Plant-Seedling Classification

(Project Report)

On

Data Mining and Analysis (CS 655C)

Project Submitted in Partial Fulfilment of the Requirements for the Award of Bachelor in Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

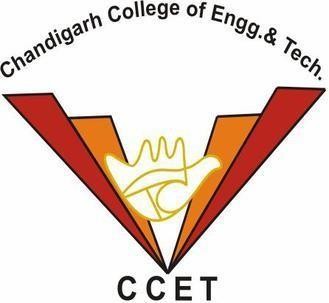
Nikhil Anand (CO17342)

Lavpreet Singh (LCO17368)

Yamini

(LCO17380)

Under the supervision of Dr. Varun Gupta



CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)

Government Institute under Chandigarh (UT) Administration, Affiliated to Punjab University, Chandigarh

Sector-26 Chandigarh PIN-160019 Jan-June, 2020

Seedling Classification Using Tensorflow

Abstract: Agriculture is very important to human continued existence and remains a key driver of many economies worldwide, especially in underdeveloped and developing economies. There is an increasing demand for food and cash crops, due to the increasing in world population and the challenges enforced by climate modifications, there is an urgent need to increase plant production while reducing costs. Preceding instrument vision methods established for selective weeding have confronted with major challenges for trustworthy and precise weed recognition. In this plant seedlings classification approach is presented with a dataset that contains images of approximately 960 unique plants belonging to 12 species at several growth stages It comprises annotated RGB images with a physical resolution of roughly 10 pixels per mm. Image classification has become one of the most important problems that Machine learning and deep learning can solve.

The problem here is the weed seedling is much like crop seedling and our goal is to be able to differentiate between them using Machine learning and deep learning techniques.

Keywords—image classification, Convolution neural networks (CNN), kaggle, machine learning, deep learning

# **I. Definition**

Image classification is the process of taking an input (like a picture) and outputting a class (like “cat”) or a probability that the input is a particular class

Image classification refers to the task of extracting information classes from a multiband raster image. The resulting raster from image classification can be used to create thematic maps. Depending on the interaction between the analyst and the computer during classification, there are two types of classification: supervised and unsupervised.

In This project we have used one of Kaggle Competition’s dataset, this dataset contains images of approximately 960 unique plants belonging to 12 species at several growth stages It comprises annotated RGB images with a physical resolution of roughly 10 pixels per mm. The problem here is the weed seedling is much like crop seedling and our goal is to be able to differentiate between them using Machine learning and deep learning techniques.

**Problem statement**

The target of this project is to distinguish between the different weed seeding and crop seeding of 12 different plant species. Hence its multiclassification problem.

As we take .png image of a weed or a crop seedling and output the correspondent specie from our 12 classes. We are using CNN model to classify the plants. CNN is widely used in the field of computer vision for doing complicated task.

**Traditional image classification approach**

Traditional image processing approach involves interconnected steps such as segmentation, feature extraction and classification.... Many methods have been proposed for classification of WBCs using neural networks. It is a supervised machinelearning algorithm which consists of input layer, hidden layer and output layer.

**Deep learning neural networks in image classification**

The convolutional neural network (CNN) is a class of **deep learning neural network**. CNNs represent a huge breakthrough in image recognition. They’re most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook’s photo tagging to self-driving cars. They’re working hard behind the scenes in everything from healthcare to security. They’re fast and they’re efficient. But how do they work? Image classification is the process of taking an **input** (like a picture) and outputting a **class** (like “cat”) or a **probability** that the input is a particular class (“there’s a 90% probability that this input is a cat”). You can look at a picture and know that you’re looking at a terrible shot of your own face, but how can a computer learn to do that?

With a convolutional neural network!

A CNN has

* Convolutional layers
* ReLU layers
* Pooling layers
* a Fully connected layer
* A classic CNN architecture would look something like this:

**Input ->Convolution ->ReLU ->Convolution ->ReLU ->Pooling ->  
ReLU ->Convolution ->ReLU ->Pooling ->Fully Connected**

A CNN **convolves** (not convolutes…) learned features with input data and uses 2D convolutional layers. This means that this type of network is ideal for processing 2D images. Compared to other image classification algorithms, CNNs actually use very little pre-processing. This means that they can **learn** the filters that have to be hand-made in other algorithms. CNNs can be used in tons of applications from image and video recognition, image classification, and recommender systems to natural language processing and medical image analysis.

CNNs are inspired by biological processes. They’re based on some [cool research done by Hubel and Wiesel in the 60s](http://klab.tch.harvard.edu/academia/classes/Neuro230/2014/readings/reading_assignment2_gk1852.pdf) regarding vision in cats and monkeys. The pattern of connectivity in a CNN comes from their research regarding the organization of the visual cortex. In a mammal’s eye, individual neurons respond to visual stimuli only in the receptive field, which is a restricted region. The receptive fields of different regions partially overlap so that the entire field of vision is covered. This is the way that a CNN works.

CNNs have an input layer, and output layer, and hidden layers. The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.

* Convolutional layers apply a convolution operation to the input. This passes the information on to the next layer.
* Pooling combines the outputs of clusters of neurons into a single neuron in the next layer.
* Fully connected layers connect every neuron in one layer to every neuron in the next layer.

In a convolutional layer, neurons only receive input from a subarea of the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer.

A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained! They’re learned while the network trains on a set of images. This makes deep learning models extremely accurate for computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

# **DATASET DESCRIPTION**

This project we will use one of Kaggle Competition’s dataset, this dataset contains images of approximately 960 unique plants belonging to 12 species at several growth stages It comprises annotated RGB images with a physical resolution of roughly 10 pixels per mm. The problem here is the weed seedling is much like crop seedling and our goal is to be able to differentiate between them using Machine learning and deep learning techniques.

1. Data exploration

● Visualize the distribution of data

2. Data preprocessing

● Check for null and missing values

● resize images

● apply segmentation and sharpening for images

● 5.1 Label encoding

● 6.1 Split training and validation set

2. CNN

● 2.1 Define the model

● 2.2 Set the optimizer and annealer

● 2.3 Data augmentation

4. Evaluate the model

● 3.1 Training and validation curves

● 3.2 Confusion matrix

The final model is expected to be useful for classify the 12 different image

species***.***

**METRICS**

Accuracy is a common metric for binary classifiers; it takes into account

both true positives and true negatives with equal weight.

● accuracy = true positives + true negatives/ dataset size

● Confusion matrix​ can be very helpful to see the model drawbacks.

● a confusion matrix is such that is equal to the number of

observations known to be in group but predicted to be in group.

**II. Analysis**

**Data exploration: -**

Plant seedling data set size is 1.6 GB divided into 12 folders in each one it contains number of images belong to certain class. I will split it later into training set and testing set.

● There are 12 types of plant seedling:

1-Black-grass

2-Common Chickweed

3-Loose Silky-bent

4- Shepherd's Purse

5-Charlock

6-Common wheat

7-Maize

8-Small-flowered Cranesbill

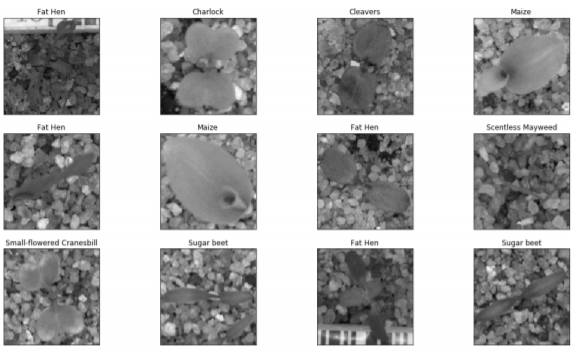
9-Cleaver0073`

10-Fat Hen

11-Scentless Mayweed

12-Sugar beet

Not all images have the same size (I make resizing to all images in the preprocess stage)



Here is a sample of one of those classes(charlock):



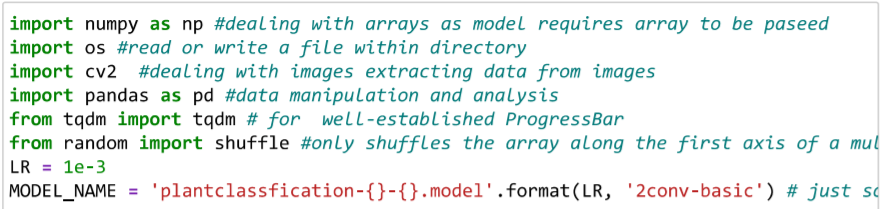
**Algorithms and Techniques**

The classifier is a CNN (Convolutional Neural Network), which is the state-of-the-art algorithm for most image processing tasks, including classification. It needs a large amount of training data compared to other approaches; and this dataset already are big enough to fit this criterion.

The algorithm outputs an assigned probability for each class; this can be used to reduce the number of false positive using a threshold.

(The tradeoff is that this increases the number of false negatives.)

Initially we are importing the libraries as shown below: -



NumPy: -NumPy is a python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

OS: - The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The \*os\* and \*os. path\* modules include many functions to interact with the file system.

CV2: - for importing the OpenCV first you need to follow these steps: - Install all packages related to OpenCV like Numpy and all into their default locations. Python will be installed to C:/Python27/. After installation, open Python IDLE. Enter import Numpy and make sure Numpy is working fine.

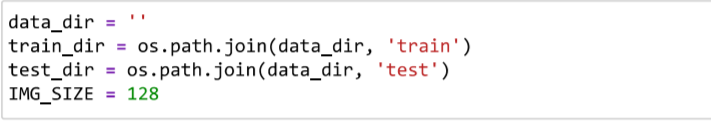
Panda: -Pandas is an open source Python package that provides numerous tools for data analysis. The package comes with several data structures that can be used for many different data manipulation tasks.

Shuffle: -Sometimes we want the computer to pick a random number in a given range, pick a random element from a list, pick a random card from a deck, flip a coin, etc. ... To get access to the random module, we add from randomimport \* to the top of our program (or type it into the python shell).

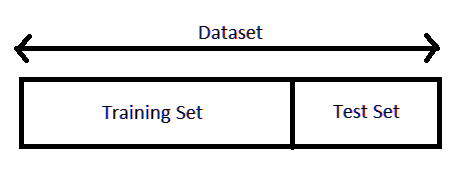
**TensorFlow: -TensorFlow** is basically a software library for numerical computation using **data flow graphs** where:

* nodes in the graph represent mathematical operations.
* edges in the graph represent the multidimensional data arrays (called tensors) communicated between them. (Please note that tensor is the central unit of data in TensorFlow).

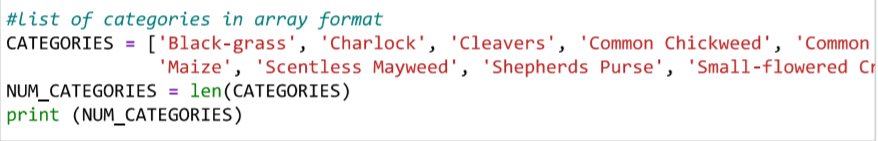
Here we are joining text and train data to the directory: -



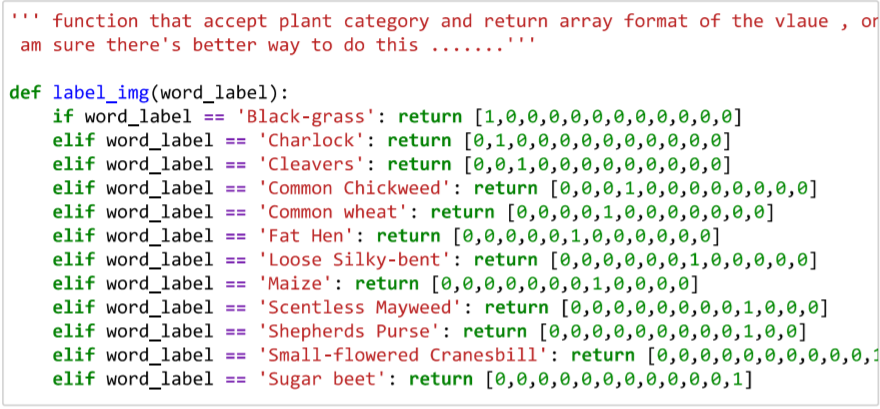
As we work with datasets, a [**machine learning algorithm**](https://data-flair.training/blogs/machine-learning-algorithm/) works in two stages. We usually split the data around 20%-80% between testing and training stages. Under supervised

learning, we split a dataset into a training data and test data in Python M

Here is the list of all species in array format

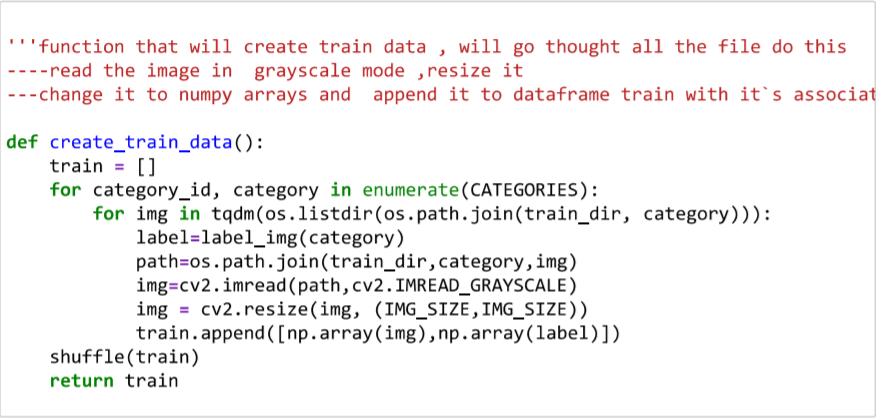


Here we are representing various species of plants in array format in computer understandable language which is binary language… 1 is representing the image is present or vice versa.

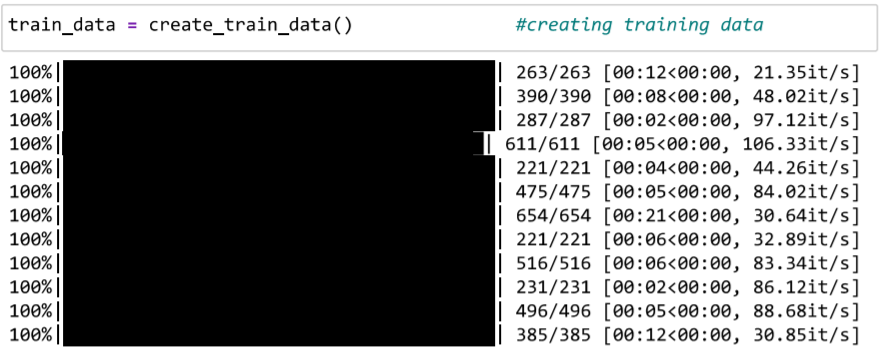


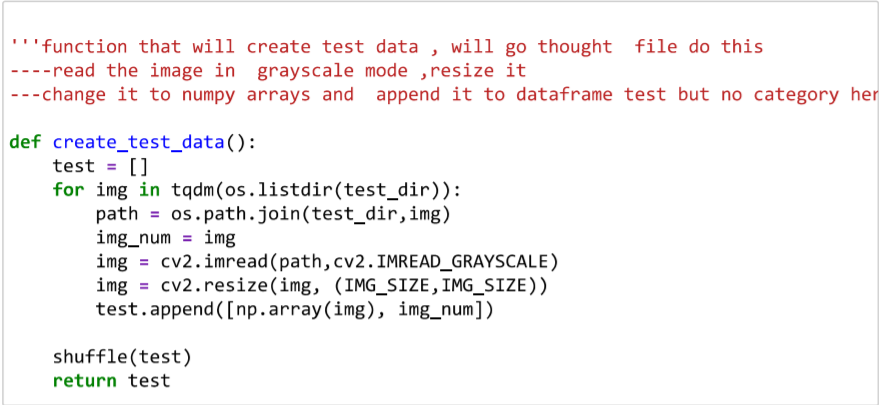
The trainingdata is an initial set of data used to help a program understand how to apply technologies like neural networks to learn and produce sophisticated results. It may be complemented by subsequent sets of data called validation and testing sets.

Firstly, function will create train data, will go through all the file do this after read the image in grayscale mode, realize it and after that change it to Numpy arrays and append it to data frame train with its associate



Machine Learning models are trained using data with specific features. The way in which the data is structured helps the models to learn and develop relationship between these features. A well-processed training set is required to build a robust model which in turn generates accurate results. In this article we shall look at some of the ways in which one can build a structured dataset for training. To build a robust model, one has to keep in mind the flow of operations involved in building a quality dataset. The data should be accurate with respect to the problem statement. For example, while trying to determine the height of a person, feature such as age, sex, weight, or the size of the clothes, among others, are to be considered. Here, the person’s clothes will account for his/her height, whereas the colour of the clothes and the material will not add any value in this case. Hence these features have very low weightage for predicting the height of a person. A golden rule of machine learning is: *Larger the data better the results.*

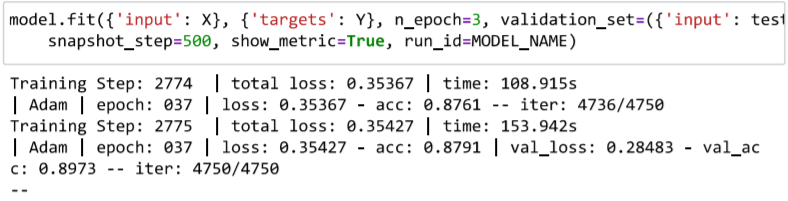


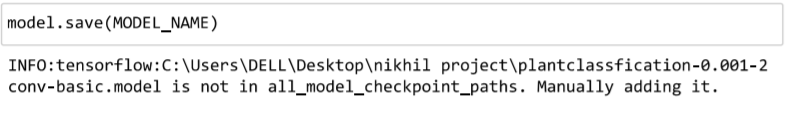


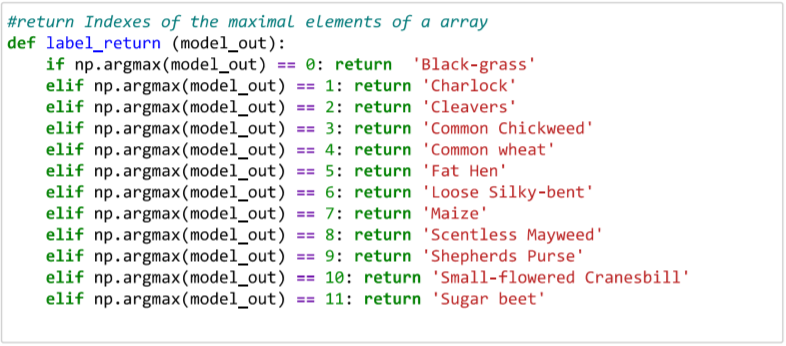


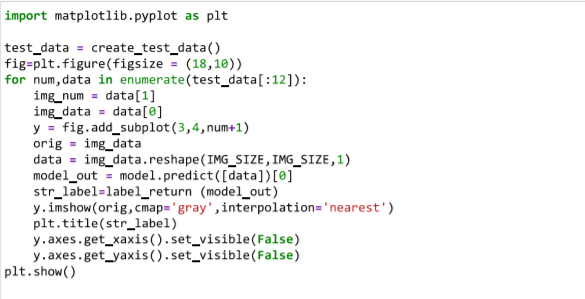


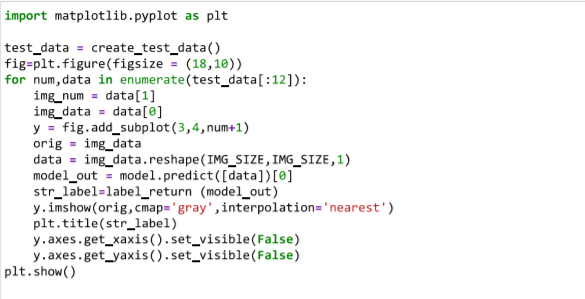
This picture showing training steps, total loss, time taken by the program to process



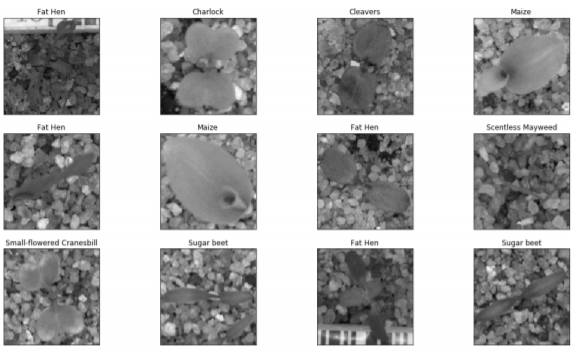




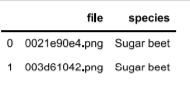






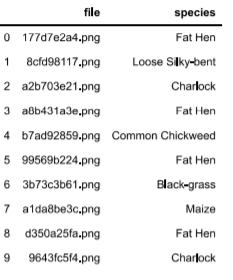


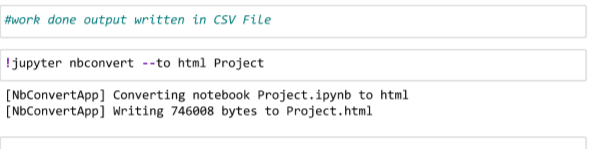












The following parameters can be tuned to optimize the classifier:

**❖**Classification threshold (see above)

* Training parameters
* Training length (number of epochs)
* Batch size (how many images to look at once during a single training step)
* Solver type (what algorithm to use for learning)
* Learning rate (how fast to learn; this can be dynamic)
* Weight decay (prevents the model being dominated by a few “neurons”)
* Neural network architecture **➢** Number of layers

**➢** Layer types ( convolutional , fully-connected , or pooling ) **➢** Layer parameters

* Preprocessing parameters (see the Data Preprocessing section)

**Benchmark:**

Plant seedling classification dataset is provided by Kaggle we benchmarked our model to another model from Kaggle kernels. It has score of 86.3% is a decent score when compared to 60.8% (benchmark model’s performance). I believe the final solution will definitely contribute significantly towards solving the current problem.

**III. Methodology**

**Implementation:**

The implementation process can be split into two main stages:

* Preprocessing data stage
* The classifier training stage

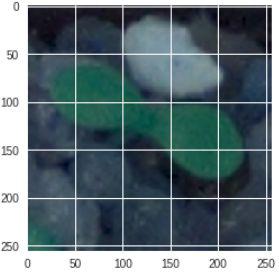
**Preprocessing:**

**1- Resizing Images:**

Images have not the same size so I have resized the images to 256\*256 pixel

to feed it later to the neural network

In this figure the image after resize:

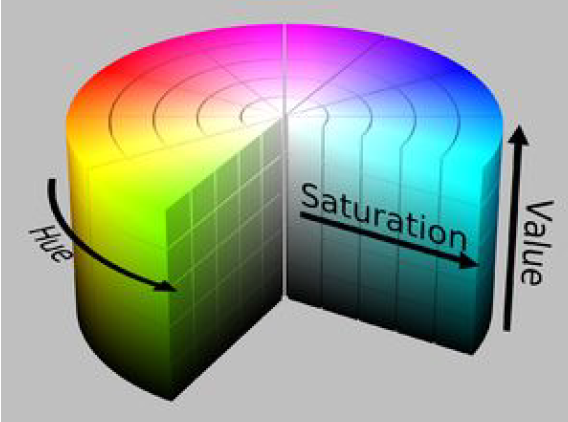


**2- Creating mask for the images: ​**

create\_mask\_for\_plant function returns an image mask: Matrix with shape (image\_height, image\_width). In this matrix there are only 0 and 1 values. The 1 value define the interesting part of the original image. I can create this mask using HSV of the image.

The HSV color-space is suitable for color detection because with the Hue we can define the color and the saturation and value will define "different kinds" of the color. (For example, it will detect the red, darker red, lighter red too). We cannot do this with the original BGR color space

This figure illustrates HSV space of the image​.



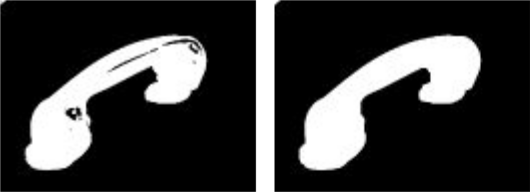
After converting RGB to HSV I started to apply ​morphological operations

[**​3- Morphological operations**](https://en.wikipedia.org/wiki/Mathematical_morphology)**​:**

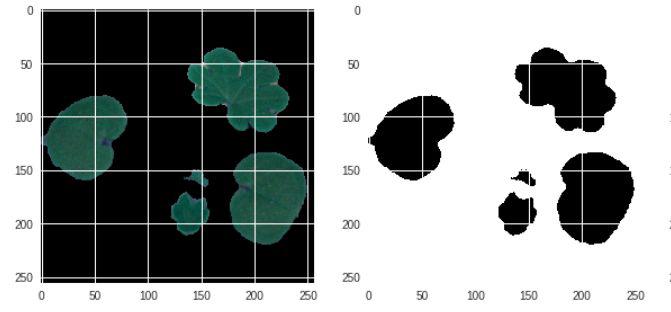
**​**one of most common morphological

operation is ​closing **​**: closing used to close the small halls in the images.

This figure below illustrates the image before and after applying closing:



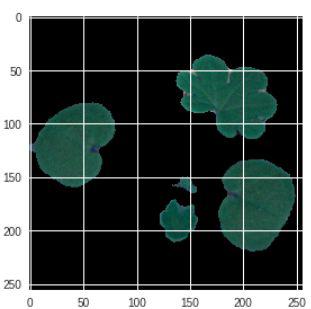
Here is sample after applying the mask on one of images:



**4- Segmentation:**

Segmentation is partitioning an image into distinct regions containing each pixels with similar attributes. To be meaningful and useful for image analysis and interpretation, the

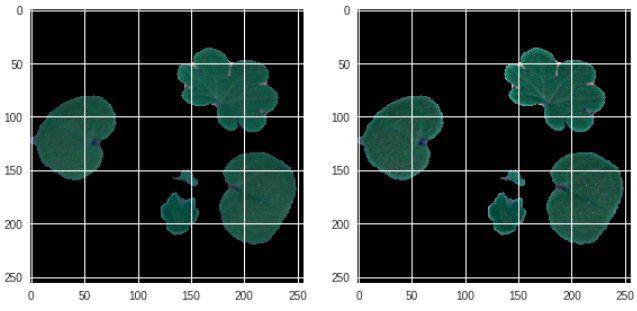
regions should strongly relate to depicted objects or features of interest. Here is a sample after applying segmentation on one of images:



**5- Sharpening:**

Sharpening an image increases the contrast between bright and dark regions to bring out features.

Image before and after sharpening:



**6- Label encoding:**

Using ​LabelBinarize**​​​** labels in a one-vs-all fashion

Input**: ​**the label of image and the​output **​**is vector represent the class in binaryform.

**7- Splitting data into training and testing set:**

In this step I used sklearn train\_test split 70% for training and 30% for testing Then we split testing data into testing and validation set 50% for testing and 50% for validation. The point of using the validation set is to try to avoid overfitting and check if an overfitting may occur

**The classifier training stage**

During the first stage, the classifier was trained on the preprocessed training data. This was done in a Jupyter notebook (titled “FINAL seedling.ipynp”), and can be further divided into the following steps:

1. Load both the training and validation images into memory, preprocessing them as described in

the previous section

the data required for a single training step (a batch of images, their labels, and the learning rate)

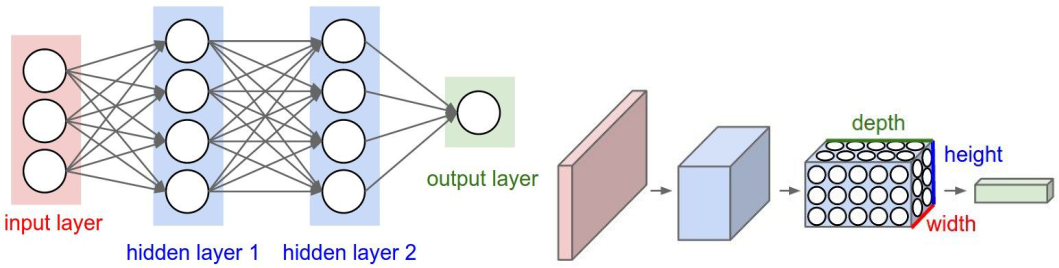
1. Define the network architecture and training parameters
2. Define the loss function, accuracy
3. Train the network, logging the validation/training loss and the validation accuracy
4. Plot the logged values
5. If the accuracy is not high enough, return to step 3
6. Save and freeze the trained network

**CNN Architecture:**

In general, Neural networks accept a single vector as input, transform it to a series of hidden layers, which in turn is made up of set of neurons that are fully connected to all neurons in the previous layer. Neurons of the same layer are independent and do not share any connections. After the hidden layers, is the last fully connected layer which is also called the ‘output layer’, where each node outputs score for each class. The downside of regular neural network is that they don’t scale well to full images. It's mainly because with images of decent size, the number of neurons and weights that the network must accommodate becomes unmanageable. This is where Convolutional Neural Network comes to rescue with its neurons arranged in 3 dimensions (width, height, depth).

Each of the layer in CNN accepts 3D input volume and transforms it into 3D output volume

Following is a simple visualization of how CNN arranges its neurons in 3 dimensions (width, height, depth):



Following are the layers that are used to build a CNN:

Input layer (w,h,d)

Input layer of shape (w,h,d) represents image of size ‘w x h’ and ‘d’ number of color channels. For example, for an image of size (256x256) with 3 color channels (HSV), the input layer will hold raw pixel values of the image as a vector of size [256,256,3]

**CONV layer:**

The CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. For eg, if we use 32 filters, CONV layer will output a volume equal to (256x256x3) Activation functions

Activation layer will apply an element wise activation functions, leaving the volume unchanged.

**POOL layer:**

Pool layer performs downsampling operation along the spatial dimensions (width, height), outputting a reduced volume than the previous layer. For eg, (128x128x32)

**Fully-Connected Layer:**

FC layer, also called as the dense layer, each neuron will be connected to all the neurons of the previous layer. FC layer when used as the output layer results in much reduced volume of size [1x1xm], where m is the number of categories that are to be predicted. Each of the nodes of fully connected layer outputs a score corresponding to a class score.

**Dropout layer**

Dropout layer is used as a method of regularization to combat over-fitting of the training set. It ‘drops’ neurons at random (depending on the probability mentioned) while calculating the forward prop and backward prop, resulting in a simpler version of the CNN for each iteration and hence giving the model a hard time to overfit the training set.

**CNN algorithm**

It **​**consists of several convolution (CNV) operations followed ofthe image sequentially which is followed by pooling operation (PL) to generate the neurons feed into fully connected (FC) layer.

We used the Keras Sequential API, where you have just to add one layer at a time, starting from the input.

**Input: ​**of CNV is typically 2D image data with HSV (hue saturation value)

**The first is the convolutional (Conv2D) layer​**: It is like a set of learnablefilters. I chose to set 32 filters for the two firsts conv2D layers and 64 filters for the 2nd convolutional layer and 128 filters for the two last ones.

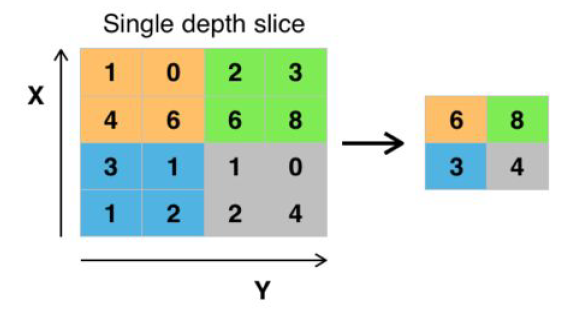
Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image.

The CNN can isolate features that are useful everywhere from these transformed images (feature maps).

**The second important layer in CNN is the pooling​ (MaxPool2D) layer.**

This layer simply acts as a down sampling filter. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size (i.e. the area size pooled each time) more the pooling dimension is high, more the down sampling is important.

This figure below illustrate Max pooling with a 2x2 filter and stride = 2



Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

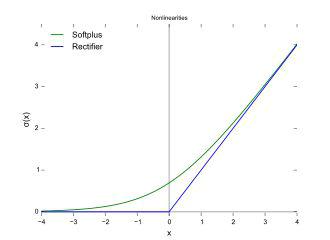
Dropout is a regularization method, where a proportion of nodes in the layer are randomly ignored (setting their weights to zero) for each training sample. This drops randomly a proportion of the network and forces the network to learn features in a distributed way. This technique also improves generalization and reduces the overfitting.

**ReLU Layer:**

ReLU is the abbreviation of Rectified Linear Units. This layer applies the non-saturating activation function. *f*(*x*) = *x*+ = *max* (0, *x*)

It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Figure below illustrate Relu function: Plot of the rectifier (blue) and softplus (green) functions near x = 0



**The Flatten layer ​**is use to convert the final feature maps into a one single 1Dvector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/ maxpool layers. It combines all the found local features of the previous convolutional layers.

In the end used the features in two fully-connected (Dense) layers which is just artificial neural networks (ANN) classifier.

In the last layer (Dense (10, activation="softmax ")) the net outputs distribution of probability of each class.

**Fully Connected (FC) Layer:**

This layer will reduce the size of input data to the size of classes that the CNN is trained for by combining output of CNV layer with different weights. Each neuron at the output of the CNV layer will be connected to all other neurons after weighted properly, Similar to CNV layer, weight of these taps in FC layer is found though backpropagation algorithm.

**Classification Layer (CL):**

This is the final layer of the CNN that converts the output of FC to probability of each object being in a certain class. Typically, soft-max type of algorithms is used in this layer.

**Refinement:**

* Decreasing learning rate from .01 to .001 avoid me the overfitting of the model.
* By adding data augmentation accuracy increased to : 91%
* Adding some additional layer also improved the model
* Changing dropout from .25 to .3 improved the accuracy for 86%

**IV. Results**

****

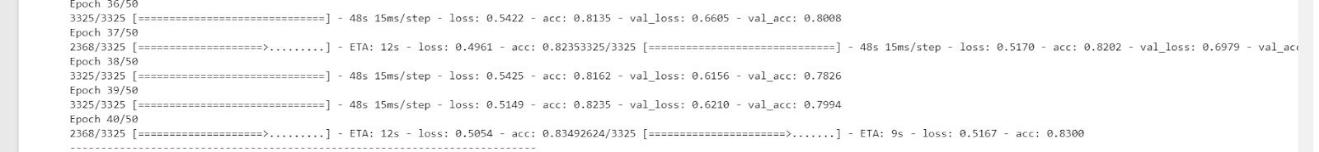
**Model Evaluation and Validation:**

During development, a validation set was used to evaluate the model. The final architecture and hyperparameters were chosen because they performed the best among the tried combinations.

complete description of the final model and the training process:

* The shape of the filters of the 1st and 2nd convolutional layers is 5\*5 and 3\*3 for the rest of convolutional layers.
* False positives are rare but present

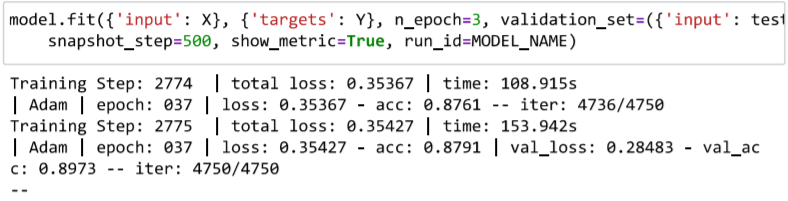
After testing on validation set it give the score in the image:



With Test loss: 0.35367 and Test accuracy: 108.915s

And after applying data augmentation

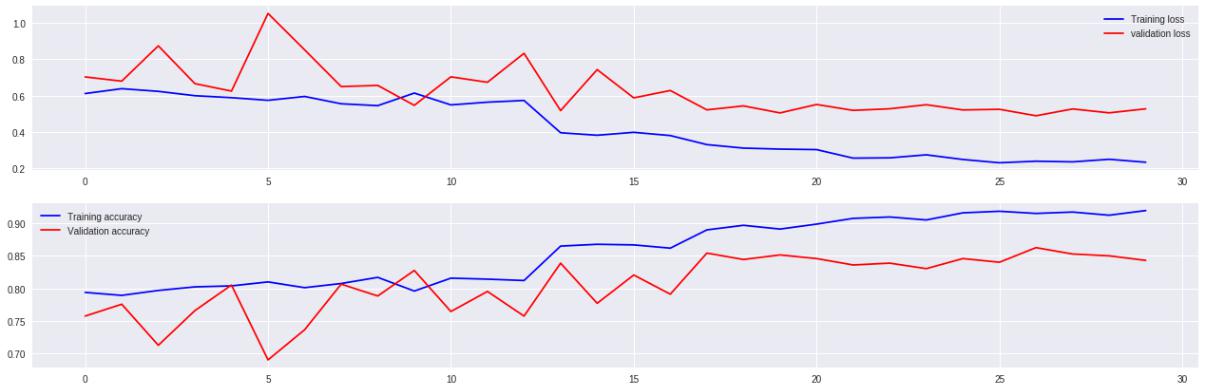
It gives a test score of .0.8973

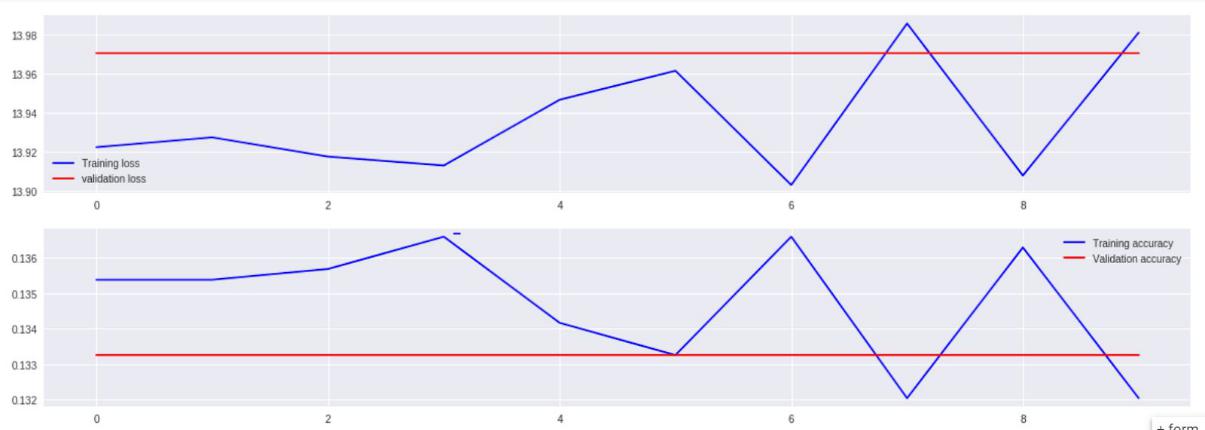


As for the evaluation and validation, I have used 2 main values to check:

Accuracy and validation loss.

Before tuning some parameters (before reducing learning rate)





From the figures we just shared, we can see that with 128 & 64 filters, we have reached a better score at 25 epochs and the validation accuracy is not dropping down

* Having gone through many trials, coding and fixing bugs, and coding again, the current accuracy is more than enough to be trusted. Of course, there'll be an error ratio, but the accuracy is very acceptable and robust enough to be depended on in real life applications.

**Justification**

The results of the final classifier are much better than that of the benchmark model. It has score of 89.7% is a decent score when compared to 60.8% (benchmark model’s performance). We believe the final solution will definitely contribute significantly towards solving the current problem and also with more training data and more preprocessing stages, there are possibilities of improving the model further.

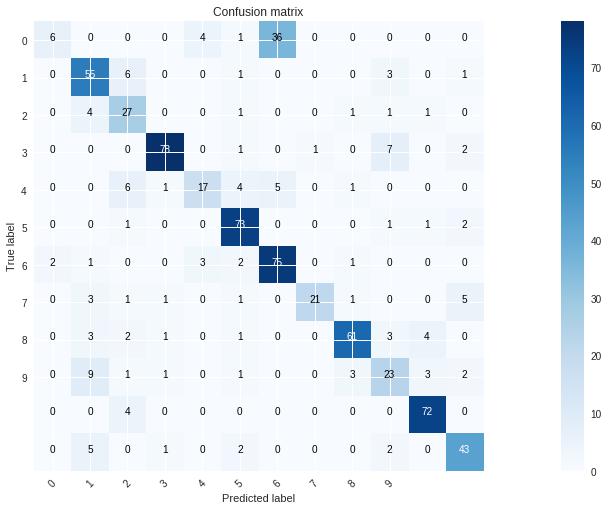
**Conclusion:**

To show the model quality we get the images which classified incorrectly in the test set this means the images that uncropped correctly can be misclassified

This model can help farmers to automate the task of classifying seedling plants and weed plants.

We plotted the confusion matrix to observe which category is poorly classified by the classifier and observe its performance visually. We can see in the below confusion matrix plot, that the major misclassification happened between Loose Silky-bent and Black-grass. It looks like the classifier is having difficulty classifying these two categories. We plotted the confusion matrix to observe which category is poorly classified by the classifier and observe its performance visually. We can see in the below confusion matrix plot, that the major misclassification happened between ​'Common wheat', 'Fat Hen' ​It looks like the classifier is having difficulty classifying these two categories. Hence, this is where the classifier needs improvement as these misclassification amount to 50% of the misclassification overall. We

believe overcoming this challenge will boost the F1-score significantly.



● Our final scores after model tuning is 89.7

**Reflection:**

We wanted to pick a problem that would give us much more clarity and exposure on how to deal with image classification problems, and we believe that choice of this problem has justified that need.

For this problem, after downloading the dataset from Kaggle, we loaded the dataset and converted the categorical file names into array of 1s and 0s using the labelbinarizer . We explored the data by visualizing the categorical distribution of the dataset by plotting a graph (using matplotlib) to check if the dataset is well balanced or not. After splitting the dataset into train, validation sets, the images were converted into 4D tensors for further processing. Once we built CNN models, we fed these tensors and evaluated the model’s performance using confusion metrics. we watched, over multiple iterations, how does the number of layers in the convolutional network have a significant effect on the performance of the classifier, how does adding dropout layers reduce potential overfitting. This project gave me a good insight on how to deal with future image classification problems and encouraged me to work on further improving my current model.

**Improvement**

As an improvement and future work, we would like to try data masking on the training set. Noise from the background of the images can be cancelled by masking images. We believe that without the background noise and restricting the visibility to the green leaves, the model can be trained better, and we may notice significant improvement in the performance.

Another implementation that can be tried is data augmentation. As the dataset is highly unbalanced, augmenting data to the under-represented classes might give a good boost to the total number of training images yielding a well-balanced dataset. Training the model on such dataset may give us significant improvement in the performance.

**References:**

* <https://en.wikipedia.org/wiki/HSL_and_HSV>
* <https://homepages.inf.ed.ac.uk/rbf/HIPR2/close.htm>
* [https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html](https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm) [/topic4.htm](https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm)
* [https://machinelearningmastery.com/evaluate-performance-deep-learning-models-kera](https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/) [s/](https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/)
* <https://www.kaggle.com/c/plant-seedlings-classification/data>