Train And Save The Model

Import ImageDataGenerator Library and Configure it

ImageDataGenerator class is used to load the images with different modifications like considering the zoomed image, flipping the image and rescaling the images to range of 0 and 1.

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2,zoom_range = 0.2,horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1)
```

Apply ImageDataGenerator functionality to Train and Test set

Specify the path of both the folders in the flow_from_directory method. We are importing the images in 128*128 pixels.

```
x_train= train_datagen.flow_from_directory('/content/drive/MyDrive/Veg-dataset/Veg-dataset/train_set',batch_size=32,target_size=(128,128),
                        color_mode='rgb',class_mode='categorical')
x_test = test_datagen.flow_from_directory('/content/drive/MyDrive/Veg-dataset/Veg-dataset/test_set',batch_size=32,target_size=(128,128),
                        color_mode='rgb',class_mode='categorical')
```

```
Found 11430 images belonging to 9 classes.
Found 3416 images belonging to 9 classes.
```

Import the required model building libraries

Import the libraries that are required to initialize the neural network layer, create and add different layers to the neural network model.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
```

from keras.layers **import** Flatten

Initialize the model

Initialize the neural network layer by creating a reference/object to the Sequential class.

Model=Sequential()

Add the convolution layer

In the first layer of the neural network model, the convolution layer will be added. To create a convolution layer, the Convolution2D class is used. It takes a number of feature detectors, features detector size, expected input shape of the image, and activation function as arguments. This layer applies feature detectors on the input image and returns a feature map (features from the image).

Activation Function:

These are the functions that help us to decide if we need to activate the node or not. These functions introduce non-linearity in the networks.

model.add(Convolution2D(32,(3,3),input_shape = (128,128,3),activation = 'relu'))

Add the pooling layer

Max Poolingselects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

After the convolution layer, a pooling layer is added. Max pooling layer can be added using MaxPooling2D class. It takes the pool size as a parameter. Efficient size of the pooling matrix is (2,2). It returns the pooled feature maps.

model.add(MaxPooling2D(pool_size = (2,2)))

Add the flatten layer

The flatten layer is used to convert n-dimensional arrays to 1-dimensional arrays. This 1D array will be given as input to ANN layers.

```
model.add(Flatten())
```

Adding the dense layers

Three dense layers are added which usually take a number of units/neurons. Specifying the activation function, kind of weight initialization is optional.

```
model.add(Dense(300,  'relu'))
model.add(Dense(150, 'relu'))
model.add(Dense(75, 'relu'))
model.add(Dense(9,  'softmax', ))
```

Compile the model

After adding all the required layers, the model is to be compiled. For this step, loss function, optimizer and metrics for evaluation can be passed as arguments.

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

Fit and save the model

Fit the neural network model with the train and test set, number of epochs and validation steps. Steps per epoch is determined by number of training images//batch size, for validation steps number of validation images//batch size.

```
model.fit(x_train,epochs=10,steps_per_epoch=89,validation_data = x_test,
validation_steps = 27)
```

Accuracy, Loss:

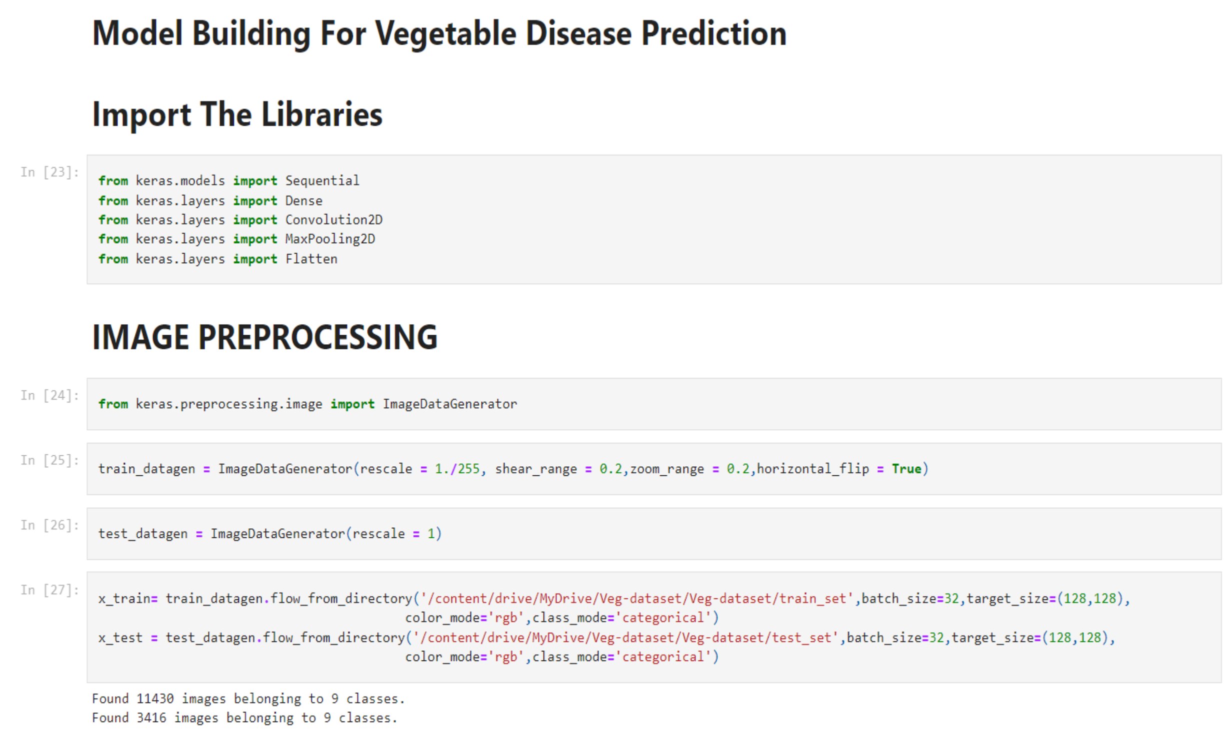
Loss value implies how poorly or well a model behaves after each iteration of optimization. An accuracy metric is used to measure the algorithm's performance in an interpretable way. The accuracy of a model is usually determined after the model parameters and is calculated in the form of a percentage.

The weights are to be saved for future use. The weights are saved in as .h5 file using save().

model.save("veg.h5")

model.summary()

can be used to see all parameters and shapes in each layer in our models.



Model Building For Vegetable Disease Prediction

Import The Libraries

In [23]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
```

IMAGE PREPROCESSING

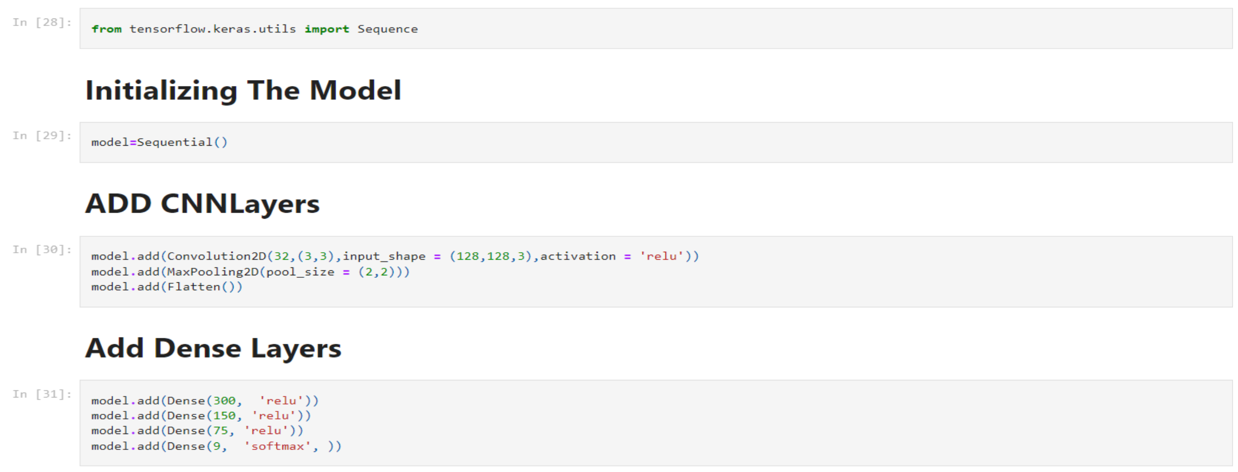
In [24]:
```
from keras.preprocessing.image import ImageDataGenerator
```

In [25]:
```
train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2,zoom_range = 0.2,horizontal_flip = True)
```

In [26]:
```
test_datagen = ImageDataGenerator(rescale = 1)
```

In [27]:
```
x_train= train_datagen.flow_from_directory('/content/drive/MyDrive/Veg-dataset/Veg-dataset/train_set',batch_size=32,target_size=(128,128),
                                         color_mode='rgb',class_mode='categorical')
x_test = test_datagen.flow_from_directory('/content/drive/MyDrive/Veg-dataset/Veg-dataset/test_set',batch_size=32,target_size=(128,128),
                                         color_mode='rgb',class_mode='categorical')
```

```
Found 11430 images belonging to 9 classes.
Found 3416 images belonging to 9 classes.
```



In [28]:
```
from tensorflow.keras.utils import Sequence
```

Initializing The Model

In [29]:
```
model=Sequential()
```

ADD CNNLayers

In [30]:
```
model.add(Convolution2D(32,(3,3),input_shape = (128,128,3),activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())
```

Add Dense Layers

In [31]:
```
model.add(Dense(300,  'relu'))
model.add(Dense(150, 'relu'))
model.add(Dense(75, 'relu'))
model.add(Dense(9,  'softmax', ))
```

Train And Save The Model

In [32]:
```
model.compile(optimizer='adam', loss = "categorical_crossentropy" , metrics =['accuracy'])
```

In [33]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 126, 126, 32)      896

 max_pooling2d_1 (MaxPooling  (None, 63, 63, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 127008)            0

 dense_7 (Dense)             (None, 300)               38102700

 dense_8 (Dense)             (None, 150)               45150

 dense_9 (Dense)             (None, 75)                11325

 dense_10 (Dense)            (None, 9)                 684

=================================================================
Total params: 38,160,755
Trainable params: 38,160,755
Non-trainable params: 0
_________________________________________________________________
```

In [35]:
```
model.fit(x_train,epochs=10,steps_per_epoch=89,validation_data = x_test, validation_steps = 27)


model.save("veg.h5")
```

```
Epoch 1/10
89/89 [==============================] - 718s 8s/step - loss: 1.4285 - accuracy: 0.5103 - val_loss: 398.4555 - val_accuracy: 0.3611
Epoch 2/10
89/89 [==============================] - 512s 6s/step - loss: 0.9607 - accuracy: 0.6552 - val_loss: 883.0972 - val_accuracy: 0.2697
Epoch 3/10
89/89 [==============================] - 361s 4s/step - loss: 0.7985 - accuracy: 0.7229 - val_loss: 664.0219 - val_accuracy: 0.3275
Epoch 4/10
89/89 [==============================] - 290s 3s/step - loss: 0.6901 - accuracy: 0.7598 - val_loss: 870.4464 - val_accuracy: 0.2859
Epoch 5/10
89/89 [==============================] - 208s 2s/step - loss: 0.6114 - accuracy: 0.7802 - val_loss: 709.7632 - val_accuracy: 0.3542
Epoch 6/10
89/89 [==============================] - 183s 2s/step - loss: 0.5603 - accuracy: 0.7978 - val_loss: 842.9805 - val_accuracy: 0.2384
Epoch 7/10
89/89 [==============================] - 148s 2s/step - loss: 0.5167 - accuracy: 0.8195 - val_loss: 1794.7992 - val_accuracy: 0.1296
Epoch 8/10
89/89 [==============================] - 118s 1s/step - loss: 0.4628 - accuracy: 0.8385 - val_loss: 1593.1969 - val_accuracy: 0.1516
Epoch 9/10
89/89 [==============================] - 103s 1s/step - loss: 0.4795 - accuracy: 0.8304 - val_loss: 1793.0253 - val_accuracy: 0.1551
Epoch 10/10
89/89 [==============================] - 94s 1s/step - loss: 0.3958 - accuracy: 0.8575 - val_loss: 1651.8546 - val_accuracy: 0.1505
```