Machine Learning Capstone Project Report

Plant Seedling Classification

https://www.kaggle.com/c/plant-seedlings-classification

1. Definition

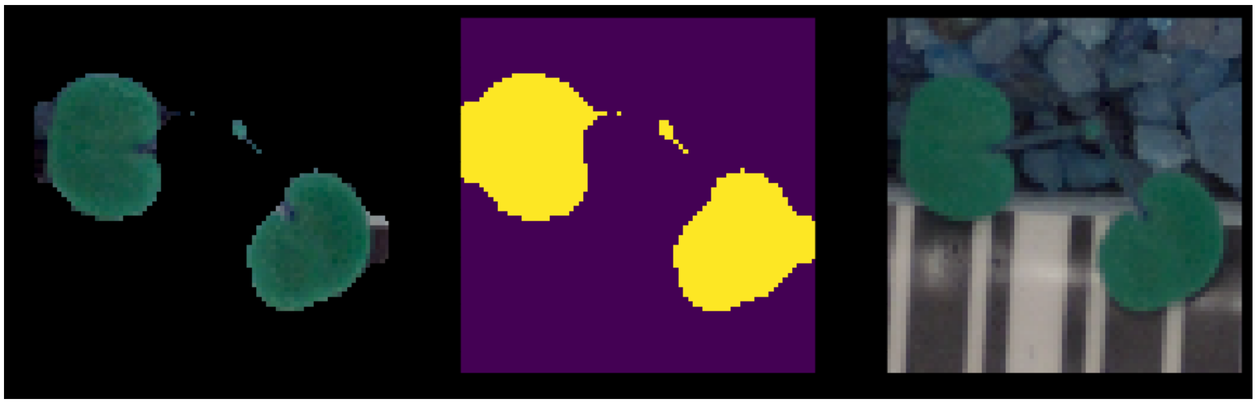
A. Project Overview

My Capstone project for the MLND is the Plant Seedling classification project obtained from the Kaggle Competition website. Classifying plants is extremely important in today's world. It helps people -especially botanists and farmers- to organize certain species of plants into their categories. It can be used by scientists studying plants in a forest, by environmentalists analyzing the local environment to determine what types of plants grow in specific areas, and maybe by explorers exploring forests to identify certain species of plants. The project statement on the Kaggle website states that "The ability to do so effectively can mean better crop yields and better stewardship of the environment." The objective of this project is to classify given images into twelve different categories. The categories are for the plants in this competition are given as follows: Black grass, Charlock, Cleavers, Common Chickweed, Common Wheat, Fat Hen, Loose Silky-Bent, Maize, Scentless Mayweed, Shepherds Purse, Small-Flowered Cransebill, and Sugarbeet. For this project, the images from the training set were used as input data and the names of the species corresponding to the images as labels. The final objective is to predict the name of the species (out of the 12) of a new image from the test set which has 794 images.

B. Problem Statement

The problem statement of the Plant Seedling classification is to build a model that can accurately classify images into given categories of twelve species. Since inputs are images, it is best to use computer vision tactics using OpenCV for image pre-processing. The model itself should be a Convolutional neural network since these networks are specifically designed to work well on images. Lastly, the test images went through the same image pre-processing to eliminate any possible errors an biases in the test set.

1. Image Pre-Processing includes:
    a. Resizing (The images were resized to 75 by 75)
    b. Color adjustment (HSV, gray_scale)
    c. Masking to remove background
    d. Segmenting - the images maintain their original color while the background is eliminated.
    e. Image Sharpening to eliminate blurs.



2. The CNN model includes:
    a. The model is built using the **keras** deep learning library.
    b. 32, 64, and 128 filters and not any deeper because the images are not very large and deepening the model will only end up overfitting it.
    c. Kernel size of (3.3)
    d. 2-dimensional max pooling with a pool size of (2,2)

e. Batch Normalization to reduce the amount by what the hidden unit values shift around (covariance shift) so that the model learns faster with significant improvement.

f. Dropout layers to avoid overfitting and avoid complex adaptions on training data.

The result of training should be that the model will be able to accurately predict the species (out of the 12) of a plant form the testing set. According to Kaggle, the species of the corresponding images of plants should be recorded in an excel document and be submitted for evaluation.

C. **Metrics**

The model was evaluated for its performance by Kaggle and me. The evaluation metrics that I chose to measure the performance of the model are Validation accuracy, Validation loss, Cross-Entropy loss, and general accuracy. I chose these particular evaluation metrics because they are global metrics and are easy to understand. The project itself revolves around how accurately the model can classify the images.The validation loss and validation accuracy are for the validation sets (“practice test sets”), and these are good representations of loss and accuracy when performed on the actual test set. The Kaggle platform uses the f1 score to measure the model's performance which is an average of precision and recall.

1) Formulas for the evaluation metrics

   a. Accuracy = (TP + TN) / (TP + TN + FP + FN)

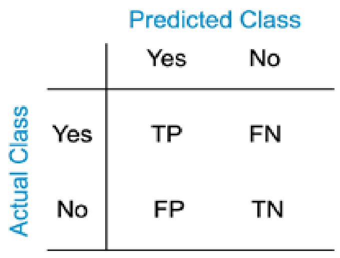
   b. Categorical Cross Entropy Loss =

      $H_{y'}(y) := -\sum_i (y'_i \log(y_i) + (1-y'_i)\log(1-y_i))$

c. Precision = TP / (TP+FP)

d. Recall = TP / (TP+FN)

e. Confusion Matrix = →

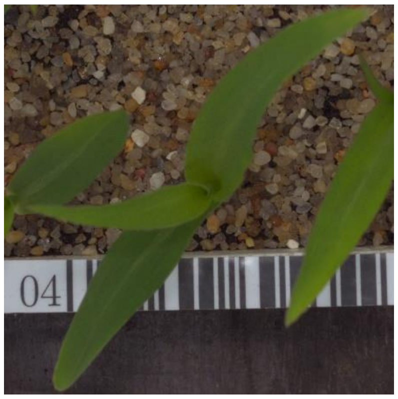


		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

2. Analysis

A. Data Exploration

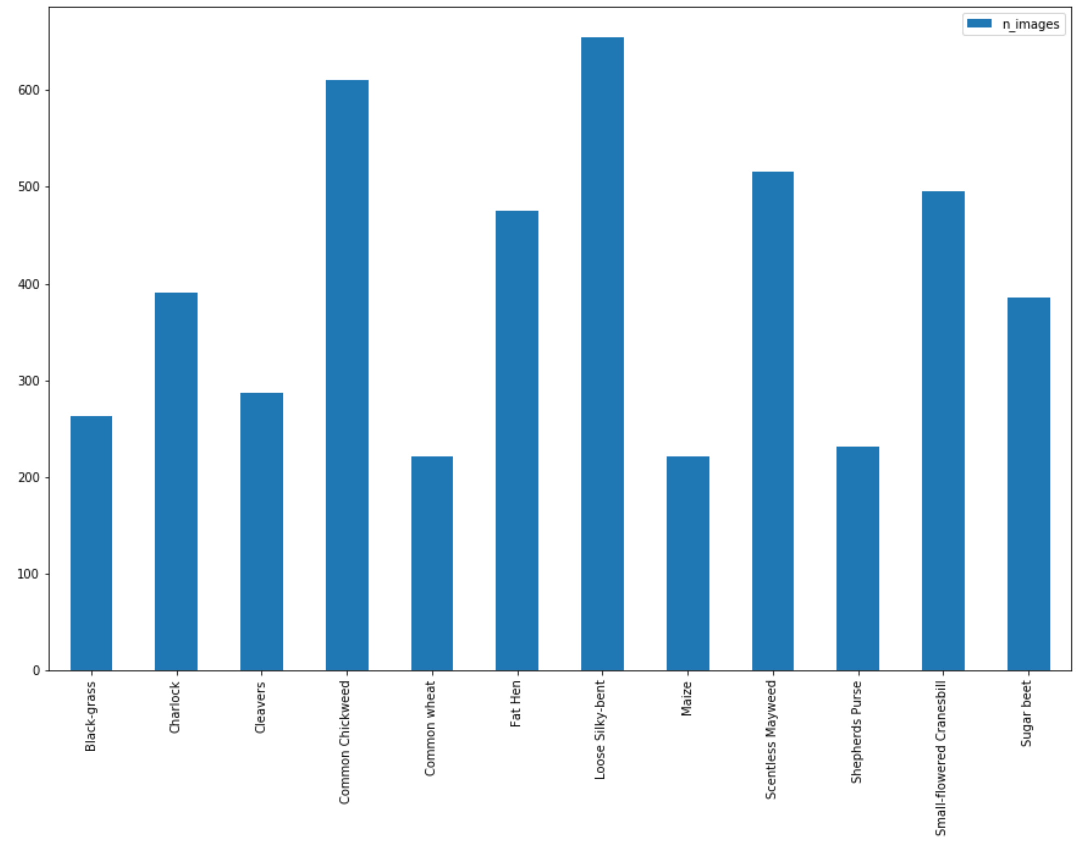
The training dataset (available at https://www.kaggle.com/c/plant-seedlings-classification/data) entirely consisting of 4750 images came in 12 separate folders that serve as labels to the pictures. The folder names are the names of species and inside the folders are the corresponding images. All photos are quadratic but vary in size, and so the images need to be resized to the similar dimensions. The images are pictures of the plants along with various



external elements such as pebbles and in some instances, a bar code (not relevant to the project). Since not all images have the exact external features placed at the precise location, this was a factor that needed consideration when feeding images to the model. The best solution is to eliminate these features to avoid overfitting and a possible decrease in accuracy. Since the dataset only has 4750 images, the accuracy might be lower than usual because of the more the input, the better the accuracy.

The test set is one folder with random images and no labels. The results are to be submitted via an excel document with the image-id and its prediction.

Data should be used because it is one of the essential pieces of machine learning and data science because, without data, machine learning is impossible. For a model to perform a required task, it has to train, and the training happens on data. In this instance, the data consists of pictures and words. Both have to be converted to numbers that the model can interpret and then switched back into to desired output. Regarding Data Distributions, each category has a different number



of images.

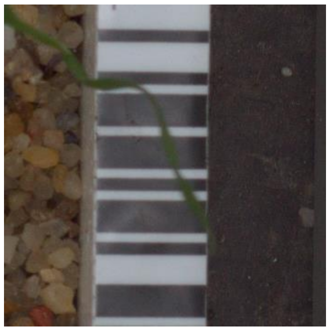
Category	Number of Images
1. Black-Grass	263 (5.5%)
2. Charlock	390 (8.2%)
3. Common Chickweed	611 (12.8%)
4. Common Wheat	221 (4.6%)

5. Fat Hen	475 (10%)
6. Loose Silky-bent	654 (13.7%)
7. Maize	221 (4.6%)
8. Scentless Mayweed	516 (10.8%)
9. Shepherds Purse	231 (4.8%)
10. Small-flowered Cranesbill	496 (10.4%)
11. Sugarbeet	385 (8.1%)
12. Cleavers	287 (6%)
TOTAL	4750 (100 %)

For this to be an even distribution, each category(species) should be about 8.3%. Alternatively, 394 images.

B. Exploratory Visualization

The images are a combination of random pictures taken in no given format or alignment. Some images are focused on the plant with the minimal background, but some have more background than the others.



The image on the left is a picture with a lot of background and bias while the picture on the right is mostly focused on the plant. Both pictures have been obtained from the Black-Grass category. Other characteristics include different image sizes. Here, the table shows the image sizes of images from the same category:

	Class	Height	Width
0050f38b3.png	Black-grass	196	196
0183fdf68.png	Black-grass	388	388
0260cffa8.png	Black-grass	886	886
05eedce4d.png	Black-grass	117	117
075d004bc.png	Black-grass	471	471
0050f38b3.png	Black-grass	196	196

C. Algorithms and Techniques

The Algorithms and techniques used in this project are:

a. Convolutional Neural Networks

- Convolutional Neural Networks are similar to neural networks that have been extended and have special shared weights. The networks are designed to detect and recognize images by edge detection and pixel calculations. The networks have convolutions inside that work with the pixels of the image.

b. Gradient Descent

- Gradient descent is an algorithm that is used to find the optimum weights by finding the minimum of the function by taking the derivative/partial derivative.

```
Repeat until convergence {
    $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
}
```

Image Source -

https://www.codeproject.com/KB/recipes/879043/GradientDescent.jpg

c. Back Propagation

- This an efficient method of performing gradient descent.This calculates derivatives using the chain rule and adjusts weights accordingly.

d. Adam Optimizer

- Adaptive Movement Estimation is the best and most commonly used gradient descent optimizers. This is used in the CNN for this project.

e. Dropout

- This feature randomly drops units and its connections during training. This prevents the network from co-adapting too much thus preventing the model from over-fitting.

f. Max- Pooling

- Max pooling reduces the image's dimensionality and allows for assumptions to be made about contained features. This is also done to help over-fitting.

g. Batch Normalization

- Batch Normalization is a regularization process that prevents the gradients from outliers and focuses them on what is relevant to a range of the mini-batches.

h. Learning Rate Decay/Reduction

- This process decreases the learning rate over time. This has two benefits of making substantial changes at the beginning of the training procedure when larger learning rate values are used and decreasing the learning rate such that a smaller rate and therefore smaller training updates are made to weights later in the training procedure.

i. Early Stopping

- The procedure/function stops the training if the loss does not decrease for a period. This can save time and allow the developer to make changes as soon as possible.

j. The CNN Architecure for this project is as follows:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 73, 73, 32)        896
_________________________________________________________________
batch_normalization_1 (Batch (None, 73, 73, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 71, 71, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 35, 35, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 35, 35, 32)        128
_________________________________________________________________
dropout_1 (Dropout)          (None, 35, 35, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 33, 33, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 33, 33, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 31, 31, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 15, 15, 64)        256
_________________________________________________________________
dropout_2 (Dropout)          (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 13, 13, 128)       73856
_________________________________________________________________
batch_normalization_5 (Batch (None, 13, 13, 128)       512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 11, 11, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 5, 5, 128)         512
_________________________________________________________________
dropout_3 (Dropout)          (None, 5, 5, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               409728
_________________________________________________________________
batch_normalization_7 (Batch (None, 128)               512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 12)                1548
```

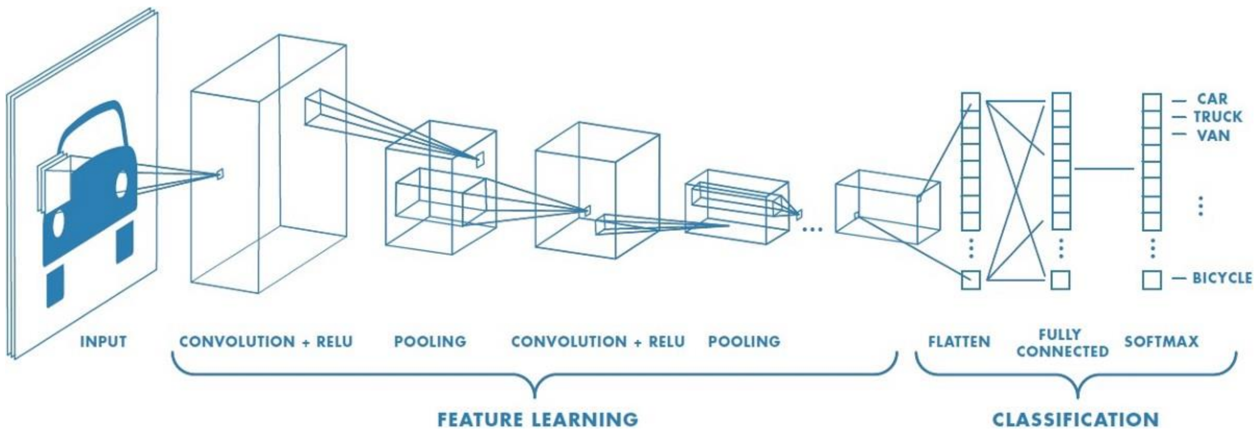


Image Source - https://cdn-images-1.medium.com/max/1200/1*XbuW8WuRrAY5pC4t-9DZAQ.jpeg

D. Benchmark

To create a benchmark model, I inputted the images to a very basic CNN that had no image augmentation, batch normalization, and dropout layers but rather a simple learning model. That model at that time did not have any callbacks such as learning rate decay or early stopping. The hidden layers, kernel sizes were also different. As I trained my model and obtained a benchmark accuracy, I added the missing parts. The resulting validation accuracy was 0.8109 and validation loss was 0.60863.

Name	Value
Validation Accuracy	0.8109
Validation Loss	0.6086
Training Accuracy	0.9134

Training Loss	0.3073
F1 Score	0.8745

3. Methodology

A. Data Pre- Processing

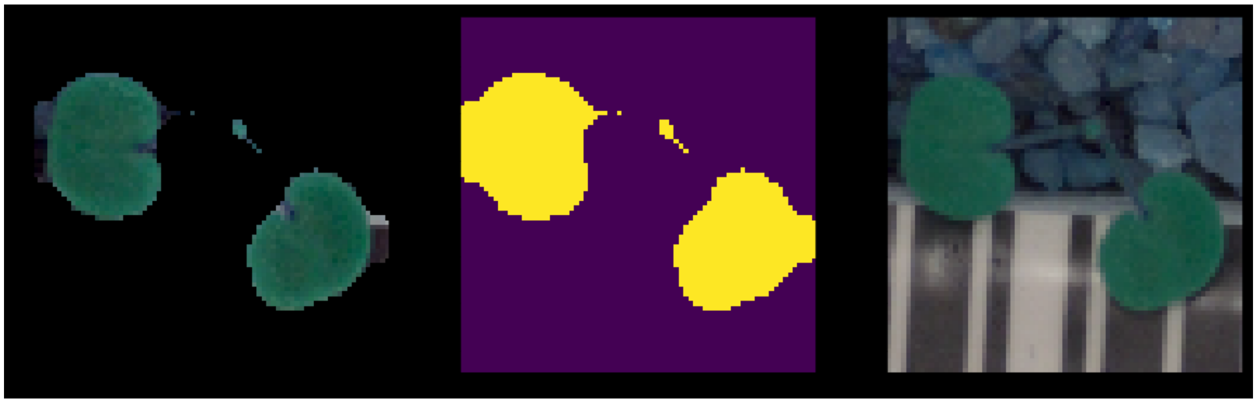
The input data (images) were different sizes, had different backgrounds and were different plants. The steps towards pre-processing were:

a. Resizing
   - Since all images were of different sizes, some very large and some very small, it is reasonable that I should resize all images to a decent size being 75 by 75 since larger images take an extended time to pre-process and train.

b. Masking
   - Masking the images eliminates the background and changes the color of the plant as well. A mask image is simply an image where some of the pixel intensity values are zero, and others are non-zero. Wherever the pixel intensity value is zero in the mask image, then the pixel intensity of the resulting masked image will is set to the background value (zero). However, masked images were not used in the training and testing process because I wanted to keep the original color of the plant.

c. Segmenting

- Segmenting images eliminates the background while keeping the original color of the plant.



d. Sharpening

- Sharpening the segmented images eliminates any blurred portions which result in improvement of the accuracy of the model.

e. Label Binarizing

- Binarizing the labels (classes) allows them to be used in the network as an input (labels). Since matching words and pixels values is not possible, binarizing transforms the words into vectors of 0’s and 1’s.

f. Image Augmentation

- Image augmentation performs geometrical transformations on the image, and each instance is used as an input to the model. Usually, image augmentation extends training time but increases the accuracy as well.

The image on the far left is the sharpened segmented image, the image in the middle is the masked version, and the image on the right is the original. All three images are resized to the equivalent dimensions.

```
[[0 0 0 ..., 1 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 ...,
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]
```

The vectors above are the binarized labels.

There were some coding complications that occurred during the implementation process of the algorithms and techniques. The major complication was image pre-processing. During the process, the image would show up as a blank gray screen because the images would go through segmentation three times instead of one even though there was no loop. I fixed this error by re-organizing the array and also saving the images to a folder to be reused instead of being segmented again. The next coding error occurred during the testing process. I could not bring the test images from the test folder to be tested. Even though I managed to bring the test images into jupyter, they were not in order with the labels so the predictions were always incorrect. I fixed the error by changing the indices of the test images so they would line up with the prediction labels.

B. Implementation

Implementation can be divided into three parts: Data Investigation and pre-processing; Model development; and Training.

DATA INVESTIGATION AND PRE-PROCESSING

1. The data was extracted from the folders and saved into arrays (tuples)

2. The data was investigated for known properties such as image size, characteristics, and frequency.
3. The images were then resized, masked, segmented, and sharpened.
4. The labels were binarized to be used in the network.

MODEL DEVELOPMENT

1. A data augmentation set was made for geometric transformations.
2. The CNN was built.
    a. A sequential Keras model
    b. 32, 64, and 128 filter hidden layers progressively.
    c. The layers were divided into three batches
    d. Each batch has 2 Conv2D, 2 Batch normalization, a max pool and a dropout layer.
    e. There is a flatten layer of the three batches of layers.
    f. After the flatten layer, there were three more batches of layers
    g. Each layer has a dense, a batch normalization and a dropout layer.
    h. The activation function for the layers is rectified linear unit and the activation function for the output layer is softmax.
    i. The decent gradient optimizer is the Adam optimizer, and the loss function is the categorical cross entropy.

MODEL TRAINING

1. For the training phase, callbacks such as model checkpoint, learning rate reduction, and early stooping were used.

2. Next, the image augmentation was used as part of the training process.

C. Refinement

In efforts to improve the model from the benchmark, I used a variety of parameters and processes. In the model itself, I added a dropout layer, batch normalization layer and the learning rate reduction callback. Next, I added image augmentation to improve the accuracy of the model. Regarding parameters, I had to test different filters and kernel sizes before reaching the optimal values. I noticed that a filter size of 256 was overfitting and I hence I concluded that 128 is the farthest I would go. I did not have a definite kernel size either, and the best value was obtained after testing a batch of images a few times. Lastly, the learning rate is something I took into consideration. I used the standard elbow method and prior knowledge to determine that 0.00001 is the ideal learning rate.

Diving into the specifics, the hyperparameters I was constantly changing were image size, filter size, kernel size and learning rate. I wanted the model to have the best accuracy and thus made the image size large. However, making the image size large adds to training time and maybe overfitting. So I tested the model with couple different image sizes and found out that the best size was between 70 and 100 and so I picked 75. In the case of filter size, I saw it similar to image size; larger is accurate but adds to training time and might overfit. The optimum filter size reasonable for this project is 128. The best kernel size for this project is (3,3). Learning rate is something that I had to pick very carefully. I picked a low value and used the learning rate reduction to avoid overshooting the minimum.

4. Results

A. Model Evaluation and Validation

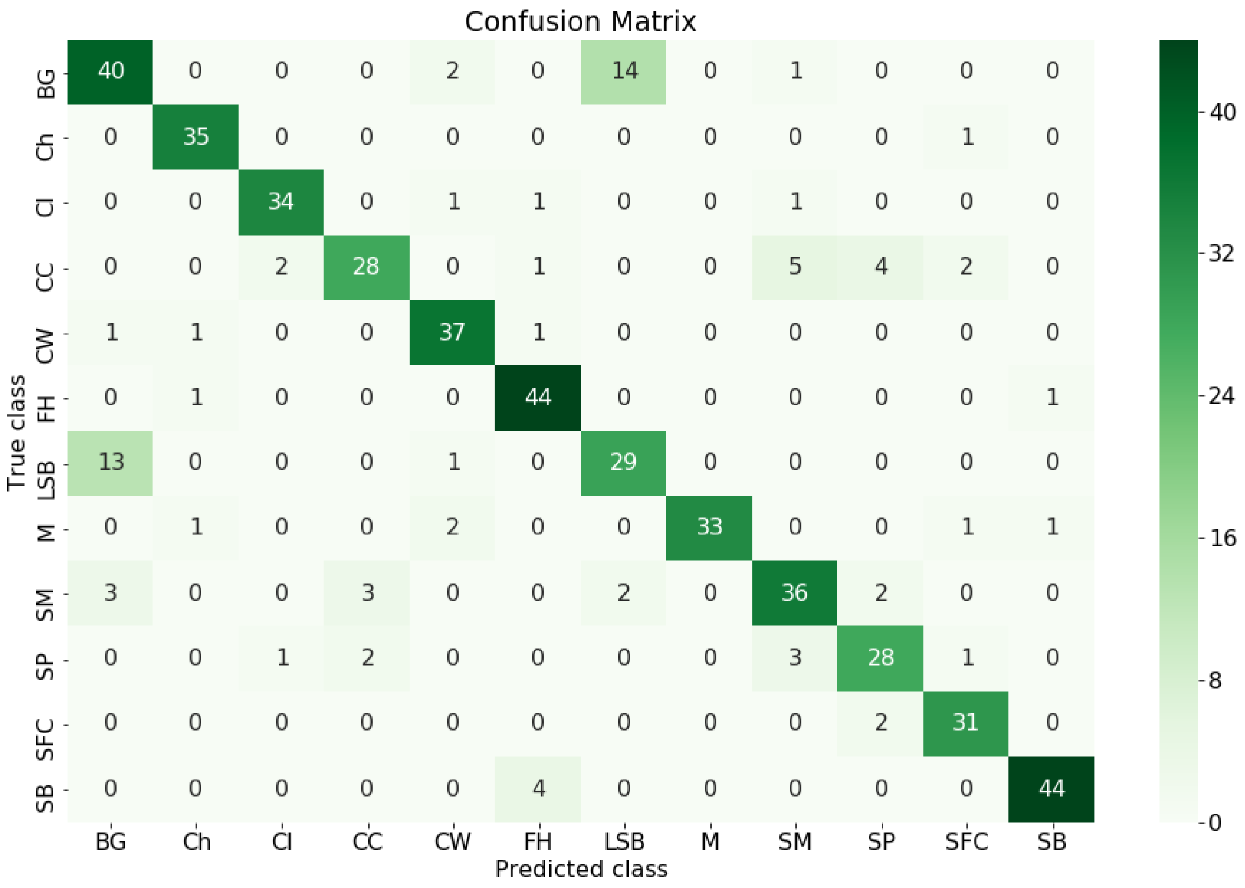
To evaluate the model, I used validation loss and accuracy and a training loss and accuracy. Additionally, Kaggle also used the f1 score to evaluate the model.

Name	Value
Validation Accuracy	0.9579
Validation Loss	0.2358
Training Accuracy	0.9968
Training Loss	0.0096

F1 Score	0.94625

The results the model produced very satisfying an excellent start to improve the model further. From here on, improving the model can be very challenging personally for me because I have already tested many cases and this stood out to be the best. I could have tested an ensemble learning machine but my GPU subscription ran out of runtime, and so I was limited. Given the complexity of the model and the straightforward outputs, I think that mode has performed well because 95 % validation accuracy is reasonably high accuracy. The model has been tested with unseen data from the test set and has produced an f1 score of 94%. Unfortunately, the labels of the test set were not provided by Kaggle for me to determine the test set accuracy. The model is robust enough for the current problem, and small changes in parameters do not affect the model's performance.

To test the robustness of the model, I used a confusion matrix that provided me with reasonable results.



This confusion matrix shows the true predictions and mis-classified predictions of all classes.

B. Justification

Benchmark:-

Name	Value
Validation Accuracy	0.8109
Validation Loss	0.6086

Training Accuracy	0.9134
Training Loss	0.3073
F1 Score	0.8745

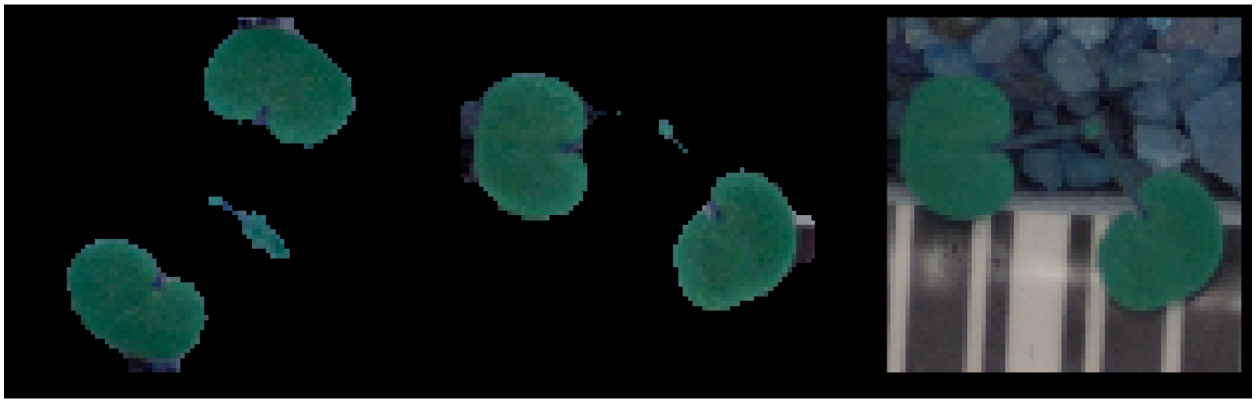
After Changes:-

Name	Value
Validation Accuracy	0.9579
Validation Loss	0.2358
Training Accuracy	0.9968
Training Loss	0.0096
F1 Score	0.9463

Improvements : -

Name	Improvements
Validation Accuracy	Increased by 15%
Validation Loss	Decreased by 61%
Training Accuracy	Increased by 9%
Training Loss	Decreased by 96%
F1 score	Increased by 8.3%

The improvements are significant enough declare that the model has improved by a lot and is a good enough model to solve the intended problem. Increasing and decreasing values by more than 5% at a peak is significant as well. Out of all the improvements, the losses have improved significantly because the room for error, bias, and overfitting was extremely low after the adjustments are parameters.



5. Conclusion

A. Free Form Visualization

The images above are from the test set and were inputted into the model for predictions . The model predicted that they are Small-flowered Cranesbill which they are.
However, there is nothing specific that happens. This project is simple regarding inputs and outp uts; they are simple images and labels. The test set goes through the same pre-processing as the tr ain set because the model did not learn the background. The model then successfully classifies th e plants.

B. Reflection

The projects requirement of classifying an image as one of 12 different classes seems like an easy task but came with many challenges. Image pre-processing took a while, primarily segmenting the images. The most time was spent in building the best model. The model used image augmentation, a 2-Dimensional Convolutional neural network which had layers including convolutional, max pool, dropout, batch normalization, flatten, and dense. The model also had callbacks such as learning rate reduction and early stopping to reduce overfitting and to save time. The images from the test set were also segmented before the prediction process.

C. Improvements

The model could be improved in many ways such as using an ensemble model or an Xception model that may produce better results.

However, there is a new type of networks called Capsule networks that perform a lot better than convolutional networks in image recognition and classification. This network was proposed by Geoffrey Hinton in October of 2017 and is relatively new. It uses Convolutions in capsules to output more accurate results. Unfortunately, my GPU service ran out of runtime an I could not change my model as desired since training would take an incredibly long time.

D. Learning Process

This project was an excellent opportunity for me to start and finish an actual machine learning process from scratch. Usually the projects in this nanodegree come with some pre-written code but that is not the case in this project. This project also forced me to learn and understand some of the concepts I missed during the lessons such as filters, kernels, dropouts and so on. The project also allowed me to make a design plan, brainstorm ideas, test multiple times and make constant changes. The most important aspect of this project was independency and implementation. It was tough to even start the project because I did not have clear goal but as I learned and recalled previous projects, things started to clear out. I had to use many online resources to clarify and doubts and questions I had. After this project, I plan on completing many other Kaggle projects or even enroll in the Deep Learning Nanodegree program. This course took me longer to complete than I initially planned because High School for me is very demanding with many after school activities, testing, sports, homeworks and projects. Time was something I did not have but as I grow not only as a machine learning enthusiast but also as a person, I am also developing a lot of time management skills. I am excited for my future in Machine learning.

Works Cited

http://www.xinapse.com/Manual/masking.html

https://towardsdatascience.com/the-10-deep-learning-methods-ai-practitioners-need-to-apply-885259f402c1

Geoffrey Hinton's Paper - https://arxiv.org/pdf/1710.09829.pdf

Capsule Network Intuition - https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b