Data Preprocessing and Model Building Report

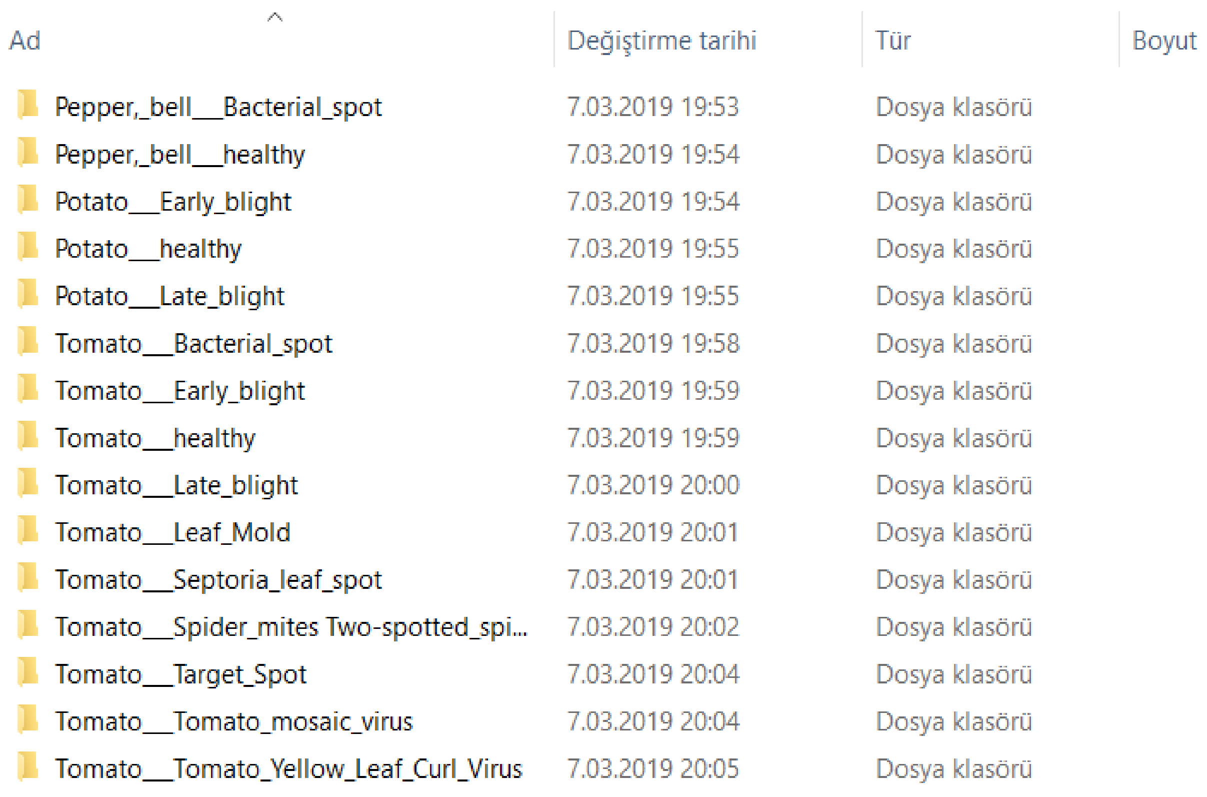
Date: 09.04.2024

A brief information will be given in this report about how Data Preprocessing applied into dataset, what kinds of layers used and architecture decisions made through the Model Building process.

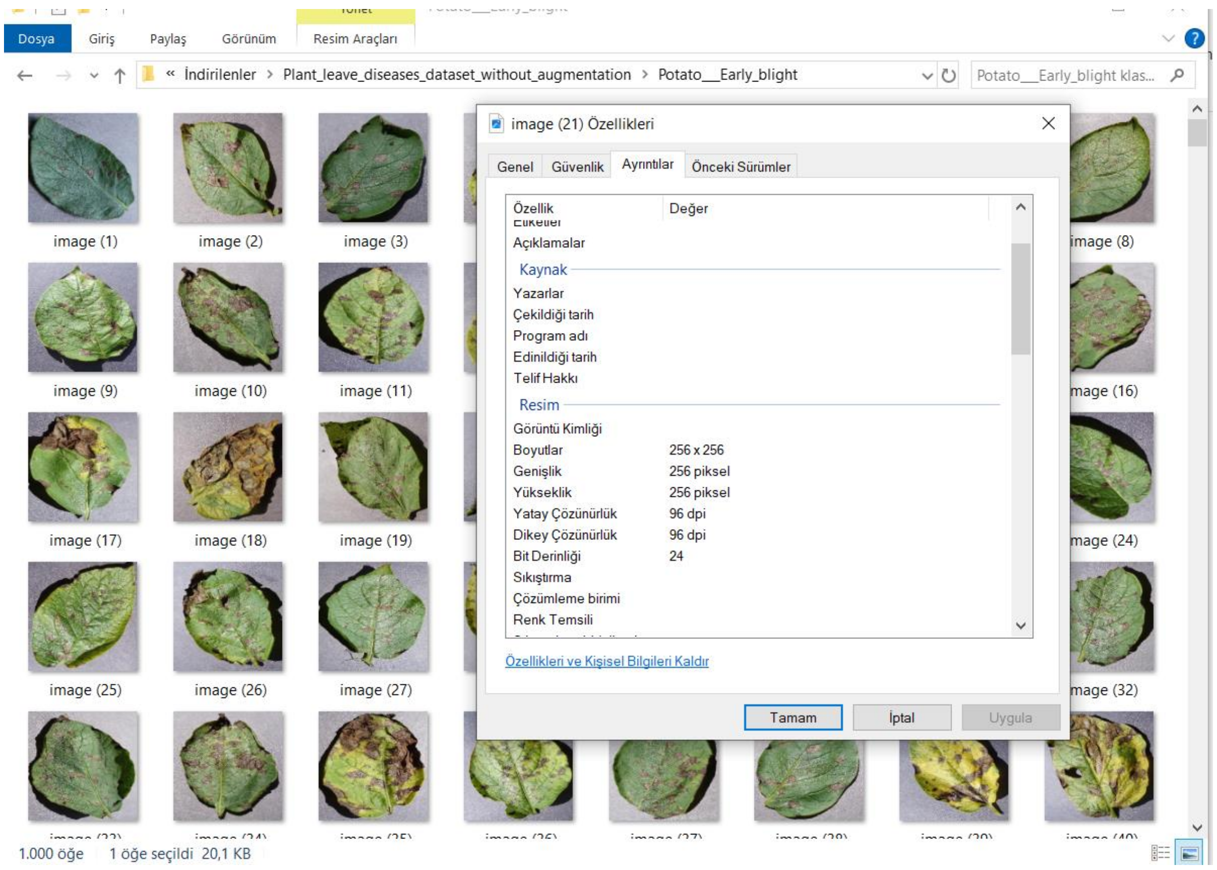
1. Data Preprocessing

Deep learning models typically accepts all inputs and their targets in a certain format which in most cases tensors of floating-point data except in some specific cases, tensors of integers or strings. Data preprocessing aims at making the raw data more amenable to neural networks. This includes vectorization, normalization or handling missing values.

The raw images in the dataset are size of 256x256 pixels and have only one feature indicating the name of disease or healthy. As discussed in Exploratory Data Analysis report, the dataset contains 38 different categories thus redundant categories are deleted by hand in order to make dataset suitable for the problem. There are screenshoots of raw dataset below:



Ad	Değiştirme tarihi	Tür	Boyut
Pepper,_bell___Bacterial_spot	7.03.2019 19:53	Dosya klasörü	
Pepper,_bell___healthy	7.03.2019 19:54	Dosya klasörü	
Potato___Early_blight	7.03.2019 19:54	Dosya klasörü	
Potato___healthy	7.03.2019 19:55	Dosya klasörü	
Potato___Late_blight	7.03.2019 19:55	Dosya klasörü	
Tomato___Bacterial_spot	7.03.2019 19:58	Dosya klasörü	
Tomato___Early_blight	7.03.2019 19:59	Dosya klasörü	
Tomato___healthy	7.03.2019 19:59	Dosya klasörü	
Tomato___Late_blight	7.03.2019 20:00	Dosya klasörü	
Tomato___Leaf_Mold	7.03.2019 20:01	Dosya klasörü	
Tomato___Septoria_leaf_spot	7.03.2019 20:01	Dosya klasörü	
Tomato___Spider_mites Two-spotted_spi...	7.03.2019 20:02	Dosya klasörü	
Tomato___Target_Spot	7.03.2019 20:04	Dosya klasörü	
Tomato___Tomato_mosaic_virus	7.03.2019 20:04	Dosya klasörü	
Tomato___Tomato_Yellow_Leaf_Curl_Virus	7.03.2019 20:05	Dosya klasörü	



Özellik	Değer
Açıklamalar	
Kaynak	
Yazarlar	
Çekildiği tarih	
Program adı	
Edinildiği tarih	
Telif Hakkı	
Resim	
Görüntü Kimliği	
Boyutlar	256 x 256
Genişlik	256 piksel
Yükseklik	256 piksel
Yatay Çözünürlük	96 dpi
Dikey Çözünürlük	96 dpi
Bit Derinliği	24
Sıkıştırma	
Çözümleme birimi	
Renk Temsili	

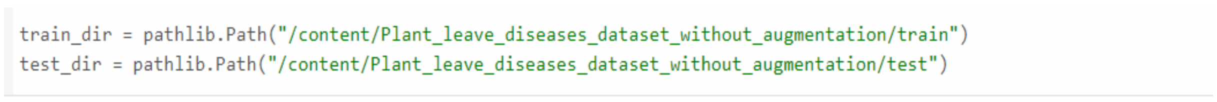
Data needs to be formatted into appropriately preprocessed floating-point tensors before giving them input into the model. Therefore image_dataset_from_directory() utility function from Keras library used in order to convert JPEG images into RGB grid of pixels and then into floating-point tensors.

Some validation techniques are investigated beforehand splitting data into train and test.

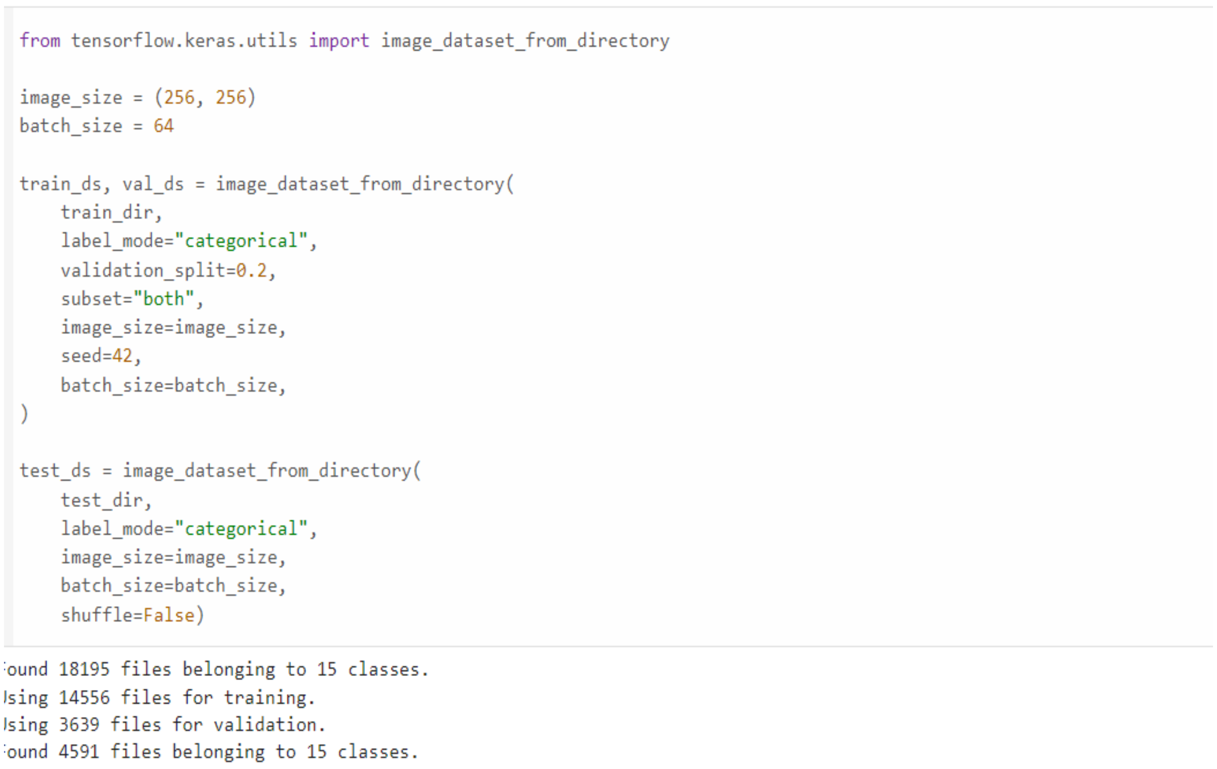
There are three types of common validation techniques used: Simple holdout validation, K-fold validation and iterated K-fold validation. K-fold validation and iterated version are generally used in situations where the dataset is small and contains a few samples. Since the dataset I used here has around 22.000 samples and not small relatively, Simple holdout validation technique is used which is splitting some fraction of data as train and the rest as test. 80% of data splitted into train folder and the rest 20% into test folder by hand manually.

Train folder path and test folder path read from Drive and passed into keras utility function image_dataset_from directory() which quickly sets up a data pipeline which can automatically turn image files into batches of preprocessed tensors. Image size is set to 256x256 as the original size and batch size is set to 64.

20% of train data splitted into validation data and shuffled before the split. Train data splitted into validation data because while training the data, performance on validation data used as a feedback signal in order to configure model size; the number of layers, the size of layers which is called as hyperparameters.

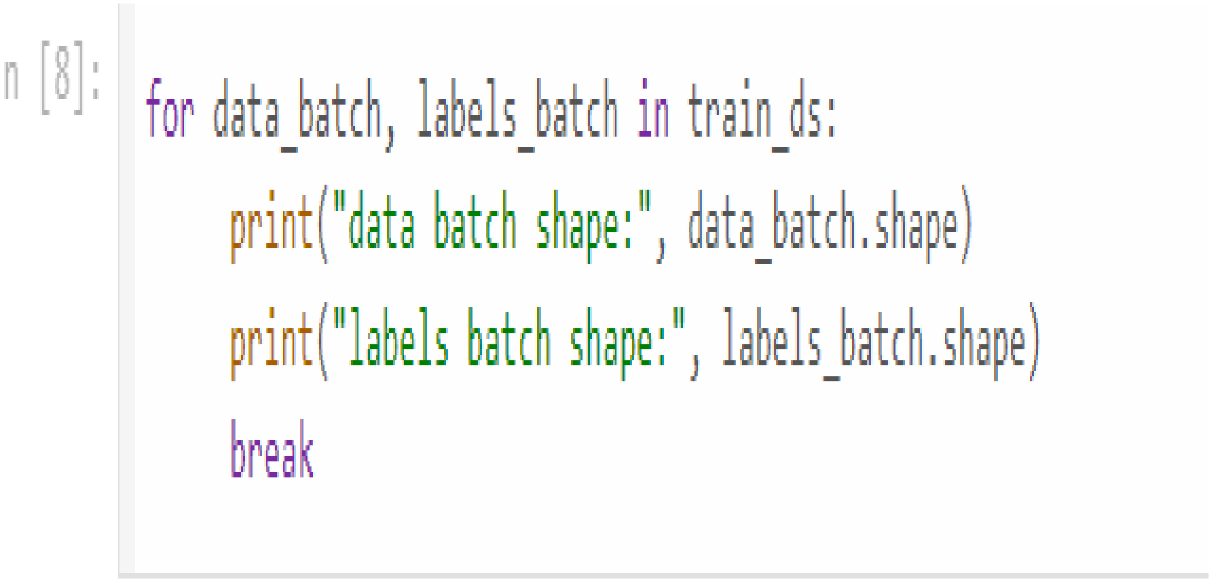


```python
train_dir = pathlib.Path("/content/Plant_leave_diseases_dataset_without_augmentation/train")
test_dir = pathlib.Path("/content/Plant_leave_diseases_dataset_without_augmentation/test")
```

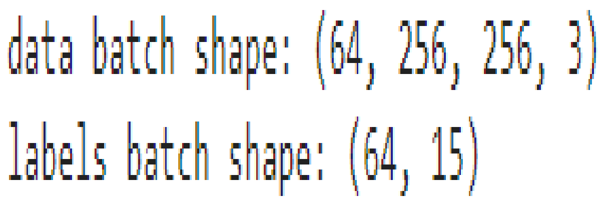


```python
from tensorflow.keras.utils import image_dataset_from_directory

image_size = (256, 256)
batch_size = 64

train_ds, val_ds = image_dataset_from_directory(
    train_dir,
    label_mode="categorical",
    validation_split=0.2,
    subset="both",
    image_size=image_size,
    seed=42,
    batch_size=batch_size,
)

test_ds = image_dataset_from_directory(
    test_dir,
    label_mode="categorical",
    image_size=image_size,
    batch_size=batch_size,
    shuffle=False)
```

```
ound 18195 files belonging to 15 classes.
sing 14556 files for training.
sing 3639 files for validation.
ound 4591 files belonging to 15 classes.
```



```python
n [8]:
for data_batch, labels_batch in train_ds:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    break
```



```
data batch shape: (64, 256, 256, 3)
labels batch shape: (64, 15)
```

2. Model Building

2.1 Initial Baseline Model

Convolutional Neural Networks (CNN) also called as convnets is a type of Deep Learning model and widely used in Computer Vision tasks such as Image classification problems. It became dominant after around 2016 and became the first choice because of its computer efficiency and feature mapping extraction in the images. Since my problem is detecting disease sympthoms on leaf images if it has then classify the condition of plant, local patterns are important in order for model to achieve better detection ability. Therefore CNN architecture is found the most suitable choice for the problem and then used through the model building stage.

Convolutional Neural Networks typically consists of Convolutional layers(Conv2d), pooling layers and a dense layer at the end in order to mapping the features into vector space. When building the model. Convolutional layers are used to extract local patterns to larger patterns sequentally, therefore increasing their filters (which indicates the output depth as well as feature map depth) is common practice while stacking the convolutional layers.

Pooling layers such as Max Pooling or Average Pooling are used for downsampling the input feature space in order to reduce the number feature-map coefficients to process, as well as to induce spatial-filter hierarchies by making succesive convolution layers look at increasingly larger windows.

To determine a baseline success for the model, a random image classifier with my dataset if it was purely distributed and balanced would be achieve accuracy rate:

100 / 15 = 6.666… %

Since the dataset I use is an imbalanced dataset and the majority class has around approximately 5000 samples over total number of approximately 23000 samples, the built model will be biased selecting the majority class roughly every 4,6 samples (which indicates the accuracy around 25%) which is computed by total number of samples 23000 divided by number of majority class 5000.

This simple calculation and observation shows that the initial model should be better than average 25% accuracy rate.

I started to building the model stacking the convolutional layers as well as increasing their filters while going deeper. This is a common approach across all convolutional layers and kernel size was set 3x3 because local patterns or small fractions are important for model to able to classify diseases like bacterial spot diseases. After taking input, rescaling layer is applied because of image inputs need to be rescaled into [0,1] range whose values are originally in [0,255] range. Max pooling layers applied after the convolutional layers in order to reduce input feature space. At the end, a single dense layer added in order to vectorize the output and to make 15-way classification since I have 15 classes in the dataset, softmax activation function was used. Screenshot of the initial model and model summary is given below:

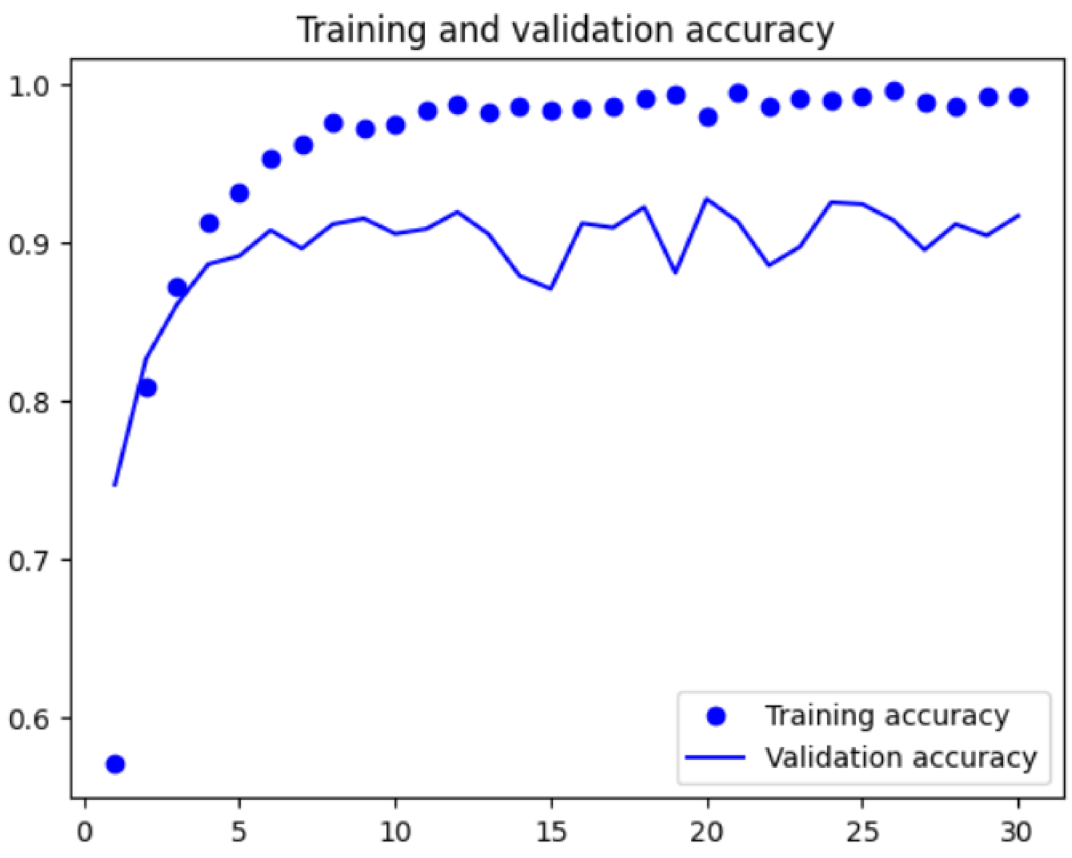
In [9]:
```python
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(256, 256, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(15, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

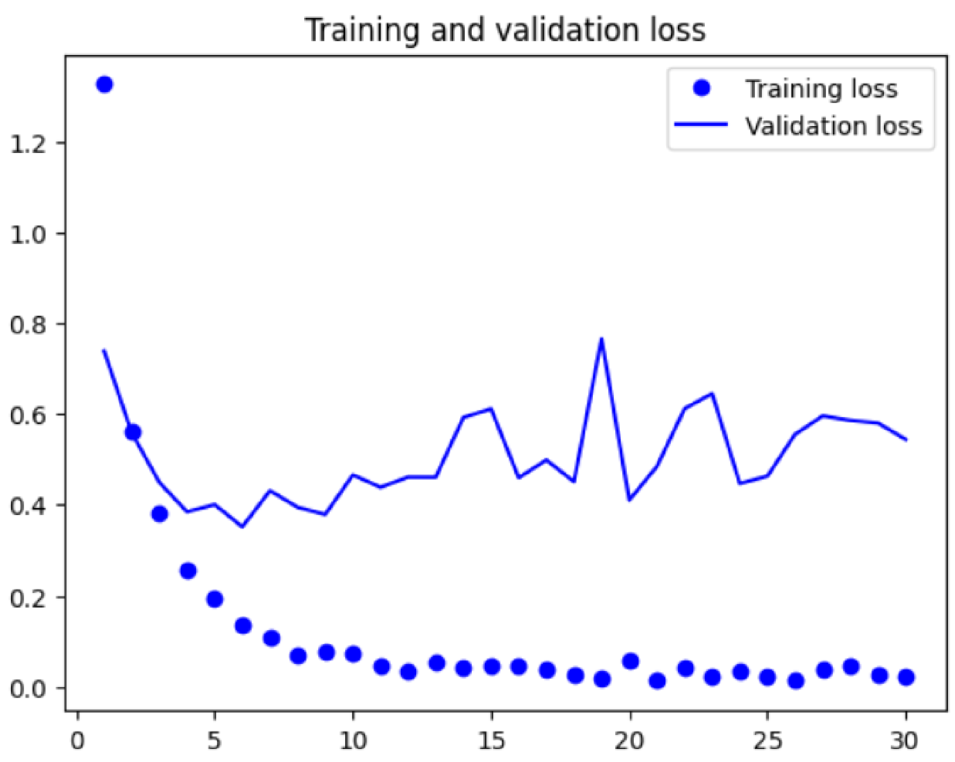
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 rescaling (Rescaling)       (None, 256, 256, 3)       0         
                                                                 
 conv2d (Conv2D)             (None, 254, 254, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856     
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 128)       0         
 g2D)                                                            
                                                                 
 conv2d_3 (Conv2D)           (None, 28, 28, 256)       295168    
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 256)       0         
 g2D)                                                            
                                                                 
 conv2d_4 (Conv2D)           (None, 12, 12, 256)       590080    
                                                                 
 flatten (Flatten)           (None, 36864)             0         
                                                                 
 dense (Dense)               (None, 15)                552975    
                                                                 
=================================================================
Total params: 1531471 (5.84 MB)
Trainable params: 1531471 (5.84 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="first_baseline_model_2.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_ds,
    epochs=30,
    validation_data=val_ds,
    callbacks=callbacks)
```

Loss function which sends feedback signals to backpropagation in order to adjust weight parameters, was choosen as categorical_crossentropy. Since the labels are not one-hot encoded which means not normalized into integer tensors. Optimizer function choosen as adam which is commonly used function lately because of its momentum advantage. Metric was choosen as accuracy. Initial model trained 30 epochs and its training / validation accuracy as well as loss charts given below:





```
5]: test_model = keras.models.load_model("first_baseline_model_2.keras")
    test_loss, test_acc = test_model.evaluate(test_ds)
    print(f"Test accuracy: {test_acc:.3f}")
```

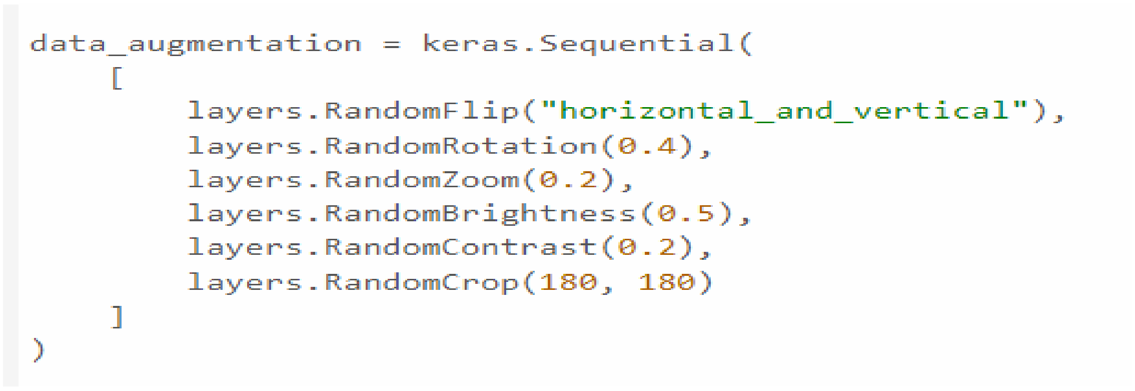
```
72/72 [==============================] - 3s 35ms/step - loss: 0.2910 - accuracy: 0.9109
Test accuracy: 0.911
```

As seen training accuracy is increasing linearly but seems to stall around 20 epoch. This indicates that model is not enough large and does not have enough representational power so that learning of the model was stucked. Also, in the training/validation loss chart, validation loss seems to stop decrasing around after 6-7 epochs and then increasing. This shows that the model began to overfitting. Test accuracy achieved 91% accuracy rate but as discussed in EDA report, other metrics need to be concerned while evaluating the success of the model.

In the second setup, model size increased by adding some additional layers to have enough representational power to learn the model. To mitigate the overfitting, data augmentation technique was applied and also dropout layer was added.

2.2 Second Model Setup

Data Augmentation takes the approach of generating more training data from existing training samples by augmenting the samples via a number of random transformations which produces believable-looking images. The goal is the model will never see the exact same picture twice thus this helps the model train more aspects of the data so it can generalize better. Since PlantVillage dataset consists of images taken in certain laboratory conditions such as better lighting, high resolution camera and plain background. This may lead the model to be trained and memorized some certain aspects and patterns in the images thus the model may yield false predictions in the greenhouses, agricultural estates where farmers or agricultural engineers typically work and going to benefit from the mobile application. Data augmentation provides better generalized results and mitigate the overfitting.



```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal_and_vertical"),
        layers.RandomRotation(0.4),
        layers.RandomZoom(0.2),
        layers.RandomBrightness(0.5),
        layers.RandomContrast(0.2),
        layers.RandomCrop(180, 180)
    ]
)
```

RandomFlip: Applies horizontal and vertical flipping to a random 50% of the images that go through it. Users of the application can take photos of images reversely in horizontal manner or vertical manner.

Random Rotation: Rotates the input images by a random value in the range [-20%

+20%]. Photos taken by users can be rotated in different directions.

RandomZoom: Zooms in or out of the image by a random factor in the range [-20%, +20%]. User can zoom in or out while taking the pictures and this may help model to yield more generalized results.

RandomBrightness: Increases or decreases the brightness of the images in the range [-0.5, +0.5]. Lighting conditions may vary based on where the user takes the images of the plant leafs. Also mobile camera lighting and quality may vary across different types of Android mobile phones since there are so much options and manufacturer in the Android market.

RandomConstrast: Increases or decreases the contrast of the images in the range [-0.2, +0.2]. This is also applied because contrast as well as lighting may vary across different type of mobile phones.

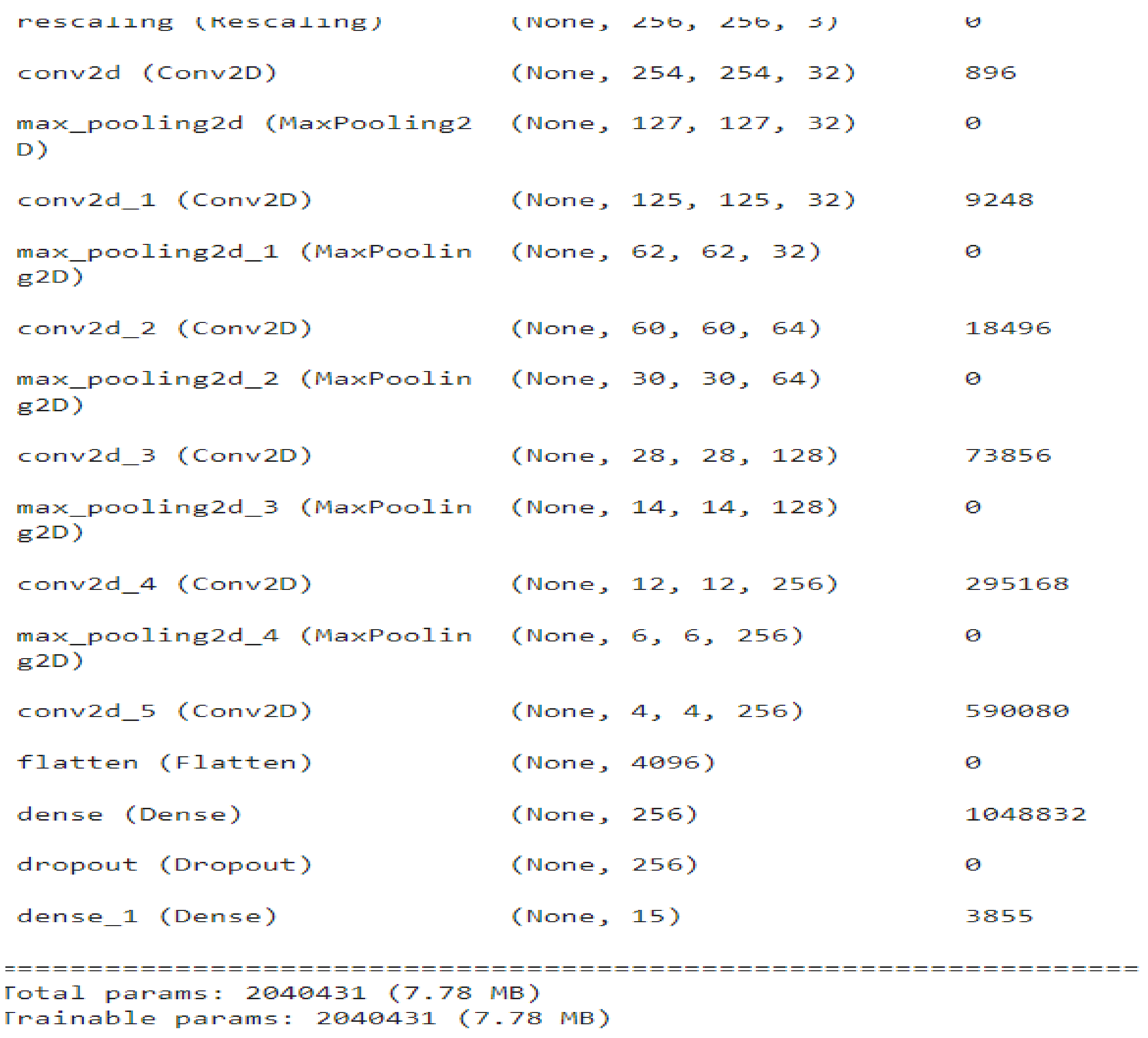
RandomCrop: Randomly crops of a batch of images into specified size, in this case 180x180 pixel. This can also be useful, because user can crop the images while taking the pictures.

Example of data augmentation applied given below:

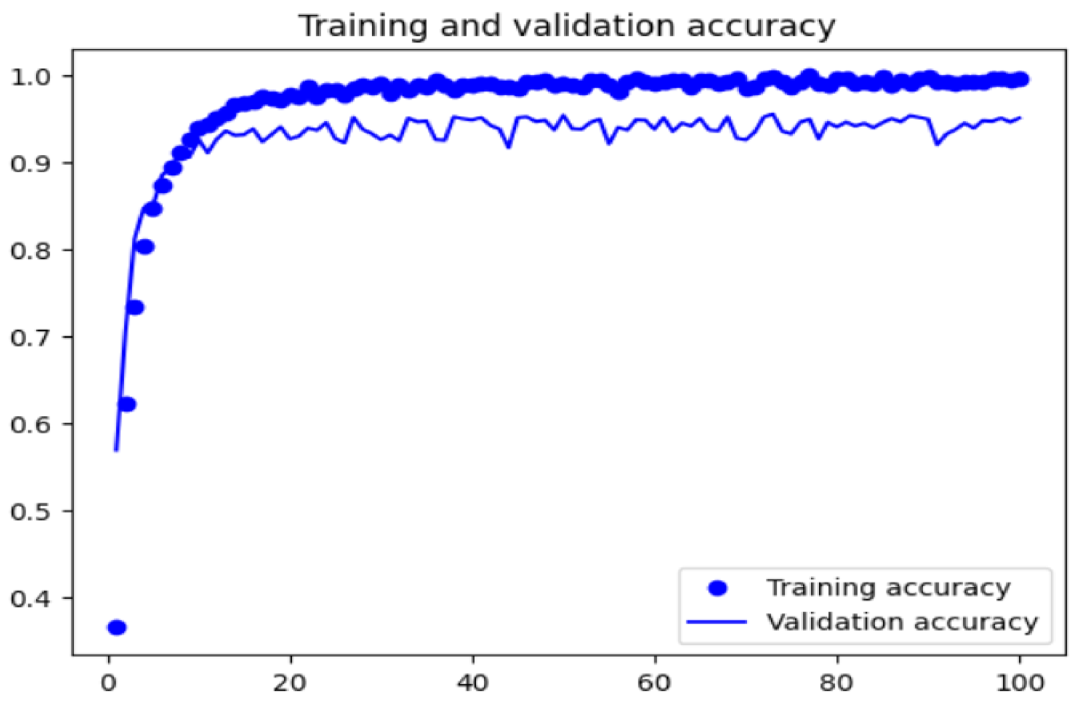
Dropout is one of the most effective and most commonly used regularization techniques for neural networks. Dropout applied to a layer, consists of randomly dropping out (setting to zero) a number of output features of the layer during training. This helps to migitate overitting by preventing parameters of the layers start memorizing the train data. The dropout rate is the fraction of the features that are zeroed out, At test time, no units are dropped out so as well as data augmentation, dropout only applied in training process not in the infere process which is testing the model. The dropout rate is the fraction of the features that are zeroed out; it's usually set between 0.2 and 0.5. Since initial model has approximately one and half million parameters, this rate selected as 0.5 and applied into the model. Below is the screenshoot given of the second model setup with increased size of the model, applied data augmentation and dropout layers. This model trained 100 epochs.

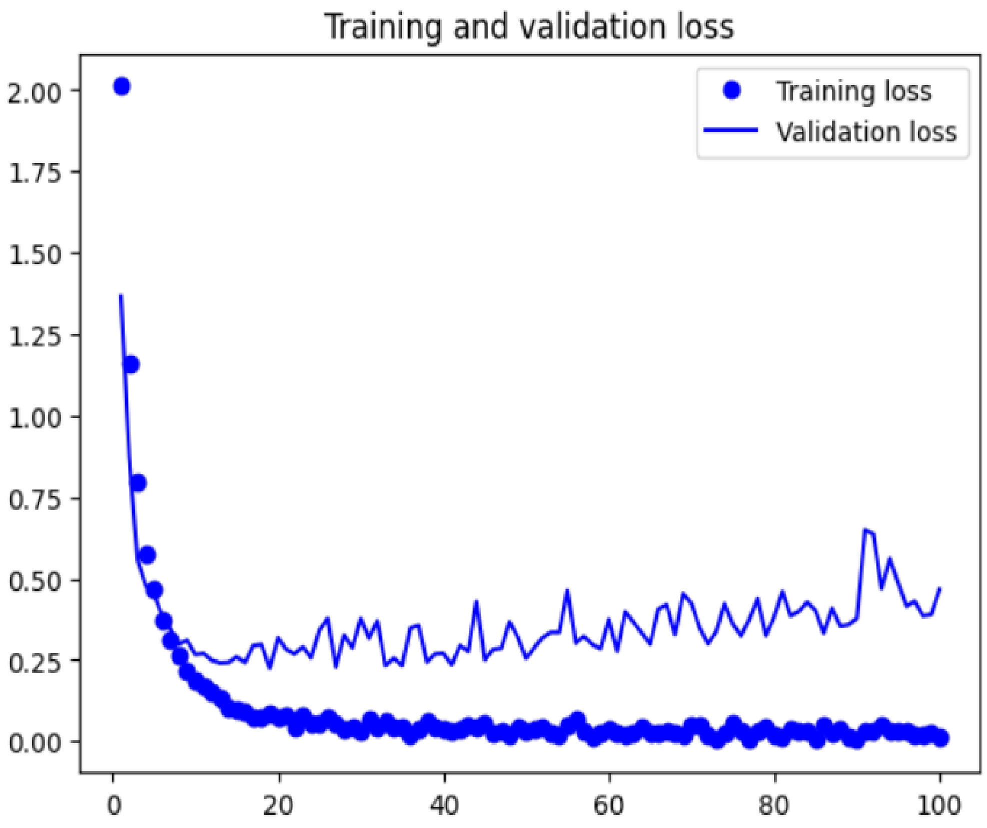
In [11]:

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(256, 256, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(15, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



```
 rescaling (Rescaling)       (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      9248

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 128)       0
 g2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 256)       295168

 max_pooling2d_4 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 256)         590080

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 256)               1048832

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 15)                3855

=================================================================
Total params: 2040431 (7.78 MB)
Trainable params: 2040431 (7.78 MB)
```





[17]:

```
test_model = keras.models.load_model("first_baseline_model_4_100epochs.keras")
test_loss, test_acc = test_model.evaluate(test_ds)
print(f"Test accuracy: {test_acc:.3f}")
```

```
72/72 [==============================] - 2s 29ms/step - loss: 0.2187 - accuracy: 0.9397
Test accuracy: 0.940
```

With this setup, the model achieved 94% test accuracy which is 3% percent higher than inital model. Also, training accuracy approximate better linearly up to 100% without any stall. The model began to overfit at around 25-30 epochs by applying data augmentation and dropout.

As a result, second model is better than initial model, with higher representational power, better generalized and achived higher accuracy than initial model. But as a consequence of adding more layers, model size (total number of parameters of the layers) increased up to 2 million which is actually bad situation for mobile phones in terms of their hardware constraints. In the next report, final model setup will be presented and modern architectures techniques applied and discussed such as residual networks and batch normalization in order to reduce model size with a better generalized model as well as accuracy. Final model evaluation and deployment will also be presented and discussed.

Student Name: Hadican Munis

Student ID: 20170702039