**1. Abstract:**

Plant disease diagnosis through optical observation of the symptoms on plant leaves, incorporates a significantly high degree of complexity. Due to this complexity and to the large number of cultivated plants and their existing Phyto-pathological problems, even experienced agronomists and plant pathologists often fail to successfully recognize specific diseases, and are consequently led to mistaken conclusions and treatments. The existence of an automated computational system for the detection and diagnosis of plant diseases, would offer a valuable assistance to the agronomist who is asked to perform such diagnoses through optical observation of leaves of infected plants.

With the development of computational systems in recent years, and in particular Graphical Processing Units (GPU) embedded processors, Machine Learning-related Artificial Intelligence applications have achieved exponential growth, leading to the development of novel methodologies and models, which now form a new category, that of Deep Learning. The introduction of these deep learning techniques into agriculture (e.g., Carranza-Rojas et al., 2017), and in particular in the field of plant disease detection (Yang and Guo, 2017), has only begun to take place in the last couple of years. The basic deep learning tool used in this work is **Convolutional Neural Networks (CNNs)** (LeCun et al., 1998). Lee et al. (2015) presented a CNNs system for the automated recognition of plants, based on leaves images.

Now, the aim of our project is to detect diseases that occur on plants in tomato fields or in their greenhouses. Tomato is a major food crop across the world. For this purpose, deep learning was used to detect the various diseases on the leaves of tomato plants. In this model, Conventional Neural Network architecture called **Xception** that is 71 layers deep, was developed to perform plant disease detection and diagnosis using simple leaves images of healthy and diseased plants, through deep learning methodologies.  Several model architectures were trained, with the best performance reaching a **93.33%** success rate and **87.82%** success rate in validation data in identifying the corresponding (plant, disease) combination (or healthy plant). At last testing the images of leaves, gave **81%** accuracy on our model. The significantly high success rate makes the model a very useful advisory or early warning tool, and an approach that could be further expanded to support an integrated plant disease identification system to operate in real cultivation conditions.

**2. Introduction:**

Tomato is one of the most produced crops all around the world. According to the statistics obtained from the Food and Agricultural Organization of the United Nations, approximately 170.750 kilotons of tomato produced in the year 2014 in all around the world. In Bangladesh, the area of cultivation is about 13,066 ha with the production of about 74,000 m tons. To ensure healthy and proper growth of tomato plants it is essential to detect any disease in time and prior to applying required treatment to the affected plants. Identifying diseases from the image of the plant is one of the interesting research areas in computer vision and agricultural field. There are some popular tomato plant diseases namely ‘Bacterial spot’, ‘Early blight’, ‘Leaf mold’, 'Septoria leaf spot’, 'Tomato Mosaic Virus' etc.

Different types of tomato plant diseases:

**Healthy tomato:** The healthy tomatoes are completely free of blemishes and bruises and should be a deep, bright red. Any tomato that looks dull or pale is going to be lackluster. Steer clear of any discolorations-even a small black spot can mean hidden rot on the inside.

**Bacterial spot of tomato:** Bacterial spot can affect all above ground parts of tomato: stems, petioles, leaves & fruits. The symptoms of the diseases are small brown circular spots surrounded by a yellow halo on the tomato leaves. The center of the leaf spots often falls out resulting in small holes.

**Tomato early blight:** Early blight infects leaves, fruits and stems of tomato. The symptoms are small dark spots form on older foliage near the ground initially. Spots are round, brown and grow up to half inch in diameter. The stem above the soil line turns brown, sunken and dry. Fruit spots are leathery and black.

**Tomato late blight:** Late blight infects leaves, fruits and stems of tomato. The symptoms are leaves have large, dark brown blotches with a green gray edge; not confined by major leaf veins. Firm and dark brown with a round edge grow on stem and to the cover large parts of fruit.

**Tomato leaf mold:**

Leaf mold infects the oldest leaves first. Pale greenish-yellow spots, usually less than ¼ inch, with no definite margins, form on upper side of leaves. Olive green to brown velvety mold forms on the lower leaf surface below leaf spots. Infected blossoms turn black and fall off.

**Tomato Septoria leaf spot:** Septoria leaf spot is caused by fungus. It usually appears on the lower leaves after the first fruit sets. Spots are circular, about 1/6 to ¼ inch in diameter with dark brown margins and tan to gray centers with small black fruiting structures.

**Tomato target spot:**  Symptoms of target spot can be confused with those of bacterial spot and early blight. The target spot fungus can infect all above ground part. The initial foliar symptoms are pinpoint-seized, water-soaked spots on the upper leaf surface. The spots develop into small, necrotic lesions that have light brown centers and dark margins .The lesions increase in size and become circular with gray to pale brown centers. Yellow halos can form around the lesions on some varieties.

**Tomato mosaic virus (ToMV):** ToMV symptoms can be varied and hard to distinguish from other common tomato viruses. Some symptoms are mottled light and dark green on leaves. If plants are infected early, they may appear yellow and stunted overall. Leaves may curled, malformed, or reduced in size. Fruits may ripe unevenly and yellowish rings may appear if fruits ripe in warm weather.

****

**(I) Bacterial spot (II) Early Blight**

****

**(III) Late blight (IV) Leaf mold**

**Figure 1:** Different types of tomato plant diseases

The process of plant disease detection is mainly divided into two parts –

**(i)** First one is Image Processing and **(ii)** other is using Transfer Learning to classify the image.

Various steps of image processing **include image rescaling, zooming, image shearing, image flipping horizontally, brightness controlling** are done in **Image Preprocessing**. At first, we do preprocessing our images by using **ImageDataGenartor** function that generates batches of tensor image data with real-time data augmentation. The output images generated by the generator will have the same output dimensions as the input images.

And then, we used Deep learning which advantage is that, it can extract features from images automatically. The neural network learns how to extract features while training. We propose a convolutional neural network architecture based entirely on depth-wise separable convolution layers. We name our proposed architecture **Xception**, which stands for **“Extreme Inception”**. The Xception architecture has 36 convolutional layers forming the feature extraction base of the network. In short, the Xception architecture is a linear stack of depth wise separable convolution layers with residual connections. This makes the architecture very easy to define and modify; it takes only 30 to 40 lines of code using a high level library such as **Keras** or **TensorFlow**.

In our experimental evaluation, we will exclusively investigate image classification and therefore our convolutional base will be followed some dense layer. In the dense layer, we used **RELU** & **Softmax** activation function for training the model. We also used **MaxPolling** here. And for compiling the model, **categorical\_crossentropy was** used as loss and **ADAM** was used as optimizer.

After that, we fit our model & get 93.33% accuracy on training dataset. **Matplotlib** was used for plotting the accuracy and loss in per epoch of training data and validation data. Then we testing some datasets and create a **Confusion matrix** by which the number of correct and incorrect predictions are summarized with count values and broken down by each class. After that, a **classification report** was used to calculate **recall, precision and F1 score.** At the end, we predict our model on test dataset and we find correct prediction most of the time. We also used a customizable User Interface called **Gradio** which core library is free and open source.

**3. Related Works:**

**(I) MNIST Handwritten Digit Classification** **Using CNN.**

**(II) [CNN\_cifar10\_dataset\_Classification.](https://github.com/codebasics/deep-learning-keras-tf-tutorial/blob/master/16_cnn_cifar10_small_image_classification/cnn_cifar10_dataset.ipynb" \o "cnn_cifar10_dataset.ipynb)**

**(III) IRIS Flower Image Classification using Transfer Leering.**

**(IV) Animal Identification Using CNN and Transfer Learning (VGG\_16, RESNET50, InceptionV3) and comparing the results between them.**

**(V**) **[Keras\_Fashion\_MNIST](https://github.com/codebasics/deep-learning-keras-tf-tutorial/blob/master/1_keras_fashion_mnist_neural_net/1_keras_fashion_mnist.ipynb" \o "1_keras_fashion_mnist.ipynb)  using CNN.**

**(VI) Image Segmentation Project.**

**4. Used Language:**

We used **Python** Programming language in our project. Python code is understandable by humans, which makes it easier to build models for machine learning. Since Python is a general-purpose language, it can do a set of complex **machine learning** tasks and enable you to build prototypes quickly that allow you to test your product for machine learning purposes.

**Python Fundamental:**

**(I)** Python basic

**(II)** OOP (Object Oriented Programming)

**(III)** Data structure and algorithm

**5. Implementation Tools:**

* Keras
* Tensorflow
* Google colab
* Numpy
* Matplotlib
* Gradio User interface

**6. Datasets:**

We took datasets from **Kaggle** website where many images of tomato leaves are included. Then we divided the datasets into 3 categories,

**(I)** Train Dataset,

(II) Valid Dataset and

(III) Test Dataset.

In this case, we used total **10083** images for our model where **8695** images for training, **1388** images for validation and **1359** images for testing. So, it is said that we took approximately **75%** images for train dataset, **13%** for validation dataset and **12%** for test dataset.

**The datasets included with total 8 classes of Tomato Plant disease which are----**

‘Bacterial spot’, ‘Early blight’, ‘Healthy Tomato’, ‘Late blight’, ‘Leaf mold’, 'Septoria Leaf Spot’, ‘Tomato target spot’, ‘Tomato Mosaic Virus;.

**7. Methodology:**

Detection and classification of tomato plant diseases can be possible with many ways. Deep convolutional neural networks (CNN) is a deep learning architecture that can be used for image classification, object detection and many more on computer vision task. Image need to some pre-processing step to prepare for feature extraction and classification.

Image acquisition

Image pre-processing

Training image dataset

Feature extraction

Image classification

**Figure 2:** Flow diagram of methodology

**(a) Image acquisition**

In **image** processing, it is defined as the action of retrieving an **image** from some source, usually a hardware-based source for processing. It is the first step in the workflow sequence because without an image no processing is possible! Image collection using a digital camera is called the image acquisition. Image can be collected from tomato field or web for making the dataset. Always try to capture the image of the object at the same distance. If the image captured very close or very far distance, then it is hard to increase the accuracy of the model. There is a standard method to find the interesting area of an image so that keep in mind this standard when collection of an image dataset. And after colleting images, some uploads the images like datasets into the website of google like **Kaggle.** We used that images for preprocessing in our Tomato plant Disease Detection.

**(b) Image pre-processing**

Keras **ImageDataGenerator** is a gem! It lets us to augment our images in real-time while our model is still training! We can apply any random transformations on each training image as it is passed to the model. This will not only make your model robust but will also save up on the overhead memory.

* Image augmentation in Keras.
* Image Augmentation techniques with Keras ImageDataGenerator.
  + Rescales
  + Flips
  + Brightness
  + Zoom
  + Shears etc are used in our model.
* Introducing the Dataset (Train, Validation & Test dataset).
* ImageDataGenerator methods: We used-

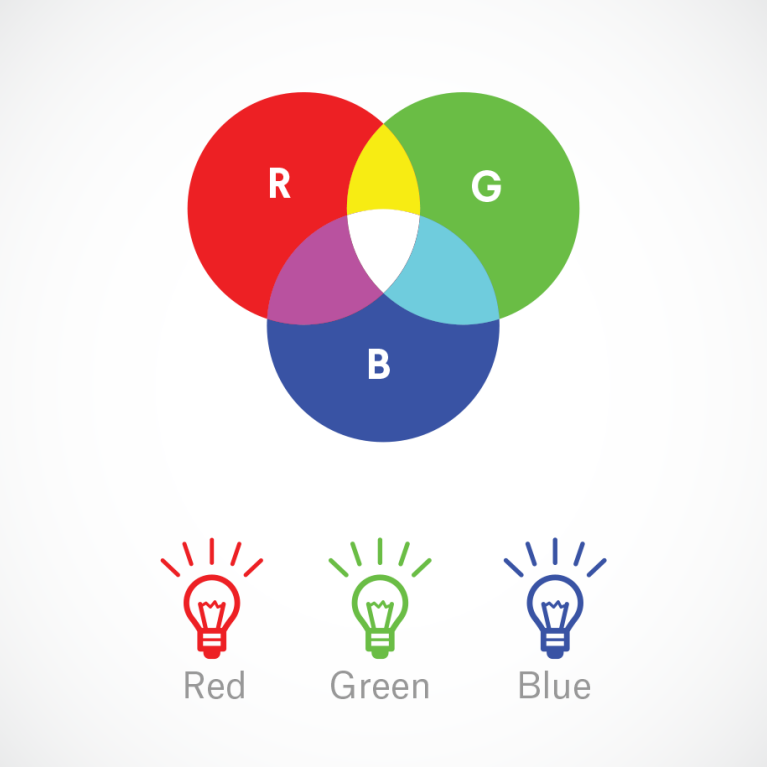
- Flow\_from\_directory (From 3 directories of Train, Valid & Test Datasets)

**(1) target \_size**=(224,224), -(same size after resizing images of datasets) **(2) color \_mode**= ‘rgb’ .

RGB (Red, Green and Blue) is the color space for digital images. Use the RGB color mode if your design is supposed to be displayed on any kind of screen.

A light source within a device creates any color you need by mixing red, green and blue and varying their intensity. This is known as additive mixing: all colors begin as black darkness and then red, green and blue light is *added* on top of each other to brighten it and create the perfect pigment. When red, green and blue light is mixed together at equal intensity, they create pure white.

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors.



**(3) batch\_size**=64.

The batch size is a number of samples processed before the model is updated. The number of epochs is the number of complete passes through the training dataset. The size of a batch must

be more than or equal to one and less than or equal to the number of samples in the training dataset. is an important hyperparameter that influences the dynamics of the learning algorithm. Batch size controls the accuracy of the estimate of the error gradient when training neural networks. Higher batch sizes leads to lower asymptotic test accuracy. The model can switch to a lower batch size or higher learning rate anytime to achieve better test accuracy.

So, I used **64** as batch size.

**(4) class \_mode**='categorical'. Machine learning models require all input and output variables to be numeric. This means that if data contains categorical data, we must encode it to numbers before we can fit and evaluate a model. The two most popular techniques are an Ordinal Encoding and a One-Hot Encoding.

**(5) shuffle**=False.

To shuffle the data, Default: True. If set to False, sorts the data in alphanumeric order. So, we make it False because we need the sorting the name of the image datasets in alphanumeric order.

**(6) seed**=42.

Optional random **seed** for shuffling and transformations- save \_to \_ dir : None or str (default: None). This allows us to optionally specify a directory to which to save the augmented pictures

being generated (useful for visualizing).

**(c) Feature Extraction using Transfer Learning**

**Feature Extraction of Image:**

The features extraction aspect of image analysis focuses on identifying inherent characteristics or features of object present within an image. In image processing, feature extraction is a special form of dimensionality reduction. The main goal of the feature extraction is to obtain the most relevant information from the original data and represent that information in a lower dimensionality space. When the input data to an algorithm is too large to be processed and it is suspected to be redundant then the input data will have transformed into a reduced representation set of features. There are three categories are extracted: color, shape and texture. The color is an important feature because it can differentiate one disease from another. The pixel in the color images are commonly represent in RGB format. Each disease may have different shape and system can differentiate diseases using shape features. Breadth and length of the image are significant characteristic to describe the shape and used to measure the count of the object pixel. Texture means how color patterns are scattered in the image.

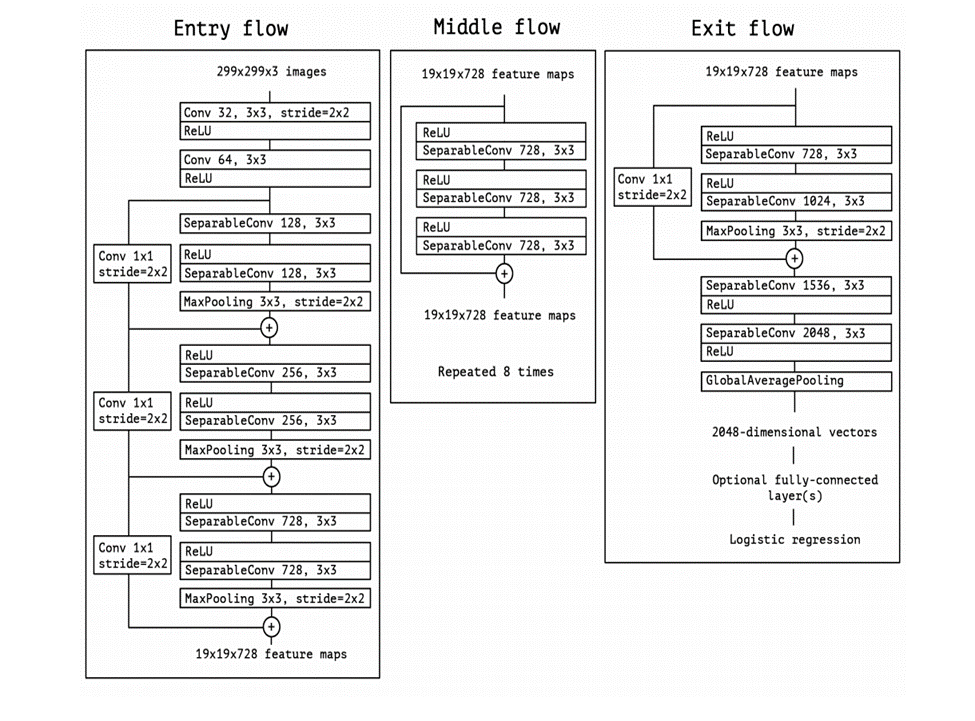
**Transfer Learning using Feature Extraction from Trained model:**

The idea of transfer learning comes from a curious phenomenon that many deep neural networks trained on natural images learn similar features. These are texture, corners, edges and color blobs in the initial layers. Such initial-layer features appear not to specific to a particular data-set or task but are general in that they are applicable to many data-sets and tasks. These standard features found on the initial layers seems to occur regardless of the exact cost function and natural image data-set. We call these initial-layer features general and can be transferred for learning specific data-set. Deep convolutional neural network models may take days or even weeks to train on very large datasets. A way to short-cut this process is to re-use the model weights from pre-trained models that were developed for standard computer vision benchmark datasets, such as the **ImageNet** image recognition tasks. Top performing models can be downloaded and used directly, or integrated into a new model for your own computer vision problems. Transfer learning involves using models trained on one problem as a starting point on a related problem. Transfer learning is flexible, allowing the use of pre-trained models directly, as feature extraction preprocessing, and integrated into entirely new models. Keras provides convenient access to many top performing models on the ImageNet image recognition tasks such as VGG, Inception, and ResNet , Xception etc. We used Xception architecture here.

**Xception Architecture Model:**

We propose a convolutional neural network architecture based entirely on depthwise separable convolution layers. In effect, we make the following hypothesis: that the mapping of cross-channels correlations and spatial correlations in the feature maps of convolutional neural networks can be entirely decoupled. Because this hypothesis is a stronger version of the hypothesis of the Inception architecture, we name our proposed architecture Xception, which stands for “Extreme Inception”. A complete description of the specifications of the network is given in figure 2. The Xception architecture has 36 convolutional layers forming the feature extraction base of the network. In our experimental evaluation we will exclusively investigate image classification and therefore our convolutional base will be followed by a logistic regression layer. Optionally one may insert fully-connected layers before the logistic regression layer, which is explored in the experimental evaluation section The 36 convolutional layers are structured into 14 modules, all of which have linear residual connections around them, except for the first and last modules. In short, the Xception architecture is a linear stack of depthwise separable convolution layers with residual connections. This makes the architecture very easy to define and modify; it takes only 30 to 40 lines of code using a high level library such as Keras [or TensorFlow, not unlike an architecture such as VGG-16, but rather unlike charchitectures such as Inception V2 or V3 which are far more complex to define.

In figure 2, the Xception architecture: the data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization (not included in the diagram). All SeparableConvolution layers use a depth multiplier of 1 (no depth expansion).



**Figure 3:** Implementation of Xception network

**We have done-**

(I) The top of the model was included as False.

(II) Used ‘imagenet’ as weights.

(III) Resized the input shape as (224,224,3).

(IV) Maxpolling was used here.

**Maxpolling**: Maximum pooling, or max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling.

Let's say we have a 4x4 matrix representing our initial input. Let's say, as well, that we have a 2x2 filter that we'll run over our input. We'll have a stride of 2 (meaning the (dx, dy) for stepping over our input will be (2, 2)) and won't overlap regions.

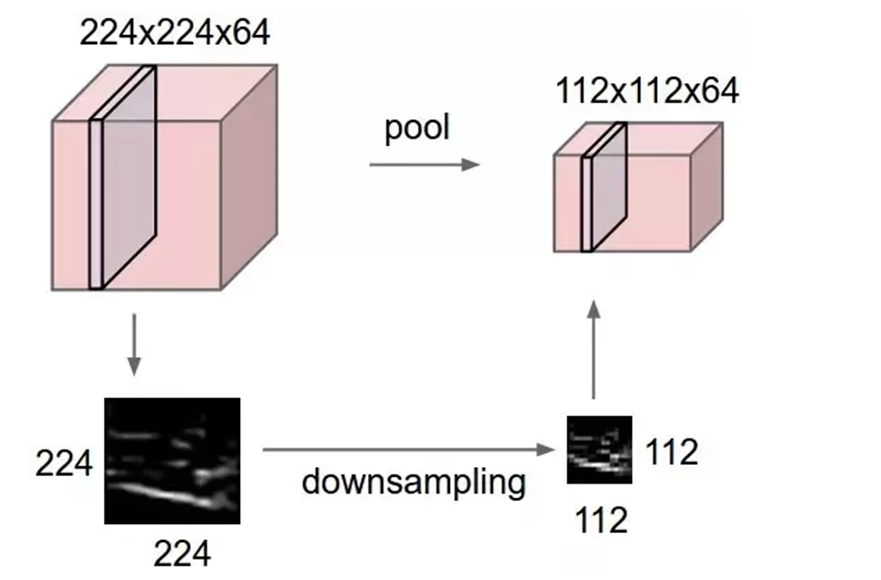
For each of the regions represented by the filter, we will take the max of that region and create a new, output matrix where each element is the max of a region in the original input.

**Pictorial representation:**

[](https://computersciencewiki.org/index.php/File:MaxpoolSample2.png)

**Figure 4:** Maxpolling

**Real-life example:**



**Figure 5:** Maxpolling in real life uses.

(V) Base Model Trainable layers are included as false.

(VI) Flatten the Base Model Output. **Flatten** is the function that converts the pooled feature map to a single column that is passed to the fully connected layer. Dense adds the fully connected layer to the **neural network**.

(VII) There are 3 Dense layer. In the middle layer, we used an activation function called ReLu. **ReLu:** A recent invention which stands for Rectified Linear Units. The formula is deceptively simple: max(0,z)max(0,z). Despite its name and appearance, it’s not linear and provides the same benefits as Sigmoid but with better performance. And last layer is output layer of 8 classes where Softmax activation function was used.

**Softmax:** Softmax function calculates the probabilities distribution of the event over ‘n’ different events. In general way of saying, this function will calculate the probabilities of each target class over all possible target classes. Later the calculated probabilities will be helpful for determining the target class for the given inputs. We used it at output layers to identify the proabibilty of every class of the Tomato Disease prediction Model.

**(d) Image classification & Detection of Diseases:**

After using Transfer Learning in our model, we had to compile and fit the model.

**Compiling the model:** For compiling the model, we need loss, optimizer and metrics.

**(I) Loss:** Neural networks are trained using stochastic gradient descent and require that to choose a loss function when designing and configuring your model.There are many loss functions to choose from and it can be challenging to know what to choose, or even what a loss function is and the role it plays when training a model.

**Categorical\_Cross\_entropy loss**, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.

**(II) Optimizer: Adam** is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

**(III) Metrics: Accuracy** is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition: Accuracy = Number of correct predictions Total number of predictions. The other *c*lassification Metrics are- **precision, recall, F1-score** etc.

**Fitting and saved the Model:**

We put training and validation datasets for fitting the model and used total 70 epochs for classify the images and detect the diseases of tomato plant. After that we saved the model and know the experimental results of the model.

By this way, we can predict the class, mainly different types of diseases of tomato plant perfectly. Though the model is not so perfect, there is some loss which is given in Expermental Result.

**8. System Architecture:**

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. A system architecture consists of **s**ystem components and the sub-systems developed, that will work together to implement the overall system.

Xception

Training set

Dataset

Data Augmentation & Data preprocessing

Validation set

Feature extraction & classification

Applying on testing data

User Interface

Tomato Plant Disease Detection

**Figure 6:** System Architecture of the model

**9. Experimental Results:**

We are not directly saying that this model or architecture is perfect. We have need some comparison between our proposed models. The comparison is based on accuracy, loss, recall, confusion matrix, F1-score.

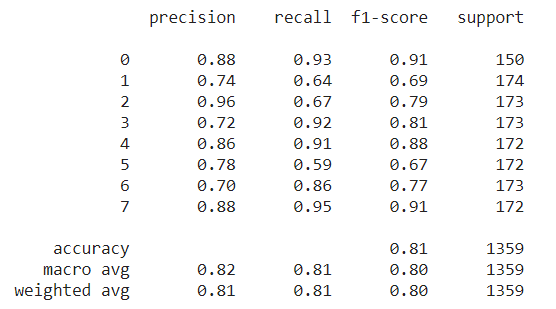
The accuracy and loss of training section and validation section are given below:

**Accuracy** **Loss**

**Training** 93.33% 18.32%

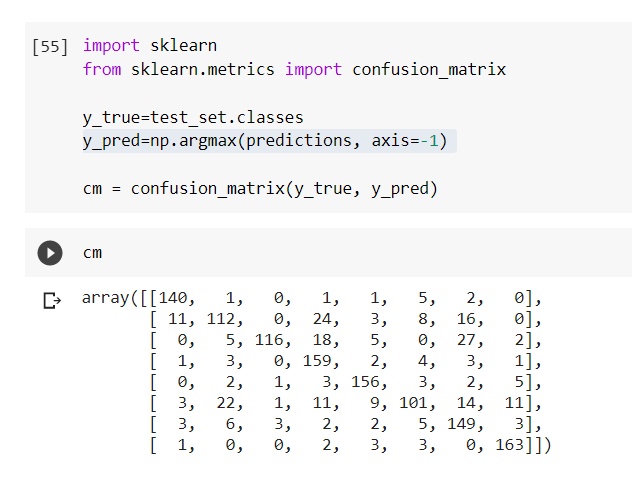
**Validation** 87.82% 37%

In this model, we used classification report for **recall** and **precision**. A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.

****

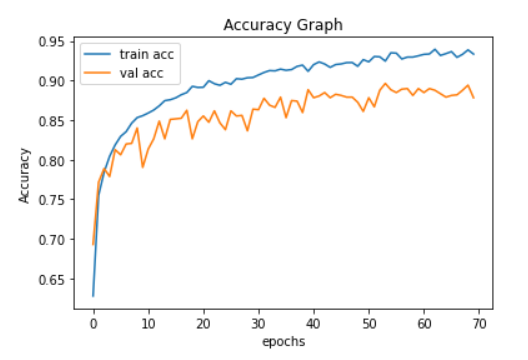
**Figure 7:** Recall and precision of the model

**Confusion matrix:** Here we used confusion matrix. In the field of machine learning, and specifically the problem of [statistical classification](https://en.wikipedia.org/wiki/Statistical_classification" \o "Statistical classification), a **confusion matrix**, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) one (in [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning" \o "Unsupervised learning) it is usually called a **matching matrix**). Each row of the [matrix](https://en.wikipedia.org/wiki/Matrix_(mathematics)" \o "Matrix (mathematics)) represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa – both variants are found in the literature. The name stems from the fact that it makes it easy to see whether the system is confusing two classes (i.e. commonly mislabeling one as another).

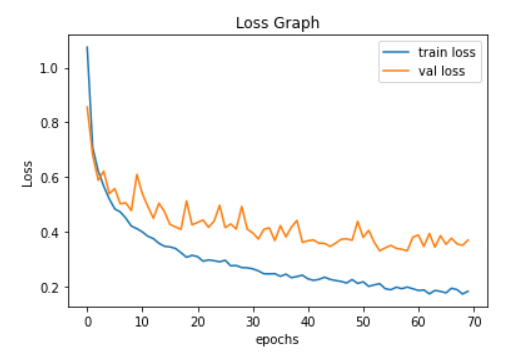
****It is a special kind of [contingency table](https://en.wikipedia.org/wiki/Contingency_table" \o "Contingency table), with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).

**Figure 8:** Confusion matrix of the model

**Plotting graph: Matplotlib** is a comprehensive library for creating static, animated, and interactive visualizations in Python. Here we used Matplotlib for our accuracy graph and loss graph. The accuracy graph and loss graph are given below. In these graphs, multiple axes are created with epochs and accuracy & epochs and loss respectively.



**Figure 9:** Accuracy graph

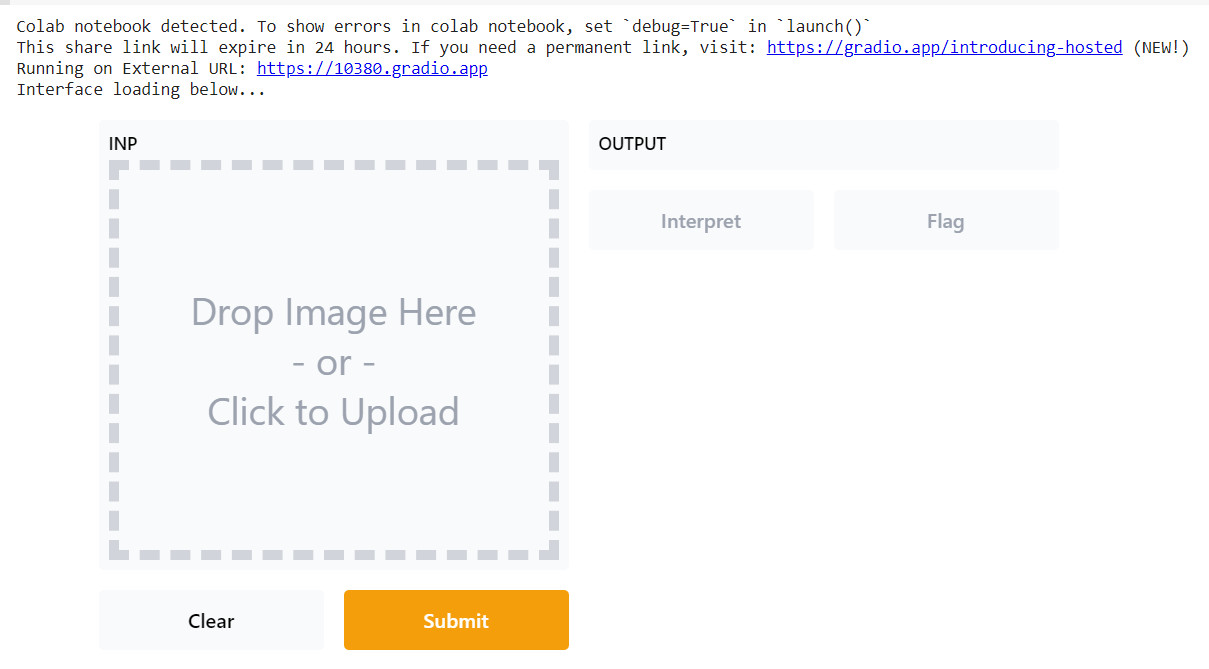
****

**Figure 10:** Loss graph

**10. User Interface:**

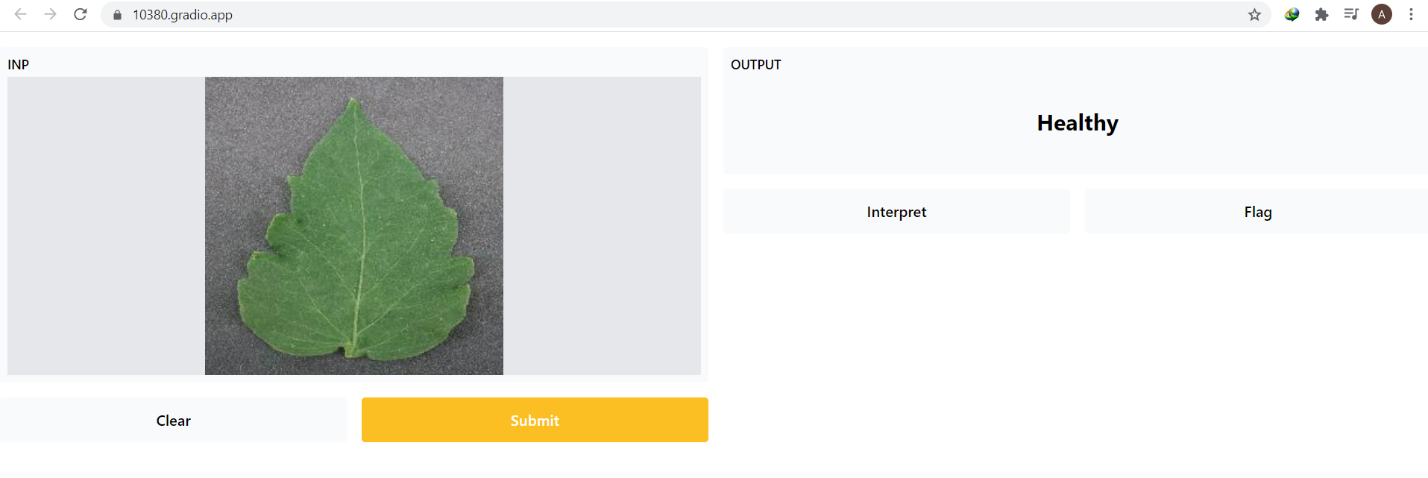
The user interface (UI) is the point of human-computer interaction and communication in a device. This can include display [screens](https://whatis.techtarget.com/definition/screen), [keyboards](https://whatis.techtarget.com/definition/keyboard), a mouse and the appearance of a [desktop](https://searchenterprisedesktop.techtarget.com/definition/desktop). It is also the way through which a user interacts with an [application](https://searchsoftwarequality.techtarget.com/definition/application) or a [website](https://whatis.techtarget.com/definition/Web-site).

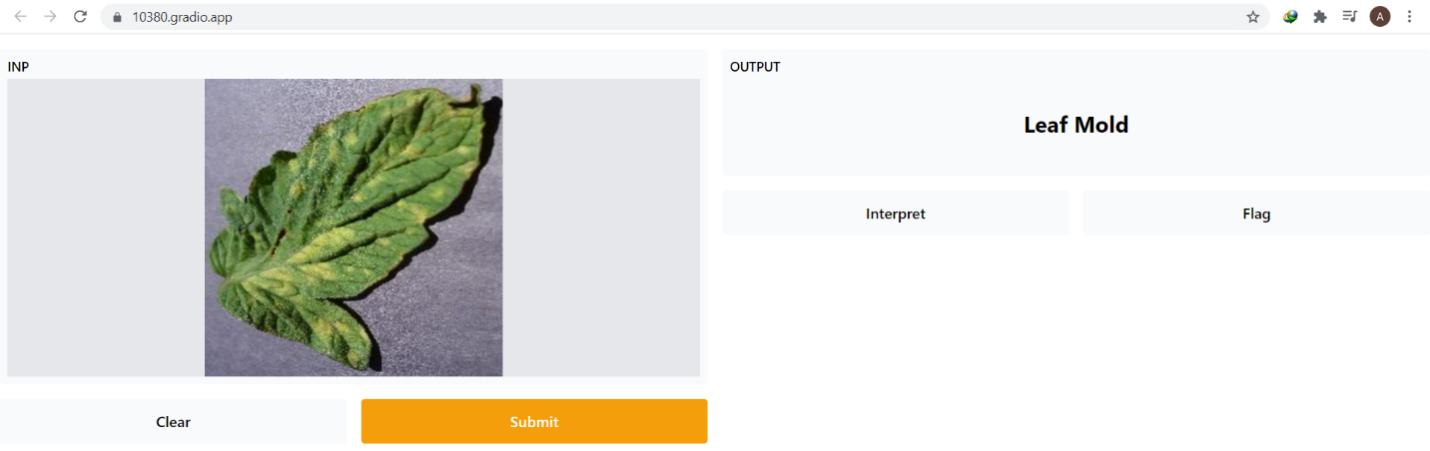
Here we have used gradio for our user interface. Gradio allows to quickly create customizable UI components around TensorFlow or PyTorch models, or even arbitrary Python functions. Mix and match components to support any combination of inputs and outputs.

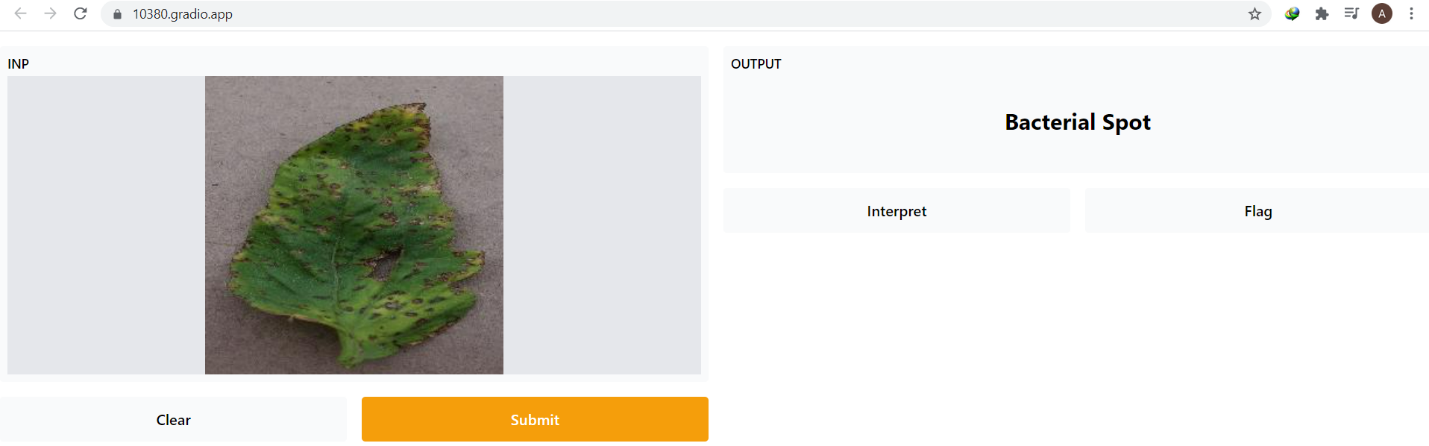


**Figure 11:** Input and output section of User Interface

After clicking the external URI link: <https://10380.gradio.app> we get our desirable output.







**Figure12:** Showing output in Gradio

**11. Challenges:**

**1**. Collection of images from tomato field

**2.** Disease segmentation with their symptoms

**3.** Hyper parameter tuning to get better accuracy

**4.** If some diseases have similar characteristics, then it is difficult to distinguish

**12. Future Work:**

(I) Rice Plant Disease Recognition & use it on an Android App.

(II) Face Mask Detection.

(III) Traffic Sign Recognition.

(IV) Image Caption Generator.

(V) Cancer Cell Detection.

(VI) COVID-19 Detection using chest x-ray.

**13. Conclusion:**

In this work, specialized deep learning models were developed, based on specific convolutional neural networks architecture (Xception), for the identification of tomato plant diseases through simple leaves images of healthy or diseased plants. The training of the models was performed using an openly available database of huge amounts of photographs, taken in both laboratory conditions and real conditions in cultivation fields. The data comprises in 8 distinct classes of plant disease combinations, including some healthy plants. The successful model architecture, a transfer learning called Xception, achieved a success rate of 81% in the classification of previously unseen plant leaves images (testing set) by the model.

Based on performance, it becomes evident that transfer learning networks are highly suitable for the automated detection and diagnosis of plant diseases through the analysis of simple leaves images. In addition, the high importance of the existence of real conditions images (captured in the cultivation fields) in the training data, which was indicated by the presented results, suggests that, in the development of such models, focus should be given in the maximization of the ratio of real-conditions images in the training data. Furthermore, the low computational power required by the trained model to classify a given image (about 2 ms on a single GPU), makes feasible its integration into applications for their uses. In the former case in particular, except for the fact that a farmer at a remote location could have an incipient warning about a possible threat for his/her cultivation, and an agronomist could have a valuable advisory tool in his/her disposal, a future possibility could be the development of an automated pesticide prescription system that would require a confirmation by the automated tomato plant disease diagnosis system to allow the purchase of appropriate pesticides by the farmers. So, The proposed deep learning approach showed its better potential, thus it is a matter of quantity and quality of available data to improve the system, and make it wider (in terms of plant species and diseases that can be identified) and more robust in real cultivation conditions.