Laboratoire d'analyse et d'architecture des systemes

Internship Report

Plant Disease Classsification

Majeed Hussain

10.09.2019

Supervised by
Ariane Herbulot
Supervised by
Michel Devy

Contents

1 Objective **1**

2 Introduction **3**

3 Software and Tools **5**

4 Procedure **7**
4.1 Dataset . . . 7
4.2 Convolutional Neural Network . . . 7
4.2.1 Image Classification . . . 9
4.3 Experiments Done . . . 18
4.3.1 Data Augmentation . . . 21
4.3.2 Transfer Learning . . . 22
4.3.3 VGG16 . . . 25
4.3.4 Resnet . . . 28
4.4 Visualization . . . 32
4.4.1 Activation Maximization . . . 33
4.4.2 Saliency maps . . . 34
4.4.3 Class Activation maps . . . 36
4.5 Embedding . . . 38
4.6 Graphical User Interface . . . 38
4.7 iOS Application . . . 39

5 Results **41**
5.1 Custom Model . . . 41
5.2 VGG16 . . . 42
5.3 Resnet 18 . . . 43
5.4 Resnet 34 . . . 45
5.5 Graphical User Interface . . . 47
5.6 Embedding Visualization . . . 48
5.7 Neural Network Visualization . . . 49
5.8 iOS Application . . . 52

6 How to Run the Code **55**
6.1 Install the Dependencies . . . 55
6.2 Run GUI . . . 55
6.3 Visualize Embeddings . . . 55

6.4 Run iOS app 56

7 Future Work **57**

8 Conclusion **59**

Bibliography **61**

Bibliography **63**

Annex **65**

1 Objective

The Objective for this Internship is to do Plant Disease Classification using Deep learning Algorithms on Agricultural robot.

Image classification is a procedure to automatically categorize all pixels in an Image of a terrain into land cover classes. Normally, multispectral data are used to Perform the classification of the spectral pattern present within the data for each pixel is used as the numerical basis for categorization. This concept is dealt under the Broad subject, namely, Pattern Recognition. Spectral pattern recognition refers to the Family of classification procedures that utilizes this pixel-by-pixel spectral information as the basis for automated land cover classification. Spatial pattern recognition involves the categorization of image pixels on the basis of the spatial relationship with pixels surrounding them. Image classification techniques are grouped into two types, namely supervised and unsupervised[1]. The classification process may also include features, Such as, land surface elevation and the soil type that are not derived from the image.

We perform here a supervised learning using deep learning techniques to accomplish the task.A supervised classification algorithm requires a training sample for each class, that is, a collection of data points known to have come from the class of interest. The classification is thus based on how âcloseâ a point to be classified is to each training sample. We shall not attempt to define the word âcloseâ other than to say that both Geometric and statistical distance measures are used in practical pattern recognition algorithms. The training samples are representative of the known classes of interest to the analyst. Classification methods that relay on use of training patterns are called supervised classification methods. The three basic steps involved in a typical supervised classification procedure are as follows:

Training stage: The analyst identifies representative training areas and develops numerical descriptions of the spectral signatures of each land cover type of interest in the scene.

The classification stage: Each pixel in the image data set IS categorized into the land cover class it most closely resembles. If the pixel is insufficiently similar to any training data set it is usually labeled âUnknown'.

The output stage: In this final stage we get the Probabilities of classes to the corresponding input examples.

2 Introduction

Agriculture and farming are the professions for most of the people. Due to increase in technology the ways of farming is changing rapidly. Agriculture is a major industry and a huge part of the foundation of our economy. The rise in Artificial Intelligence is playing a key role in most of the fields. As climates are changing and populations are increasing, AI is becoming a technological innovation that is improving and protecting crop yield in many parts of the world.

For this Project to detect diseases occur in vineyard we use Neural Networks.Neural networks, as its name suggests, is a machine learning technique which is modeled after the brain structure. It comprises of a network of learning units called neurons. These neurons learn how to convert input signals (e.g. picture of a cat) into corresponding output signals (e.g. the label âcatâ), forming the basis of automated recognition.There are different types of Neural networks depending upon the problem. Here we use Convolutional Neural network which are widely popular for Image Classification and Recognition.

Lets take the example of automatic image recognition. The process of determining whether a picture contains a cat involves an activation function. If the picture resembles prior cat images the neurons have seen before, the label âcatâ would be activated. Hence, the more labelled images the neurons are exposed to, the better it learns how to recognize other unlabelled images. We call this the process of training neurons.The intelligence of neural networks is uncanny. While artificial neural networks were researched as early in 1960s by Rosenblatt, it was only in late 2000s when deep learning using neural networks took off. The key enabler was the scale of computation power and datasets with Google pioneering research into deep learning. In July 2012, researchers at Google exposed an advanced neural network to a series of unlabelled, static images sliced from YouTube videos. To their surprise, they discovered that the neural network learned a cat-detecting neuron on its own.

There different kinds of neural networks depending upon the tasks for Image recognition Convolutional neural networks are widely popular. It differs from regular neural networks in terms of the flow of signals between neurons. Typical neural networks pass signals along the input-output channel in a single direction, without allowing signals to loop back into the network. This is called a forward feed.While forward feed networks were successfully employed for image and text recognition, it required all neurons to be connected, resulting in an overly-complex network structure. The cost of complexity grows when the network has to be trained on large datasets which, coupled with the limitations of computer processing speeds, result in grossly long training times.

Hence, forward feed networks have fallen into disuse from mainstream machine learning in today's high resolution, high bandwidth, mass media age. A new solution was needed.

In 1986, researchers Hubel and Wiesel were examining a cat's visual cortex when they discovered that its receptive field comprised sub-regions which were layered over each other to cover the entire visual field. These layers act as filters that process input images, which are then passed on to subsequent layers. This proved to be a simpler and more efficient way to carry signals. In 1998, Yann LeCun and Yoshua Bengio tried to capture the organization of neurons in the cat's visual cortex as a form of artificial neural net, establishing the basis of the first CNN. In the same year a breakthrough for CNN was happened when researchers namely Bengio, Le Cun, Bottou and Haffner released a paper on CNNs called **"LeNet-5"** it was able to classify the handwritten numbers.The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images.

3 Software and Tools

Here we list out all the software's and tools used in this Project. There are many deep learning libraries like tensorflow from Google, Pytorch from Facebook, CNTK from Microsoft and also API's like Keras, Fastai and so on. Here in this Project I used Keras since I'm very much comfortable with it. Following are all the libraries used in this Project.

- **Keras**: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.
- **Matplotlib**: Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.
- **Numpy**: NumPy is the fundamental package for scientific computing with Python.
- **OpenCV**: OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.
- **Scikit-learn**: Scikit-learn is a free software machine learning library for the Python programming language.
- **PyQT**: PyQt is a set of Python v2 and v3 bindings for The Qt Company's Qt application framework and runs on all platforms supported by Qt including Windows, OS X, Linux, iOS and Android. We built a Graphical User Interface with this library.
- **Keras-vis**: keras-vis is a high-level toolkit for visualizing and debugging your trained keras neural net models. Currently supported visualizations include Activation maximization,Saliency maps,Class activation maps.
- **Lucid-keras**: Lucid is a collection of infrastructure and tools for research in neural network interpretability. We use this library to visulize the neural networks.
- **Coremltools**: Core ML community tools contains all supporting tools for Core ML model conversion and validation. We convert our keras model to Core ML to deploy our app on ios platform.
- **Tensorboard**: One of the famous visualization tools is Tensorboard. It is used to visualize your TensorFlow graph, plot quantitative metrics about the execution of your graph, and show additional data like images that pass through it.

4 Procedure

4.1 Dataset

In this Project we combined various data some are taken from Kaggle and and few Annotated data. The Dataset Contains seven categories of Plant diseases, they are as follows

- **Black Measles**
- **Blackrot**
- **Isariopsis**
- **No Disease**
- **MDB**
- **Mildiou**
- **Spyder**

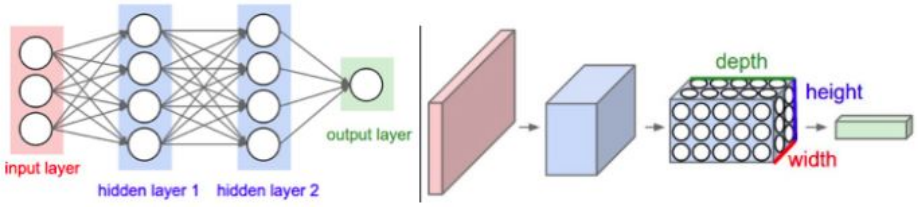
Before we begin the Procedure let's try to understand what are Convolutional Networks and how does they work.

4.2 Convolutional Neural Network

Convolutional neural networks â CNNs or convnets for short â are at the heart of deep learning, emerging in recent years as the most prominent strain of neural networks in research. They have revolutionized computer vision, achieving state-of-the-art results in many fundamental tasks, as well as making strong progress in natural language processing, computer audition, reinforcement learning, and many other areas. Convnets have been widely deployed by tech companies for many of the new services and features we see today.Convolutional neural networks. Sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15% percent, an astounding improvement at the time.Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google

for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

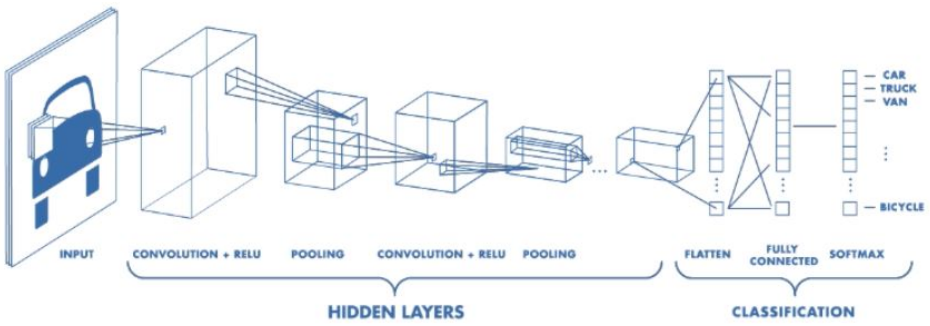
Convolutional Neural Networks are a bit different. First of all, the layers are organised in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Normal NN vs CNN. — Source: http://cs231n.github.io/convolutional-networks/

This CNN's will perform series of convolutions and pooling operations during which features are extracted and the last fully connected layers will serve as classifiers, it gives the predictions of the class the object belongs to in terms of probabilities.



Architecture of a CNN. — Source: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

However, the classic, and arguably most popular, use case of these networks is for image processing. Within image processing, let's take a look at how to use these CNNs for image classification.

4.2.1 Image Classification

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.

What We See

What Computers See

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for cat, .15 for dog, .05 for bird, etc).

Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat. This is the process that goes on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional

layers. This is a general overview of what a CNN does. Lets get into the specifics.

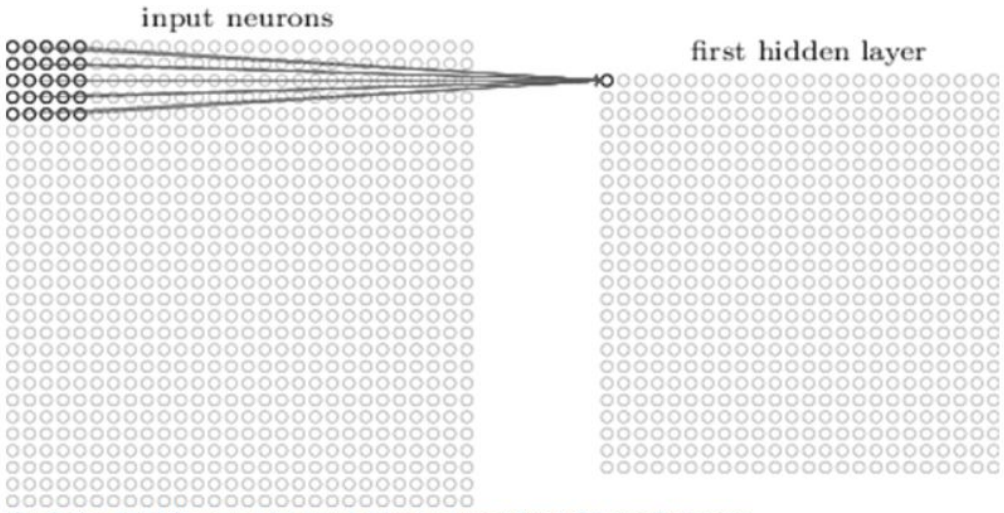
But first, a little background. When you first heard of the term convolutional neural networks, you may have thought of something related to neuroscience or biology, and you would be right. Sort of. CNNs do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks (the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well, and is the basis behind CNNs.

A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now, the hard part is understanding what each of these layers do. So lets get into the most important one.

First Layer

The first layer in a CNN is always a Convolutional Layer. First thing to make sure you remember is what the input to this conv (I'll be using that abbreviation a lot) layer is. Like we mentioned before, the input is a 32 x 32 x 3 array of pixel values. Now, the best way to explain a conv layer is to imagine a flashlight that is shining over the top left of the image. Let's say that the light this flashlight shines covers a 5 x 5 area. And now, let's imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a filter(or sometimes referred to as a neuron or a kernel) and the region that it is shining over is called the receptive field. Now this filter is also an array of numbers (the numbers are called weights or parameters). A very important note is that the depth of this filter has to be the same as the depth of the input (this makes sure that the math works out), so the dimensions of this filter is 5 x 5 x 3. Now, let's take the first position the filter is in for example. It would be the top left corner. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing element wise multiplications). These multiplications are all summed up (mathematically speaking, this would be 75 multiplications in total). So now you have a single number. Remember, this number is just representative of when the filter is at the top left of the image. Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right again

by 1, and so on). Every unique location on the input volume produces a number. After sliding the filter over all the locations, you will find out that what you're left with is a 28 x 28 x 1 array of numbers, which we call an activation map or feature map. The reason you get a 28 x 28 array is that there are 784 different locations that a 5 x 5 filter can fit on a 32 x 32 input image. These 784 numbers are mapped to a 28 x 28 array.Let's say now we use two 5 x 5 x 3 filters instead of one. Then our output volume would be 28 x 28 x 2. By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

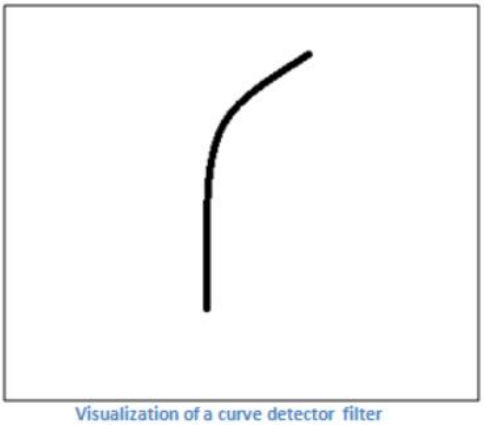


Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

However, lets talk about what this convolution is actually doing from a high level. Each of these filters can be thought of as feature identifiers. When I say features, I'm talking about things like straight edges, simple colors, and curves. Think about the simplest characteristics that all images have in common with each other. Let's say our first filter is 7 x 7 x 3 and is going to be a curve detector. In this section, let's ignore the fact that the filter is 3 units deep and only consider the top depth slice of the filter and the image, for simplicity. As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve .

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

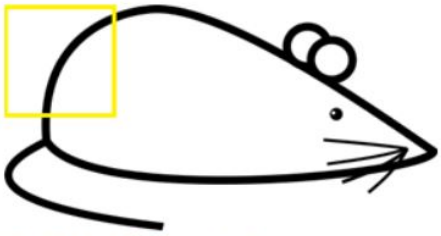
Pixel representation of filter



Visualization of a curve detector filter

Now, let's go back to visualizing this mathematically. When we have this filter at the top left corner of the input volume, it is computing multiplications between the filter and pixel values at that region. Now let's take an example of an image that we want to classify, and let's put our filter at the top left corner.

Original image



Visualization of the filter on the image

Remember, what we have to do is multiply the values in the filter with the original pixel values of the image.

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

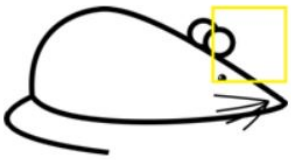
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter.



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

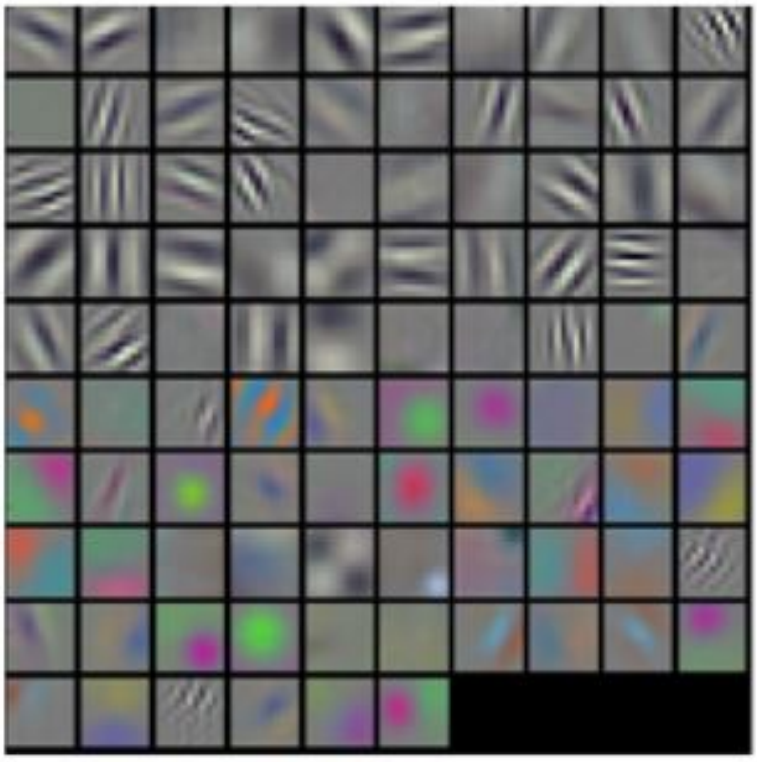
$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

The value is much lower. This is because there wasnt anything in the image section that responded to the curve detector filter. Remember, the output of this conv layer is an activation map. So, in the simple case of a one filter convolution (and if that filter is a curve detector), the activation map will show the areas in which there at mostly likely to be curves in the picture. In this example, the top left value of our 26 x 26 x 1 activation map (26 because of the 7x7 filter instead of 5x5) will be 6600. This high value means that it is likely that there is some sort of curve in the input volume that caused the filter to activate. The top right value in our activation map will be 0 because there wasnt anything in the input volume that caused the filter to activate. Remember this is just for one filter. This is just a filter that is going to detect lines that curve outward and to the right. We can have other filters for lines that curve to the left or for straight edges. The more filters, the greater the depth of the activation map, and the more information we have about the input volume.

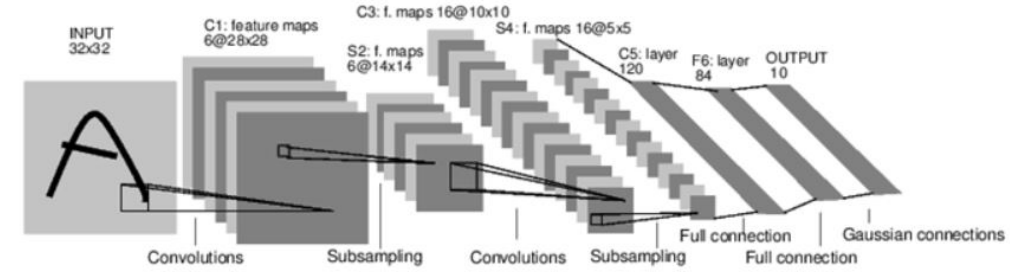


Now in a traditional convolutional neural network architecture, there are other layers that are interspersed between these conv layers. I'd strongly encourage those interested to read up on them and understand their function and effects, but in a general sense, they provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control overfitting. A classic CNN architecture would look like this.

The last layer, however, is an important one and one that we will go into later on. Lets just take a step back and review what weve learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would imagine, in order to predict whether an image is a type of object, we need the network to be able to recognize higher level features such as hands or paws or ears. So let's think about what the output of the network is after the first conv layer. It would be a 28 x 28 x 3 volume (assuming we use three 5 x 5 x 3 filters). When we go through another conv layer, the output of the first conv layer becomes the input of the 2nd conv layer. Now, this is a little bit harder to visualize. When we were talking about the first layer, the input was just the original image. However, when we're talking about the 2nd conv layer, the input is the activation map(s) that result from the first layer. So each layer of the input is basically describing the locations in the original image for where certain low level features appear. Now when you apply a set of filters on top of that (pass it through the 2nd conv layer), the output will be activations that represent higher level features. Types of these features could be semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As you go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, you may have some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc. If you want more information about visualizing filters in ConvNets, Matt Zeiler and Rob Fergus had an excellent research paper discussing the topic. Jason Yosinski also has a video on YouTube that provides a great visual representation. Another interesting thing to note is that as you go deeper into the network, the filters begin to have a larger and larger receptive field, which means that they are able to consider information from a larger area of the original input volume.

Now that we can detect these high level features, the icing on the cake is attaching a fully connected layer to the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from. For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class. For example, if the resulting vector for a digit classification program is [0 .1 .1 .75 0 0 0 0 0 .05], then this represents a 10% probability that the image is a 1, a 10% probability that the image is a 2, a 75% probability that the image is a 3, and a 5% probability that the image is a 9 (Side note: There are other ways that you can represent the output, but I am just showing the softmax approach). The way this fully connected layer works is that it looks at the output of the previous layer (which as we remember should represent the activation maps of high level features) and determines which features most correlate to a particular class.

For example, if the program is predicting that some image is a dog, it will have high values in the activation maps that represent high level features like a paw or 4 legs, etc. Similarly, if the program is predicting that some image is a bird, it will have high values in the activation maps that represent high level features like wings or a beak, etc. Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.



A Full Convolutional Neural Network (LeNet)

Next Part is Training, it is probably the most important part. There may be a lot of questions you had while reading. How do the filters in the first conv layer know to look for edges and curves? How does the fully connected layer know what activation maps to look at? How do the filters in each layer know what values to have? The way the computer is able to adjust its filter values (or weights) is through a training process called backpropagation.

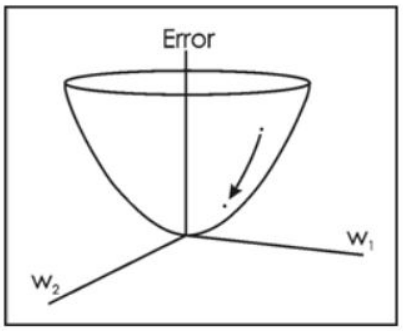
Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didnt know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values are randomized. The filters dont know to look for edges and curves. The filters in the higher layers dont know to look for paws and beaks. As we grew older however, our parents and teachers showed us different pictures and images and gave us a corresponding label. This idea of being given an image and a label is the training process that CNNs go through. Before getting too into it, lets just say that we have a training set that has thousands of images of dogs, cats, and birds and each of the images has a label of what animal that picture is.

So backpropagation can be separated into 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. During the forward pass, you take a training image which as we remember is a 32 x 32 x 3 array of numbers and pass it through the whole network. On our first training example, since all of the weights or filter values were randomly initialized, the output will probably be something like [.1 .1

.1 .1 .1 .1 .1 .1 .1 .1 .1], basically an output that doesn't give preference to any number in particular. The network, with its current weights, isn't able to look for those low level features or thus isn't able to make any reasonable conclusion about what the classification might be. This goes to the loss function part of backpropagation. Remember that what we are using right now is training data. This data has both an image and a label. Let's say for example that the first training image inputted was a 3. The label for the image would be [0 0 0 1 0 0 0 0 0 0]. A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is Â$\frac{1}{2}$ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Let's say the variable L is equal to that value. As you can imagine, the loss will be extremely high for the first couple of training images. Now, let's just think about this intuitively. We want to get to a point where the predicted label (output of the ConvNet) is the same as the training label (This means that our network got its prediction right).In order to get there, we want to minimize the amount of loss we have. Visualizing this as just an optimization problem in calculus, we want to find out which inputs (weights in our case) most directly contributed to the loss (or error) of the network.



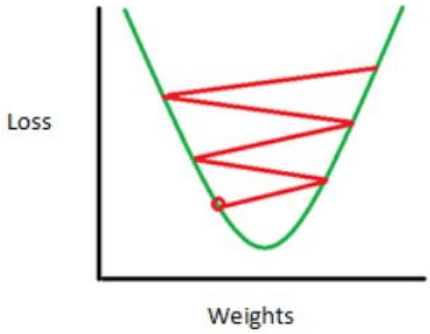
One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in our bowl shaped object. To do this, we have to take a derivative of the loss (visual terms: calculate the slope in every direction) with respect to the weights.

This is the mathematical equivalent of a dL/dW where W are the weights at a particular layer. Now, what we want to do is perform a backward pass through the network, which is determining which weights contributed most to the loss and finding ways to adjust them so that the loss decreases. Once we compute this derivative, we then go to the last step which is the weight update. This is where we take all the weights of the filters and update them so that they change in the opposite direction of the gradient.

$$w = w_i - \eta \frac{dL}{dW}$$

w = Weight
w_i = Initial Weight
η = Learning Rate

The learning rate is a parameter that is chosen by the programmer. A high learning rate means that bigger steps are taken in the weight updates and thus, it may take less time for the model to converge on an optimal set of weights. However, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal point.



Consequence of a high learning rate where the jumps are too large and we are not able to minimize the loss.

The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a batch). Once you finish the parameter update on the last training example, hopefully the network should be trained well enough so that the weights of the layers are tuned correctly.

Final Part would be the testing our trained model,we have a different set of images and labels and pass the images through the CNN. We compare the outputs to the ground truth and see if our network works.

4.3 Experiments Done

For this project we have used different architectures of model it includes custom models and also there are few Award winning model we will be using , we start with Building a Simple Custom Model. The Architecture of our Custom Model is as follows:

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 256, 256, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 85, 85, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 85, 85, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 85, 85, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 85, 85, 64)        256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 85, 85, 64)        36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 85, 85, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 42, 42, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 42, 42, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 42, 42, 128)       73856
_________________________________________________________________
batch_normalization_3 (Batch (None, 42, 42, 128)       512
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 42, 42, 128)       147584
_________________________________________________________________
batch_normalization_4 (Batch (None, 42, 42, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 21, 21, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 21, 21, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 56448)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              57803776
_________________________________________________________________
batch_normalization_5 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 7175
=================================================================
Total params: 58,094,471
Trainable params: 58,091,591
Non-trainable params: 2,880
_________________________________________________________________
```

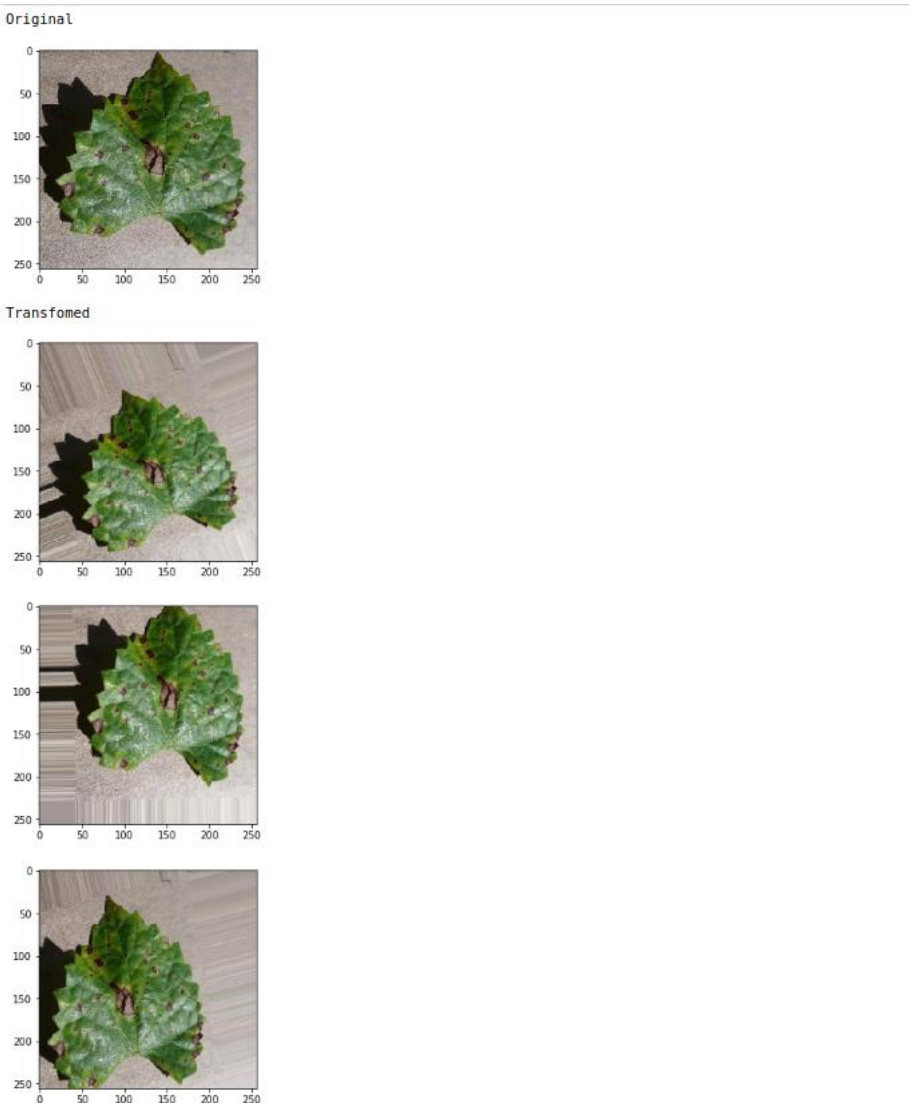
Using the above architecture we have acquired a good performance but to increase the accuracy more we can do Transfer learning with different techniques which we will be using in next section.

4.3.1 Data Augmentation

Later we used Data augmentation,it is a technique to make your existing dataset even larger, just with a couple easy transformations. when a computer takes an image as an input, it will take in an array of pixel values. Letâs say that the whole image is shifted left by 1 pixel. To you and me, this change is imperceptible. However, to a computer, this shift can be fairly significant as the classification or label of the image doesnât change, while the array does. Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation techniques. They are a way to artificially expand your dataset. Some popular augmentations people use are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more. By applying just a couple of these transformations to your training data, you can easily double or triple the number of training examples. keras has Image-generator class to generate the samples. We can perform many transformation some of them are listed below

- Scaling
- Translation
- Rotation (at 90 degrees)
- Rotation (at finer angles)
- Flipping
- Adding Salt and Pepper noise
- Lighting condition
- Perspective transform

Following example shows how a sample image looks like after applying random transformation.



After data augmentation we train the network again with this generated data to improve in accuracy, as of now we have used custom models let's do transfer learning.

4.3.2 Transfer Learning

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest. The three major Transfer Learning scenarios look as follows:

- **ConvNet as fixed feature extractor**: Take a VGG16 pretrained on ImageNet, remove the last fully-connected layer (this layerâs outputs are the 1000 class scores

for a different task like ImageNet), then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. In an AlexNet, this would compute a 4096-D vector for every image that contains the activations of the hidden layer immediately before the classifier. We call these features CNN codes. It is important for performance that these codes are ReLUd (i.e. thresholded at zero) if they were also thresholded during the training of the ConvNet on ImageNet (as is usually the case). Once you extract the 4096-D codes for all images, train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset.

- **Fine-tuning the ConvNet**: The second strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or itâs possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset. In case of ImageNet for example, which contains many dog breeds, a significant portion of the representational power of the ConvNet may be devoted to features that are specific to differentiating between dog breeds.

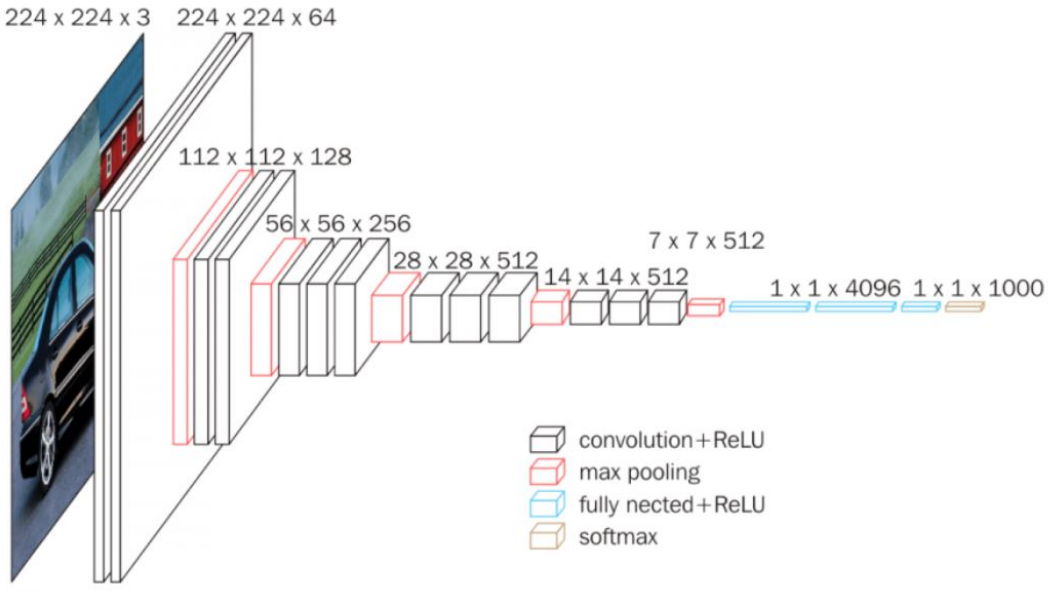
- **Pretrained models**: Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning. For example, the Caffe library has a Model Zoo where people share their network weights.

Following are the points to remember to know when to use transfer learning.

- **New dataset is small and similar to original dataset**: Since the data is small, it is not a good idea to fine-tune the ConvNet due to overfitting concerns. Since the data is similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes.
- **New dataset is large and similar to the original dataset**: Since we have more data, we can have more confidence that we wonât overfit if we were to try to fine-tune through the full network.
- **New dataset is small but very different from the original dataset**: Since the data is small, it is likely best to only train a linear classifier. Since the dataset is very different, it might not be best to train the classifier form the top of the network, which contains more dataset-specific features. Instead, it might work better to train the SVM classifier from activations somewhere earlier in the network.
- **New dataset is large and very different from the original dataset**: Since the dataset is very large, we may expect that we can afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. In this case, we would have enough data and confidence to fine-tune through the entire network.

4.3.3 VGG16

Keras provides convenient access to many top performing models on the ImageNet image recognition tasks such as VGG, Inception, and ResNet. Let's first use VGG16 model, It is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper âVery Deep Convolutional Networks for Large-Scale Image Recognitionâ. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3 by 3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPUâs. The architecture of VGG16 is shown in the below figure



The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3Ã3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1Ã1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). 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 by3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2Ã2 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.
The model summary of VGG16 is as follows

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

As we could see the final layer is of 1000 classes since it has trained on imagenet dataset we will remove that layer and replace with seven classes which is in our case and then freeze all the layers and just fine tune on final layer. The reason we are freezing is not to train all the layers again since it has trained on huge dataset and learned many patterns since we are adjusting by finetuning the last layer.After the modification on VGG16 model our final model's summary would be as follows.

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 256, 256, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 256, 256, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 256, 256, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 128, 128, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 128, 128, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 128, 128, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 64, 64, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 64, 64, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 64, 64, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 64, 64, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 32, 32, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 32, 32, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 32, 32, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 32, 32, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 16, 16, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 8, 8, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               16777728
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 3591
=================================================================
Total params: 31,496,007
Trainable params: 16,781,319
Non-trainable params: 14,714,688
```

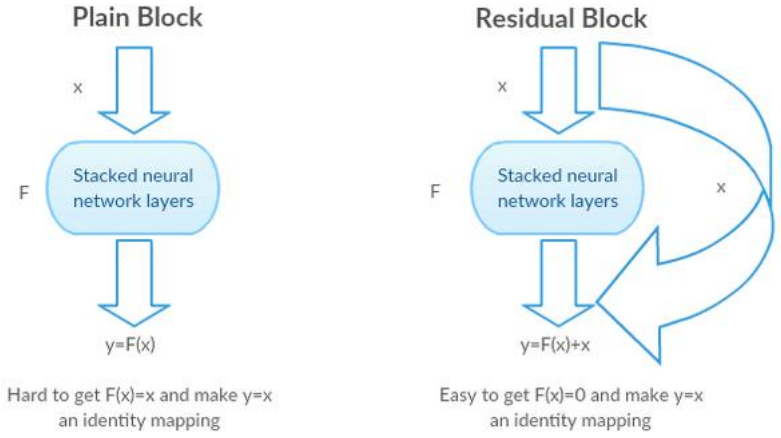
Now after training the above model we have got 92% of accuracy after few epochs.

4.3.4 Resnet

As we saw from above model results as the number of layers increases the accuracy also increases but increasing layers and having less data can lead to Overfitting issue and When deeper networks starts converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly. Inorder to deal this kind of problem we use Residual networks where the model performs very well even with adding many layers.

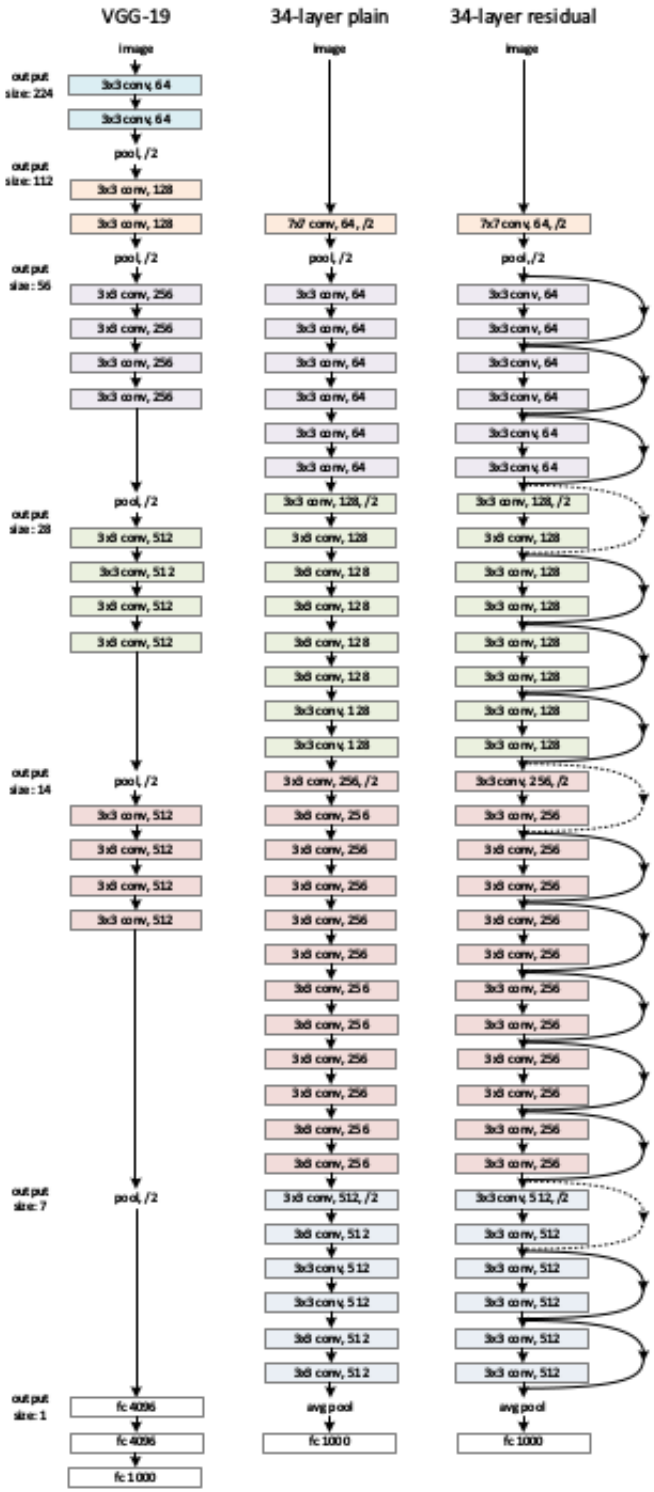
Imagine a deep CNN architecture. Take that, double the number of layers, add a couple more, and it still probably isn't as deep as the ResNet architecture that Microsoft Research Asia came up with in late 2015. ResNet is a new 152 layer network architecture that set new records in classification, detection, and localization through one incredible architecture. Aside from the new record in terms of number of layers, ResNet won ILSVRC 2015 with an incredible error rate of 3.6% Depending on their skill and expertise, humans generally hover around a 5-10% error rate.

So inorder to solve this problem we could do direct mapping of x -¿y with a function H(x) (A few stacked non-linear layers). Let us define the residual function using F(x) = H(x) â x, which can be reframed into H(x) = F(x)+x, where F(x) and x represents the stacked non-linear layers and the identity function(input=output) respectively.If the identity mapping is optimal, We can easily push the residuals to zero (F(x) = 0) than to fit an identity mapping (x, input=output) by a stack of non-linear layers. In simple language it is very easy to come up with a solution like F(x) =0 rather than F(x)=x using stack of non-linear cnn layers as function. So, this function F(x) is what the authors called Residual function.



Identity mapping in Residual blocks

Following is how the VGG19, plain 34 layer and Resnet connections lookalike



For residual network there are two connections

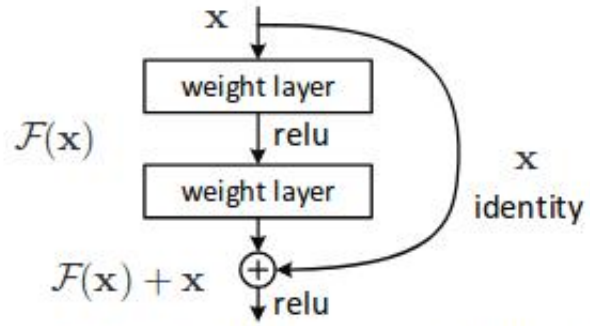


Figure 2. Residual learning: a building block.

Residual block

The identity shortcuts (x) can be directly used when the input and output are of the same dimensions.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \quad (1)$$

Residual block function when input and output dimensions are same

When the dimensions change, A) The shortcut still performs identity mapping, with extra zero entries padded with the increased dimension. B) The projection shortcut is used to match the dimension (done by 1*1 conv) using the following formula

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \quad (2)$$

Residual block function when the input and output dimensions are not same.

The first case adds no extra parameters, the second one adds in the form of W_s

There are different layered resnet here we used 18 and 34 layered networks

Resnet 18

In keras library resnet18 is missing so we used here image-classifiers library to use resnet18. Following is the architecture of resnet 18

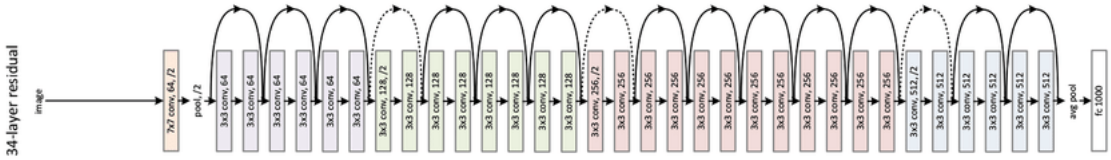
Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

ResNet-18 Architecture.

As we see the final softmax layer is 1000 classes but for our case we have only seven classes there fore we remove the final layer and replace with our number of classes value here I didn't use any previous weights the training has done on all the layers unlike the above VGG16 model. The accuracy obtained is around 98% with 50 epochs.

Resnet34

For this architecture too we used image-classifiers library. Following is the architecture of resnet 34



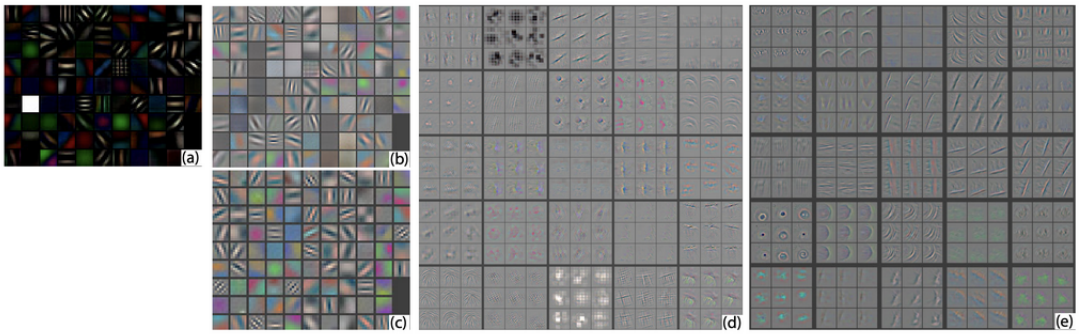
As we see here too the final softmax layer is 1000 classes but for our case we have only seven classes there fore we remove the final layer and replace with our number of classes value here I didn't use any previous weights the training has done on all the layers unlike the above VGG16 model. The accuracy obtained is around 99% with 50 epochs.

4.4 Visualization

Visualization helps us see what features are guiding the model's decision for classifying an image. In modern machine learning progress is impressive, it isnât with unique challenges. For example, the lack of interpretability and transparency of neural networks, from the learned features to the underlying decision processes, is an important problem to address. Making sense of why a particular model misclassifies data or behaves poorly can be challenging for model developers. Similarly, end-users interacting with an application that relies on deep learning to make decisions may question its reliability if no explanation is given by the model, or may become confused if the explanation is convoluted.While explaining neural network decisions is important, there are numerous other problems that arise from deep learning, such as AI safety and security (e.g., when using models that could affect a personâs social, financial, or legal wellbeing), and compromised trust due to bias in models and datasets, just to name a few. These challenges are often exacerbated due to the large datasets required to train most deep learning models. As worrisome as these problems are, they will likely become even more widespread as more AI-powered systems are deployed in the world. Therefore, a general sense of model understanding is not only beneficial, but often required to address the aforementioned issues.

Data visualization and visual analytics excel at communicating information and discovering insights by using visual encodings to transform abstract data into meaningful representations. For visualization in deep learning, in the seminal work by Zeiler and Fergus, a technique called deconvolutional networks enabled projection from a modelâs learned feature space back to the pixel space, or in other words, gave us a glimpse at what neural networks were seeing in large sets of images. Their technique and results give insight into what types of features deep neural networks are learning at specific layers in the model, and provide a debugging mechanism for improving a model. This work is often credited for popularizing visualization in the deep learning and computer vision communities in recent years, highlighting visualization as a powerful tool that

helps people understand and improve deep models.



A figure from "Visualizing and Understanding Convolutional Networks" by Zeiler and Fergus, 2013, shows early results for a technique called **feature visualization** that visualizes the learned features in intermediate hidden layers of a deep learning model.

However, visualization research for neural networks started well before. Now, over just a handful of years, many different techniques have been introduced to help interpret what neural networks are learning. For example, many techniques generate static visualizations indicating which parts of an image are most important to a modelâs classification. However, interaction has also been incorporated into visual analytics tools to help people understand a modelâs decision process. This hybrid research area has grown in both academia and industry, forming the basis for new research and communities that wish to explain models clearly.

In this Project we have used two main libraries for Visualization

- **lucid-keras**
- **Keras-vis**

Lucid: Lucid is a collection of infrastructure and tools for research in neural network interpretability. We tried to visualize few specific neurons there are alot things we can visualize using this library since lucid is built for tensorflow we have lucid-keras version which is made to work on keras code.

Keras-vis: keras-vis is a high-level toolkit for visualizing and debugging your trained keras neural net models. Currently supported visualizations include:

- Activation maximization
- Saliency maps
- Class activation maps

4.4.1 Activation Maximization

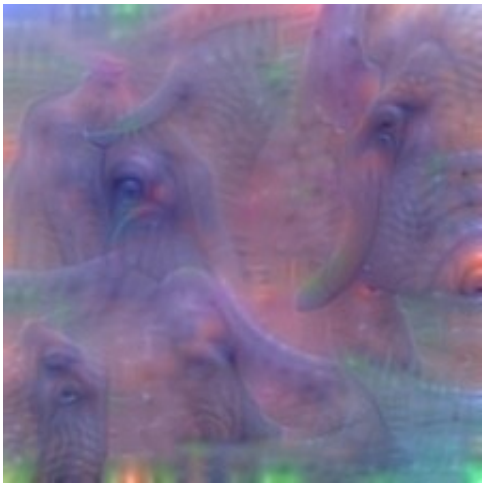
In a CNN, each Conv layer has several learned template matching filters that maximize their output when a similar template pattern is found in the input image. First Conv

layer is easy to interpret; simply visualize the weights as an image. To see what the Conv layer is doing, a simple option is to apply the filter over raw input pixels. Subsequent Conv filters operate over the outputs of previous Conv filters (which indicate the presence or absence of some templates), making them hard to interpret.

The idea behind activation maximization is simple in hindsight - Generate an input image that maximizes the filter output activations. i.e., we compute

$$\frac{\partial ActivationMaximizationLoss}{\partial input}$$

For example if we take an image of elephant and perform activation maximization then you will get the following output.



From the above image, we can observe that the model expects structures like a tusk, large eyes, and trunk. Now, this information is very important for us to check the sanity of our dataset. For example, letâs say that the model was focussing on features like trees or long grass in the background because Indian elephants are generally found in such habitats. Then, using activation maximization, we can figure out that our dataset is probably not sufficient for the task and we need to add images of elephants in different habitats to our training set.

4.4.2 Saliency maps

Suppose that all the training images of bird class contains a tree with leaves. How do we know whether the CNN is using bird-related pixels, as opposed to some other features

such as the tree or leaves in the image? This actually happens more often than you think and you should be especially suspicious if you have a small training set.Saliency maps was first introduced in the paper: Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.

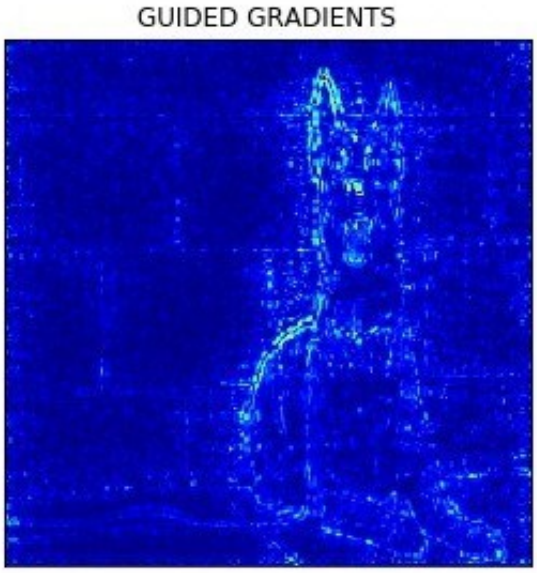
The idea is pretty simple. We compute the gradient of output category with respect to input image. This should tell us how output category value changes with respect to a small change in input image pixels. All the positive values in the gradients tell us that a small change to that pixel will increase the output value. Hence, visualizing these gradients, which are the same shape as the image should provide some intuition of attention.

The idea behind saliency is pretty simple in hindsight. We compute the gradient of output category with respect to input image.

$$\frac{\partial output}{\partial input}$$

This should tell us how the output value changes with respect to a small change in inputs. We can use these gradients to highlight input regions that cause the most change in the output. Intuitively this should highlight salient image regions that most contribute towards the output.let's see how to generate saliency maps for the following dog image.

The output image after guided backpropagation is as follows:



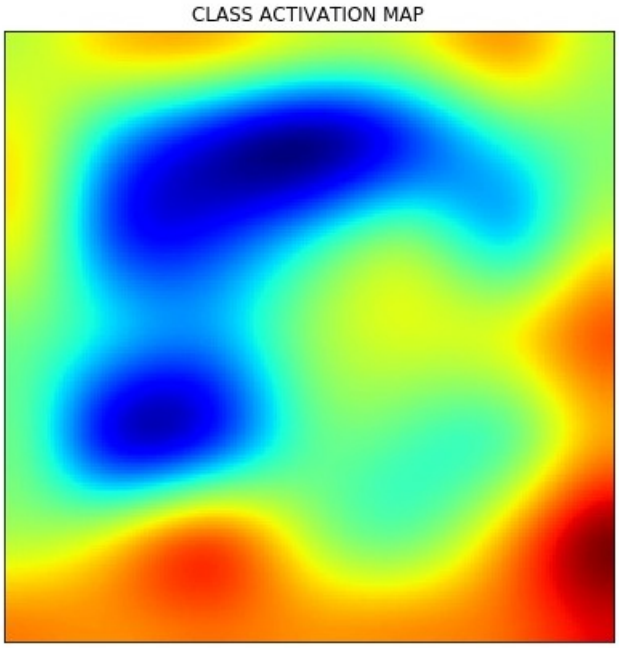
4.4.3 Class Activation maps

class activation maps or grad-CAM is another way of visualizing attention over input. Instead of using gradients with respect to output (see saliency), grad-CAM uses penultimate (pre Dense layer) Conv layer output. The intuition is to use the nearest Conv layer to utilize spatial information that gets completely lost in Dense layers.Grad-CAM involves the following steps:

- Take the output feature map of the final convolutional layer. The shape of this feature map is 14x14x512 for VGG16.
- Calculate the gradient of the output with respect to the feature maps.
- Apply Global Average Pooling to the gradients.
- Multiply the feature map with corresponding pooled gradients.

For example let's see the input image and its corresponding Class Activation Map below:

Now let's generate the Class activation map for the above image.



All the outputs for our plants disease are showed in results section.

4.5 Embedding

An embedding is a mapping of a discrete â categorical â variable to a vector of continuous numbers. In the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of discrete variables. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space.Neural network embeddings have 3 primary purposes:

- Finding nearest neighbors in the embedding space. These can be used to make recommendations based on user interests or cluster categories.
- As input to a machine learning model for a supervised task.
- For visualization of concepts and relations between categories.

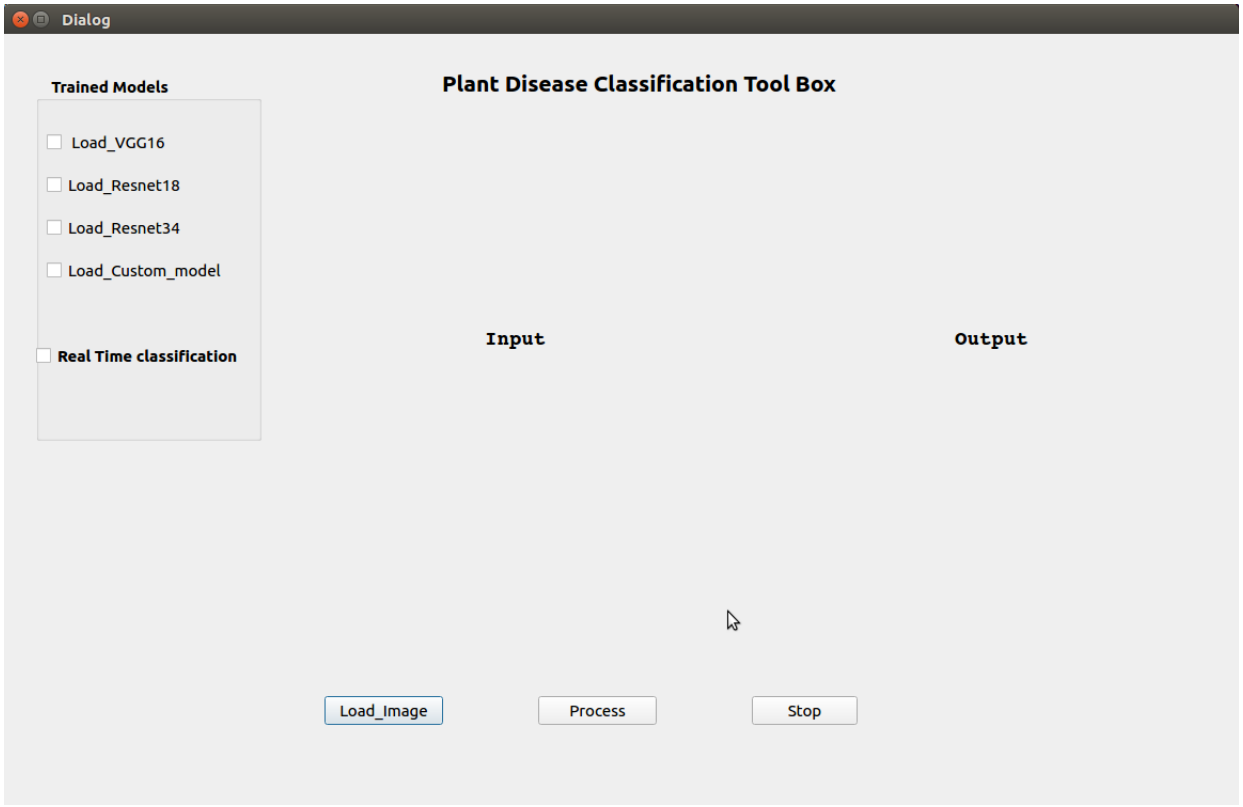
This means in terms of the book project, using neural network embeddings, we can take all training examples and represent as seven dimensional vector since seven classes i.e, final layer. Moreover, because embeddings are learned, samples that are more similar in the context of our learning problem are closer to one another in the embedding space.

In simpler terms we are passing our samples to our trained model as in the final layer it spits a seven dimensional vector i.e, final layer of model. We then save all the vectors information and project those vectors in space and look how close the clusters are formed. Each sample is nothing but a datapoint in embedding space. We will be using tensorboard for visualization. All the ouputs will be shown in result section.

4.6 Graphical User Interface

For User friendly experience I have created a GUI for our project using PyQT library. PyQt is a GUI widgets toolkit. It is a Python interface for Qt, one of the most powerful, and popular cross-platform GUI library. PyQt is a blend of Python programming language and the Qt library. PyQt is compatible with all the popular operating systems including Windows, Linux, and Mac OS. It is dual licensed, available under GPL as well as commercial license.

The Interface looks as follows:



The explanation for how to run this would be discussed in later section.

4.7 iOS Application

Deploying machine learning models into android or ios platform is pretty simple now a days since huge development in software there are lot of libraries for conversion of our trained models. I have used coreml convertor to convert our trained keras model into core Ml. Later used those model to deploy in iOS platform.

Core ML Tools is a Python package that converts a variety of model types into the Core ML model format. Table 1 lists the supported models and third-party frameworks.

Table 1 Models and third-party frameworks supported by Core ML Tools

Model type	Supported models	Supported frameworks
Neural networks	Feedforward, convolutional, recurrent	Caffe v1 Keras 1.2.2+
Tree ensembles	Random forests, boosted trees, decision trees	scikit-learn 0.18 XGBoost 0.6
Support vector machines	Scalar regression, multiclass classification	scikit-learn 0.18 LIBSVM 3.22
Generalized linear models	Linear regression, logistic regression	scikit-learn 0.18
Feature engineering	Sparse vectorization, dense vectorization, categorical processing	scikit-learn 0.18
Pipeline models	Sequentially chained models	scikit-learn 0.18

I have used Vision framework listed in Apple developer site and with some preprocessing functions and with converted ml model. I deployed the app in simulator. In order to Compile, Build, Run in simulator I have used Xcode 11.

Xcode 11: It includes everything you need to create amazing apps and to bring your apps to even more devices. Take advantage of SwiftUI, an all-new user interface framework with a declarative Swift syntax. Start bringing your iPad app to Mac with just a click. And with support for Swift packages, Xcode 11 lets you share code among all of your apps or use packages created by the community.

The Outputs will be displayed in results section.

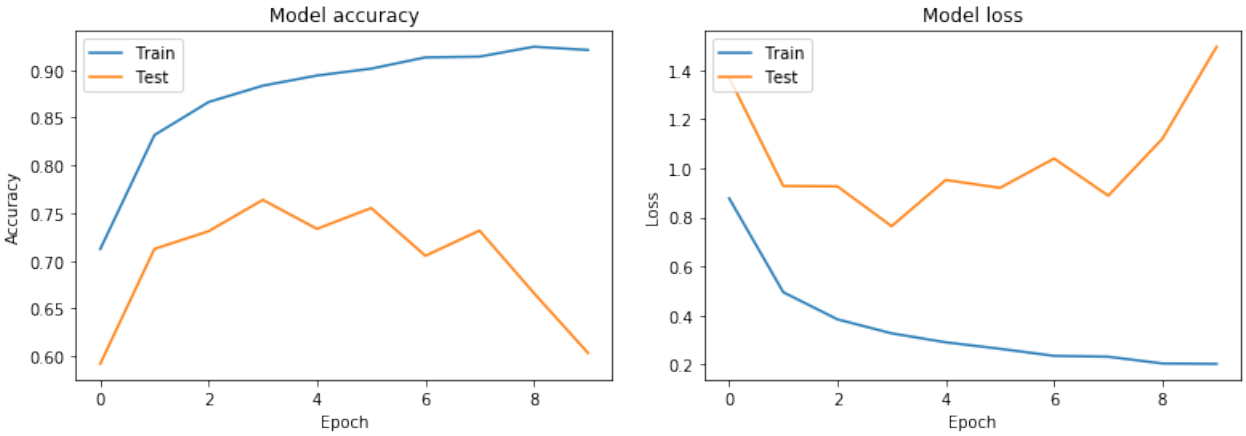
5 Results

5.1 Custom Model

Firstly we started out with Custom model so the results after training ten epochs obtained is just 60% accuracy as follows:

```
Epoch 00009: val_acc did not improve from 0.76393
Epoch 10/10
350/350 [==============================] - 147s 420ms/step - loss: 0.2020 - acc: 0.9210 - val_loss: 1
cc: 0.6036

Epoch 00010: val acc did not improve from 0.76393
```

And following are the plot of acuuracies and loss after each epoch



Later we trained the same model with augmented data you can see the improvement in accuracy no raised to 86%

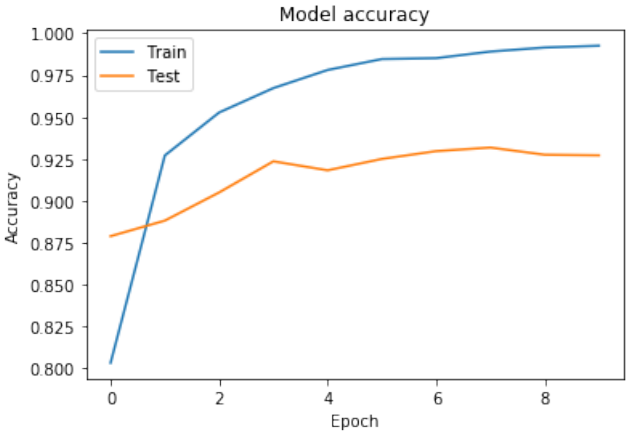
```
Epoch 00009: val_acc did not improve from 0.92179
Epoch 10/10
700/700 [==============================] - 198s 283ms/step - loss: 0.2140 - acc: 0.9207 - val_loss: 0.3
cc: 0.8679

Epoch 00010: val_acc did not improve from 0.92179
```

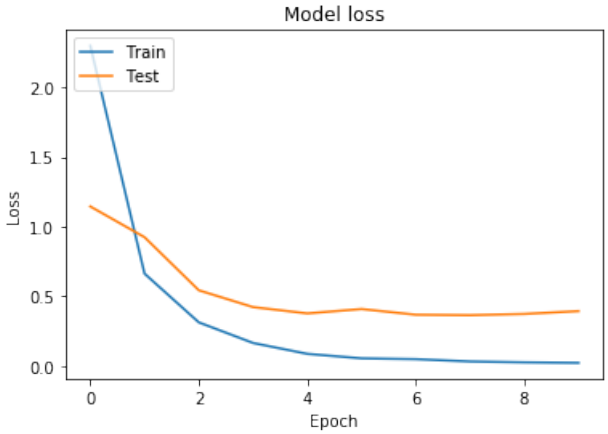
5.2 VGG16

Here we used transfer learning techniwue as explained in above section we have acquired an accuracy of 92% using VGG16 model

```
Epoch 00009: loss improved from 0.03371 to 0.02741, saving model to VGG16_1.h5
Epoch 10/10
350/350 [==============================] - 230s 657ms/step - loss: 0.0238 - acc: 0.9925 - val_lo
cc: 0.9271

Epoch 00010: loss improved from 0.02741 to 0.02383, saving model to VGG16_1.h5
```

Following are the training and validation plots



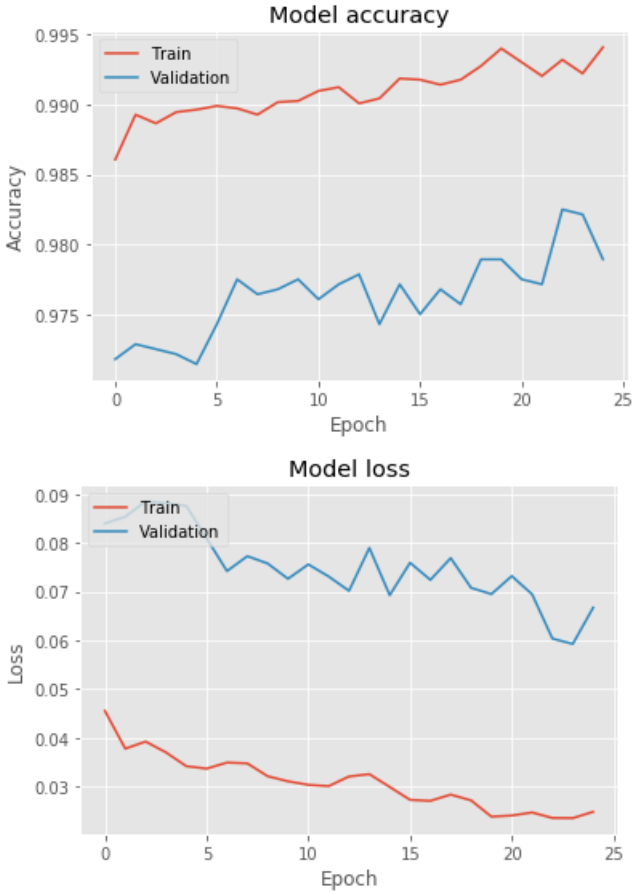


5.3 Resnet 18

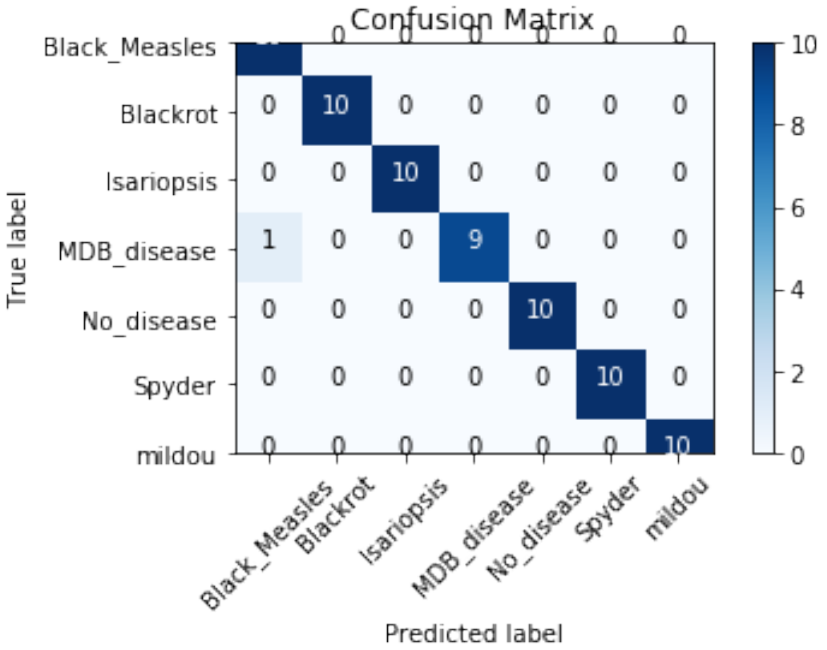
Later we have used Resnet 18 architecture for this model the training is done all the layers, here we obtained around 98% accuracy

```
Epoch 00049: loss improved from 0.02356 to 0.02354, saving model to resnet18.h5
Epoch 50/50
350/350 [==============================] - 167s 478ms/step - loss: 0.0248 - acc: 0.9941 - val_loss: 0.0
_acc: 0.9789

Epoch 00050: loss did not improve from 0.02354
```

Following are the training and validation accuracies of our resnet18 model



Following figure shows the Confusion matrix and Classification reports



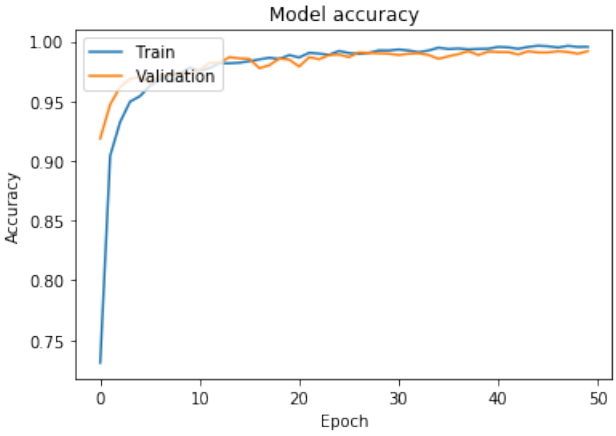
	precision	recall	f1-score	support
Black_Measles	0.91	1.00	0.95	10
Blackrot	1.00	1.00	1.00	10
Isariopsis	1.00	1.00	1.00	10
MDB_disease	1.00	0.90	0.95	10
No_disease	1.00	1.00	1.00	10
Spyder	1.00	1.00	1.00	10
mildiou	1.00	1.00	1.00	10
accuracy			0.99	70
macro avg	0.99	0.99	0.99	70
weighted avg	0.99	0.99	0.99	70

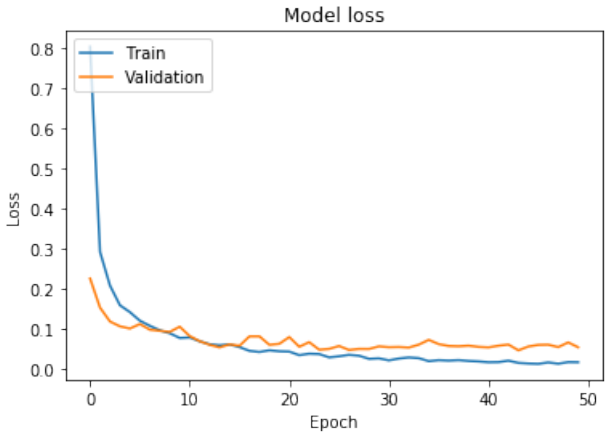
5.4 Resnet 34

Later we used Resnet34 architecture and obtained around 99% accuracy as shown in the figure.

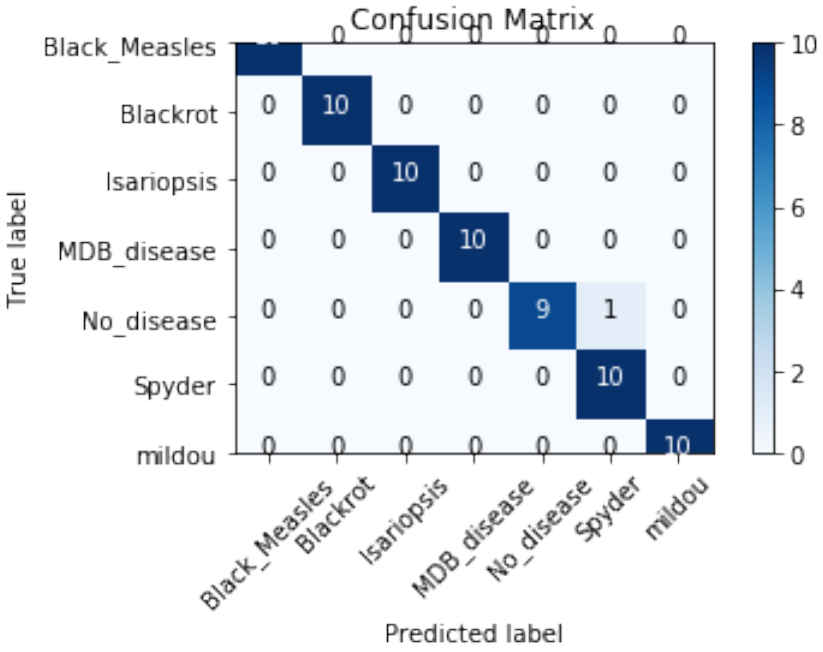
```
Epoch 00049: loss did not improve from 0.01130
Epoch 50/50
700/700 [==============================] - 262s 375ms/step - loss: 0.0154 - acc: 0.9954 - val_loss: 0.0
_acc: 0.9918

Epoch 00050: loss did not improve from 0.01130
```

Following are the training and Validation plots





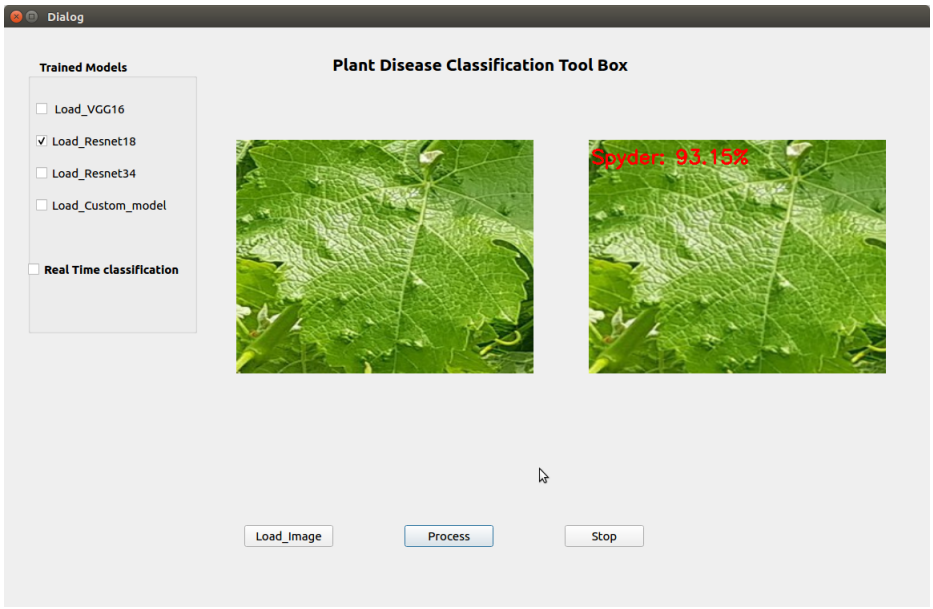
Following are the Confusion matrix and Classification reports



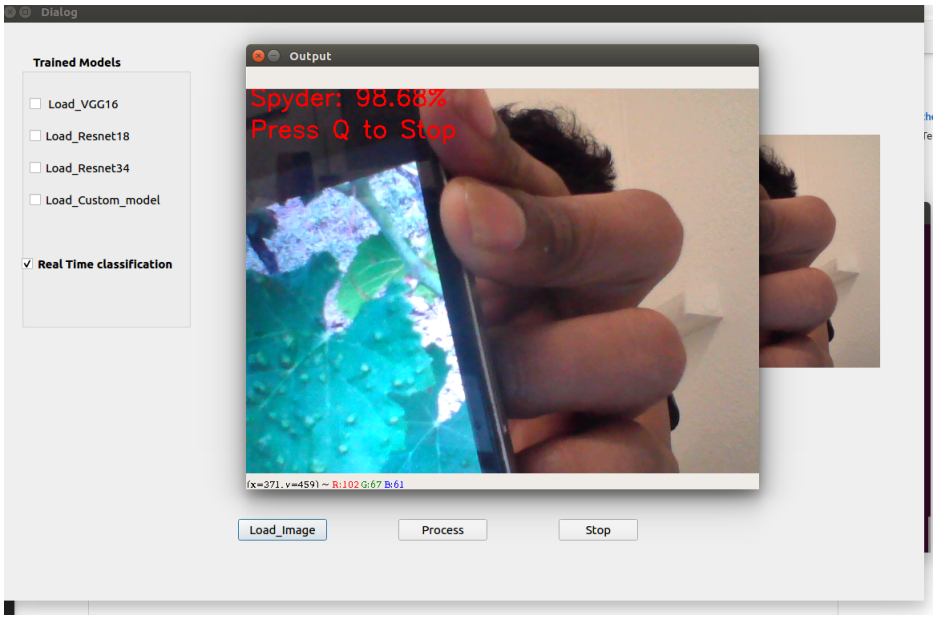
	precision	recall	f1-score	support
Black_Measles	1.00	1.00	1.00	10
Blackrot	1.00	1.00	1.00	10
Isariopsis	1.00	1.00	1.00	10
MDB_disease	1.00	1.00	1.00	10
No_disease	1.00	0.90	0.95	10
Spyder	0.91	1.00	0.95	10
mildiou	1.00	1.00	1.00	10
micro avg	0.99	0.99	0.99	70
macro avg	0.99	0.99	0.99	70
weighted avg	0.99	0.99	0.99	70

5.5 Graphical User Interface

For Graphical User Interface we used PyQT and integrated all the model and made user friendly application. Following figure shows the Interface and results both on image and real time.



Real Time Classification Output



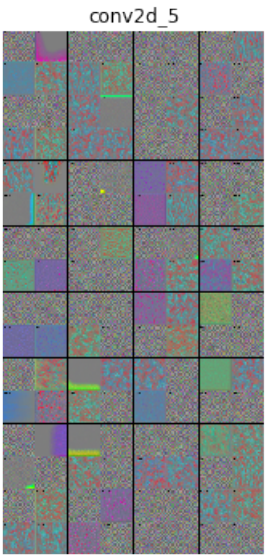
5.6 Embedding Visualization

Here we have generated embeddings and visualized using tensorboard the following figure figure shows how our images looks in feature space.

5.7 Neural Network Visualization

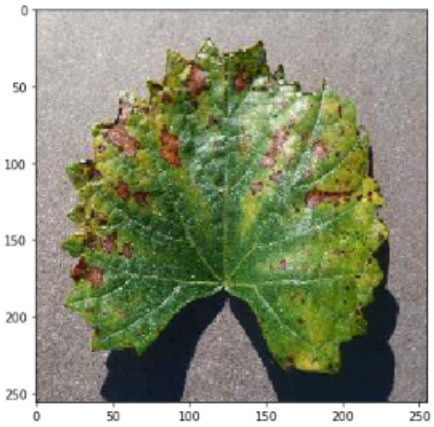
As we discussed above we have used two main libraries to interpret the neural networks i.e, keras-vis and lucid following are some of the outputs we tried to visualize.

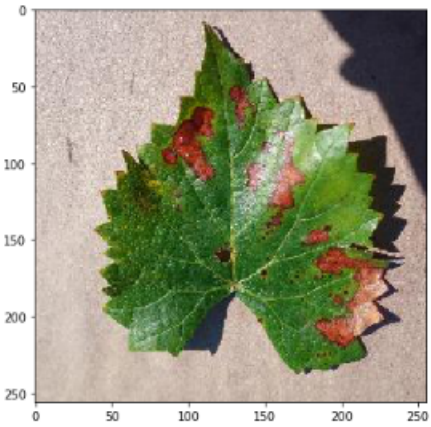
Intermediate layer:



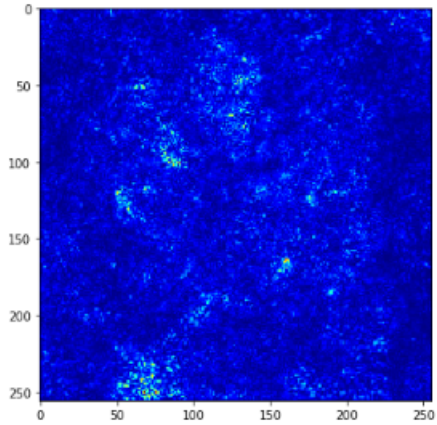
Following are the saliency maps.

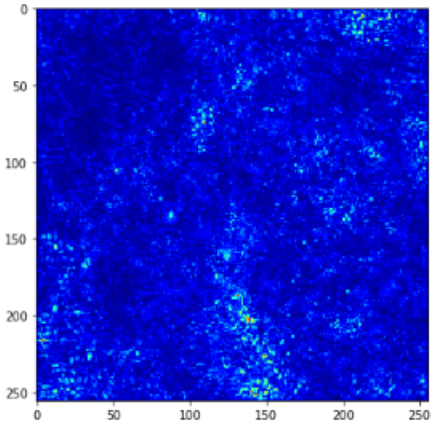
Input image:



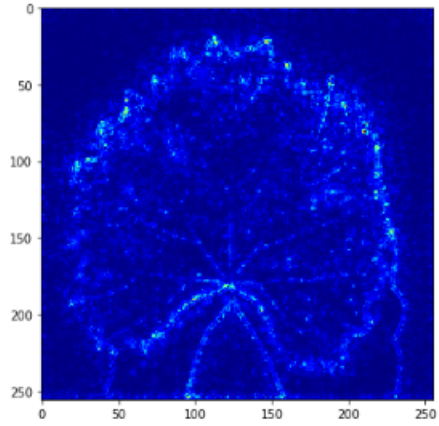


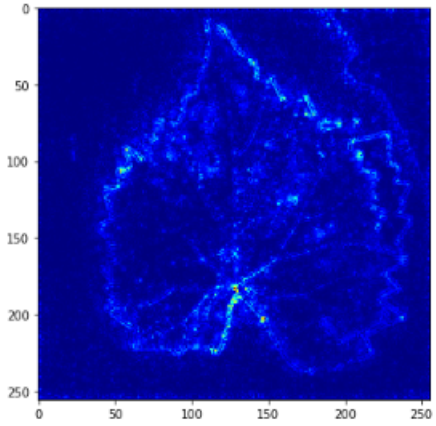
Saliency maps:





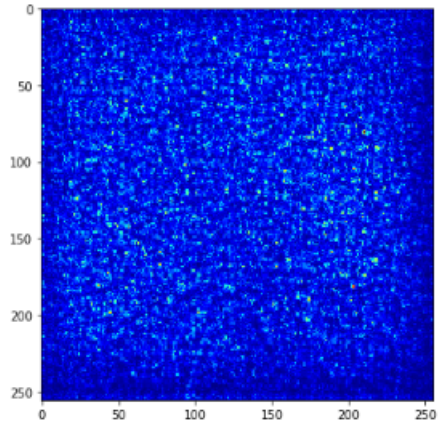
guided

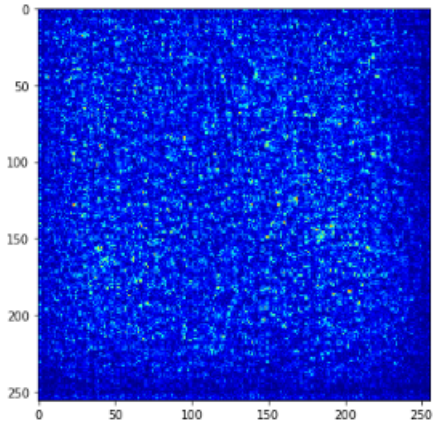




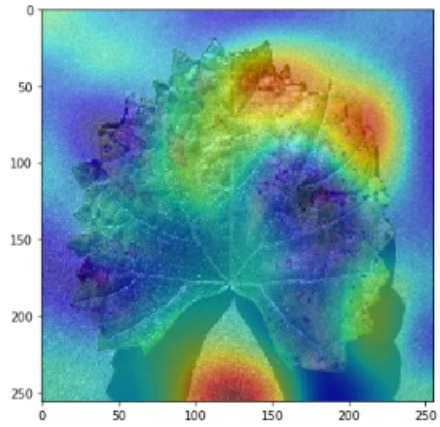
<Figure size 1296x432 with 0 Axes>

relu

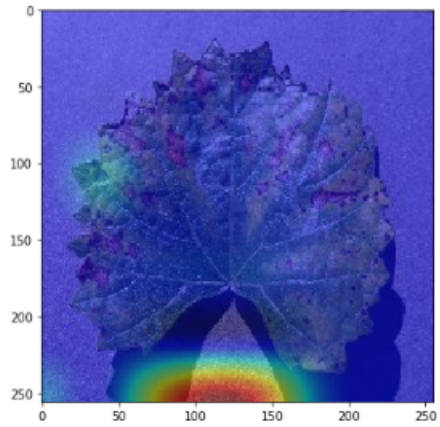




Heatmaps:




```
<Figure size 1296x432 with 0 Axes>
```

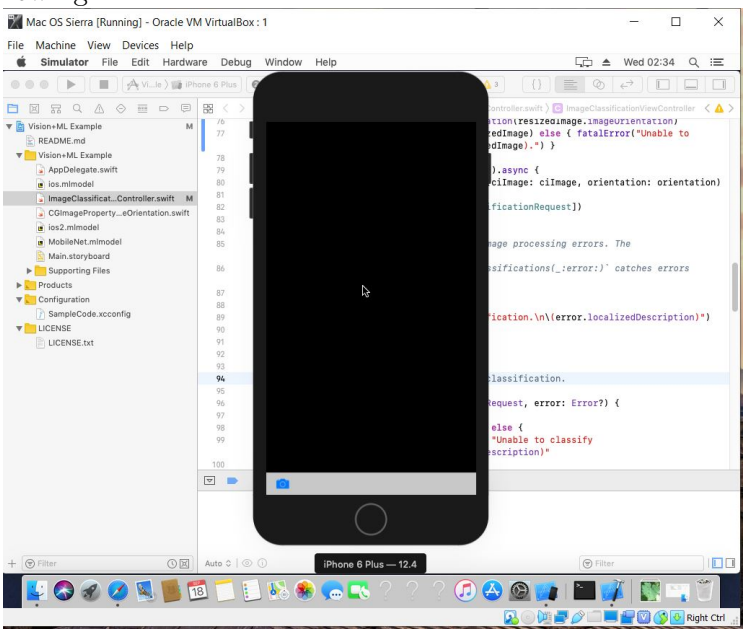



```
<Figure size 1296x432 with 0 Axes>
```

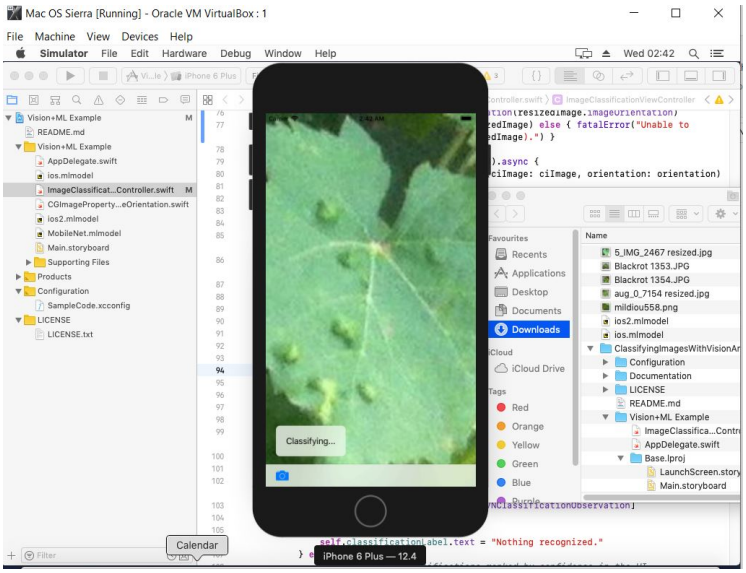
Specific Neuron:

5.8 iOS Application

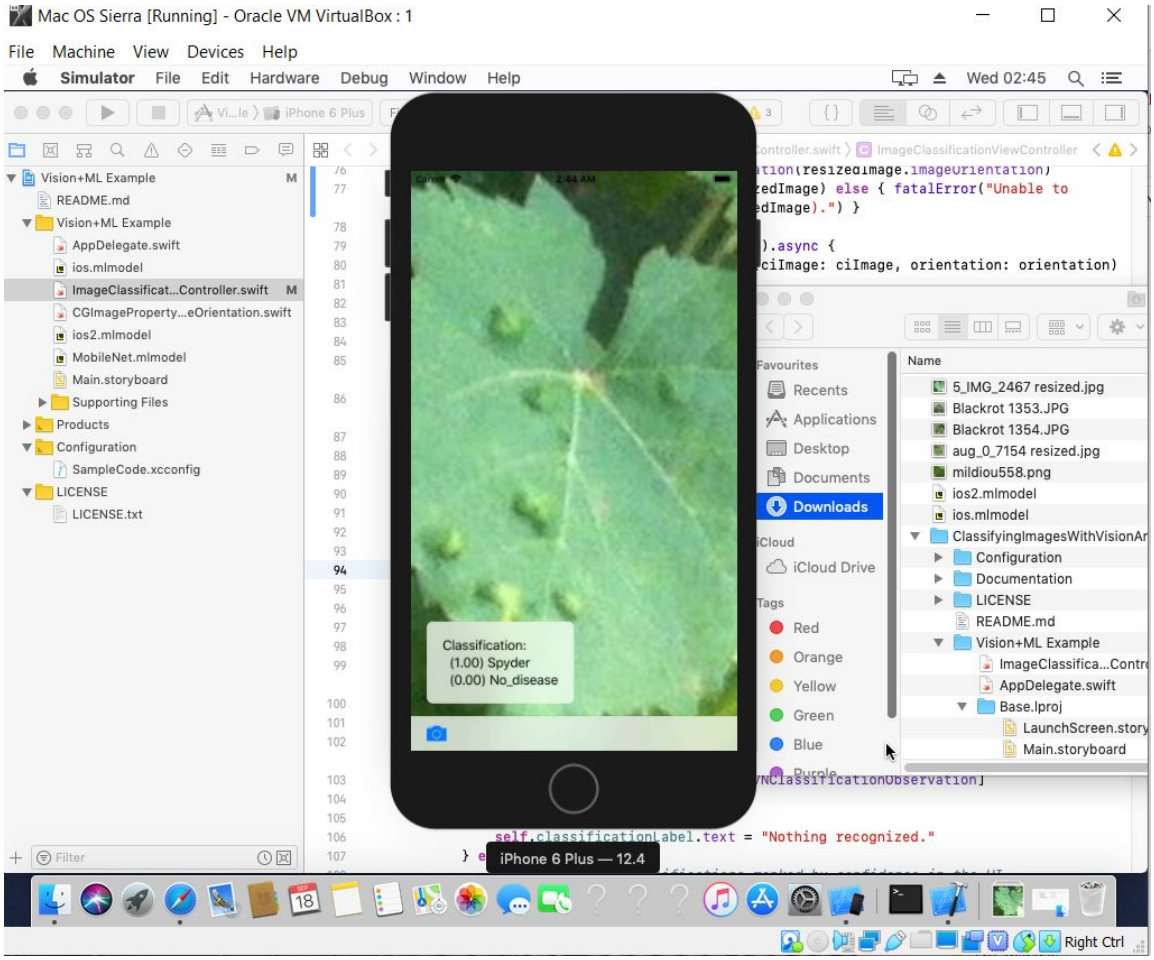
Open the application from iphone homescreen, the interface of the application looks as following



Now select camera and choose test image from gallery, now the model takes few seconds to do prediction



Following figure shows the Classification results on iOS App



6 How to Run the Code

6.1 Install the Dependencies

Inoder to install all the dependencies go to intern folder and look for the file named "requirements.txt", this file consists of all the libraries used in this project. Open the terminal in that specific directory and run the following command to install all the dependencies.

```
pip install -r requirements.txt
```

After running the above command you are good to go and run your code jupyter notebook.

Now go to the notebook directory through terminal and enter the following command.

```
jupyter notebook
```

Now you may open any notebook and run any part of the code.

6.2 Run GUI

Inorder to run GUI go to Gui directory through terminal and enter the following command.

```
python toolbox.py
```

6.3 Visualize Embeddings

In order visualize the embeddings using tensorboard all you have to do is point your tensorboard to embeddins-logs folder by following command.

```
tensorboard --logdir=./embeddings-logs
```

6.4 Run iOS app

To run iOS App you need to switch to mac OS and download first xcode from apple store and install, then copy the ios_app folder from intern folder and transfer into mac OS, Now you need to open project file namely "project.pbxproj" it is located at **(ios_App/Classification_coreML/Vision+ML Example.xcodeproj/project.pbxproj)** once you double click this file it open xcode application in the top you will find **"Product"** under that select run after succesfull build run now click **destination** and select the simulator as your wish and wait for few minutes you will find your app is launching in iphone simulator. You can drag and drop the test images in the simulator's gallery from your local test dataset. As soon as the application is launched press camera button and select the test image and in few seconds you will see the output as displayed in result section above.

7 Future Work

This Project can be push forward to object detection and recognition and later for semantic segmentation. As of now as it is only performing classification the test images should only consist of one type of diseases since the model has been trained on images which consist single class in each image. If it does both object detection and recognition giving probabilities and bounding box it can be used to classify multiple diseases in a every frame. Later we could use Mask RCNN and Unet to go further into semantic segmentation. For the navigation part as of now GPS is used we could use deep reinforcement learning techniques for vision based navigation.

8 Conclusion

I would like to thank my Supervisors Ariane Herbulot and Michel Devy for giving this opportunity. I have learnt many things during this Internship. I feel that I could do a lot more but due to time constraint i.e, just two months it wasn't able to continue, hopefully I look forward to work on these type of topics in future.

Bibliography

[1] Wikipedia

[2] Medium posts

[3] Towards Datascience posts

[4] opencv.com

[5] keras.io

[6] Other Blog posts

Bibliography

Annex

```
<?xml version="1.0" encoding="UTF-8"?>
<widget>
         <debug>off</debug>
         <window name="myWindow" title="Hello Widget" visible="true">
                 <height>120</height>
                 <width>320</width>
                 <image src="Resources/orangebg.png">
                        <name>orangebg</name>
                        <hOffset>0</hOffset>
                        <vOffset>0</vOffset>
                </image>
                 <text>
                         <name>myText</name>
                         <data>Hello Widget</data>
                         <color>#000000</color>
                         <size>20</size>
                         <vOffset>50</vOffset>
                         <hOffset>120</hOffset>
                 </text>
        </window>
</widget>
```

Listing 1: Sourcecode Listing

```
INVITE sip:bob@network.org SIP/2.0
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
Max-Forwards: 70
To: Bob <sip:bob@network.org>
From: Alice <sip:alice@ims-network.org>;tag=42
Call-ID: 10@100.101.102.103
CSeq: 1 INVITE
Subject: How are you?
Contact: <sip:xyz@network.org>
Content-Type: application/sdp
Content-Length: 159
v=0
o=alice 2890844526 2890844526 IN IP4 100.101.102.103
s=Phone Call
t=0 0
c=IN IP4 100.101.102.103
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.network.org:5060;branch=z9hG4bK83842.1
;received=100.101.102.105
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
To: Bob <sip:bob@network.org>;tag=314159
From: Alice <sip:alice@network.org>;tag=42
Call-ID: 10@100.101.102.103
CSeq: 1 INVITE
Contact: <sip:foo@network.org>
Content-Type: application/sdp
Content-Length: 159
v=0
o=bob 2890844526 2890844526 IN IP4 200.201.202.203
s=Phone Call
c=IN IP4 200.201.202.203
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Listing 2: SIP request and response packet[**?**]