Plant Seedlings Classification

Determine the species of a seedling from an image using various CNN architectures

by a1804938

School of Computer Science, The University of Adelaide

Report submitted for **4020_COMP_SCI_7209 Big Data Analysis and Project** at, University of Adelaide towards the Master of Data Science

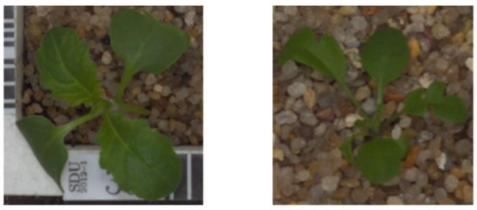
Abstract

Testing accuracy of 87.04% was achieved using VGG16 CNN architecture while a testing accuracy of 86.62% was achieved using Resnet-50 architecture. Model was trained on data acquired from Aarhus University Department of Engineering made available on Kaggle[1]. The dataset consists of various images of different species of seedlings on which extensive data processing is done for removing the background by methods like using Gaussian blur to remove noise, converting color to HSV, applying Boolean mask on the images and then normalizing the data which is then fit into various CNN models to acquire accuracy and loss.

1. Introduction

As we know that plants are crucial to human beings for various necessities such as for wood, food, medicines etc. In current time and also in future it will be critical to take care of plants with this constant rise in population of humans, increase demand for cheap and better quality food, and continuous climate changes. There are many plants and weeds which may look very similar and hence, there is a requirement to have a solution which can differentiate between them in early stages of their growth to increase the production of required plants and reduction in weeds and hence, increasing the efficiency of agriculture sector which can be seen the paper where researchers have developed a system to check site-specific weed control [2]. These old methods are being replaced by the implementation of deep machine learning in the process.

In below image we can see 2 different species which look so similar that it will be difficult to differentiate between them manually by naked eyes.



Charlock Shepherds Purse

Examples of Charlock and Shepherds Purse. We can see that they look very similar.

In machine learning one of the most frequently used technique is Image classification. This technique classify images such as classification of various objects, buildings, faces etc. There are various techniques for image classification, to name a few, SVM, Random Forest, XGB, K-NN, CNN etc. Although, CNNs - Convolutional Neural Networks are becoming more popular and widely used architecture for most of the image classification tasks since a boost in its popularity after a CNN model, won the ILSVRC – 2012 (ImageNet Large Scale Visual Recognition Challenge)[3].

In this project, we have used the dataset of Plant Seedlings which is used to classify seedling images into different plants categories which prevent the manual labor of individually distinguishing the weeds from plants. This report is focused on different Convolutional Neural Network (CNN) models to classify images from dataset published by Aarhus University Signal Processing group, in collaboration with University of Southern Denmark [4]. The dataset consists of training ad test set. The training set has 4750 color images under 12 classes, and test set contains 794 pictures to be classified.

2. Method

In our day to day life we see many things, hundreds and thousands of images every second which is processed by us, in human brain, which is the most powerful machine in the world. Our brain automatically recognizes any pattern to identify the object based on our focus on the image, different angles and views of the image. This principle of searching various patterns and differentiating the images is the foremost principle behind CNN - Convolutional Neural Networks.

So when a bunch of images are passed to a CNN model for training, the model will look for some patterns in the same way a human brain does, and then, when we ask CNN to identify the image, it will do so by recognizing the patterns in the image.

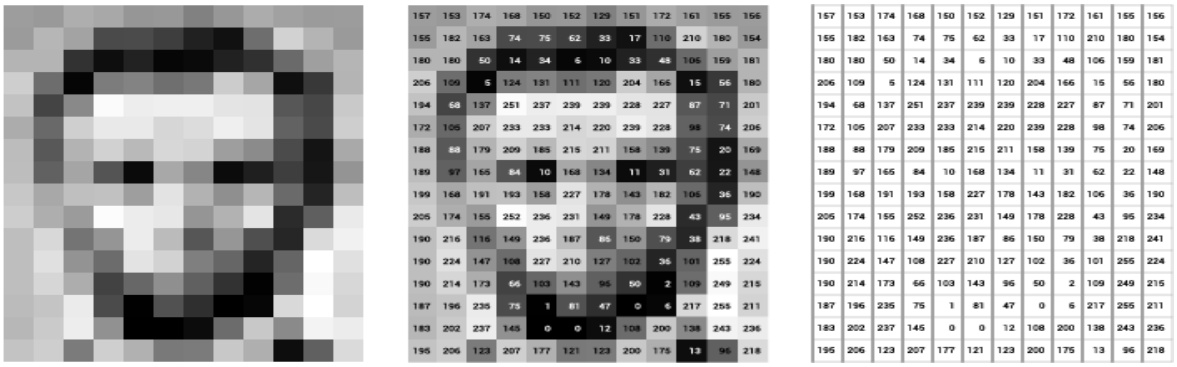


Fig 1[5]

In machines, images are read by the value of pixel. As we know that every image is a combination of its pixel values and hence, if we change any pixel value, the image as a whole will be altered. So, in CNN before these pixels are fed into the model, they need to be processed. Depending on the size of the image, number of pixel values will vary. So, if the image size is too large, we will face a problem when a fully connected network will flatten a 2 dimensional array of these pixels into one dimensional array to identify the image, that's where convolution layer comes.

The Convolution Layer:-

Convolution layer along with pooling reduces the size of image by extracting few features from the image. So if an image of 6*6 matrix is there then after convolution layer it can be converted into a 3*3 matrix based on the shape of our weight let's say 3*3. The weight matrix runs when applied on the image matrix, it will give an output of a convoluted matrix of size 3*3. The weight should cover all the pixels in the image matrix at least once by performing element wise multiplication.

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

*

1	0	-1
1	0	-1
1	0	-1

=

6	-9	-8
-3	-2	-3
-3	0	-2

In element wise multiplication, the elements/pixels on the edges are taken into consideration only once as compared to rest of the elements. This issue is solved by padding (shown in the below figure) which helps in preserving the size of the image. To minimize the loss function or to correctly predict the images, weights are learned similar to a multi-layer perceptron. In a real world problem, more than one convolution layer is applied and the number of filters will change the depth of convoluted layer and hence, the output is depended on the number of filters used.

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

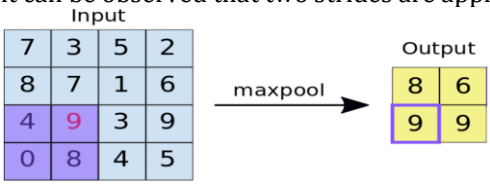
114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	421

Activation Function:

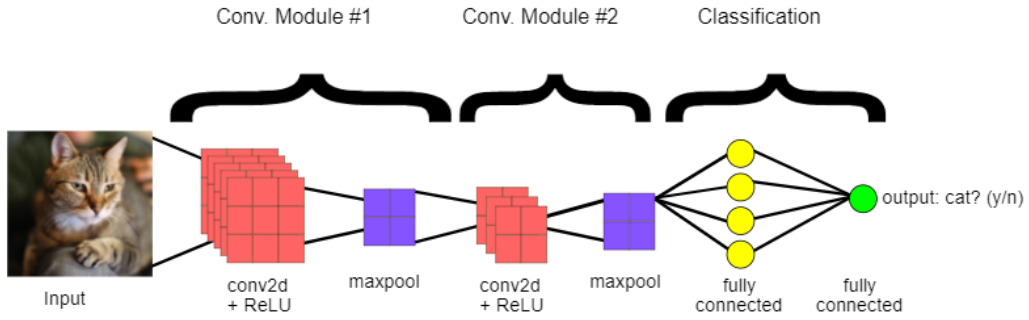
After each convolution layer, in a CNN model, an activation function is applied. In CNN usually, a Rectified Linear Unit (ReLU) is applied to the convolved feature to introduce non linearity in the model. The ReLU, G(x) = max (0, x) returns x for values x > 0, and will return 0 for x ≤ 0.
For FC- Fully connected layers, the output of previous layer is flattened to a 1-D vector and then softmax activation function is applied to determine the probabilities of each class

$$Softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

The Pooling Layer:
The pooling is used to reduce size of the images basically to save on processing time by preserving the most critical features information but reducing the dimension of the feature matrix. Depth of the convolution layers will remain unchanged because pooling is applied on each layer. In most of the case Max pooling is chosen. In the below image it can be observed that two strides are applied at a time. The size will decrease as strides increases.



Fully Connected Layers:
After CNN, are FS's or fully connected layers. There can be one or more fully connected layers. 2 layers will be fully connected when all the nodes from the first layer will be connected to the every nodes of the second layer. In FC's classification is performed on the features outputted by the previous convolutions layers. Usually there will be a softmax activation function in the final fully connected layer. Softmax is to give an output probability of 0 to 1 for the labels above model is predicting.
In the below image [6], the CNN consists of 2 convolution layers, comprising of convolution + ReLU + max pooling which will extract the features, and at the end 2 FC layers for classification.



3. Implementation

In this experiment, Keras and tensorflow is used. First the experiment is done in basic way following the architectural principles of the VGG model with 3 Convolutional layer and then a VGG16 model, a basic Resnet-50 model, with weights pre-trained on ImageNet is applied on the dataset along with the SVM and XGB classification on the dataset.

Classification process in this experiment consists of the below steps:

Data loading — where images and labels are read after loading of the dataset for both train and test.

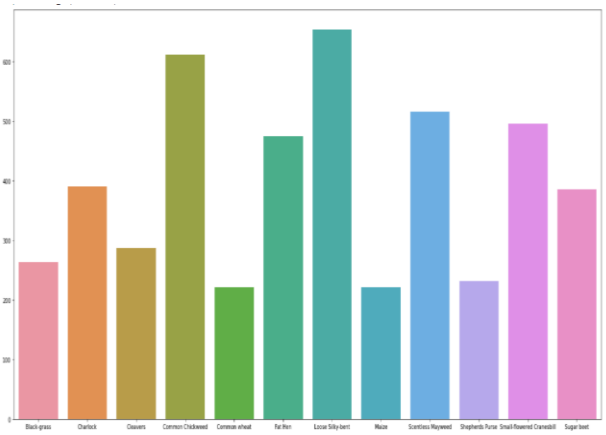
Data cleaning — where various image preprocessing is done like removing background, input normalization and label preparation
Model building — splitting data into training and validation sets, creation and fitting of created models
Model evaluation — evaluation of model, and prediction is made.

Data loading —

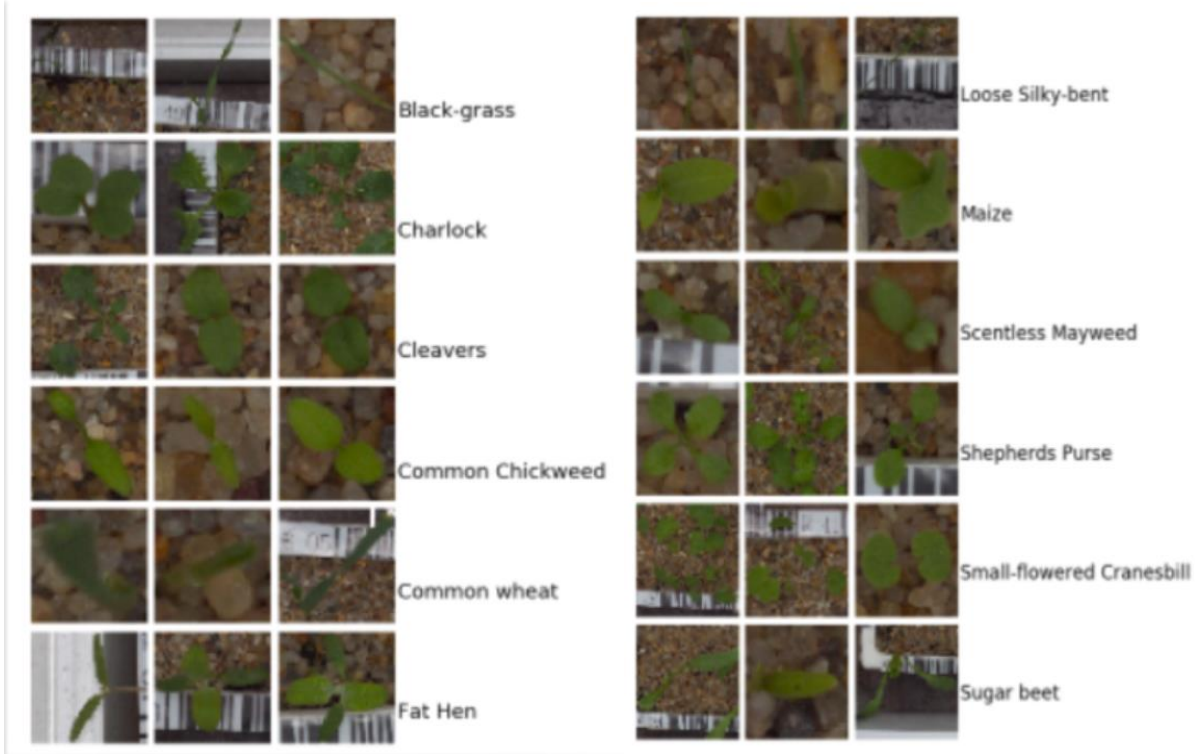
The Plant Seedling dataset was formed by Thomas Mosgaard Giselsson, Rasmus Nyholm Jørgensen, Peter Kryger Jensen, Mads Dyrmann and Henrik Skov Midtiby, at the Aarhus University Signal Processing group, in collaboration with University of Southern Denmark [2] which was made available of Kaggle in 2018.
The training image set consists of 4750 labelled examples and the test set consists of 750 unlabeled testing images, which have its labels retained for evaluation on submission. The training set images consists of 12 seedling species. The figure below shows the labels and the total number of each species, which range from 221 - 654.

```
Species wise images in the dataset:

654 images for  Loose Silky-bent category
231 images for  Shepherds Purse category
516 images for  Scentless Mayweed category
221 images for  Maize category
475 images for  Fat Hen category
221 images for  Common wheat category
611 images for  Common Chickweed category
385 images for  Sugar beet category
496 images for  Small-flowered Cranesbill category
390 images for  Charlock category
263 images for  Black-grass category
287 images for  Cleavers category
```



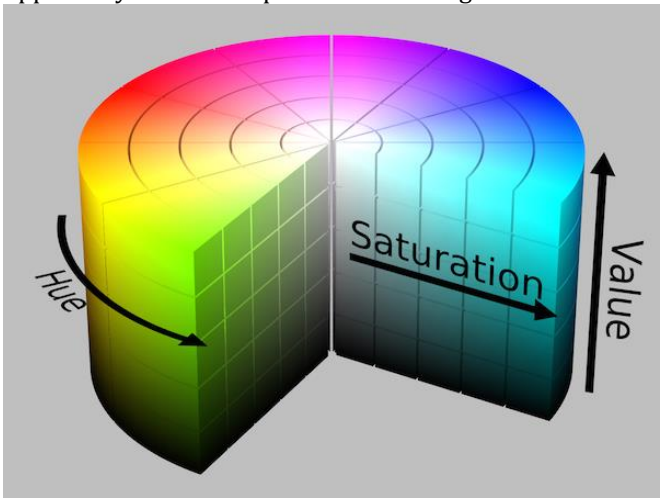
Seedlings and their number in the input



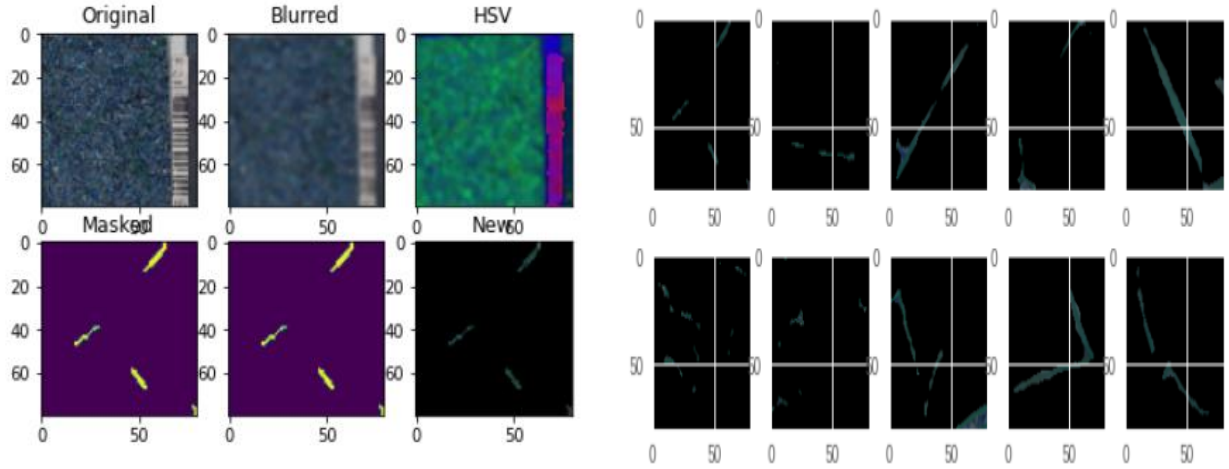
Few examples of the twelve kinds of seedlings in the input data

The loaded images are resized to 80 x 80 px (pixels). We use 80 x 80 px size so that models are trained quickly. Also image labels are loaded from folder name.

Data cleaning —
In the above photo of plant seedling we can see that there is background, so we tried to remove it which helps us to achieve better model accuracy. For background removal, we use the knowledge that all plants in our input are green. A mask will be created which leaves only some range of green color in the image while removing other parts. For masking of green plant for background removal, our RGB image is converted to HSV. In HSV, it's apparently easier to represent color range than in RGB and hence, HSV is used RGB alternative.

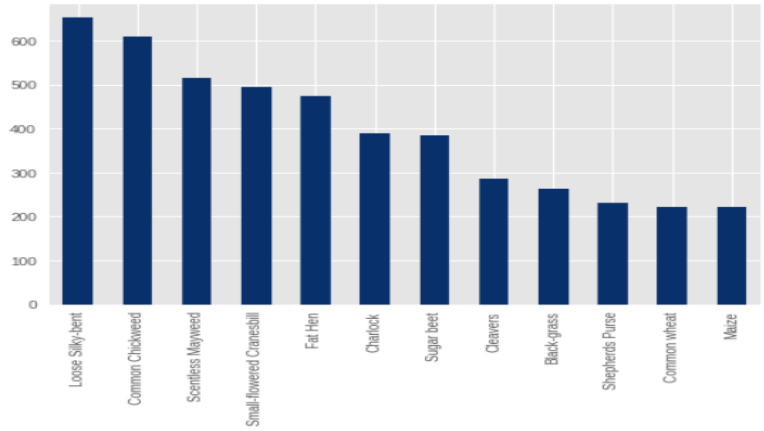


For removing noise we will first blur the image then convert the image color to HSV. Then a mask will be created on the HSV green color images and then it will be converted to Boolean mask and applied to the original image. Below figure shows the process and the output of HSV conversion and masking of the images.



Now that the background is removed, the images are normalized from RGB color space [0…255] to [0…1] which will be useful to train CNN faster.

Now labels are categorized by encoding image labels. These are 12 string names, which will create classes array and encode every label by their position in the array. For example 'Black Grass' -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].



Model building →

Data Splitting ->
First the input dataset is split for training and validation set. 10% of data is appointed as the validation set
Since, our data is unbalanced as seen in the above image, to avoid inaccurate evaluation of model we will set stratify= clearall. Stratify is used so that the proportion of values in the sample which will be produced by splitting will be in the same proportion as of the value provided to the parameter. See below code snippet:

```
from sklearn.model_selection import train_test_split

#splitting the dataset into training and testing for validation
x_train,x_test,y_train,y_test = train_test_split(new_train_data,clearall,test_size=0.1,random_state=seed,stratify=clearall)
```

Data generator ->

To prevent over fitting of data, image generator is used which randomly rotate, zoom, shift or flip images during model fitting. We can set rotation between 0 to 180 degrees, horizontal flips, vertical flips etc. See below code snippet for the same.

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
        rotation_range=180,  # randomly rotate images in the range
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally
        height_shift_range=0.1,  # randomly shift images vertically
        horizontal_flip=True,  # randomly flip images horizontally
        vertical_flip=True  # randomly flip images vertically
    )
datagen.fit(x_train)
```

Model creation ->

3 different type of models are created for this experiment.

a) Basic Keras Sequential multi layer architecture->

This model is created with 6 convolutional layers and three FC layers in the end. The 1st two Conv2D layers have filters set to 64, next 2 layers have 128 filters & the last 2 layers having 256 filters. A max pooling layer is added after every pair of Conv2D layer.

Preventing Overfitting ->
In machine learning models, one of the major concern is convolutional neural network when trained is overfitting.

2techniques to prevent overfitting in CNN are:

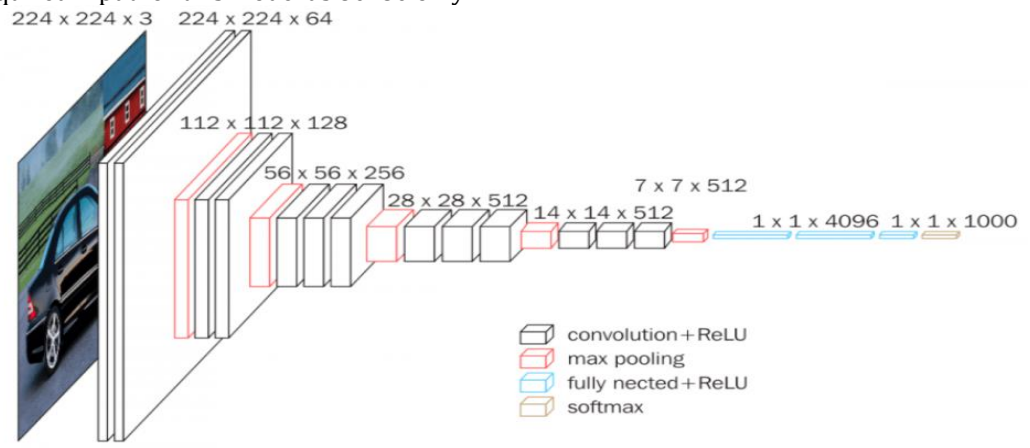
Data augmentation: Is useful when training dataset is small. It is basically, boosting the number of train examples and diversity of the dataset, by doing some transformations on the original images to create new variants.

Dropout regularization: removing random units the model while training gradient step - 10% between convolutional layers and 50% between fully connect layers

Then batch normalization is done to stabilize the learning and accelerate the learning process. Between each layer we use batch normalization layer.

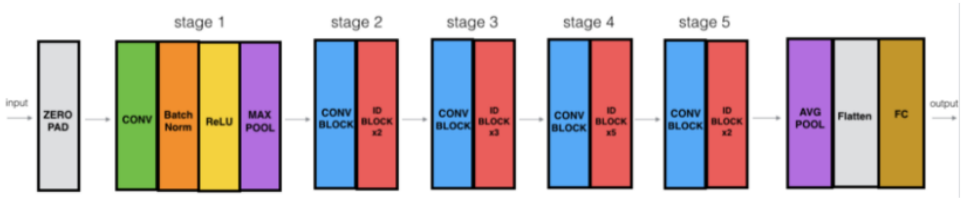
In the end, 3 fully-connected layers are used for classifying. In last layer the dense network will output distribution of probability for 12 classes.

b) VGG16 architecture->
Then Using VGG16 model[9], with weights pre-trained on ImageNet is used.
VGG-16 was introduced by K. Simonyan and A. Zisserman [10]. It is made up of 13 Conv2D and 3 FC layers. Input images for vgg16 are of size 224x224 but since, we have resized our data to 80x80, we will enter the required input for this model as 80x80 only.



```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 80, 80, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 80, 80, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 40, 40, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 40, 40, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 40, 40, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 20, 20, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 20, 20, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 20, 20, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 20, 20, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 10, 10, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 10, 10, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 10, 10, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 10, 10, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 5, 5, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 5, 5, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 5, 5, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 5, 5, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
flatten_9 (Flatten)          (None, 2048)              0
_________________________________________________________________
hidden1 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_27 (Dropout)         (None, 512)               0
_________________________________________________________________
hidden2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_28 (Dropout)         (None, 256)               0
_________________________________________________________________
predictions (Dense)          (None, 12)                3084
=================================================================
Total params: 15,898,188
Trainable params: 15,898,188
Non-trainable params: 0
_________________________________________________________________
```

c) Resnet-50 architecture ->
In resnet50 architecture we will use only the layers from 143 to 179 and freezing all others as we don't need all the 179 layers as our data is already preprocessed.



ResNet-50 Model

Below is the code snippet to show the resnet50 architectures and layers added.

```
import tensorflow.keras as K

input_t = K.Input(shape=(80,80,3))
resnet50_base_model = ResNet50(weights='imagenet',
                    include_top=False,
                    input_tensor=input_t# input: 80x80 images with 3 channels -> (80, 80, 3) tensors.
                   )

for layer in resnet50_base_model.layers[:142]:
    layer.trainable = False
```

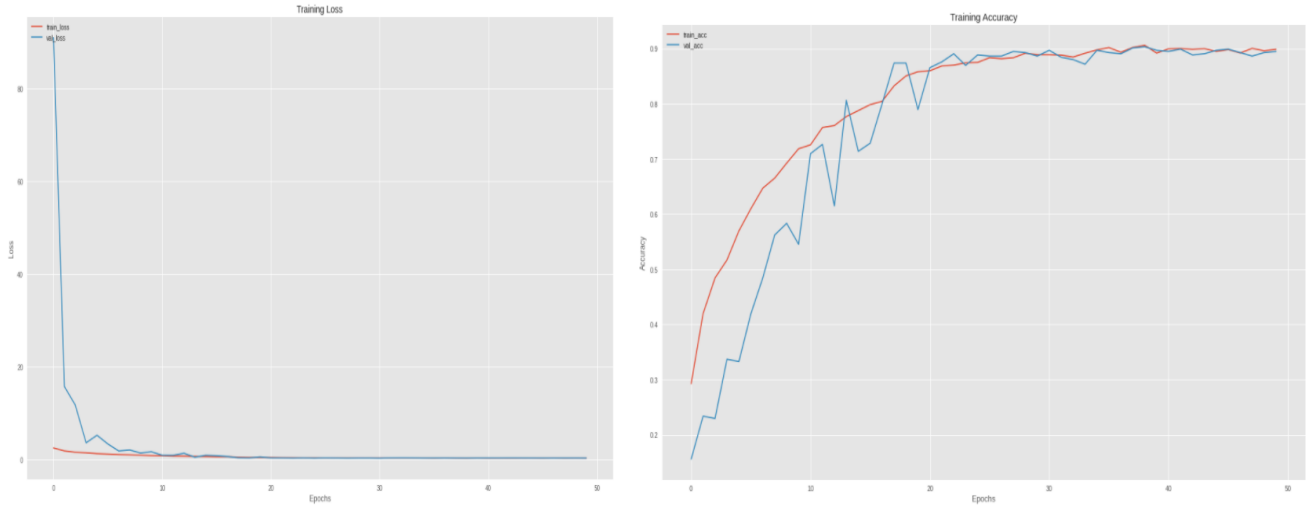
```
to_res = (80, 80)
resnet50_model = K.models.Sequential()
resnet50_model.add(K.layers.Lambda(lambda image: tf.image.resize(image, to_res)))
resnet50_model.add(resnet50_base_model)
resnet50_model.add(K.layers.Flatten())
resnet50_model.add(K.layers.BatchNormalization())
resnet50_model.add(K.layers.Dense(256, activation='relu'))
resnet50_model.add(K.layers.Dropout(0.5))
resnet50_model.add(K.layers.BatchNormalization())
resnet50_model.add(K.layers.Dense(128, activation='relu'))
resnet50_model.add(K.layers.Dropout(0.5))
resnet50_model.add(K.layers.BatchNormalization())
resnet50_model.add(K.layers.Dense(64, activation='relu'))
resnet50_model.add(K.layers.Dropout(0.5))
resnet50_model.add(K.layers.BatchNormalization())
resnet50_model.add(K.layers.Dense(12, activation='softmax'))
```

Model fitting ->
We train our model by first setting several callbacks. 1st callback is to reduce the LR – learning rate of the model. With high LR model convergence is quick, although the model could fall into a local minimum. To avoid this LR is decreased at the process of fitting by reducing it, if the accuracy is not improved after five epochs. Other two callbacks save best and last weights of model.

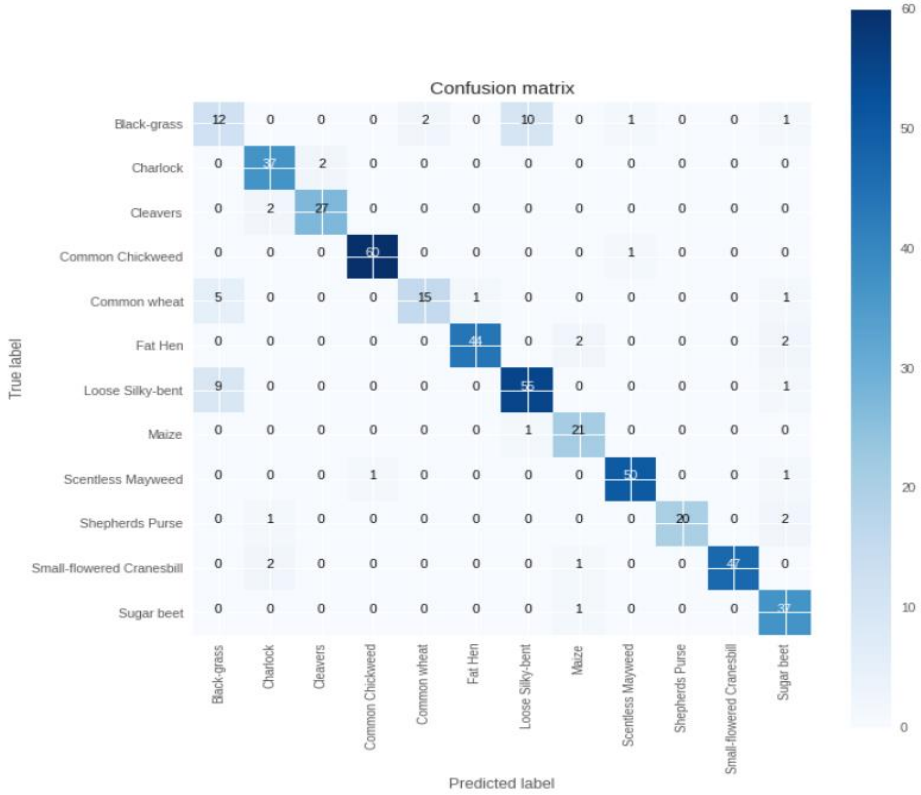
Model Evaluation →

Model is evaluated by looking at the Confusion matrix which helps to look at model errors. Also, we can see the categorical cross entropy loss function which keeps a tab on the classification accuracy as well.

a) Basic Multi-layer CNN architecture:
Below images shows the cross entropy loss of training and validation set, and accuracy of the training and validation set for the model.
Below that we can see the confusion/correlation matrix of the predictions of the model on the input data.



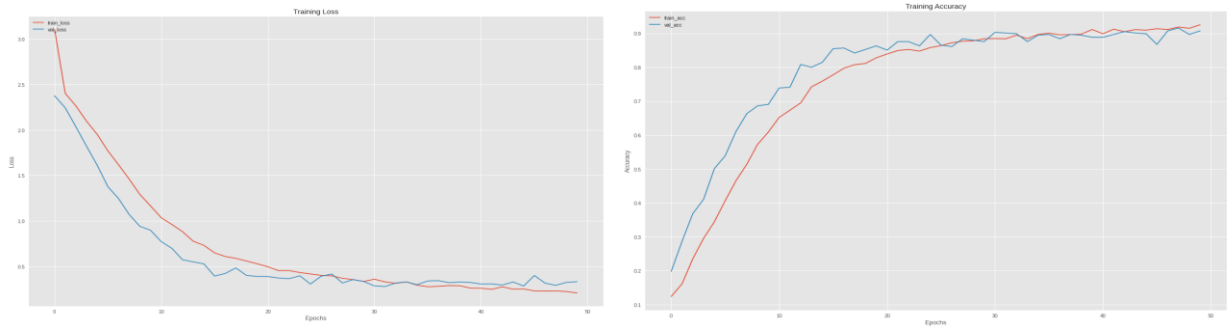
We got an accuracy of **90.526**% on the training set and **89.474**% on the testing set for 50 epochs and 0.00001 LR and a batch size of 64.



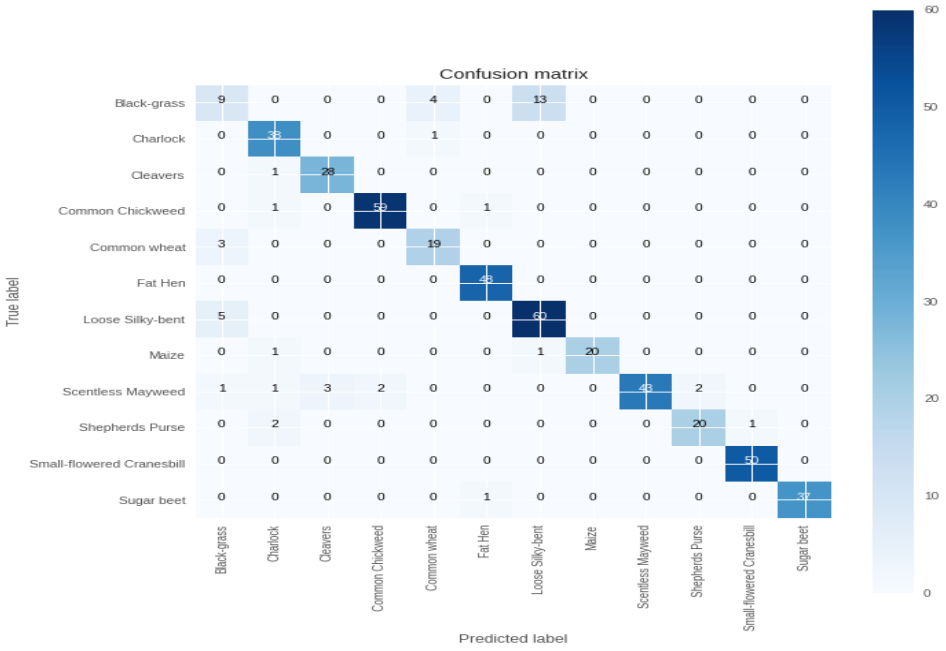
VGG16:

The model is defined as sequential model and VGG layers are added to the model. For 50 epochs and a batch size of 64, an accuracy of **91.438**% was achieved on training set and **90.736**% on the test set.

Below shows the cross entropy loss and the accuracy of the VGG16 model,



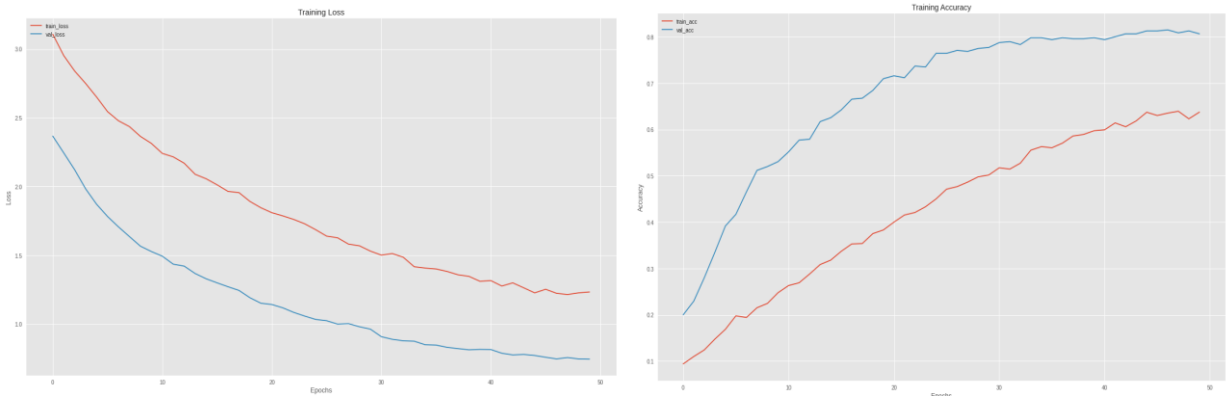
Confusion matrxi of the VGG16 model,



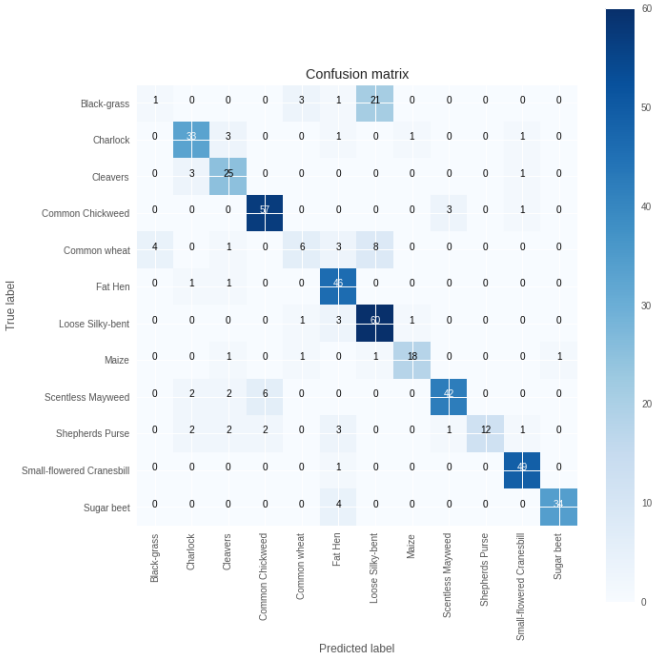
Resnet50:

The model is defined as sequential model and Resnet50 layers are added to the model. For 50 epochs and a batch size of 64, an accuracy of **82.882**% was achieved on training set and **80.632**% on the test set.

Below shows the cross entropy loss and the accuracy of the resnet50 model,



Confusion matrix of resnet50:



In this experiment, using the best VGG16 model we have applied SVM and XGB.
SVM gave an accuracy of 93.895% on train and 93.052% on validation set over VGG16 model.
XGB gave an accuracy of 99.228% on train and 92.632% on validation set over VGG16 model.

5. Future Work

The future work for this experiment will be to try and implement the model with pytorch, applying different architectures like AlexNet, InceptionV3, Xception, VGG19 and seeing the effects of different features of the models, learning rate, weights initialization and activation functions for much higher epochs and number of filters. Pixel scaling, such as centering and standardization, learning Rates effects needs to be analyzed.
Also, implementation of XGB, LGBM and SVM on top of CNN models as they have given much better accuracies on basic VGG16 model.

Analysis:

The accuracy from 6 Convolutional and 3 fully connected model is 90.526%.
The accuracy of VGG16 model on train set is 91.438%.
The accuracy of Resnet50 model on train set is 82.882%.

SVM gave an accuracy of 93.895% on train set when trained on VGG16 model.
XGB gave an accuracy of 99.228% on train set when trained on VGG16 model.

References

[1] https://www.kaggle.com/c/house-prices-advanced-regression-techniques

[2] Giselsson, T.M., Jørgensen, R.N., Jensen, P.K., Dyrmann, M. and Midtiby, H.S., 2017. A public image database for benchmark of plant seedling classification algorithms. arXiv preprint arXiv:1711.05458.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge.

[4] Giselsson, T.M., Jørgensen, R.N., Jensen, P.K., Dyrmann, M. and Midtiby, H.S., 2017. A public image database for benchmark of plant seedling classification algorithms. arXiv preprint arXiv:1711.05458.

[5] https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

[6] https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

https://harrymoreno.com/2018/10/15/gradient-boosting-decisions-trees-xgboost-vs-lightgbm.html

[9] https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16

[10] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.