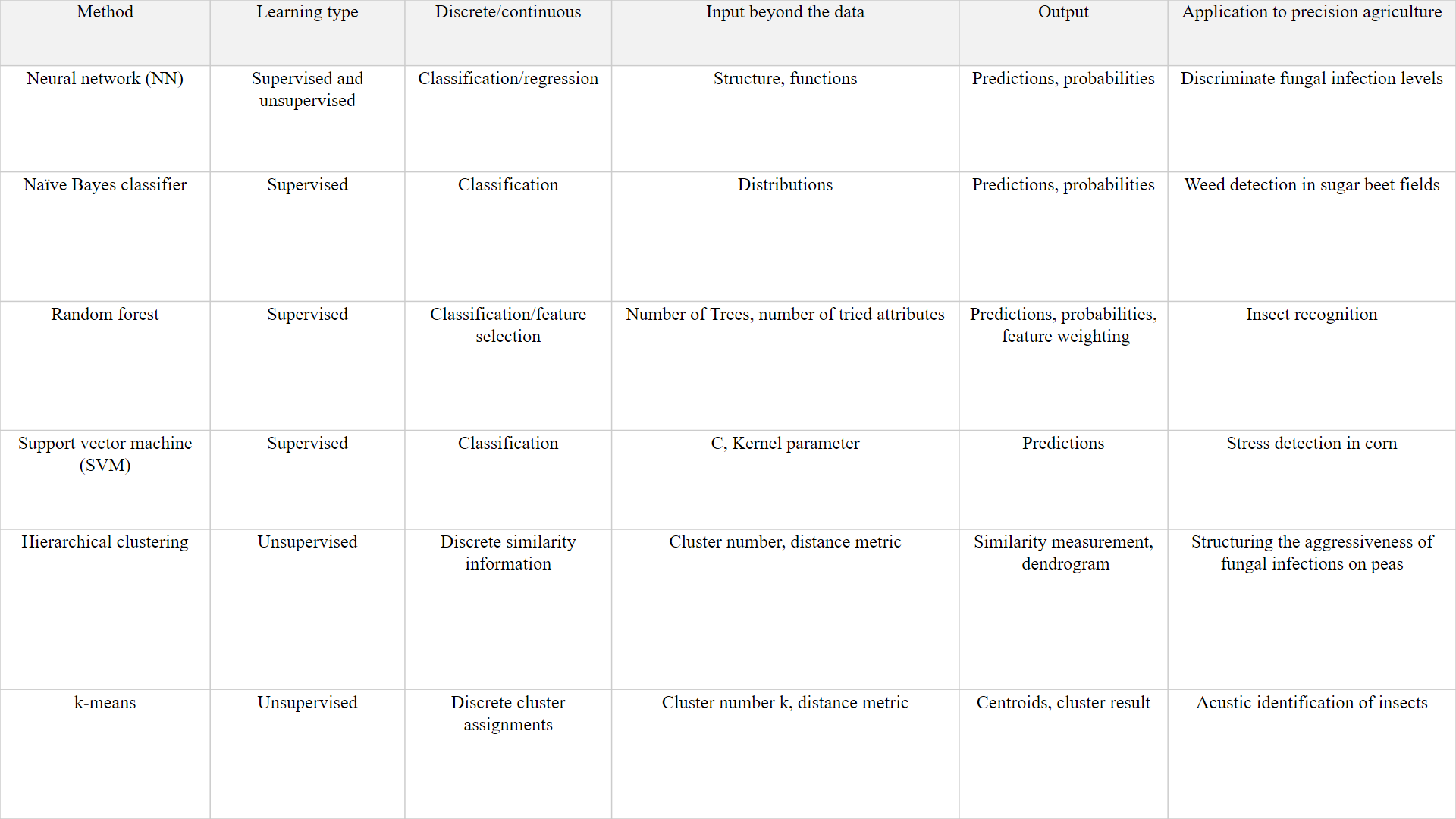
# 4.1 Image Analysis Techniques

Given a framework that gathers the necessary data, the decision making to be performed requires knowledge extraction from these data. Hyperspectral images contain a lot of useful information that can be analyzed to generate many insightful results. These results can then help to produce a reliable prescription map and present the solutions in a way that can be understood by the farmers. The potential of deep learning can also be used to analyze the RGB images of the crops taken from the farms.

Over the years, a number of vegetation indices (VIs) have been developed by combining two or more wavebands in the hyperspectral images in ratios and/or differences, to highlight various crop conditions. However, one of the problems in applying VIs to crop yield estimation is the difficulty in choosing the most appropriate vegetation index in a specific situation (Barrett and Curtis 1999). In fact, various environmental factors, such as background effects and crop canopy conditions, have been shown to be potential sources of noise, which affect the spectral reflectance in canopy level (Aparicio et al., 2000). Ironically, these difficulties, to identify the most useful wavelengths or VIs under specific environmental conditions, have been heightened with the recent proliferation of large volume of data available from hyperspectral and broadband sensors.

Artificial intelligence and especially machine learning have contributed to the creation of control systems in agriculture. Machine learning is the process of discovering previously unknown and potentially useful information from data.There are a lot of machine learning algorithms like Support Vector Machines(SVM), Simple Multivariate Linear Regressions(SMLR), Decision Trees(DT), Neural Networks(NN), etc. available that can be utilized for recognizing intricate patterns in data. Implementation of these algorithms in agricultural domain have shown inspiring results as shown in Table 4.1.

Table 4.1: Comparison between different machine learning algorithms along with their application in agricultural domains.

-

Deep Neural Networks (DNNs), have also generated a strong interest in their potential effectiveness in estimating various field and crop conditions from remotely sensed images. The ability of DNNs to associate complicated information with target attributes without any constraints for sample distribution, make them ideal for describing the intricate and complex non-linear relationships which exist between canopy-level spectral signatures and various crop condition. In fact, successful applications have already been reported for surface water quality assessment, soil moisture estimation, biomass estimation, and yield prediction. A Deep Neural Network (DNN) is a computational model which mimics the human nervous system and decision-making process. Although some technical difficulties, such as the low interpretability of the developed models, the complexity involved in optimizing the model structure, and the high processing power required for the training process, once made the intensive application of this techniques difficult, recent improvements in computing power and learning algorithms has increased the applicability of the method in various fields.

The strong noise from various environment factors, such as background soil effects and the existence of gaps in the canopy produces low prediction accuracies. Indeed, it is generally recognized that VI based methods are quite sensitive to these kinds of environmental factors, so that various standardizing process are normally required to achieve the highest performance. In particular, high prediction accuracies obtained with DNNs in various studies conducted across the world demonstrate that DNNs can be an effective alternative to conventional VI based method and hence can be used reduce the overall cost of the system by employing a simple camera(even using a smartphone camera) instead of the costly hyperspectral one. The widespread distribution of smartphones among crop growers around the world with an expected 5 billion smartphones by 2020 offers the potential of turning the smartphone into a valuable tool for diverse communities growing food.

In this study we assessed the potential of VI based method Normalized Difference Vegetation Index(NDVI) on hyperspectral data and Deep Learning on RGB data.

Dataset

RGB data: We used the PlantVillage database[] consisting of about 50,000 expertly curated images on healthy and infected leaves of crops plants with 38 classes.



Figure 4xxx: **Example of leaf images from the PlantVillage dataset, representing every crop-disease pair used. (1)** Apple Scab, Venturia inaequalis **(2)** Apple Black Rot, Botryosphaeria obtusa **(3)** Apple Cedar Rust, Gymnosporangium juniperi-virginianae **(4)** Apple healthy **(5)** Blueberry healthy **(6)** Cherry healthy **(7)** Cherry Powdery Mildew, Podoshaera clandestine **(8)** Corn Gray Leaf Spot, Cercospora zeae-maydis **(9)** Corn Common Rust, Puccinia sorghi **(10)** Corn healthy **(11)** Corn Northern Leaf Blight, Exserohilum turcicum **(12)** Grape Black Rot, Guignardia bidwellii, **(13)** Grape Black Measles (Esca), Phaeomoniella aleophilum, Phaeomoniella chlamydospora **(14)** Grape Healthy **(15)** Grape Leaf Blight, Pseudocercospora vitis **(16)** Orange Huanglongbing (Citrus Greening), Candidatus Liberibacter spp. **(17)** Peach Bacterial Spot, Xanthomonas campestris **(18)** Peach healthy **(19)** Bell Pepper Bacterial Spot, Xanthomonas campestris **(20)** Bell Pepper healthy **(21)** Potato Early Blight, Alternaria

solani **(22)** Potato healthy **(23)** Potato Late Blight, Phytophthora infestans **(24)** Raspberry healthy **(25)** Soybean healthy **(26)** Squash Powdery Mildew, Erysiphe cichoracearum **(27)** Strawberry Healthy **(28)** Strawberry Leaf Scorch, Diplocarpon earlianum **(29)** Tomato Bacterial Spot, Xanthomonas campestris pv. vesicatoria **(30)** Tomato Early Blight, Alternaria solani **(31)** Tomato Late Blight, Phytophthora infestans **(32)** Tomato Leaf Mold, Passalora fulva **(33)** Tomato Septoria Leaf Spot, Septoria lycopersici **(34)** Tomato Two Spotted Spider Mite, Tetranychus urticae **(35)** Tomato Target Spot, Corynespora cassiicola **(36)** Tomato Mosaic Virus **(37)** Tomato Yellow Leaf Curl Virus **(38)** Tomato healthy.

Hyperspectral data: We worked upon hyperspectral images taken from drone at Hayes Township Farm, Harrison, Michigan, USA.







Fig 4.xxx : Hyperspectral images taken from drone (Courtesy:6omvi.org).

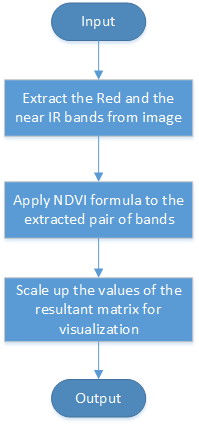


Figure 4xxx: Hayes Township Farm,Harrison,Michigan,USA.

4.2 Image Analysis Using Indexing

The hyperspectral images we have carry vital information that can be analyzed to deduce important parameters about crops. VI based methods provide many ways to explore this using specific algorithms designed to perform analysis on such images. Some of those which we wish to use are Normalized Difference Vegetation Index (NDVI), to know the

chlorophyll content in leaves.



**Figure 4.1:** Image analysis using VI based methods

# 4.2.1 Normalized Difference Vegetation Index

The Normalized Difference Vegetation Index (NDVI) is a numerical indicator that uses the visible and near-infrared bands of the spectrum, and is adopted to analyze remote sensing measurements and assess whether the target being discovered contains live inexperienced vegetation or not. NDVI has found a wide application in vegetative studies because it has been accustomed estimate crop yields, pasture performance, and rangeland carrying capacities among others. It is often directly associated with different ground parameters like percentage of ground cowl, photosynthetic activity of the plant, surface water, leaf area index and the quantity of biomass. NDVI was first used in 1973 by Rouse et al. from the Remote Sensing Centre of Texas A&M University.

Generally, healthy vegetation will absorb most of the visible lightweight that falls on that, and reflects a large portion of the near-infrared lightweight. Unhealthy or sparse vegetation reflects additional visible lightweight and fewer near-infrared lightweight. Bare soils on the different hand mirror moderately in each the red and infrared portion of the spectrum. Since we recognize the behavior of plants across the spectrum, we derived NDVI info by specializing in the satellite bands that area unit most sensitive to vegetation info (near-infrared and red). The bigger the distinction so between the near-infrared and therefore the red coefficient, the more vegetation there has to be. The NDVI algorithm subtracts the red reflectance values from the near-infrared and divides it by the sum of near-infrared and red bands.

This formulation allows us to cope with the fact that two identical patches of vegetation could have different values if one were, for example in bright sunshine, and another under a cloudy sky. The bright pixels would all have larger values, and therefore a larger absolute difference between the bands. This is avoided by dividing by the sum of the reflectance.

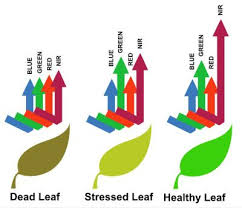


Figure 4.xxx:Reflectances from different types of leaves.

Theoretically, NDVI values are represented as a ratio ranging in value from -1 to 1 but in practice extreme negative values represent water, values around zero represent bare soil and values over 0.6 represent dense green vegetation.

In our application we calculated NDVI using QGIS (Quantum GIS) Python programing language APIs. We use QgsRasterLayer class that provides QGIS with the ability to render raster (image) datasets onto the map canvas. From loaded raster we are extracting IR and Red bands for NDVI calculation. QgsRasterCalculator (expression, output file, "GTiff", extent, width, height, entries)class generates NDVI image with the help of QgsRasterCalculator .processCalculation () method.

|  |  |
| --- | --- |
|  |  |
| Figure 4.xx: Input stitched image | Figure 4.xx: NDVI output Image |

Running python script on input image (Fig 4.xx) generates NDVI output file Figure 4.xx.

# 4.2.2 Clustering and Classification

After all the index calculations are done we are left with areas with different discrete patches that may have the same index range. The next step would be to cluster the similar data(pixel values) into a single color coded areas. The clustering is done using K-means algorithm.This clustered information is then used to train a softmax regression model for classification of new images.

## K-means Algorithm

K-means is one of the best unsupervised learning algorithms that solve the well-known bunch downside. Initially number of centers, which is k, is decided according to the Elbow Rule which minimizes the cost function with an appropriate number of clusters. The main idea is to outline k centers, one for each cluster. These centers should be placed in a crafty method attributable to totally different location causes different result. So, the better selection is to put them the maximum amount as potential secluded from one another.

The next step is to assign the closest center to each data point. At this point we'd like to re-calculate k new centroids as center of mass of the clusters ensuing from the previous step. After we have a tendency to have these k new centroids, a new binding must be done between constant data set points and therefore the nearest new center. A loop has been generated. As a result of this loop we could notice that the k centers amendment their location step by step till no additional changes square measure done or in different words centers don't move from now on. Finally, this algorithm aims at minimizing associate objective operate is aware of as square error operate given by:

is the Euclidean distance between and .

‘’ is the number of data points in ith cluster.

‘c’ is the number of cluster centers.

Algorithmic steps for k-means clustering:

Let X = {} be the set of data points and V = {} be the set of centers.

1) Randomly select ‘c’ cluster centers.

2) Calculate the distance between each data point and cluster centers.

3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.

4) Recalculate the new cluster center using:

where, ‘’ represents the number of data points in cluster.

5) Recalculate the distance between each data point and new obtained cluster centers.

6) If no data point was reassigned then stop, otherwise repeat from step 3.

We clustered the data into k = 4 classes based on the ranges of of NDVI.

# 4.2.3 Softmax Regression Model

Softmax regression is a generalization of logistic regression to the case where we want to handle multiple classes.

 It is a regression model which [generalizes the logistic regression](http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression) to classification problems where the output can take more than two possible values.

SoftMax regression is competitive in terms of CPU and memory consumption. The Softmax Regression is preferred when we have features of different type (continuous, discrete, dummy variables etc), nevertheless given that it is a regression model, it is more vulnerable to multicollinearity problems and thus it should be avoided when our features are highly correlated.

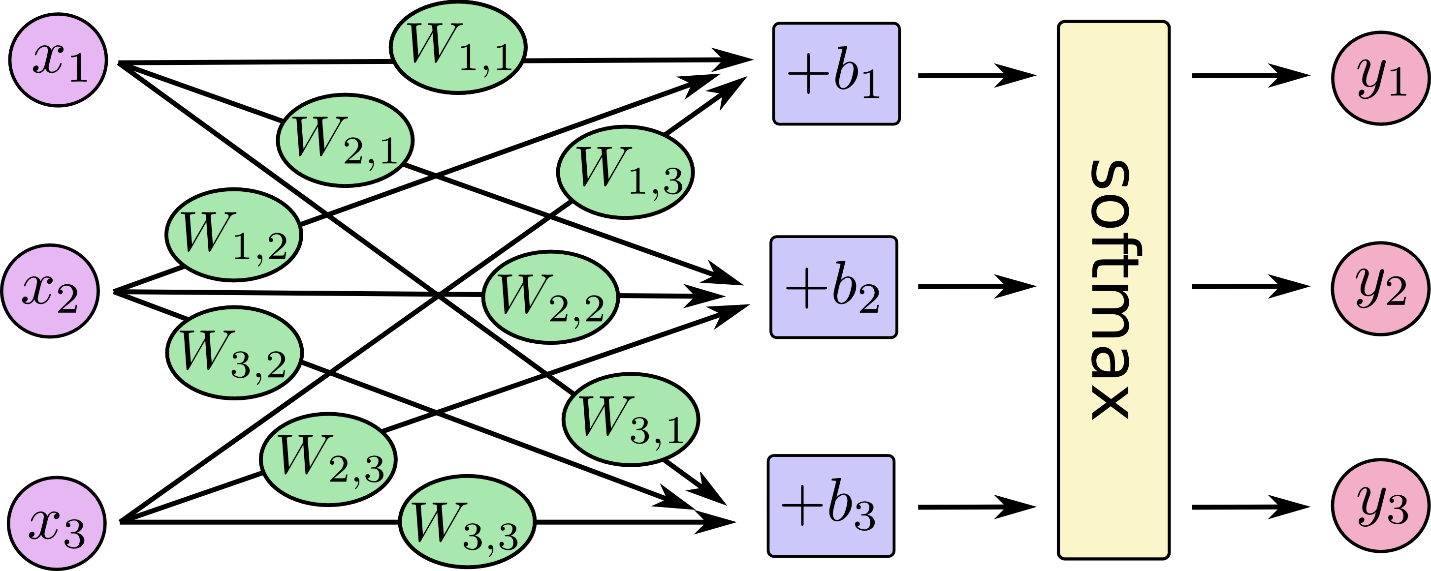


Figure 4xx: Basic structure of Softmax Model

We built a softmax regression model using Google’s tensorflow and trained it upon the clustered dataset.With 80% training and 20% test data we achieved accuracy of about 70%.This can be improved once we have better labelled dataset.

The model is shown in the image below:

# D:\graph-run=.png

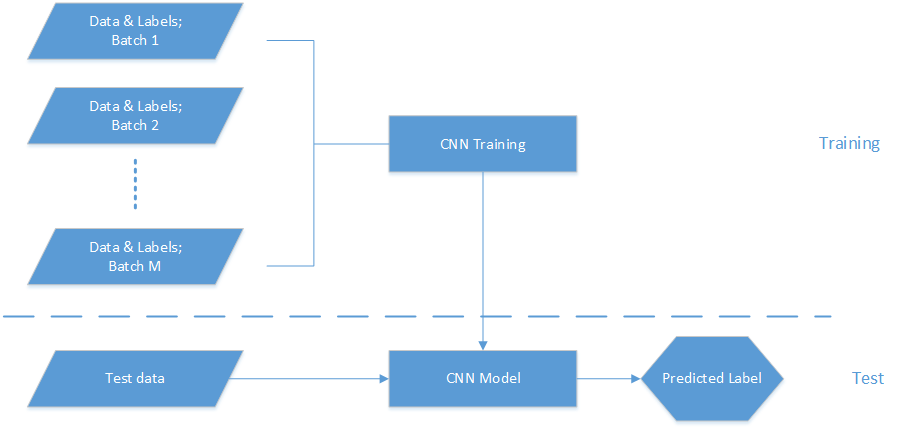
# Figure4;;; : Softmax Regression Model(Visualized in Tensorboard)

# This model divides the input farm image into different color coded regions depending upon the NDVI values.This image is then overlayed onto the Google Map for better visualization of the farm.

# 4.3 Image Analysis Using Deep Learning

Deep learning  is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data. In a simple case, you could have two sets of neurons: ones that receive an input signal and ones that send an output signal. When the input layer receives an input it passes on a modified version of the input to the next layer. In a deep network, there are many layers between the input and output (and the layers are not made of neurons but it can help to think of it that way), allowing the algorithm to use multiple processing layers, composed of multiple linear and non-linear transformations [4].

Many deep learning architectures such as Deep Neural Networks(DNNs), Convolutional Neural Networks(CNNs), and Recurrent Neural Networks(RNNs) have shown to produce state-of-the-art results on various tasks [5]. As a powerful visual model, CNNs have demonstrated remarkable performance in various visual recognition problems, and attracted considerable attention in recent years.



**Figure 4.2:** Proposed model for image analysis using CNN [13]

# 4.3.1 Convolutional Neural Networks

Convolutional Neural Networks are very similar to ordinary Neural Networks, they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer. CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network [9].

Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a CNN have neurons arranged in 3 dimensions: width, height, depth.



**Figure 4.3:** A regular 3-layer Neural Network [9]



**Figure 4.4:** A CNN arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a CNN 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) [9]

A simple CNN is a sequence of layers, and every layer transforms one volume of activations to another through a differentiable function. We use three main types of layers to build CNN architectures: **Convolutional Layer, Pooling Layer**, and **Fully-Connected Layer.** In this way, CNNs transform the original image layer by layer from the original pixel values to the final class scores.

# 4.3.2 Deep Learning Frameworks

Deep learning methods have resulted in significant performance improvements in several application domains and as such several software frameworks have been developed to facilitate their implementation to foster research and development in artificial intelligence. We can leverage these open-source tools and frameworks to implement many complex algorithms for different applications. Some of the very common frameworks include Caffe, Neon, TensorFlow, Theano, Torch, etc.

Deep learning generally means building large scale neural networks with many layers. Simply put, these networks are simply functions which generate outputs Y given inputs X. In addition to the input X, the functions make use of a bunch of parameters (also called weights). These can include scalar values, vectors, and most expensively, matrices and higher-order tensors. A tensor is just a generalization of vectors and matrices into higher dimensions. The particular functions in vogue today involve tens of computationally expensive linear algebra operations, including matrix products and convolutions. Before we can train the network, we define a loss function. Common loss functions include squared error for regression problems and cross-entropy loss for classification. To train a network, we need to successively present many batches of new inputs to the network. After each is presented, we update the model by taking the derivative of the loss with respect to all of our parameters. So right away there are a few obvious problems. First, multiplying tens or hundreds or tensors together millions of times to process even a moderately sized dataset is terribly expensive. Second, taking the derivative of giant ugly functions by hand is a pain and could consume days or weeks that would be better spent imagining new experiments. This is why we need libraries like Theano, Caffe, Torch, and TensorFlow. The takeaway here is first that with any of these libraries you can write only the prediction code or forward pass and the framework will figure out how to take derivatives for you, that is to calculate the backwards pass. Second, you write your code once in a nice high-level language without ever learning the ugly details of GPU coding, and the framework will compile for whatever CPU or NVIDIA hardware you have access to.

# Why TensorFlow?

TensorFlow is an open source software library by Google for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows us to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. The best part is that we can build every algorithm using easy to use data flow graphs which helps to visualize our algorithm’s architecture. TensorFlow seems to have a faster compile time and its computational graphs can be distributed on a cluster for computations. TensorFlow is both an R&D and deployment framework. Support from such a huge company as Google is a big advantage for TensorFlow. TensorFlow can work with any gradient-based machine learning algorithm, which opens up a much broader range of uses. Written in C++ for speed, it doesn't require the developer to know anything about the underlying hardware. It also runs across multiple devices and architectures, so it's intended to scale from SoC devices, like phones, all the way up to distributed systems using dozens of GPUs. The main barrier to using any framework for math, statistics, or machine learning is ease-of- use. Likewise, one of TensorFlow's proffered advantages is ease-of-use.

**Features:**

* Multi GPU support
* Training across distributed systems.
* Visualize the graphs using TensorBoard
* Model checkpointing

In this report we used the concept of transfer learning to retrain the final layer of Google’s inception v-3 model based upon TensorFlow.

# 4.3.3 Transfer Learning

Modern object recognition models have millions of parameters and can take weeks to fully train. Transfer learning is a technique that shortcuts a lot of this work by taking a fully-trained model for a set of categories like ImageNet, and retrains from the existing weights for new classes. Transfer learning provides the opportunity to adapt a pre-trained model to new classes of data with several advantages.

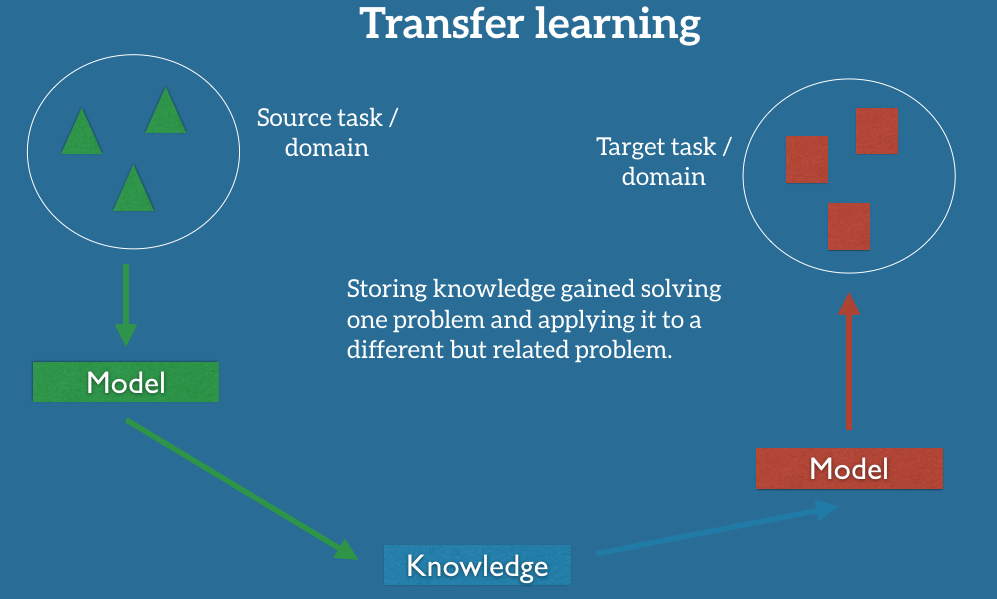


Figure 4xxxx: Transfer Learning setup

In practice, we seek to transfer as much knowledge as we can from the source setting to our target task or domain. This knowledge can take on various forms depending on the data: it can pertain to how objects are composed to allow us to more easily identify novel objects; it can be with regard to the general words people use to express their opinions, etc.

Inception V-3 model

 Inception v3 is the 2015 iteration of Google’s Inception architecture for image recognition trained upon ImageNet data. Inception is a really great architecture and it’s the result of multiple cycles of trial and error.

Inception is basically:

* An 299x299x3 input representing a visual field of 299 pixels and 3 color (RGB) channels
* Five vanilla convolution layers, with a few interspersed max-pooling operations
* Successive stacks of “Inception Modules”
* A softmax ouput layer at the end (logits) and at an intermediate output layer (aux\_logits) just after the mixed 17x17x768e layer.

It’s the repeated stacking of the Inception modules that makes this architecture “deep.” While stacking Inception modules leads to depth, each module is also “wide” and architected to recognize features at multiple length scales.

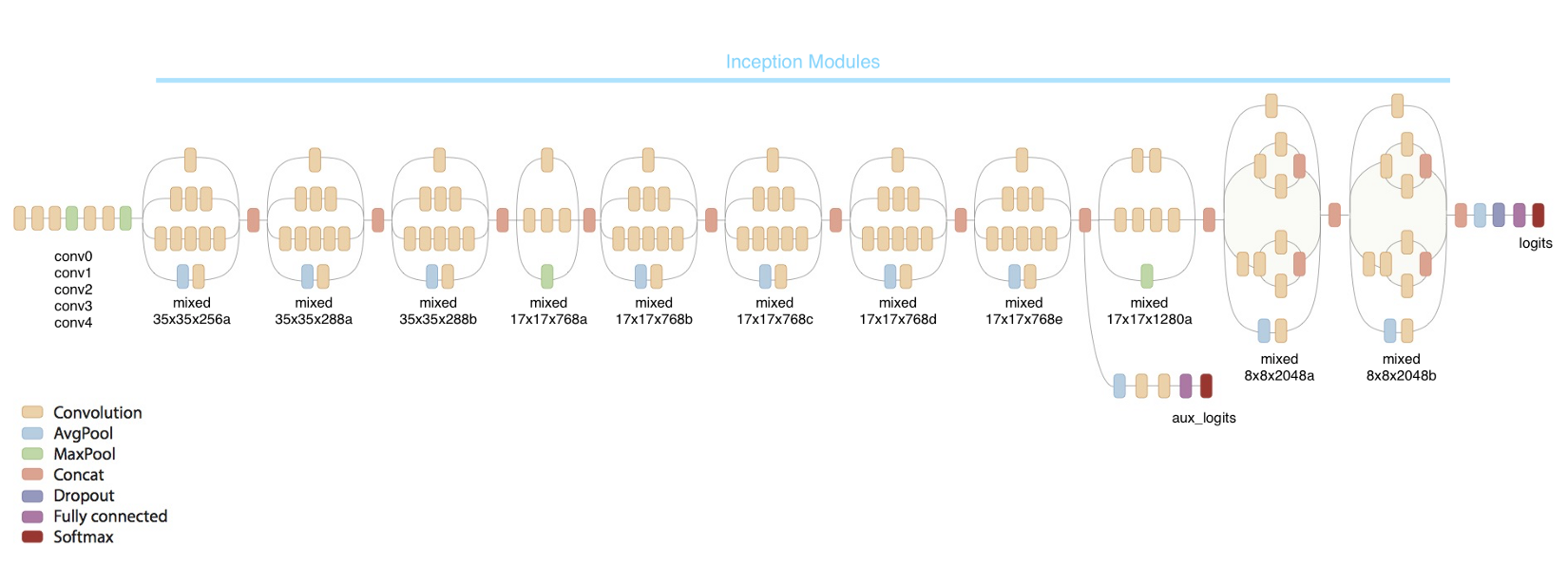


Figure 4xxx: overview of inception architecture

We fine-tuned the final layer of this model upon our dataset with 70% training and 30% validation sets, achieving an accuracy of about 99%.

Once the image is uploaded on the server and evaluated against the trained model, the farmer gets a proper prescription of the crops on the GUI.