Data Collection & Preprocessing

Import Essential Libraries:

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
```

Preprocessing:

Change the current working Directory:

```
import os

# Change the current working directory to the parent directory
os.chdir("..")
```

```
IMAGE_SIZE= 256
CHANNELS= 3
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "CNN Project\Dataset"
)
```

Found 2152 files belonging to 3 classes.

`shuffle=True` (By default)

`image_size=(256, 256)` (By default)

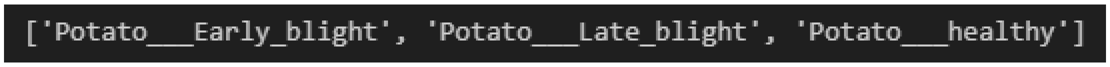
`batch_size=32` (By default)

`tf.keras.preprocessing.image_dataset_from_directory()` is a helper function that:

- Reads images from folders.
- Automatically labels them based on subfolder names.
- Returns a `tf.data.Dataset` object (ready to use in model training).

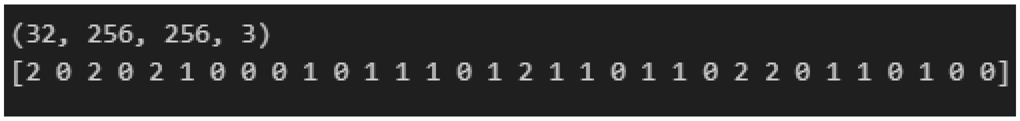
Class Names:

```
class_names= dataset.class_names
class_names
```



```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```



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

`dataset.take(1)` → Batch 1 (We have loaded images in batches)

- 0,1,2 → Labels
- 32→ Batch size
- 256,256 -Image size
- 3 → RGB Channels

Visualize the Images

```
for image_batch, labels_batch in dataset.take(1):
    plt.imshow(image_batch[0]/255)
    plt.axis('off')
```

- `/255` normalizes the image
- **Note**: Each time it will show different image because **`shuffle=True`**

Display Label:

`plt.title(class_names[labels_batch[0]])`

`class_names` → **['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']**

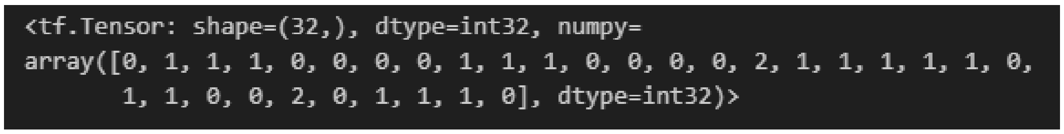
`labels_batch` → **0,1,2**

`[0]` → **Accesses the label of 0th batch**

For example:

- If `labels_batch[0] = 0`, then `class_names[0] = 'Potato___Early_blight'`, and the plot title will be `'Potato___Early_blight'`.
- If `labels_batch[0] = 1`, the title will be `'Potato___Late_blight'`.
- If `labels_batch[0] = 2`, the title will be `'Potato___healthy'`.

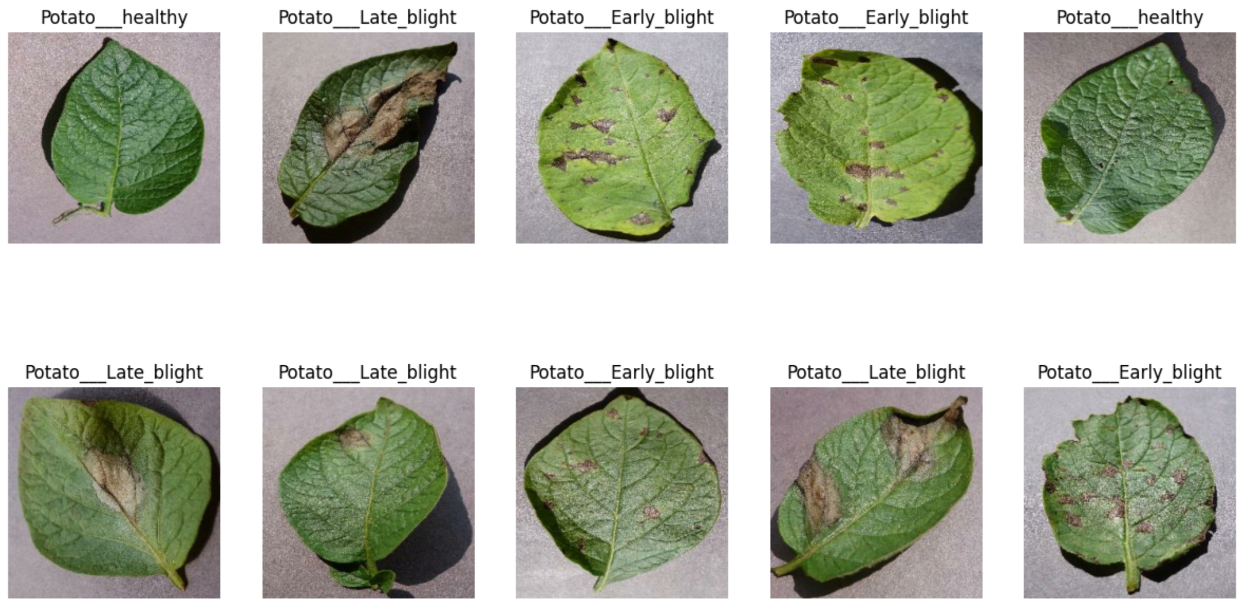
```
labels_batch
```



```
<tf.Tensor: shape=(32,), dtype=int32, numpy=
array([0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 2, 0, 1, 1, 1, 0], dtype=int32)>
```

Display Multiple Images

```
for image_batch, labels_batch in dataset.take(1):
    plt.figure(figsize=(15,8))
    for i in range (10):
        ax= plt.subplot(2,5,i+1)
        plt.imshow(image_batch[i]/255)
        plt.title(class_names[labels_batch[i]])
        plt.axis('off')
```



Split the Data

Dataset should be bifurcated into 3 subsets, namely:

1. **Training**: Dataset to be used while training **(80%)**
2. **Validation**: Dataset to be tested against while training **(10%)**
3. **Test**: Dataset to be tested against after we trained a model **(10%)**

```
len(dataset)

68
```

- Batch size = 32
    - 68∗32= 2176
    - Total Images = ***2152***
- **Train**:
    - 68∗0.8= **54.40**
    - **Take first 54 batches**
- **Test & validation**:
    - 68∗0.1= 6.80
    - **Take 7 batches**

```
train_ds= dataset.take(54)
```

- First 54 batches for testing
- **14 Batches remaining**

```
test_ds= dataset.skip(54)
val_dataset= test_ds.take(7)
test_ds= test_ds.skip(7)
```

- Temporarily, `skip(54)` assigned all 14 batches to `test_ds`

- `test_ds.take(7)` → Out of the above 14, `val_dataset` takes 7
- `test_ds.skip(7)` → Remaining 7 are assigned to `test_ds`

```
print(len(test_ds))
print(len(val_dataset))

7
7
```

Wrap all this in a function:

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

    ds_size = len(ds)

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

    train_size = round(train_split * ds_size)
    val_size = round(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
```

- It takes dataset → `ds`
- `ds_size = len(ds)` (68)
- `train_size = round(train_split * ds_size)` → round(.8*68) → 7
- `val_size = round(val_split * ds_size)` round(0.1*68) → 7

Why Shuffle Again Later?

1. **Ensure deterministic shuffling** with a fixed `seed=12`
2. Be certain the dataset is **freshly and consistently shuffled before splitting**

This is often done to **remove reliance on external shuffling** and **guarantee reproducibility** during experiments.

Apply the above function on the dataset:

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

Caching & Prefetching

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

val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)

test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

`train_ds.cache()`

- **Purpose**: Caches the dataset in memory **after it's loaded for the first time**.
- **Why**: So you don't reload or reprocess images from disk every epoch.
- **Effect**: Speeds up training after the first epoch.

Note:

- If your dataset is too big to fit in RAM, you can also cache to a file like `.cache('my_cache')`.

`.shuffle(1000)`

- **Purpose**: Shuffles the dataset so that the model doesn't learn the order.
- **Parameter (1000)**: The size of the **shuffling buffer**.
  - Larger buffer = better randomness, more memory.
  - Think of it as a queue: it keeps 1000 samples at a time and randomly picks from them.

`.prefetch(buffer_size=tf.data.AUTOTUNE)`

- **Purpose**: Tells TensorFlow to prepare the **next batch while** the current one is being processed (e.g., on GPU).
- **Effect**: Reduces data loading bottlenecks → speeds up training.
- `AUTOTUNE`: Lets TensorFlow pick the best buffer size for your system.

Preprocessing the Image

Creating a Layer for Resizing and Normalization

Before we feed our images to network, we should be resizing it to the desired size. Moreover, to improve model performance, we should normalize the image pixel value (keeping them in range 0 and 1 by dividing by 256). This should happen while training as well as inference. Hence we can add that as a layer in our Sequential Model.

```
from keras import Sequential
from tensorflow.keras.layers import Resizing, Rescaling

resize_and_rescale = Sequential([
    Resizing(IMAGE_SIZE, IMAGE_SIZE),
    Rescaling(1.0 / 255)
])
```

Data Augmentation:

```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.RandomRotation(0.1),
])
```