



FINAL PROJECT REPORT

COMPUTER VISION – CNN

INTERNSHIP – 2021

MENTOR ASSIGNED – NAG. SIR (Data Scientist at Innomatics Research Labs, Hyderabad).

PROJECT NAME: Plant Disease Recognition

Domain: Agriculture

Duration Of Project: 31-12-2021 TO 25-01-2022



TEAM MEMBERS:

- 1.Manuj Kumar Joshi
- 2.Kashetti Prashanth
- 3.Priyanka Nandibhatla
- 4.Soujanya Vattikolla
- 5.Pooja Roy Choudhary

Plant Disease classification:

Attribute Information:

This project is about collecting images of various infected, good and seemingly infected plant leafs. Then apply image processing on the images and predict the infected plant leaf's using Deep Learning+ImageProcessing.

Project Domain: OpenCV (Computer vision) and CNN and Transfer Learning

Prerequisites:


Deeplearning, Computer vision/OpenCV, Image Segmentation

Steps Involved in Image Processing: -

1. Image Acquisition
2. Image Enhancement
3. Image Restoration
4. Color Image Processing
5. Compression
6. Segmentation
7. Mobile Application

Libraries Involved:

- TensorFlow
- Keras
- Scikit Learn
- Pickle
- OpenCV

Our goal  Goal is clear and simple. We need to build a model, which can classify between healthy and diseased crop leaves and also if the crop has any disease, predict which disease is it.

OBJECTIVE: To detect plant leaf disease using CNN.

The main purpose is to detect the diseased part of the plant leaf. Using python convolutional neural networks are implemented in order to classify the diseased part. Aim is to detect the diseased part by finding the optimum way with minimum cost.

PROJECT OUTLINE: Here we have almost 1 GB of images and we have multiple subfolders that are (i) Apple___Apple_scab, (ii) Apple___Black_rot, (iii) Apple___Cedar_apple_rust, (iv) Apple___healthy, (v) Blueberry___healthy, (vi) Cherry_(including_sour)___healthy, (vii) Cherry_(including_sour)___Powdery_mildew, (viii) Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot. Each of this folder represents a disease or the name of the disease that occurs in plant on their leaves. The agenda over here is to train all our images with respect to the classes and then we are going to provide a new image somewhere from the google or some other image that we have not worked on and see whether if a model is capable of generating or if a model is capable of detecting whether a particular image belongs to that particular class of a disease.

TOOL USED: Google.colab

Technologies to be used: DL, Computer Vision

Abstract:

- Identification of the plant diseases is the key to preventing the losses in the yield and quantity of the agricultural product.
- Plant leaf diseases and destructive insects are a major challenge in the agriculture sector.
- It is very difficult to monitor the plant diseases manually. It requires tremendous amount of work, expertise in the plant diseases, and also require the excessive processing time.
- Faster and an accurate prediction of leaf diseases in crops could early treatment technique while considerably reducing economic losses.
- Hence, image processing is used for the detection of plant diseases.

About the dataset:

The plant leaf disease dataset consists of 87,900 images of leaves spanning 38 classes. Each class denotes a combination of the plant the leaf is from and the disease present in the leaf.

The dataset is divided into three parts as follows:

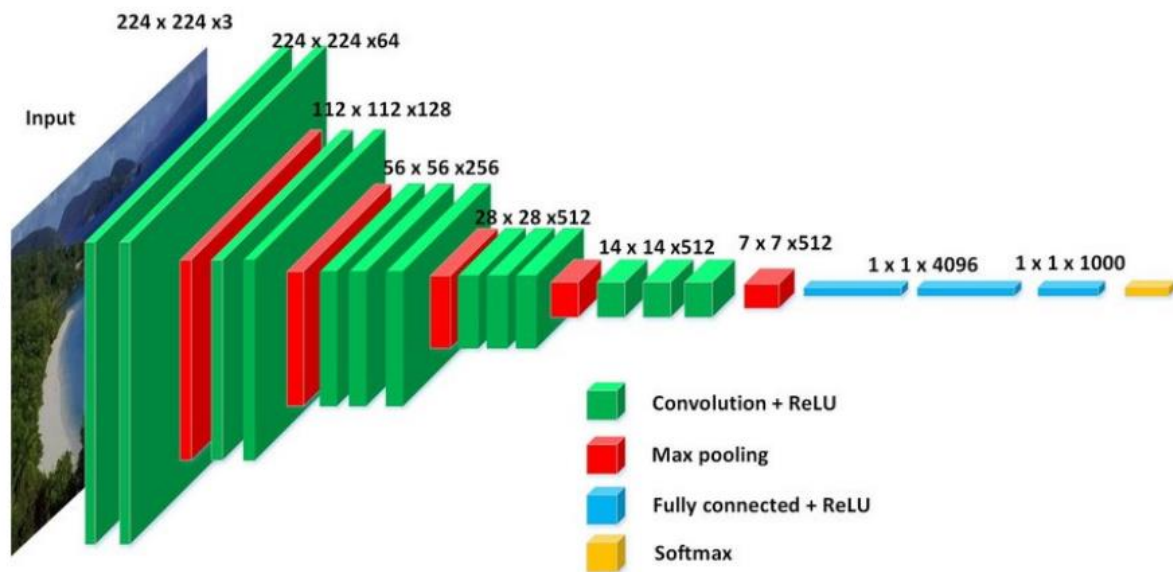
- **train** - 70,295 images divided into 38 classes.
- **valid** - 17,572 images divided into 38 classes.
- **test** - 33 images

Project flow:

- Load and pre-process the image dataset.
 - Some leaves are taken as input and it is pre-processed for the improvement of image to remove the unwanted distortion for feature processing.
 - In the feature extraction the shape, colour and texture are identified.
- Train the image classifier on the dataset.
 - Database consists of many images with healthy and unhealthy leaves and the images are trained to identify the particular image from the database.
- Use the trained classifier to predict image content.
 - In Neural Network classification the leaf is recognised whether the leaf is normal or abnormal.
 - If the leaf is abnormal the defect region is classified. The affected area and type of disease is identified.

MODEL ARCHITECTURE:

Vgg16



VGG-16 network architecture for feature extraction

Input: Images

An image is nothing but a matrix of pixel values we flatten the image and feed it to a Multi-Level Perceptron for classification purposes.

Output:

Knowledge of the scene (recognize objects, people, activity happening there, distance of the object from camera and each other, ...)

Introduction on VGGNet

The full name of VGG is the Visual Geometry Group, which belongs to the Department of Science and Engineering of Oxford University. It has released a series of convolutional network models beginning with VGG, which can be applied to face recognition and image classification, from VGG16 to VGG19. The original purpose of VGG's research on the depth of convolutional networks is to understand how the depth of convolutional networks affects the accuracy and accuracy of large-scale image classification and recognition. It is considered to be one of the excellent vision model architectures till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It

follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx.) parameters.

Vgg16 Architecture:

The input to conv1 layer is of fixed size 224×224 RGB image. The image is passed through a stack of convolutional layers where the filters were used with a very small receptive field: 3×3 . In one of the configurations, it also utilizes 1×1 convolutional filter, which can be seen as a linear transformation of the input channels. The convolution stride is fixed to 1 pixel; the spatial padding of conv. Layer input is such that the spatial resolution is preserved after convolution i.e. the padding is 1 pixel for 3×3 conv layers. Spatial pooling is carried out by five max pooling layers which follow some of the conv layers. Max pooling is performed over a 2×2 -pixel window with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class).

The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalisation (LRN), such normalization does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time.

Inception V3

The Inception V3 is a deep learning model based on Convolutional Neural Networks, which is used for image classification.

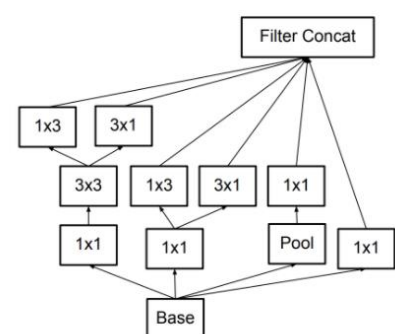
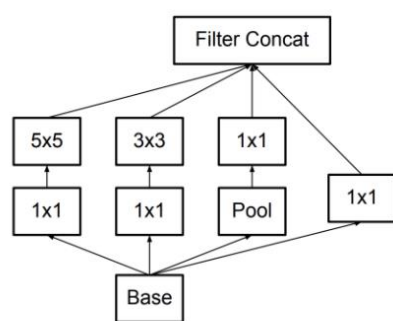
The Inception V3 model uses several techniques for optimizing the network for better model adaptation.

1. It has higher efficiency
2. It has a deeper network compared to the Inception V1 and V2 models, but its speed isn't compromised.
3. It is computationally less expensive.
4. It uses auxiliary Classifiers as regularizes.

Inception v3 Architecture

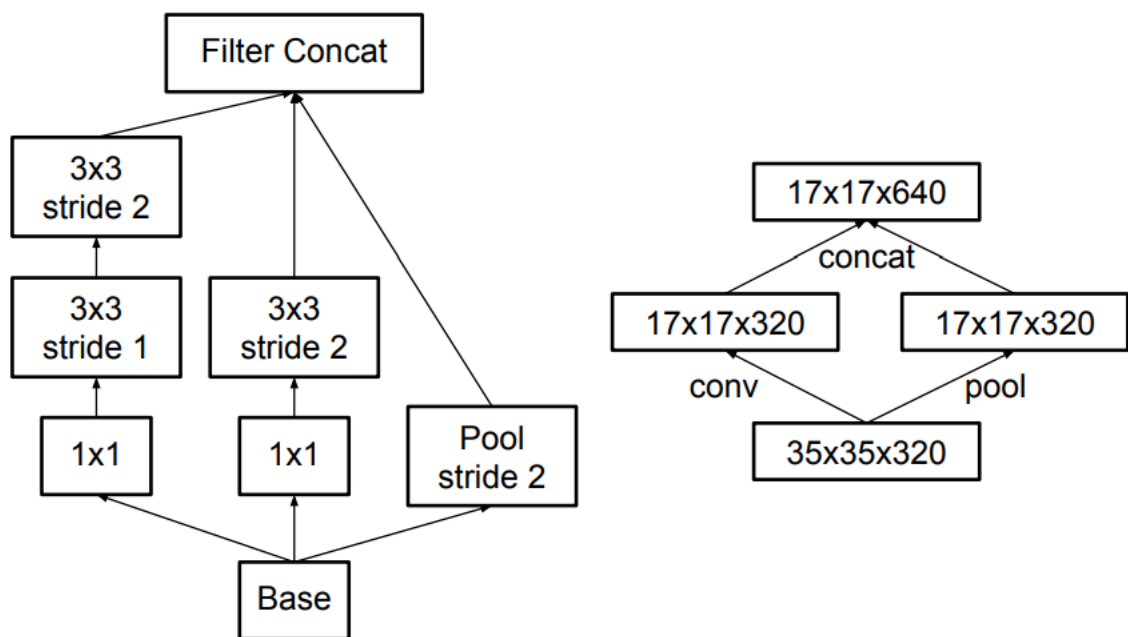
The architecture of an Inception v3 network is progressively built, step-by-step, as explained below:

1. **Factorized Convolutions:** this helps to reduce the computational efficiency as it reduces the number of parameters involved in a network. It also keeps a check on the network efficiency.
2. **Smaller convolutions:** replacing bigger convolutions with smaller convolutions definitely leads to faster training. Say a 5×5 filter has 25 parameters; two 3×3 filters replacing a 5×5 convolution has only 18 ($3 \times 3 + 3 \times 3$) parameters instead.
3. **Asymmetric convolutions:** A 3×3 convolution could be replaced by a 1×3 convolution followed by a 3×1 convolution. If a 3×3 convolution is replaced by a 2×2 convolution, the number of parameters would be slightly higher than the asymmetric convolution proposed.

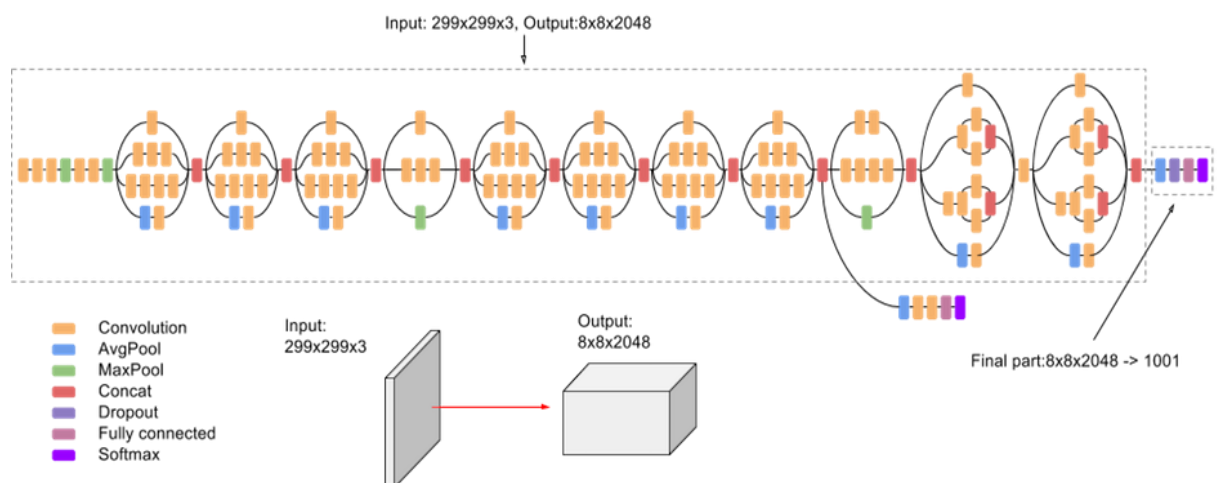


4. **Auxiliary classifier:** an auxiliary classifier is a small CNN inserted between layers during training, and the loss incurred is added to the main network loss. In Inception v3 an auxiliary classifier act as a regularizer.

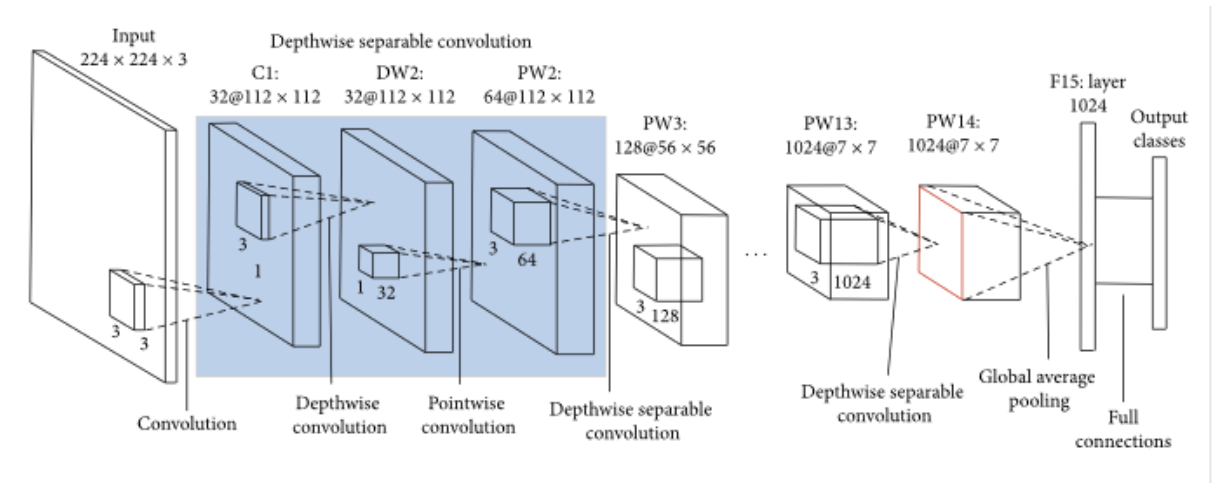
5. **Grid size reduction:** Grid size reduction is usually done by pooling operations. However, to combat the bottlenecks of computational cost, a more efficient technique is proposed:



All the above concepts are consolidated into the final architecture.



Mobile Net



What is Mobile Net?

Mobile Net model is designed to be used in mobile applications, and it is TensorFlow's first mobile computer vision model.

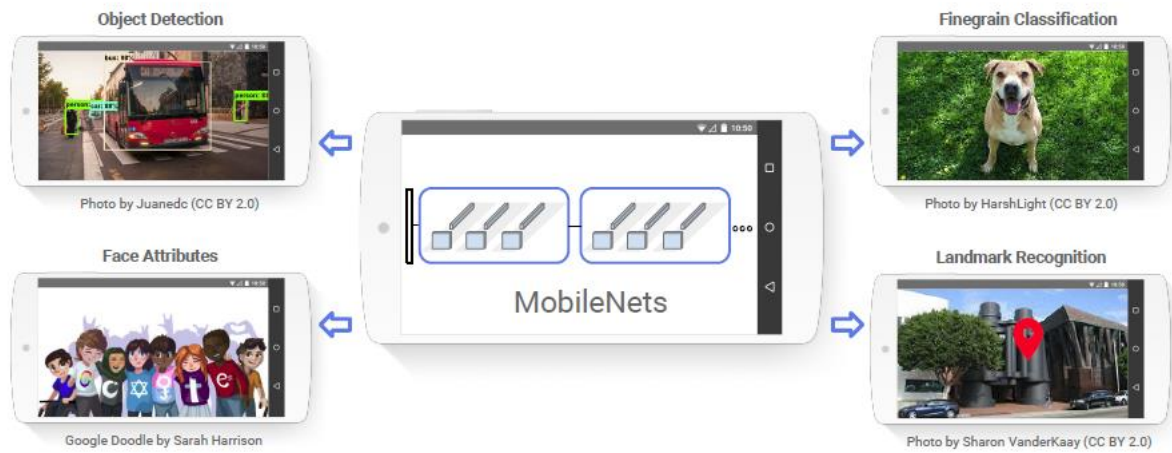
Mobile Net uses **depth wise separable convolutions**. It significantly **reduces the number of parameters** when compared to the network with regular convolutions with the same depth in the nets. This results in lightweight deep neural networks.

A depth wise separable convolution is made from two operations.

Depth wise convolution.

Pointwise convolution.

Mobile Net is a class of CNN that was open-sourced by Google, and therefore, this gives us an excellent starting point for training our classifiers that are insanely small and insanely fast.



The speed and power consumption of the network is proportional to the number of MACs (Multiply-Accumulates) which is a measure of the number of fused Multiplication and Addition operations.

Layers of Mobile Net Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5 \times$	Conv dw / s1 $3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	Conv / s1 $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Vgg19 Architecture

VGG19 is a variant of the VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer). There are other variants of VGG like VGG11, VGG16 and others. VGG19 has **19.6 billion FLOPs**.

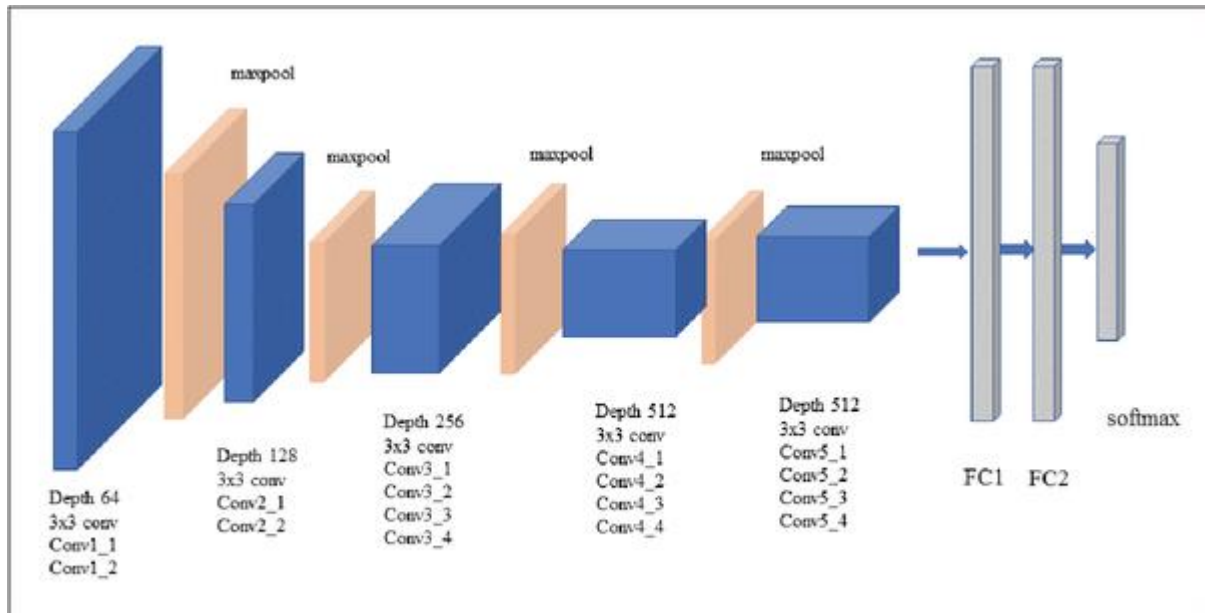


Fig. 3. VGG-19 network architecture

Architecture

- Fixed size of (224 * 224) RGB image was given as input to this network which means that the matrix was of shape (224,224,3).
- The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set.
- Used kernels of (3 * 3) size with a stride size of 1 pixel, this enabled them to cover the whole notion of the image.
- spatial padding was used to preserve the spatial resolution of the image.
- max pooling was performed over a 2 * 2 pixel windows with stride 2.
- this was followed by Rectified linear unit(ReLU) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used tanh or sigmoid functions this proved much better than those.
- implemented three fully connected layers from which the first two were of size 4096 and after that, a layer with 1000 channels for 1000-way *ILSVRC* classification and the final layer is a softmax function.

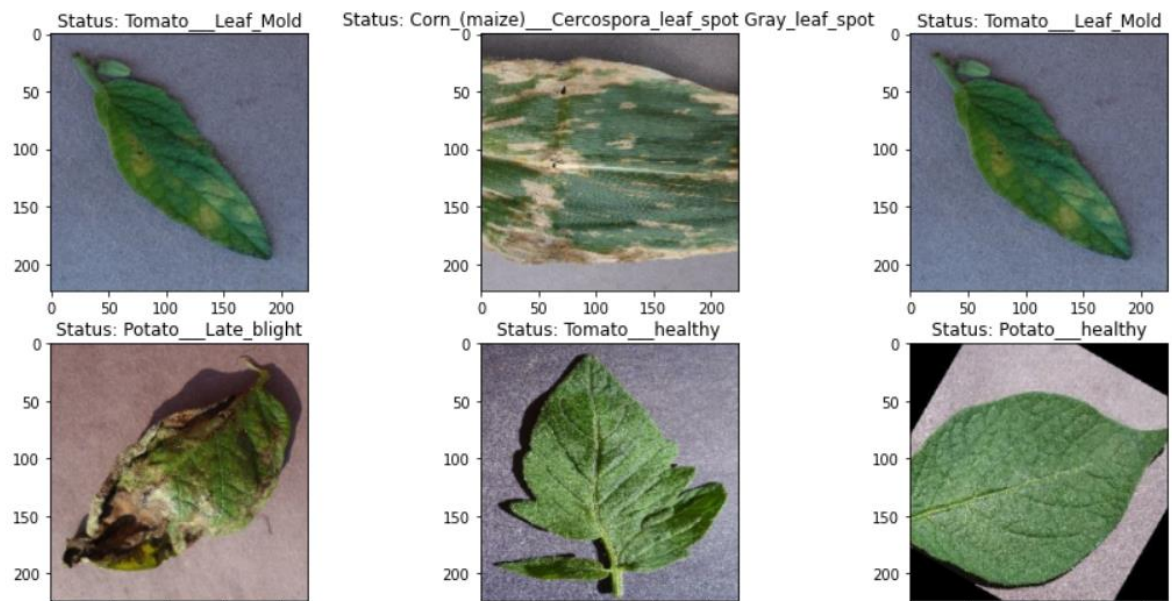
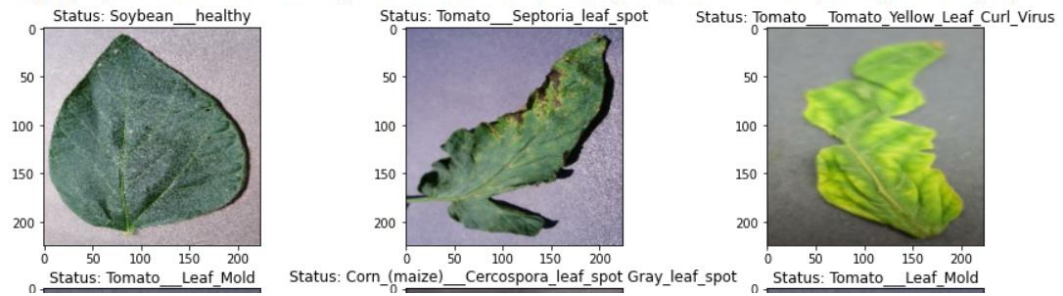
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

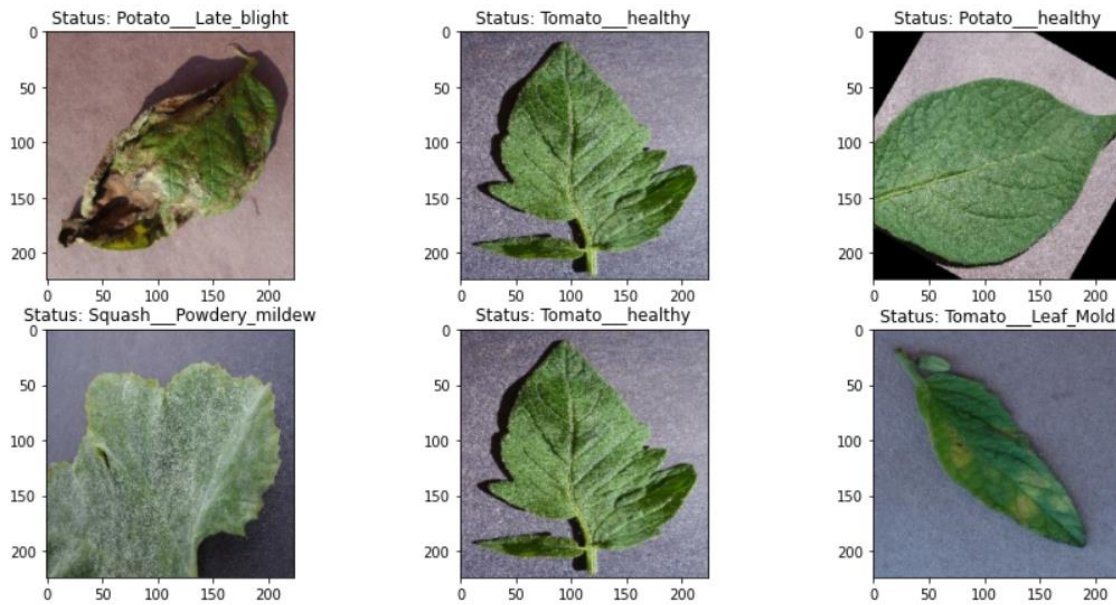
How does it look like?

```
✓ [10] classes = train_path.class_names
```

```
✓ [11] fig,ax=plt.subplots(4,3)
fig.set_size_inches(15,15)
for next_element in train_path:
    x_batch, y_batch = next_element
    for i in range(0,4):
        for j in range(3):
            random_example = np.random.randint(0, BATCH_SIZE)
            ax[i,j].imshow(x_batch[random_example]/255)
            ax[i,j].set_title('Status: ' + classes[y_batch[random_example].numpy()])
        break
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).





We note that the data is already augmented. This is relevant to have a predictive model that generalizes well: the predictions will not be dependent on the quality of the image, or the rotation. Data augmentation in image processing is mainly the following operation on the original image: rotating/flipping, blurring.

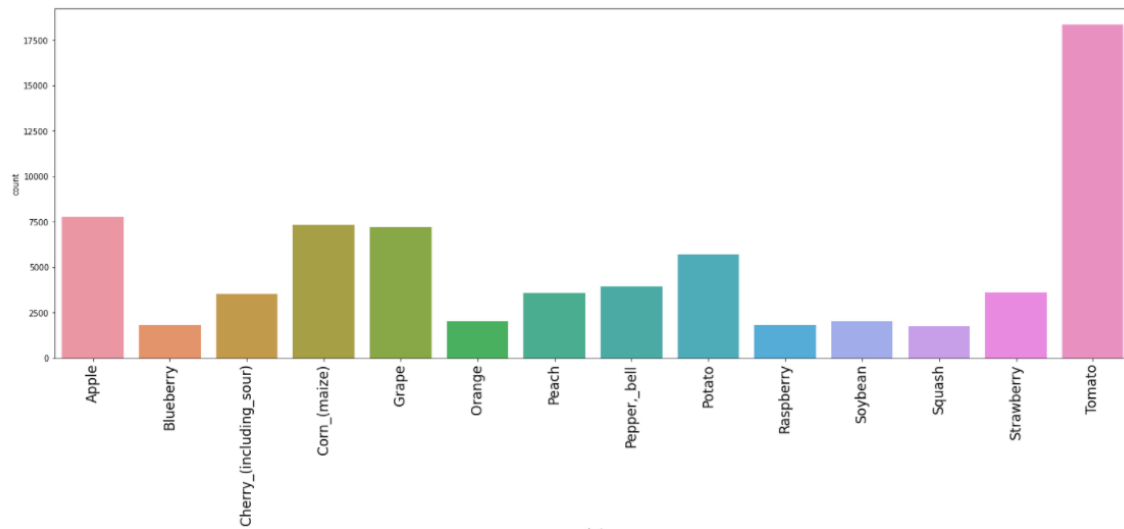
Visualisation:

Number of images for the particular plant

```
import seaborn as sns
plt.figure(figsize=(24,8))
sns.countplot(train_dataset_df['plant'])
plt.xticks(rotation=90,fontsize=17)
```

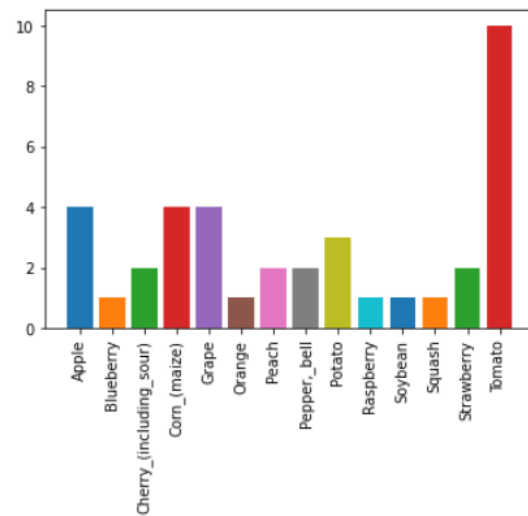
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
FutureWarning

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13]),  
<a list of 14 Text major ticklabel objects>)
```



Number of diseases present in that plant including healthy one

```
In [90]: for i in train_dataset_df['plant'].unique():
plt.bar(i,train_dataset_df[train_dataset_df['plant']== i]['diseases'].nunique())
plt.xticks(rotation=90)
```



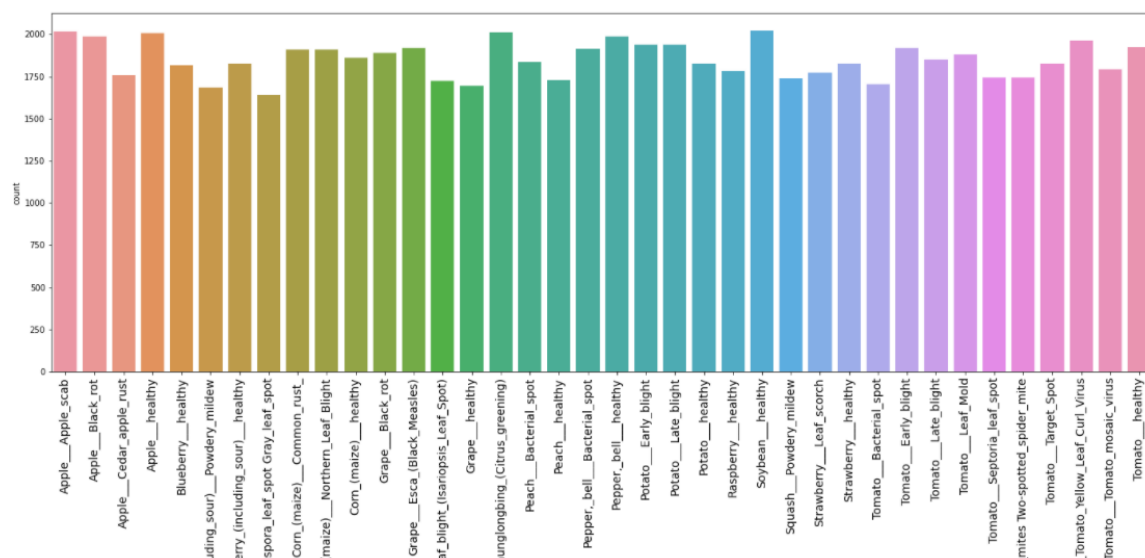
Distribution Of number of images per class:

```
: import seaborn as sns
plt.figure(figsize=(24,8))
sns.countplot(train_dataset_df['Labels'])
plt.xticks(rotation=90,fontsize=14)
```

/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

FutureWarning

```
: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37]), <a list of 38 Text major ticklabel objects>)
```



Model Used :

Vgg16 :

accuracy: 0.9847

val_accuracy: 0.9439

Inception v3:

accuracy: 96%

val_accuracy: 0.9525

Mobilenet

accuracy: 0.9904

val_accuracy: 0.9801

MODEL: Mobilenet

BATCH_SIZE = 32

EPOCHS=5

Metrics='accuracy'

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

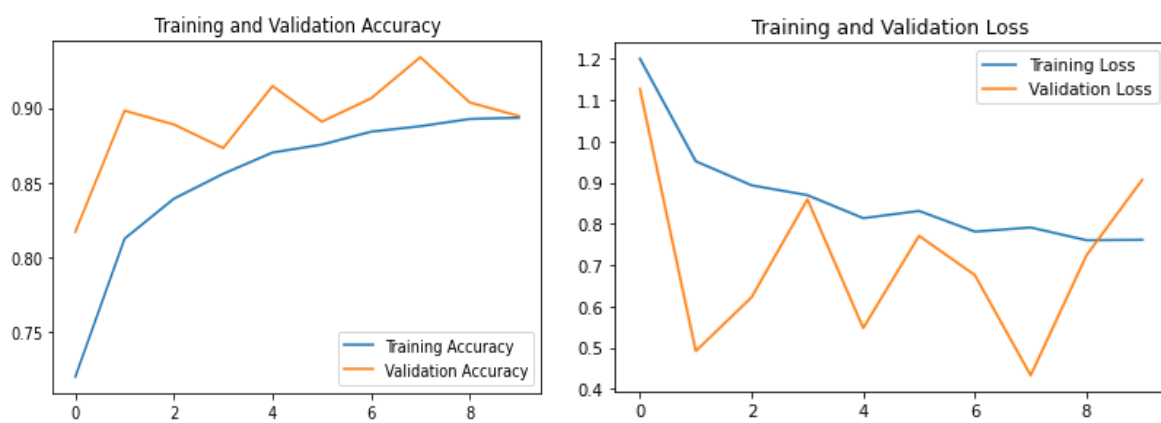
plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Vgg19 :

accuracy: 0.8934

val_accuracy: 0.8945



PROJECT EXECUTION:

Under the guidance of Nag Sir. (Data Scientist
Innomatics Research Labs, Hyderabad)



Thank You Sir From Team 3PMS