Potato Leaf Classification using CNN for Large Pakistan Potato leaf dataset

Inspiration from: https://www.youtube.com/watch?v=dGtDTjYs3xc&list=PLeo1K3hjS3ut49PskOfLnE6WUoOp_2lsD&index=1

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

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

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
# Constants
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3 #RGB
EPOCHS = 50
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/Colab Notebooks/Dissertation/Code/PLD_larger_dataset",
    shuffle = True,
    image_size = (IMAGE_SIZE, IMAGE_SIZE),
    batch_size = BATCH_SIZE
)
```

```
Found 4072 files belonging to 3 classes.
```

```
class_names = dataset.class_names
class_names # 0 = early blight, 1 = late blight, 2 = healthy
```

```
['Potato_Early_blight', 'Potato_Late_blight', 'Potato_healthy']
```

```
len(dataset)
```

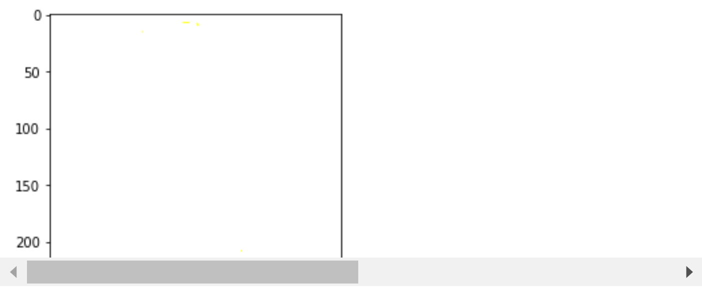
```
128
```

```
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy()) #tensor to numpy
    # 32 images in first batch, each image is (256 x 256) and RGB
```

```
(32, 256, 256, 3)
[0 1 2 1 1 2 1 0 2 0 2 2 1 1 2 2 0 1 0 1 0 0 0 1 0 2 1 0 0 1 2 0]
```

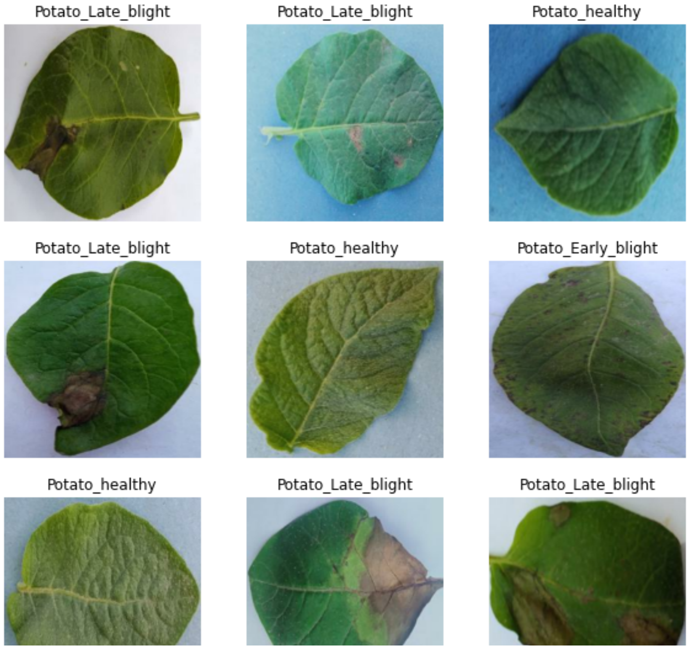
```
for image_batch, label_batch in dataset.take(1):
    plt.imshow(image_batch[0].numpy()) # bad image due to it being a float
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for
```




```
plt.figure(figsize=(10, 10))

for image_batch, label_batch in dataset.take(1):
    for i in range(9): # print first 9 images
        ax = plt.subplot(3, 3, i+1)
        plt.imshow(image_batch[i].numpy().astype("uint8")) # converted to integer, change i to 0 for 1st img
```

```
        plt.title(class_names[label_batch[i]]) # gives the label name
        plt.axis("off") #removes access
```




```
# train, validation, test split

def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)

    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

```
len(train_ds)
```

    102

```
len(val_ds)
```

    12

```
len(test_ds)
```

    14

Scale the data

```
resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)
]) # if images are not 256*256, ^^ will take care of it
```

```
# Data augmentation
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
```

```
    layers.experimental.preprocessing.RandomRotation(0.2),
])
```

CNN model

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3
model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(32, (3,3), activation ='relu', input_shape=input_shape), #layers, filter_size
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3), activation ='relu'), #layers, filter_size
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3), activation ='relu'), #layers, filter_size
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3), activation ='relu'), #layers, filter_size
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3), activation ='relu'), #layers, filter_size
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3), activation ='relu'), #layers, filter_size
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64, activation ='relu'),
    layers.Dense(n_classes, activation ='softmax'),

])

model.build(input_shape=input_shape)
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0         
                                                                 
 sequential_1 (Sequential)   (32, 256, 256, 3)         0         
                                                                 
 conv2d (Conv2D)             (32, 254, 254, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928     
                                                                 
 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928     
                                                                 
 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928     
                                                                 
 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928     
                                                                 
 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (32, 256)                 0         
                                                                 
 dense (Dense)               (32, 64)                  16448     
                                                                 
 dense_1 (Dense)             (32, 3)                   195       
                                                                 
=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

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

```
history = model.fit(
    train_ds,
    epochs = EPOCHS,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data = val_ds
)
```

```
Epoch 1/50
102/102 [==============================] - 114s 59ms/step - loss: 1.0363 - accuracy: 0.4407 - val_loss: 1.0148 - val_accuracy: 0
Epoch 2/50
102/102 [==============================] - 8s 58ms/step - loss: 0.8385 - accuracy: 0.6534 - val_loss: 0.7158 - val_accuracy: 0.6
Epoch 3/50
102/102 [==============================] - 8s 56ms/step - loss: 0.5627 - accuracy: 0.7759 - val_loss: 0.5340 - val_accuracy: 0.7
Epoch 4/50
102/102 [==============================] - 8s 56ms/step - loss: 0.4579 - accuracy: 0.8253 - val_loss: 0.6257 - val_accuracy: 0.7
Epoch 5/50
102/102 [==============================] - 8s 56ms/step - loss: 0.4045 - accuracy: 0.8434 - val_loss: 0.2663 - val_accuracy: 0.9
Epoch 6/50
102/102 [==============================] - 8s 56ms/step - loss: 0.2669 - accuracy: 0.9068 - val_loss: 0.2661 - val_accuracy: 0.9
Epoch 7/50
102/102 [==============================] - 8s 56ms/step - loss: 0.2352 - accuracy: 0.9096 - val_loss: 0.1540 - val_accuracy: 0.9
Epoch 8/50
102/102 [==============================] - 8s 56ms/step - loss: 0.2271 - accuracy: 0.9198 - val_loss: 0.2269 - val_accuracy: 0.9
Epoch 9/50
102/102 [==============================] - 8s 56ms/step - loss: 0.1822 - accuracy: 0.9380 - val_loss: 0.1607 - val_accuracy: 0.9
Epoch 10/50
102/102 [==============================] - 8s 56ms/step - loss: 0.1631 - accuracy: 0.9435 - val_loss: 0.0915 - val_accuracy: 0.9
Epoch 11/50
102/102 [==============================] - 8s 56ms/step - loss: 0.1471 - accuracy: 0.9515 - val_loss: 0.1308 - val_accuracy: 0.9
Epoch 12/50
102/102 [==============================] - 8s 57ms/step - loss: 0.1091 - accuracy: 0.9654 - val_loss: 0.1021 - val_accuracy: 0.9
Epoch 13/50
102/102 [==============================] - 8s 56ms/step - loss: 0.1201 - accuracy: 0.9599 - val_loss: 0.1212 - val_accuracy: 0.9
Epoch 14/50
102/102 [==============================] - 8s 56ms/step - loss: 0.1287 - accuracy: 0.9562 - val_loss: 0.0814 - val_accuracy: 0.9
Epoch 15/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0990 - accuracy: 0.9679 - val_loss: 0.0438 - val_accuracy: 0.9
Epoch 16/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0834 - accuracy: 0.9710 - val_loss: 0.0969 - val_accuracy: 0.9
Epoch 17/50
102/102 [==============================] - 8s 57ms/step - loss: 0.0967 - accuracy: 0.9672 - val_loss: 0.0403 - val_accuracy: 0.9
Epoch 18/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0835 - accuracy: 0.9744 - val_loss: 0.0907 - val_accuracy: 0.9
Epoch 19/50
102/102 [==============================] - 8s 57ms/step - loss: 0.0715 - accuracy: 0.9793 - val_loss: 0.0654 - val_accuracy: 0.9
Epoch 20/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0771 - accuracy: 0.9747 - val_loss: 0.0749 - val_accuracy: 0.9
Epoch 21/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0769 - accuracy: 0.9731 - val_loss: 0.1645 - val_accuracy: 0.9
Epoch 22/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0766 - accuracy: 0.9731 - val_loss: 0.0745 - val_accuracy: 0.9
Epoch 23/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0718 - accuracy: 0.9776 - val_loss: 0.0768 - val_accuracy: 0.9
Epoch 24/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0855 - accuracy: 0.9744 - val_loss: 0.0787 - val_accuracy: 0.9
Epoch 25/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0668 - accuracy: 0.9787 - val_loss: 0.0840 - val_accuracy: 0.9
Epoch 26/50
102/102 [==============================] - 8s 57ms/step - loss: 0.0626 - accuracy: 0.9759 - val_loss: 0.0196 - val_accuracy: 0.9
Epoch 27/50
102/102 [==============================] - 8s 56ms/step - loss: 0.0741 - accuracy: 0.9775 - val_loss: 0.0347 - val_accuracy: 0.9
Epoch 28/50
102/102 [==============================] - 8s 57ms/step - loss: 0.0464 - accuracy: 0.9855 - val_loss: 0.0749 - val_accuracy: 0.9
Epoch 29/50
```

```
scores = model.evaluate(test_ds)
scores # [loss, accuracy]
scores_test_acc = scores[1]*100
scores_test_loss = scores[0]
print("Test Loss: ", scores_test_loss)
print("Test Accuracy: ", scores_test_acc)
```

```
14/14 [==============================] - 2s 12ms/step - loss: 0.0471 - accuracy: 0.9911
Test Loss:  0.04708237573504448
Test Accuracy:  99.10714030265808
```

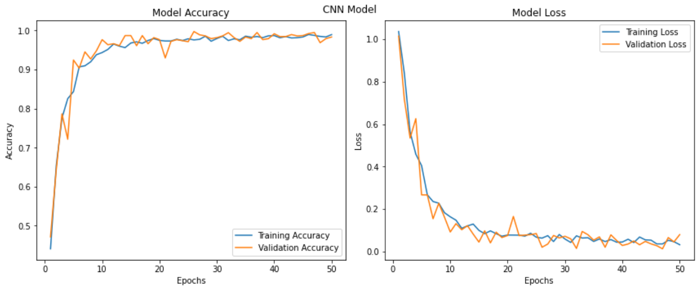
```
history
```

```
<keras.callbacks.History at 0x7fa36a472100>
```

```
CNN_acc = history.history["accuracy"]
CNN_val_acc = history.history["val_accuracy"]

CNN_loss = history.history["loss"]
CNN_val_loss = history.history["val_loss"]
n = range(1,EPOCHS + 1)
```

Plot the model accuracy and loss

```
plt.figure(figsize = (12,5))
plt.subplot(1,2,1)
plt.plot(n,CNN_acc,label= "Training Accuracy")
plt.plot(n,CNN_val_acc,label= "Validation Accuracy")
plt.legend(loc = "best")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Model Accuracy")
plt.tight_layout()


plt.subplot(1,2,2)
plt.plot(n, CNN_loss, label='Training Loss')
plt.plot(n, CNN_val_loss, label='Validation Loss')
plt.legend(loc='best')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.tight_layout()
plt.suptitle('CNN Model')
plt.savefig("CNN_model.png")
plt.show()
```

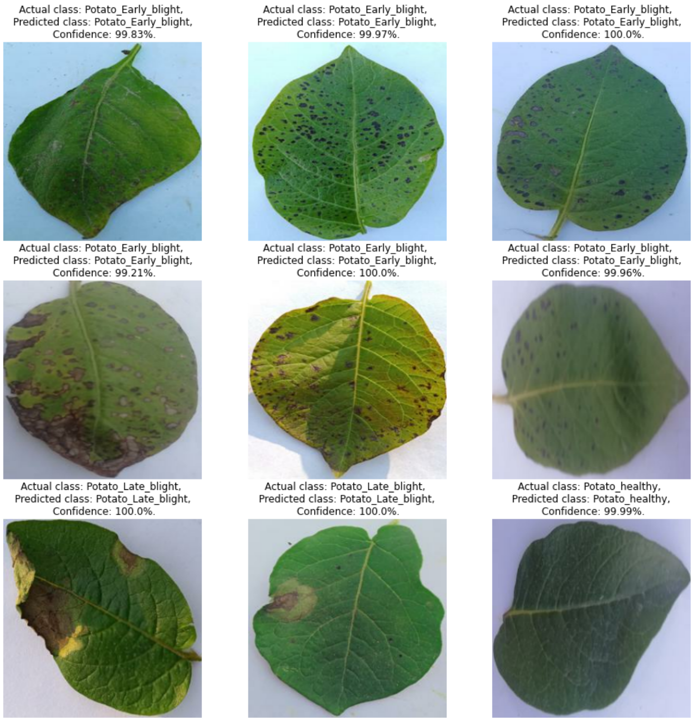


Test how well the model is at prediciting

```
def predict(model, img):
  img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy()) # convert to numpy
  img_array = tf.expand_dims(img_array, 0) # batch

  predictions = model.predict(img_array)

  predicted_class = class_names[np.argmax(predictions[0])]
  confidence = round(100 * (np.max(predictions[0])), 2)
  return predicted_class, confidence


plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i+1)
    plt.imshow(images[i].numpy().astype("uint8"))

    predicted_class, confidence = predict(model, images[i].numpy())
    actual_class = class_names[labels[i]]
```

```
    plt.title(f"Actual class: {actual_class}, \n Predicted class: {predicted_class}, \n Confidence: {confidence}%.")

    plt.axis("off")

plt.savefig("CNN_prediction images")
```

```
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 15ms/step
```



Convert .ipynb to .html

```
!jupyter nbconvert --to html Large_Pakistan_Potato_Leaf_CNN.ipynb
```

```
[NbConvertApp] WARNING | pattern 'Large_Pakistan_Potato_Leaf_CNN.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
```

```
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in the cell output (the d
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
```