**IMPLEMENTATION**

**MODULES:**

* Dataset
* Importing the necessary libraries
* Retrieving the images
* Splitting the dataset
* Building the model
* Apply the model and plot the graphs for accuracy and loss
* Accuracy on test set
* Saving the Trained Model

**MODULES DESCSRIPTION:**

**Dataset:**

In the first module, we developed the system to get the input dataset for the training and testing purpose. We give the data set in model folder. The dataset consists of 2311 Potato Disease Detection images. The dataset is collected from the internet sources. We have not used any particular dataset from kaggle or other particular websites, as it’s not available. The collected dataset consists of 5 Classes: Hollow heart of potato, potato leaf roll virus, early blight, scab of potato & Soft rot of potato.

**Importing the necessary libraries:**

We will be using Python language for this. First we will import the necessary libraries such as keras for building the main model, sklearn for splitting the training and test data, PIL for converting the images into array of numbers and other libraries such as pandas, numpy, matplotlib and tensorflow.

**Retrieving the images:**

We will retrieve the images and their labels. Then resize the images to (224,224) as all images should have same size for recognition. Then convert the images into numpy array.

**Splitting the dataset:**

Split the dataset into train and test. 80% train data and 20% test data.

# VGG-16 | CNN model

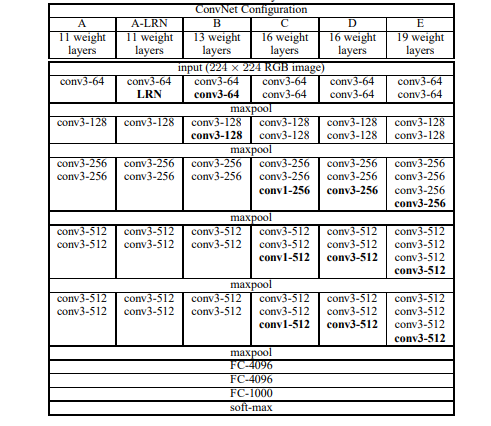
**Architecture:**

The input to the network is image of dimensions (224, 224, 3). The first two layers have 64 channels of 3\*3 filter size and same padding. Then after a max pool layer of stride (2, 2), two layers which have convolution layers of 256 filter size and filter size (3, 3). This followed by a max pooling layer of stride (2, 2) which is same as previous layer. Then there are 2 convolution layers of filter size (3, 3) and 256 filter. After that there are 2 sets of 3 convolution layer and a max pool layer. Each have 512 filters of (3, 3) size with same padding.This image is then passed to the stack of two convolution layers. In these convolution and max pooling layers, the filters we use is of the size 3\*3 instead of 11\*11 in AlexNet and 7\*7 in ZF-Net. In some of the layers, it also uses 1\*1 pixel which is used to manipulate the number of input channels. There is a padding of 1-pixel (same padding) done after each convolution layer to prevent the spatial feature of the image.

After the stack of convolution and max-pooling layer, we got a (7, 7, 512) feature map. We flatten this output to make it a (1, 25088) feature vector.After this there are 3 fully connected layer, the first layer takes input from the last feature vector and outputs a (1, 4096) vector, second layer also outputs a vector of size (1, 4096) but the third layer output a 1000 channels for 1000 classes of ILSVRC challenge, then after the output of 3rd fully connected layer is passed to softmax layer in order to normalize the classification vector. After the output of classification vector top-5 categories for evaluation. All the hidden layers use ReLU as its activation function. ReLU is more computationally efficient because it results in faster learning and it also decreases the likelihood of vanishing gradient problem.

**Configuration:**

The table below listed different VGG architecture. We can see that there are 2 versions of VGG-16 (C and D). There is not much difference between them except for one that except for some convolution layer there is (3, 3) filter size convolution is used instead of (1, 1). These two contains 134 million and 138 million parameters respectively.

[](https://media.geeksforgeeks.org/wp-content/uploads/20200217112031/VGG16conf.PNG)

*Different VGG Configuration*

**Building the model:**

They are very successful in image recognition. The key part to understand, which distinguishes CNN from traditional neural networks, is the **convolution** operation. Having an image at the input, CNN scans it many times to look for certain **features**. This scanning (convolution) can be set with 2 main parameters: stride and padding type. As we see on below picture, process of the first convolution gives us a set of new frames, shown here in the second column (layer). Each frame contains an information about one feature and its presence in scanned image. Resulting frame will have larger values in places where a feature is strongly visible and lower values where there are no or little such features. Afterwards, the process is repeated for each of obtained frames for a chosen number of times. In this project I chose a classic VGG-16 model which contains only two convolution layers.

The latter layer we are convolving, the more high-level features are being searched. It works similarly to human perception. To give an example, below is a very descriptive picture with features which are searched on different CNN layers. As you can see, the application of this model is face recognition. You may ask how the model knows which features to seek. If you construct the CNN from the beginning, searched features are random. Then, during training process, weights between neurons are being adjusted and slowly CNN starts to find such features which enable to meet predefined goal, i.e. to recognize successfully images from the training set.

Between described layers there are also **pooling** (sub-sampling) operations which reduce dimensions of resulted frames. Furthermore, after each convolution we apply a non-linear function (called **ReLU**) to the resulted frame to introduce non-linearity to the model.

Eventually, there are also fully connected layers at the end of the network. The last set of frames obtained from convolution operations is flattened to get a one-dimensional vector of neurons. From this point we put a standard, fully-connected neural network. At the very end, for classification problems, there is a softmax layer. It transforms results of the model to probabilities of a correct guess of each class

**Apply the model and plot the graphs for accuracy and loss:**

We will compile the model and apply it using fit function. The batch size will be 2. Then we will plot the graphs for accuracy and loss. We got average validation accuracy of 1.00% and average training accuracy of 1.00%.

**Accuracy on test set:**

We got an accuracy of 100% on test set.

**Saving the Trained Model:**

Once you’re confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into a .h5 or .pkl file using a library like pickle.

Make sure you have pickle installed in your environment.

Next, let’s import the module and dump the model into .h5 file