Plant Seedling Classification

Course: Big Data Analysis and Project
Name: Abhishek Das
ID: a1772359

ABSTRACT

Most of the economies in the world are based on agriculture. Farming and food productions has become survival of human beings. With unreliable weather conditions and increased pest growth, robust steps are required to help increase production of crops. This project aims to add to this effort by developing a model which will assist to classify plant and weed seedlings. The use of CNN network to visualize the dataset will be an improvement over other models as it leads to an increase in precision and accuracy. This increased accuracy increases the overall reliability of the model. This model has been developed using 4,275 images of seedlings of both plant and weed. The application of this model is just not limited to this specific dataset but can be used to classify other datasets as well. This increases the flexibility of the model making it more valuable.

INTRODUCTION

There has been a phenomenal growth in agriculture in recent times with the rise in population across the globe. One of the most critical tasks in agriculture is to differentiate between plants and weeds. It usually takes a lot of time and much money. In this project, *Kaggle* Competition datasets have been used, which contains images of 960 different plants belonging to 12 species at different growth stages. *Aarhus University Flakkebjerg Research* station recorded the database in a collaboration between the *University of Southern Denmark and Aarhus University*. The problem here is to differentiate between weed seedling and plant seedling. This type of problem is known as image classification. Image classification is one of the most critical tasks. Different classification methods can be used to classify images. *Convolution Neural Network* (CNN) is one of state-of-the-art algorithm which is used for computer vision and image classification. It helps farmers to automate this task (classify plants).

The project aims to differentiate between weed and crop seeds of 12 different plant species. The training data set size is 1.6 GB, which is divided into 12 folders; each one contains a different number of images belong to a different class. The 12 labels are as follows: *Black-grass; Loose Silky-bent; Charlock; Common Chickweed; Shepherd's Purse; Common wheat; Maize; Small-flowered Cranesbill; Cleavers; Fat Hen; Scentless Mayweed; Sugar beet*. The test data set is 85.97 MB. To fulfil the aim of our project, we need to build a multi-class classification model that can predict the unseen images accurately from our test data set. A CNN model is used for this classification problem. This model is expected to classify 12 different plan species. This type of Machine Learning algorithm can learn from a large volume of datasets and make predictions of future datasets.

ALGORITHM AND TECHNIQUES

In machine learning, a convolution neural network is a feed forward neural network. This state-of-the-art algorithm has wide applications in image and video recognition and natural language processing. *LeNet* was one of the first convolutional neural networks created by *Yann LeCun*. His remarkable work propelled the advancement in *Deep Learning*

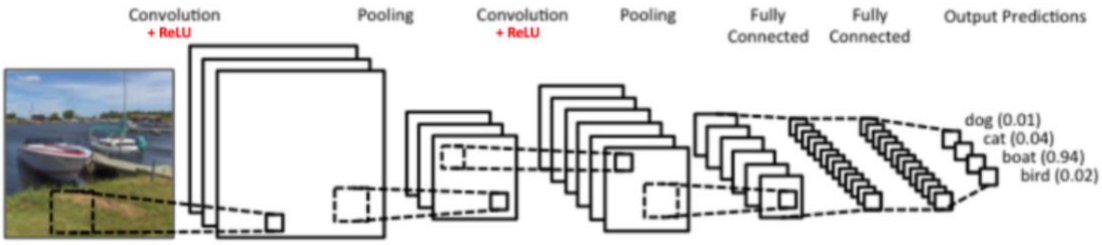


Figure 1: CNN model

There are mainly 4 important components in a CNN model:

1. Convolutional Layer:

Convolution is the first layer which extracts features from an input image. It preserves the relationship between pixels by learning image features using small squares of input data. Mathematically, it takes two inputs such as image matrix and a filter. This results in different operations such as edge detection, blur and sharpen.

2. Activation Function

The purpose of Activation Function is used to bring the non-linearity of ConvNet.They are also important to squash the linear weighted sum from the neurons. ReLU which stands for Rectified Linear Unit is the most widely used activation function

3. Pooling Layer

The objective of this layer is to reduce the number of parameters when images are very large. Spatial pooling which is also known as downsampling reduces the dimensionality of the data.

Spatial pooling can be of different types:

Max Pooling
Average Pooling
Sum Pooling

For instance, in max pooling we take the largest element from the feature matrix.

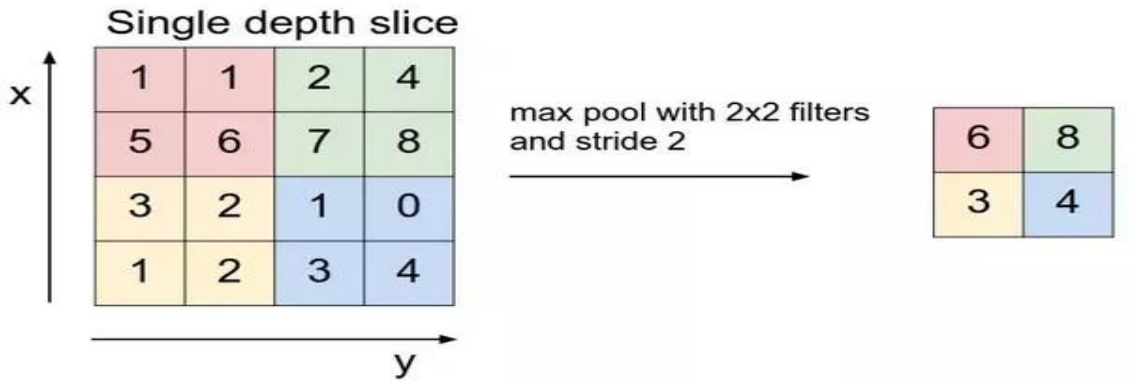


Figure 2: Example of Max Pooling

4. Fully Connected Layer

In this layer, every neuron in the previous layer is connected to every neuron on the next layer. It is simple a traditional Multi-layer Perceptron that uses a SoftMax activation function in the output layer.

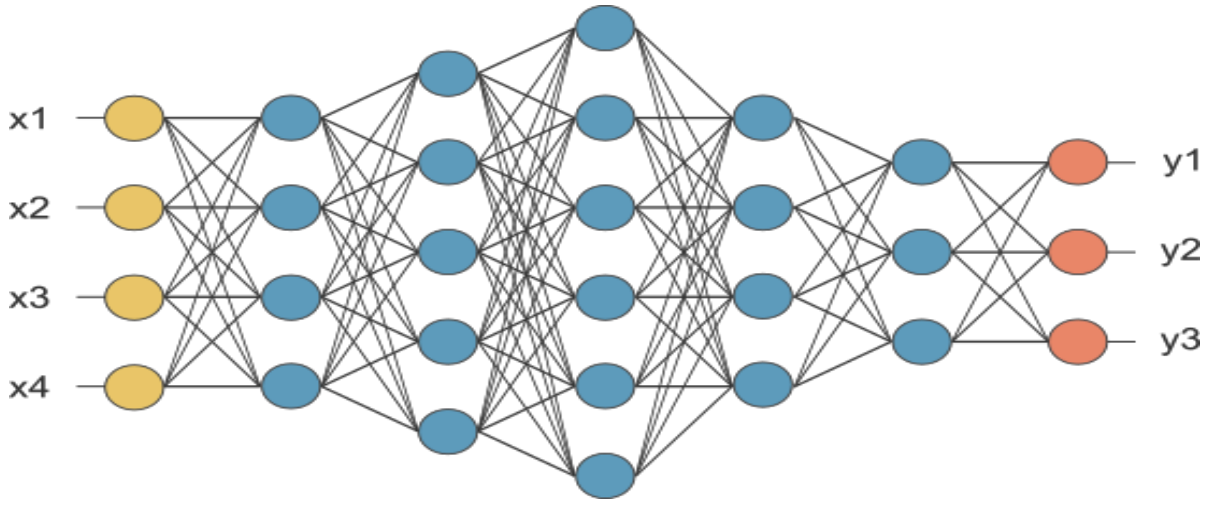


Figure 3: FC layer

There are a lot of other algorithms that can be used for image classification before CNN became popular. However, CNN has emerged as the model of choice for multiple reasons. As explained above, CNN architecture effectively uses adjacent pixel information to downsample the image first by convolution and then combines the benefits obtained by a standard neural network.

IMPLEMENTATION:

EXPLORATION AND PRE_PROCESSING

As with any machine learning algorithm, we start with data exploration. We start with unzipping our training data folder and exploring the folders. The data set was divided into 12 folder each containing 12 folders containing .png images for each class.

```
In [118]:
import zipfile as zf
train_zip = zf.ZipFile(file_name)
train_zip.extractall()

In [3]:
!ls train
```

```
Black-grass                 Common wheat                Scentless Mayweed
Charlock                    Fat Hen                     Shepherds Purse
Cleavers                    Loose Silky-bent            Small-flowered Cranesbill
Common Chickweed            Maize                       Sugar beet
```

We read the images and find the number of images. We also observed that the Images do have not the same size. Therefore, we resized the images before moving forward to pre-processing. One thing to keep in mind while scaling down the images is to keep the aspect ratio, as otherwise it will impact the structural ratio in the data.

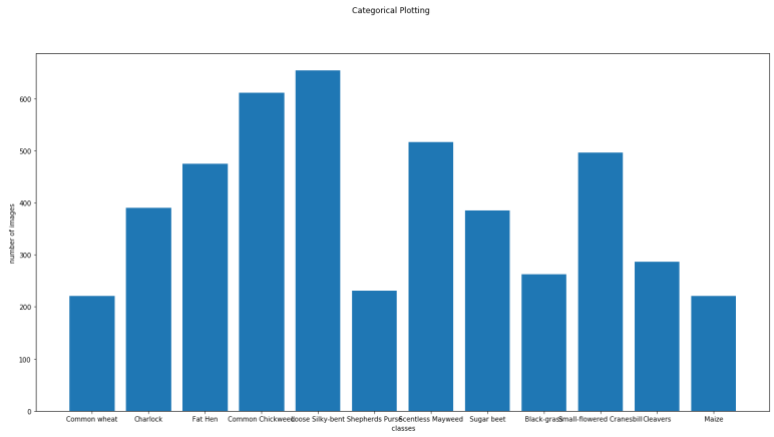
```
In [8]:
#resizing images
def resize_images(img):

  img = np.array(img).astype(np.uint8)

  res = cv2.resize(img,(256,256), interpolation = cv2.INTER_CUBIC)
  return res

In [9]:
#save resized images into images.
images = [resize_images(img) for img in images]
```

We have then explored the distribution of images in each class. We have used bar for the same purpose.

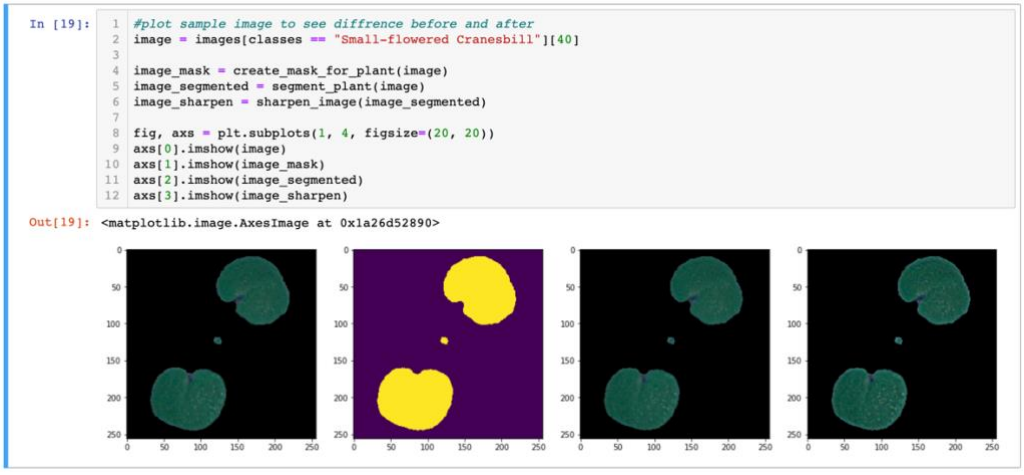


The function *create_mask_for_plant* will return an image mask. The input for this function is our image represented as a matrix of size with image length and image width and values 0 and 1. The purpose is to remove the background. We have created this mask using HSV of the image. The HSV is an alternative colour-space which is suitable for colour detection in our images as with the Hue we can define the colour i.e. green in our case and the saturation and value will define "different shades" of the green. Here, *cv2.MORPH_CLOSE* is used for closing small holes inside the foreground objects, or small black points on the object.

The function *segment_plant* takes an image as an input and partition it into multiple segments of pixels with similar features. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images.

The function *sharpen_image* is used to sharpen our image. *cv2.GaussianBlur* and *cv2.addWeighted* is used to remove the noise with the help of a smoothing filter and subtract the smoothed version from the original image (in a weighted way so the values of a constant area remain constant).

We will apply the image processing on the images and plot a sample image to see difference.



LabelEncoder() is used to encode labels and create classes. We then used *to_categorical* to convert labels into categorical values.

TRAINING AND EVALUATION

To train our data we will first split our data into training and testing set.

```
In [38]:
#splitting data into training and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(images, y, test_size=0.3, random_state=50)

In [23]:
X_train.shape

Out[23]: (3325, 256, 256, 3)
```

We will also create a validation set to evaluate our model.

```
In [39]:
#Create validation set
random_seed = 2
from sklearn.model_selection import train_test_split
X_test, X_val, y_test, Y_val = train_test_split(X_test,y_test, test_size = 0.5, random_state=random_seed)

In [40]:
print(X_test.shape)
print(X_val.shape)

(712, 256, 256, 3)
(713, 256, 256, 3)
```

We will use a Convolution Neural Network Model to train our data. Each layer in CNN accepts 3-dimension input in terms of RBG colour channel and it transforms in into a 3-dimension output. We have used a Kera Sequential model which adds each layer in a sequence with the following architecture.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 256, 256, 32)      2432
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 256, 256, 32)      25632
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 128, 128, 32)      0
_________________________________________________________________
dropout_5 (Dropout)          (None, 128, 128, 32)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 128, 128, 64)      18496
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 128, 128, 64)      36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 64, 64, 128)       73856
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 64, 64, 128)       147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
dropout_7 (Dropout)          (None, 32, 32, 128)       0
_________________________________________________________________
global_max_pooling2d_2 (Glob (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 12)                3084
=================================================================
Total params: 341,036
Trainable params: 341,036
Non-trainable params: 0
```

Conv2D is a set of filters that transforms a part of the image using kernel matrix It slides your kernel over the input, calculate the element-wise multiplications and sum them up. We have used 32 filters for the two firsts conv2D layers, 64 filters for the 2nd conv2D layer and 128 filters for the two last ones.

MaxPool2D is a downsampling strategy used in CNN. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions

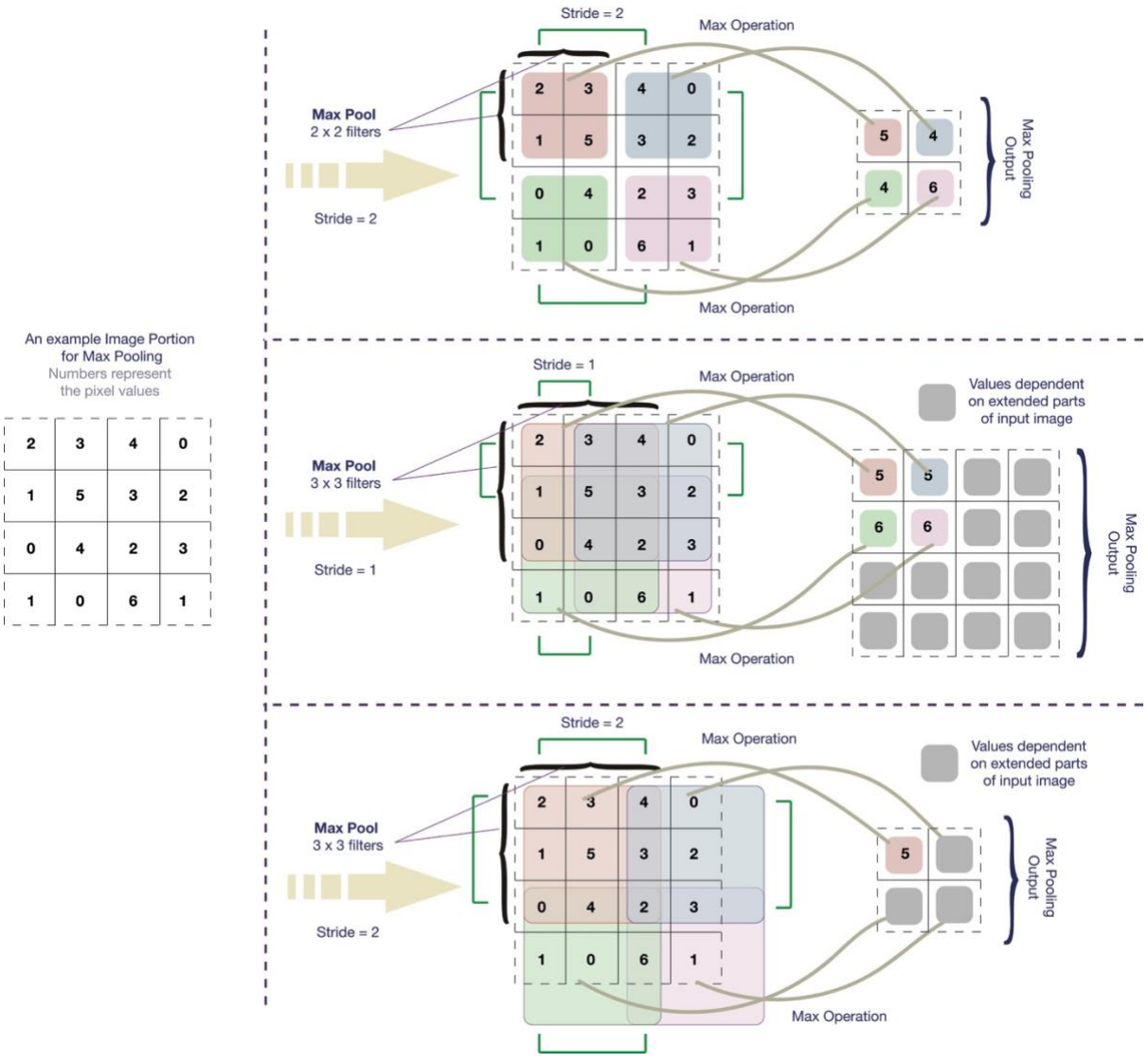


Figure 4: Max Pool

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.

Rectified Linear Units or RELU is used for the application of the non-saturating activation function. This function ca be expressed as:

$$f(x) = x^{+} = max\,(0, x)$$

The RELU layer increases nonlinearity of the decision function and also of the overall network. This is achieved without affecting the receptive fields in the convolution layer. The working of this function is illustrated in figure for better understanding.

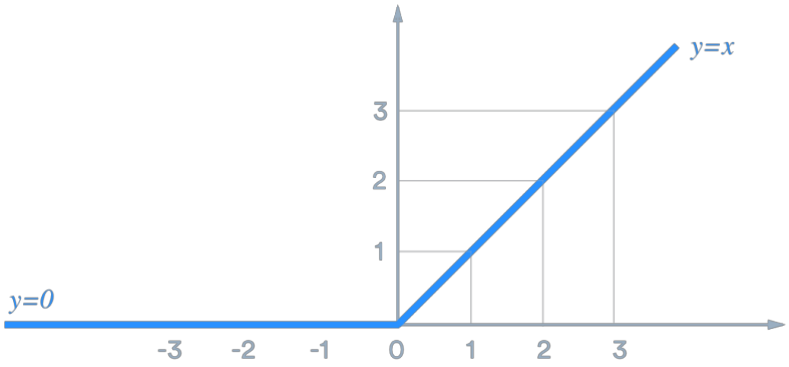


Figure 5: ReLU function

As the name suggest, the flatten layer is used to implement a flattening step by converting the ultimate feature map into a single one-dimensional vector. By flattening the fully connected layers can be fully utilised. It also combines all the local features of the previous convolutional layers. At the end this flattened vector has been fed into two fully connected dense layers.

The fully connected (FC) layer in a CNN represents the feature vector for the input. This feature vector/tensor/layer holds information that is vital to the input. After the training of the network is complete, this vector is used for further classification or as an input to another network. It may also be used as an encoded vector. This feature has been used in training for determination of loss and helps in the network training.

The final layer of the CNN is the classification layer. It is used to convert the output of FC layer to probability of each object being in a certain class.

```
In [42]:
opt = Adam(lr=0.001)

optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
model.compile(optimizer = optimizer, loss = "categorical_crossentropy", metrics = ["accuracy"])
model.fit(X_train, y_train, epochs = 50, validation_data = (X_val,Y_val),batch_size = batch_size)
```

We have used Adam optimization and RMSProp algorithm to as our update rul. Adaptive Moment Estimation (Adam) is a method that computes adaptive learning rates for each parameter. It stores both the decaying average of the past gradients. The main idea behind RMSProp is Divide the gradient by a running average of its recent magnitude. Both the optimisers are based on *Momentum*. It means the final slope is the weighted sum of current slope plus the previous slope. Intuitively, it means that if the current slope tells you to move in one direction, but the previous direction was some other direction, then continue to move in the previous direction. We tried both the optimiser and found that RMSProp works better in our dataset.
We used the validation set to evaluate our model. We set the number of epochs to 50. After training our model, we did an *early stop* when got a comparatively good accuracy of 0.8403. Early stopping is often done to avoid overfitting.

```
al_accuracy: 0.7798
Epoch 38/50
3325/3325 [==============================] - 585s 176ms/step - loss: 0.4961 - accuracy: 0.8361 - val_loss: 0.4951 - v
al_accuracy: 0.8555
Epoch 39/50
3325/3325 [==============================] - 585s 176ms/step - loss: 0.4722 - accuracy: 0.8412 - val_loss: 0.5936 - v
al_accuracy: 0.8205
Epoch 40/50
3325/3325 [==============================] - 586s 176ms/step - loss: 0.4750 - accuracy: 0.8361 - val_loss: 0.5664 - v
al_accuracy: 0.8205
Epoch 41/50
3325/3325 [==============================] - 589s 177ms/step - loss: 0.4900 - accuracy: 0.8334 - val_loss: 0.7220 - v
al_accuracy: 0.7321
Epoch 42/50
3325/3325 [==============================] - 587s 176ms/step - loss: 0.4887 - accuracy: 0.8403 - val_loss: 0.4354 - v
al_accuracy: 0.8640
Epoch 43/50
3104/3325 [===========================>..] - ETA: 36s - loss: 0.5330 - accuracy: 0.8338
```

We tried to improve the accuracy using data augmentation. The performance of neural networks often improves with more data. Data augmentation is a technique to artificially create new training data from existing training data. Image data augmentation is supported in the Keras deep learning library via the *ImageDataGenerator* class.

```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

The intent is to expand the training dataset with new, plausible examples. This means, variations of the training set image that are likely to be seen by the model. We have used the *callbacks* to get a log of internal states and statistics during model training
After applying data augmentation, we tried data augmentation again to get an accuracy of 0.9252.

```
Epoch 00023: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
Epoch 24/30
 - 595s - loss: 0.2274 - accuracy: 0.9169 - val_loss: 0.4109 - val_accuracy: 0.8569
Epoch 25/30
 - 595s - loss: 0.2183 - accuracy: 0.9197 - val_loss: 0.4052 - val_accuracy: 0.8654
Epoch 26/30
 - 602s - loss: 0.2259 - accuracy: 0.9177 - val_loss: 0.4010 - val_accuracy: 0.8569
Epoch 27/30
 - 592s - loss: 0.2124 - accuracy: 0.9189 - val_loss: 0.4095 - val_accuracy: 0.8513
Epoch 28/30
 - 604s - loss: 0.2185 - accuracy: 0.9197 - val_loss: 0.4048 - val_accuracy: 0.8555

Epoch 00028: ReduceLROnPlateau reducing learning rate to 1e-05.
Epoch 29/30
 - 606s - loss: 0.2115 - accuracy: 0.9229 - val_loss: 0.4080 - val_accuracy: 0.8527
Epoch 30/30
 - 596s - loss: 0.2108 - accuracy: 0.9252 - val_loss: 0.3935 - val_accuracy: 0.8569
```

RESULT AND ANALYSIS

Experiment	**Accuracy**
Without augmentation	0.82
With Augmentation	0.86

As the dataset is highly unbalanced and augmenting our dataset to the under-represented classes might give a good boost to the total number of training images resulting into a more-balanced dataset. Training the model on such dataset has given us significant improvement in the accuracy.

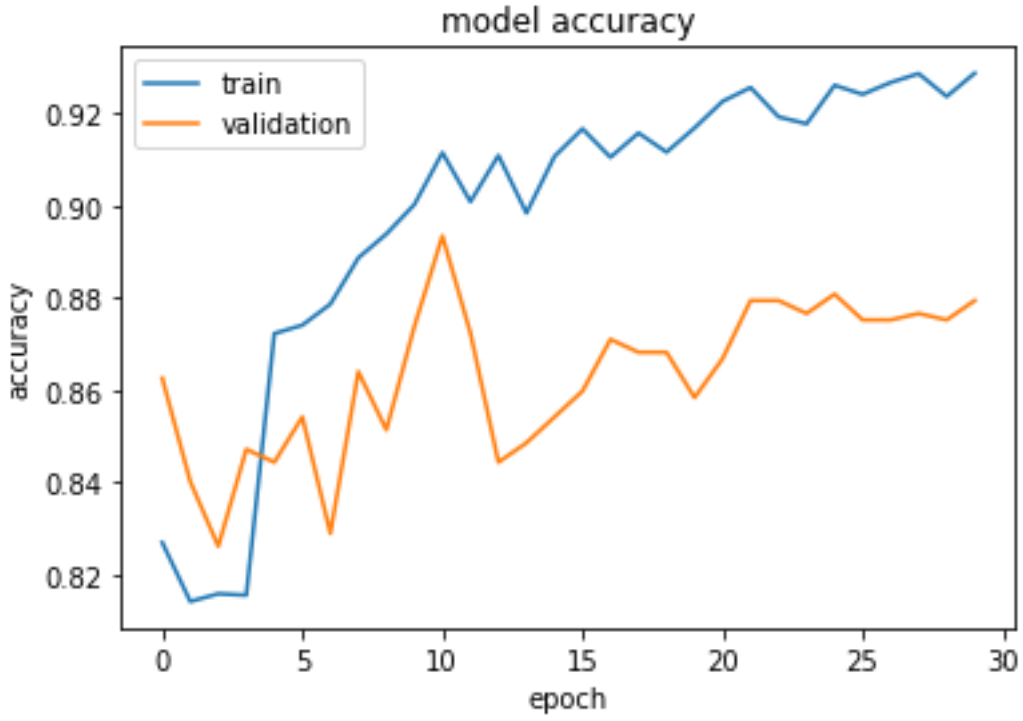


Figure 6: Model Accuracy

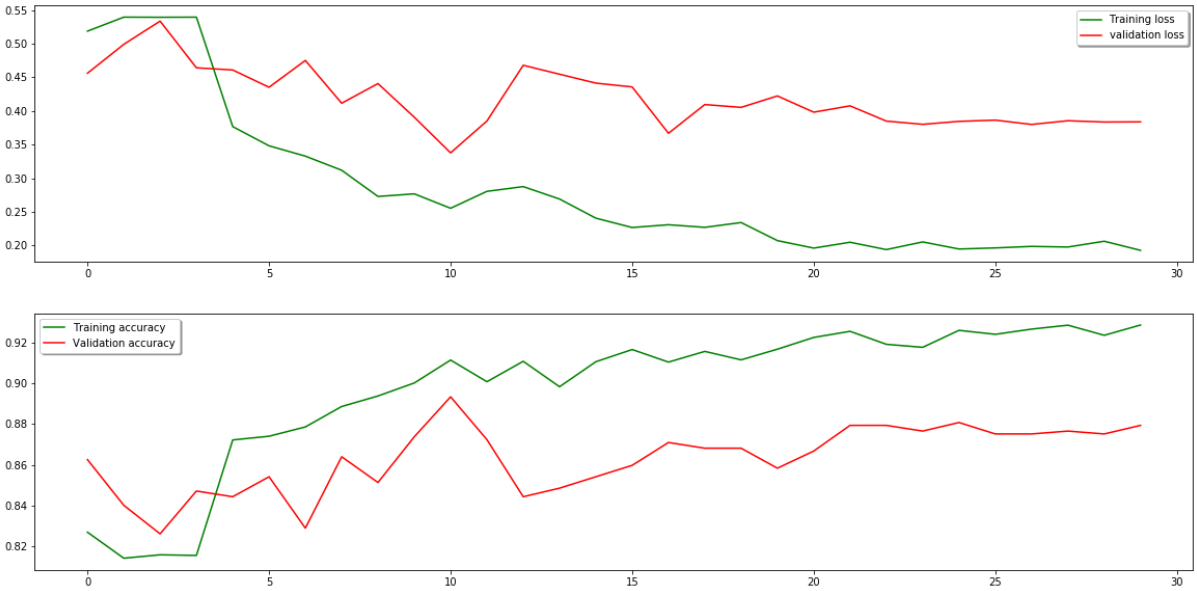


Figure 7: Model Accuracy vs Loss

In figure 6, we can observe that the training and validation monotonically increase with the number of epochs. Also, there is not a huge gap between our training accuracy and validation accuracy. Thus our model doesn't overfits It can also be seen that there is more fluctuation in the validation set in between 0 and 5 epochs. A dip in accuracy might just be some noise. From the figure 7, we can observe that we have attained a comparatively better score after 25 epochs. Also tuning the hyperparameters has helped to achieve a far better accuracy.

CONCLUSION:

In this project we developed a deep convolutional neural network method for plant seedlings classification. The dataset contains images of approximately 960 plants belonging to 12 species at several growth stages. The model can detect and differentiate a weed from other plants. Using our CNN model for image classification has given a good accuracy and precision in classifying complex datasets containing images of plant and weed seedling. This precision can be utilized to revolutionize farming industry by providing much required technology to farmers and other stake holders. We can also conclude that image preprocessing is an important part of image classification and give significant boost in the accuracy. Tuning right hyperparameters helps to improve the accuracy as well. This takes a lot of practice and experience. The long-term goal of this project would be improving the accuracy and make it more generalise. It can be done by trying different combination of layers in our CNN model or maybe tuning the hyperparameters.