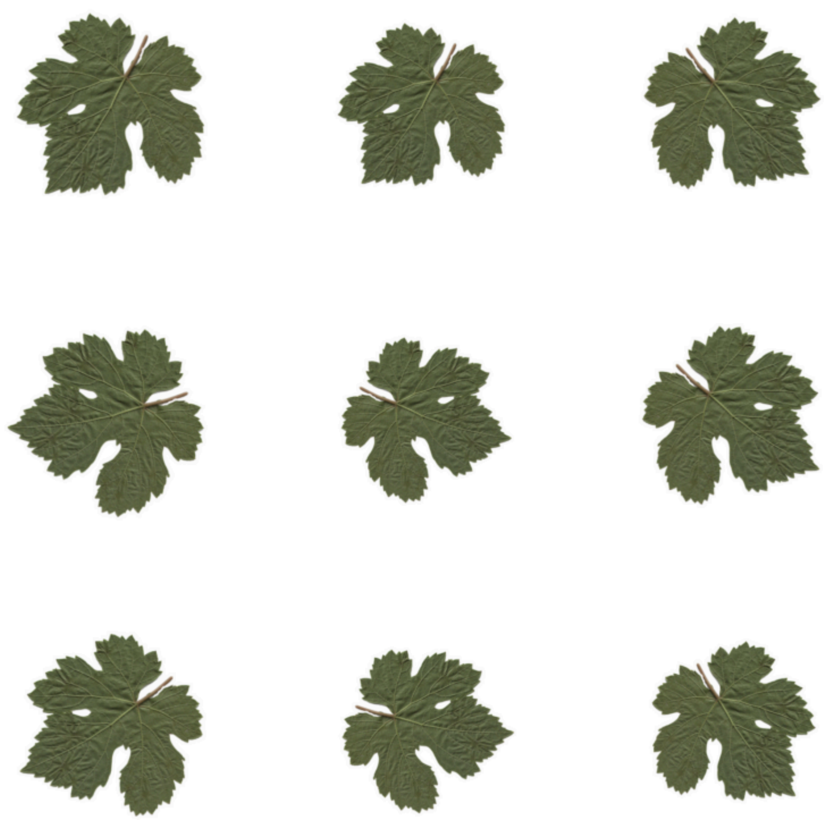
Data augmentation

Overfitting occurs when the model is able to generalize on the training model but not able to generalize on validation data (unseen data). Data augmentation generates additional data from the training sample, by augmenting them using random tranformations that yield belivable looking images. This helps expose the model to more aspects of the data and able to generalize better.

```
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

Lets visualize few imahes with augmented data

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```



Dropout

To help reduce overfitting using dropout, a form of regularization.

When you set 'dropout' to a layer, it randomly drops out (by setting the activation function to 0) a number of outputs units from the layer during the training process. The dropout technique is applied only to the training set. Dropout takes fractional numbers as input such as 0.1, 0.2. This means 10%, 20% of the output units will be randomly set to 0.

```
model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Compile the model

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_10 (Sequential)  (None, 511, 511, 3)       0

 rescaling_11 (Rescaling)    (None, 511, 511, 3)       0

 conv2d_24 (Conv2D)          (None, 511, 511, 16)      448

 max_pooling2d_22 (MaxPooli  (None, 255, 255, 16)      0
 ng2D)

 conv2d_25 (Conv2D)          (None, 255, 255, 32)      4640

 max_pooling2d_23 (MaxPooli  (None, 127, 127, 32)      0
 ng2D)

 conv2d_26 (Conv2D)          (None, 127, 127, 64)      18496

 max_pooling2d_24 (MaxPooli  (None, 63, 63, 64)        0
 ng2D)

 dropout_2 (Dropout)         (None, 63, 63, 64)        0

 flatten_8 (Flatten)         (None, 254016)            0

 dense_20 (Dense)            (None, 128)               32514176

 dense_21 (Dense)            (None, 5)                 645

=================================================================
Total params: 32538405 (124.12 MB)
Trainable params: 32538405 (124.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/15
13/13 [==============================] - 142s 11s/step - loss: 4.3783 - accuracy: 0.1925 - val_loss: 1.6082 - val_accuracy:
Epoch 2/15
13/13 [==============================] - 136s 11s/step - loss: 1.6095 - accuracy: 0.1925 - val_loss: 1.6087 - val_accuracy:
Epoch 3/15
13/13 [==============================] - 139s 11s/step - loss: 1.6026 - accuracy: 0.2800 - val_loss: 1.6063 - val_accuracy:
Epoch 4/15
13/13 [==============================] - 139s 11s/step - loss: 1.5988 - accuracy: 0.2700 - val_loss: 1.6178 - val_accuracy:
Epoch 5/15
13/13 [==============================] - 138s 11s/step - loss: 1.5739 - accuracy: 0.2750 - val_loss: 1.6103 - val_accuracy:
Epoch 6/15
```

```
13/13 [==============================] - 137s 11s/step - loss: 1.5540 - accuracy: 0.3225 - val_loss: 1.5662 - val_accuracy:
Epoch 7/15
13/13 [==============================] - 139s 11s/step - loss: 1.5448 - accuracy: 0.3075 - val_loss: 1.5802 - val_accuracy:
Epoch 8/15
13/13 [==============================] - 145s 11s/step - loss: 1.5147 - accuracy: 0.3400 - val_loss: 1.5195 - val_accuracy:
Epoch 9/15
13/13 [==============================] - 139s 11s/step - loss: 1.5055 - accuracy: 0.3575 - val_loss: 1.6798 - val_accuracy:
Epoch 10/15
13/13 [==============================] - 138s 11s/step - loss: 1.4731 - accuracy: 0.3850 - val_loss: 1.4842 - val_accuracy:
Epoch 11/15
13/13 [==============================] - 138s 11s/step - loss: 1.4130 - accuracy: 0.4175 - val_loss: 1.7418 - val_accuracy:
Epoch 12/15
13/13 [==============================] - 145s 11s/step - loss: 1.3798 - accuracy: 0.4000 - val_loss: 1.3421 - val_accuracy:
Epoch 13/15
13/13 [==============================] - 139s 11s/step - loss: 1.3861 - accuracy: 0.4300 - val_loss: 1.5959 - val_accuracy:
Epoch 14/15
13/13 [==============================] - 139s 11s/step - loss: 1.2702 - accuracy: 0.5025 - val_loss: 1.4498 - val_accuracy:
Epoch 15/15
13/13 [==============================] - 138s 11s/step - loss: 1.3020 - accuracy: 0.4650 - val_loss: 1.7912 - val_accuracy:
```

Visualize training results

After applying data augmentation and Dropout, there is less overfitting than before, and training and validation accuracy are closer aligned.

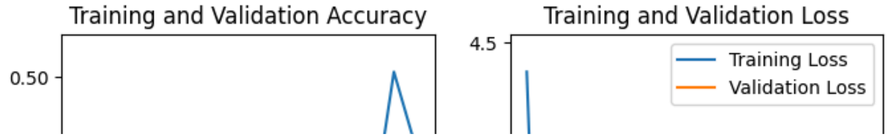
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```


Training and Validation Accuracy
0.50
Training and Validation Loss
4.5
Training Loss
Validation Loss