An Investigation of Algorithms for Reliable and Explainable Weed Classification

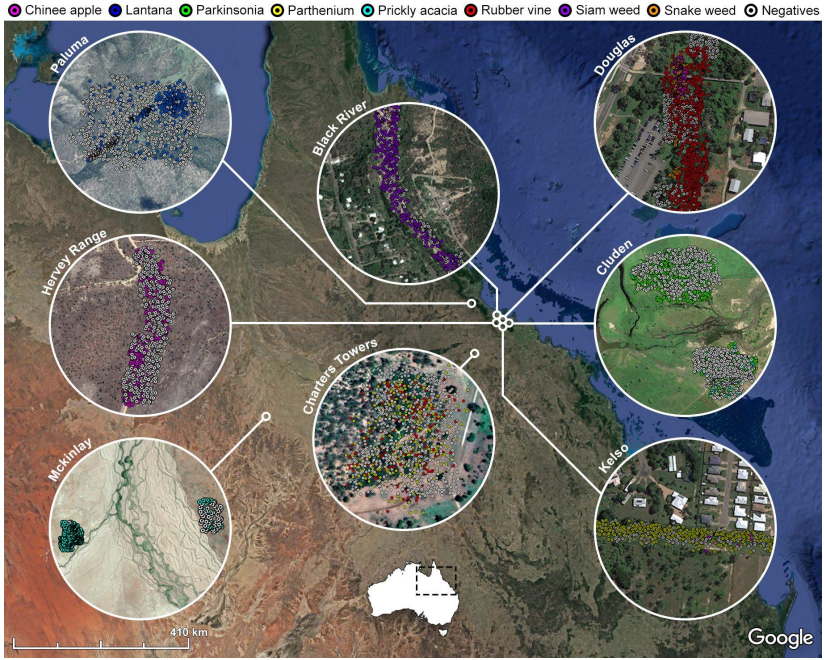
Members:
Aryan Sarswat
Lim Wei Liang
Yuichiro Fukushima
Lim Jin Feng
Jin Yixuan

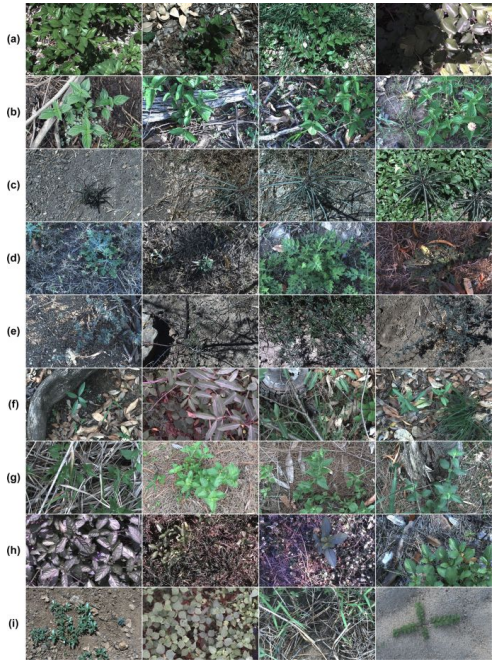
Group Project for CS3244: Machine Learning

Introduction

Dataset Description

The DeepWeeds dataset consists of 17,509 images capturing eight different weed species native to Australia in situ with neighbouring flora.



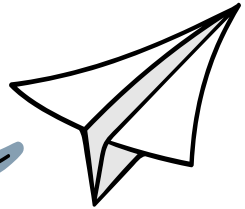


Motivation

Native Flora

Human agriculture

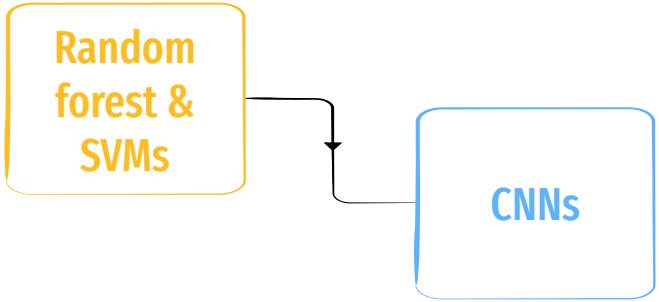
Biodiversity



Generalizable solutions

Control the spread & Agricultural productivity

Literature Survey



DeepWeeds: A Multiclass Weed Species Image Dataset for Deep Learning

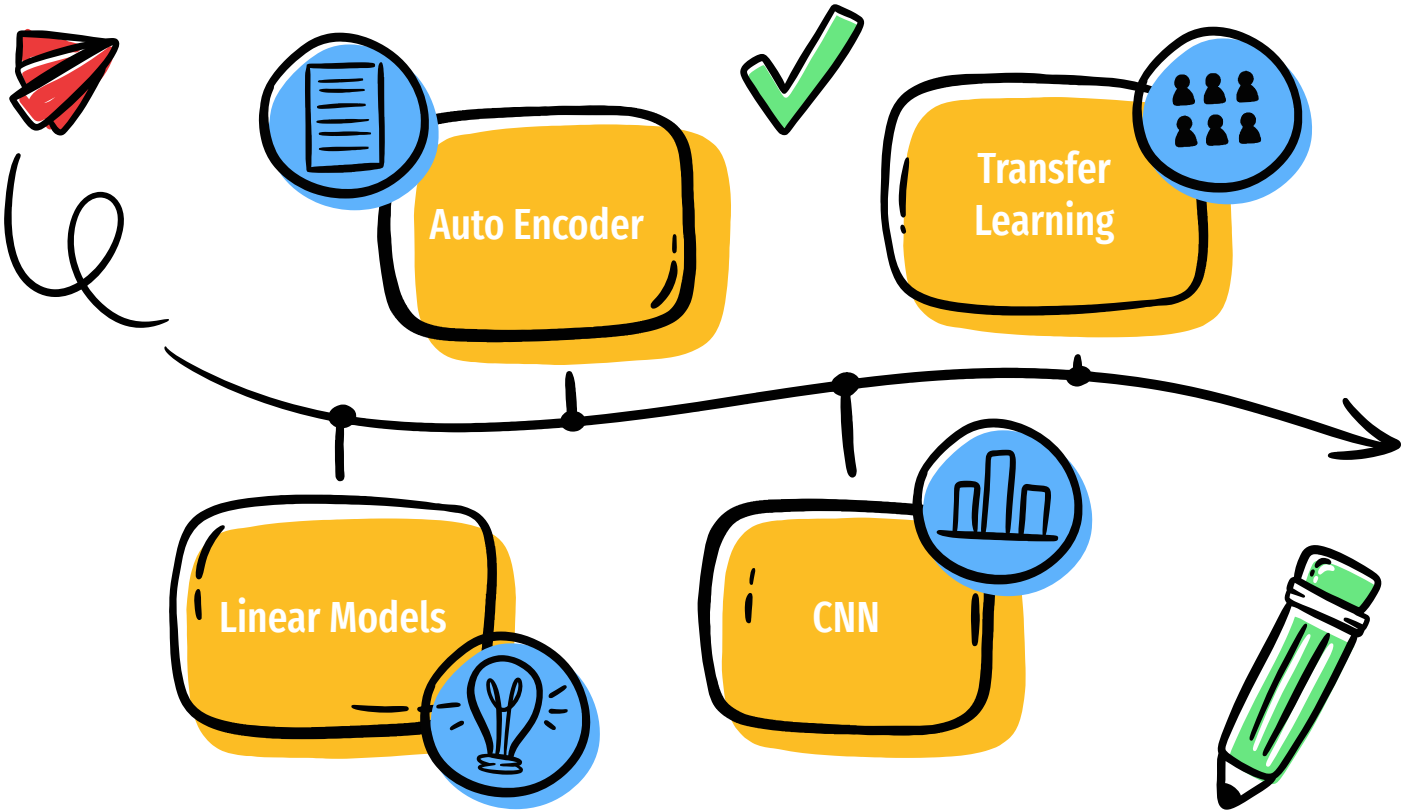
Alex Olsen, Dmitry A. Konovalov, Bronson Philippa, Peter Ridd, Jake C. Wood, Jamie Johns, Wesley Banks, Benjamin Girgenti, Owen Kenny, James Whinney, Brendan Calvert, Mostafa Rahimi Azghadi & Ronald D. White

A Survey of Deep Learning Techniques for Weed Detection from Images

A S M Mahmudul Hasan, Ferdous Sohel, Dean Diepeveen, Hamid Laga, Michael G.K. Jones

The rapid advances in Deep Learning (DL) techniques have enabled rapid detection, localisation, and recognition of objects from images or videos. DL techniques are now being used in many applications related to agriculture and farming. Automatic detection and classification of weeds can play an important role in weed management and so contribute to higher yields. Weed detection in crops from imagery is inherently a challenging problem because both weeds and crops have similar colours ('green-on-green'), and their shapes and texture can be very similar at the growth phase. Also, a crop in one setting can be considered a weed in another. In addition to their detection, the recognition of specific weed species is essential so that targeted controlling mechanisms (e.g. appropriate herbicides and correct doses) can be applied. In this paper, we review existing deep learning-based weed detection and classification techniques. We cover the detailed literature on four main procedures, i.e., data acquisition, dataset preparation, DL techniques employed for detection, location and classification of weeds in crops, and evaluation metrics approaches. We found that most studies applied supervised learning techniques, they achieved high classification accuracy by fine-tuning pre-trained models on any plant dataset, and past experiments have already achieved high accuracy when a large amount of labelled data is available.

Overview of Models Used



Data Preprocessing

Feature Engineering

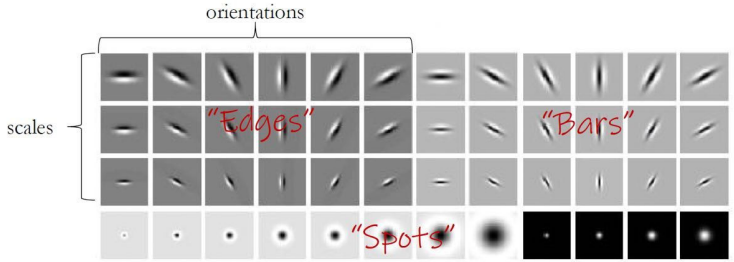
What features were extracted?

- Average RGB pixels values
- Average Intensity pixels values
- Average Average Laplace Means and Sobel Filter mean values
- Standard deviation of pixels values

However it is widely known these values alone will not work, which why is why we decided to try something new known as **Gabor filters** to create Filter banks.

$$f_{mn} = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{m^2+n^2}{2\sigma^2}\right] \sin\left[\frac{2\pi(\cos[\omega]m + \sin[\omega]n)}{\lambda} + \phi\right]$$

[Equation Defining the Filter]



Each of these filters try to detect different properties of an image, be it an edge, bar or spot. This will hopefully allows us to differentiate between classes

However due to the extremely large nature of the dataset we have had to take the mean values of the image after applying the filters. This has the drawback of losing information regarding the relationship between pixels.

Train-Val-Test Split

- Before training any models, split the dataset to prevent **data leakage**
  - 70% Train
  - 20% Validation
  - 10% Test
- Validation was used to tune hyperparameters and conduct model experiments
- Test was only used for the final reporting on model performance

Dealing with Class Imbalances - Simple Approaches

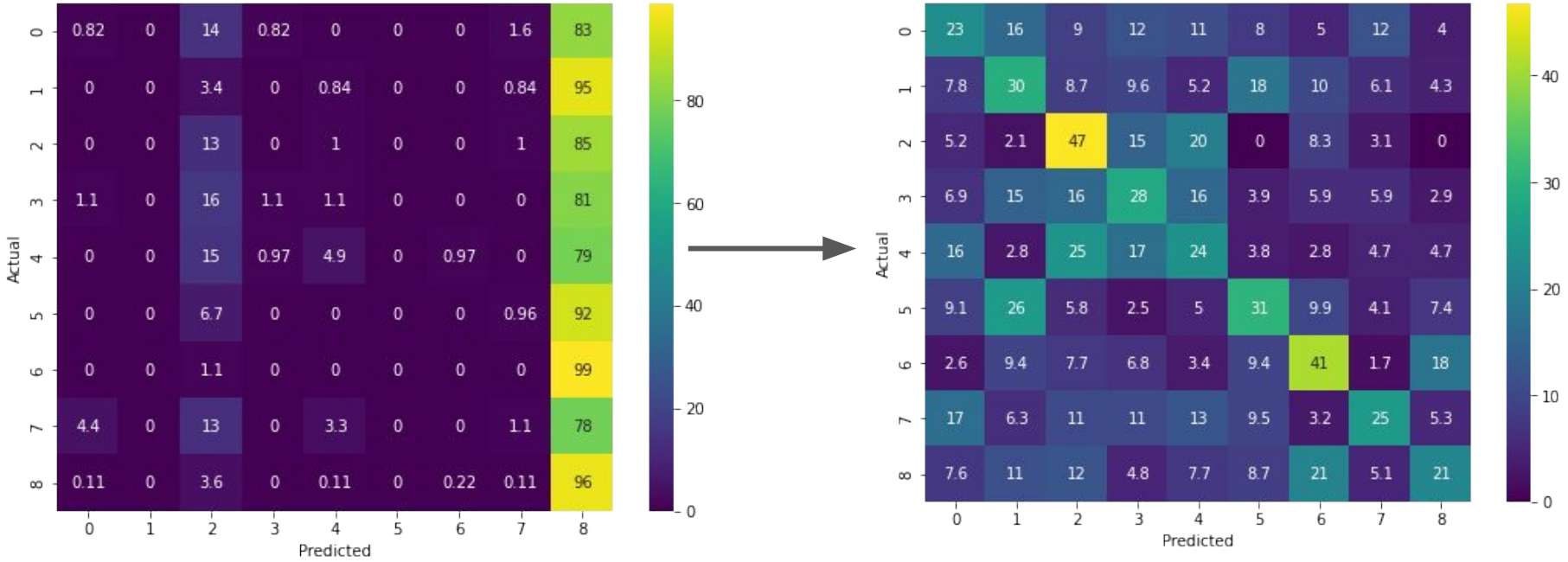
The Problem (Severe Class Imbalance):

~9000 examples in the negative class, but only **~1000** examples for each of the 8 weed types → poor model performance on minority classes

Approaches Tested

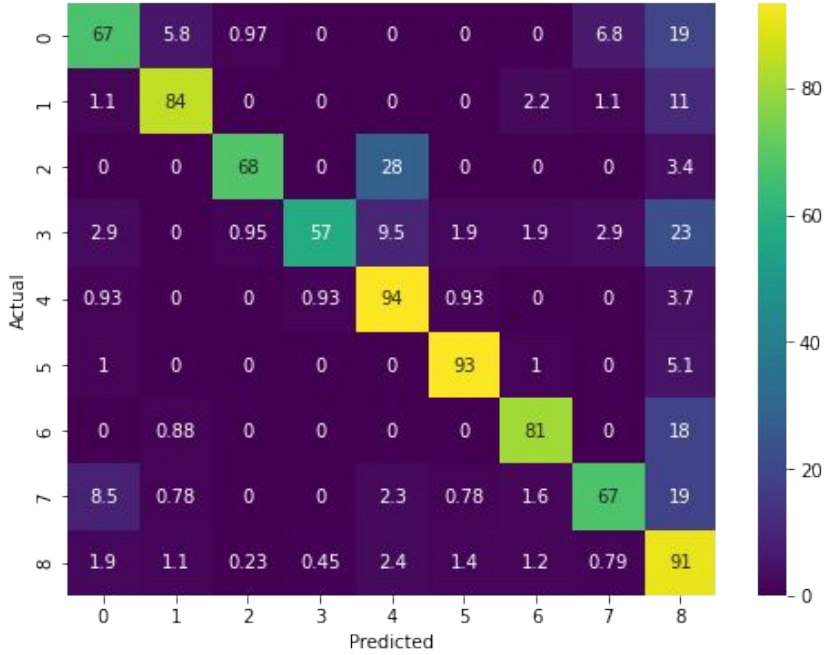
1. **Overweighting Minority Classes** - during the weight update step of gradient descent, introduce a multiplier = #class/#majority so that the model pays more attention to the minority classes
2. **Random Undersampling**

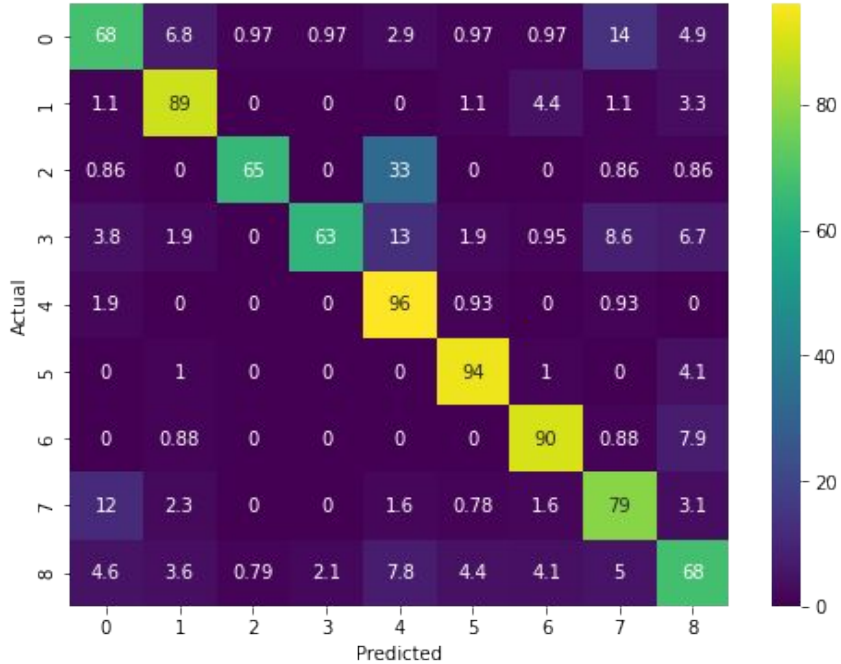
Effect of Undersampling on a Linear Model (kNN)



Significantly improves minority class recall and reduces overprediction of the majority class label

Effect of Overweighting Minority Classes on a CNN Model

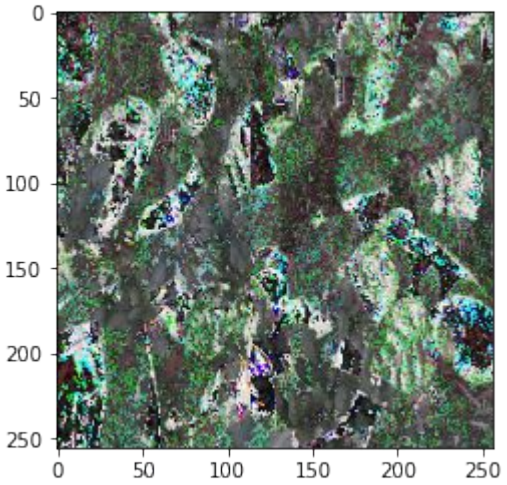




- Smaller improvement in recall and decline in precision → less useful for CNNs vs linear models
- Chose not to use this for our CNN models because we want to avoid automatic pruning of non-weed species by mistake

Dealing with Class Imbalances - SMOTE

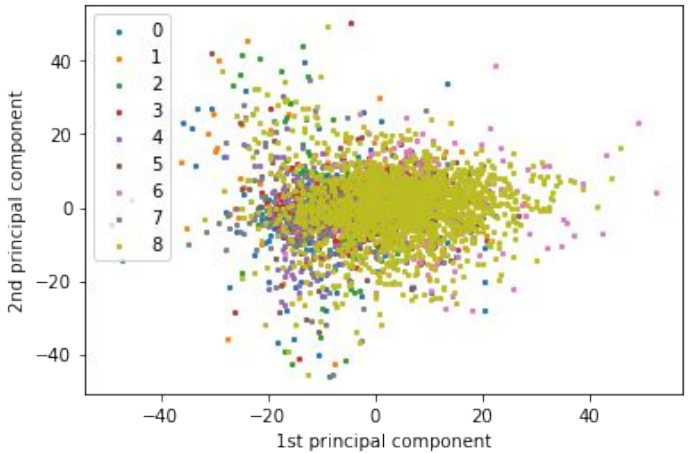
- Simple approaches did not address the underlying issue of limited minority class image data → **generate synthetic data?**
- **SMOTE (Synthetic Minority Over-sampling Technique)**
  - Performed experiment with subset of 500 images
  - Generated synthetic data for minority classes by interpolating between 5 nearest neighbours (with the same label) of a data point
  - Trained CNN model on oversampled dataset
  - **Outcome:**
    - Generated images were unrealistic, with a lot of noise
    - No improvement in model performance on all metrics



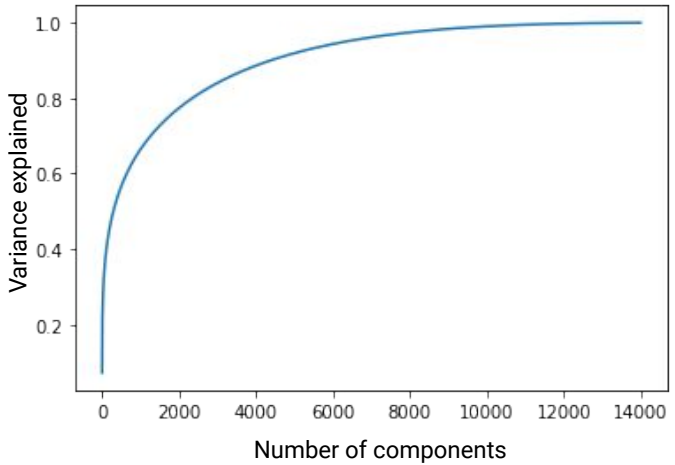
PCA

Tried to reduce dimension and extract useful features for classification model using PCA

Attempt to cluster weeds by 1st and 2nd principle component



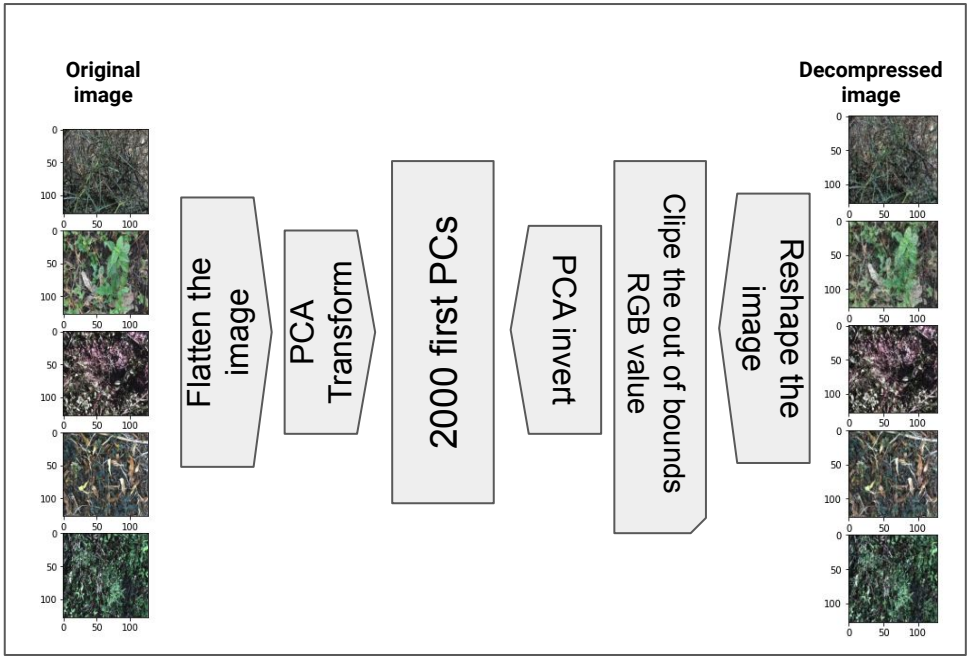
How well PCA captures the variance of the data



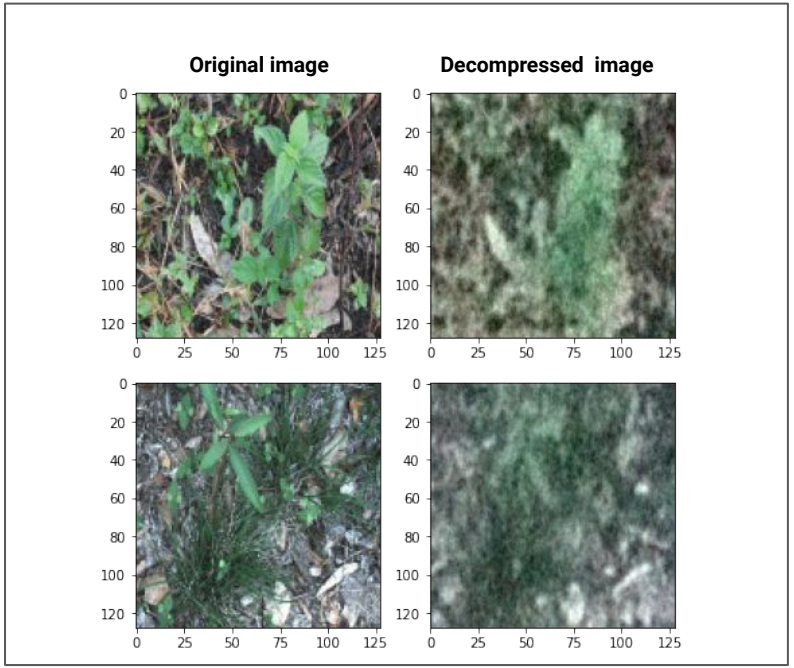
New Image compression with PCA

- PCA does a good job of compressing and decompressing an weed image

Normal compression/decompression architecture



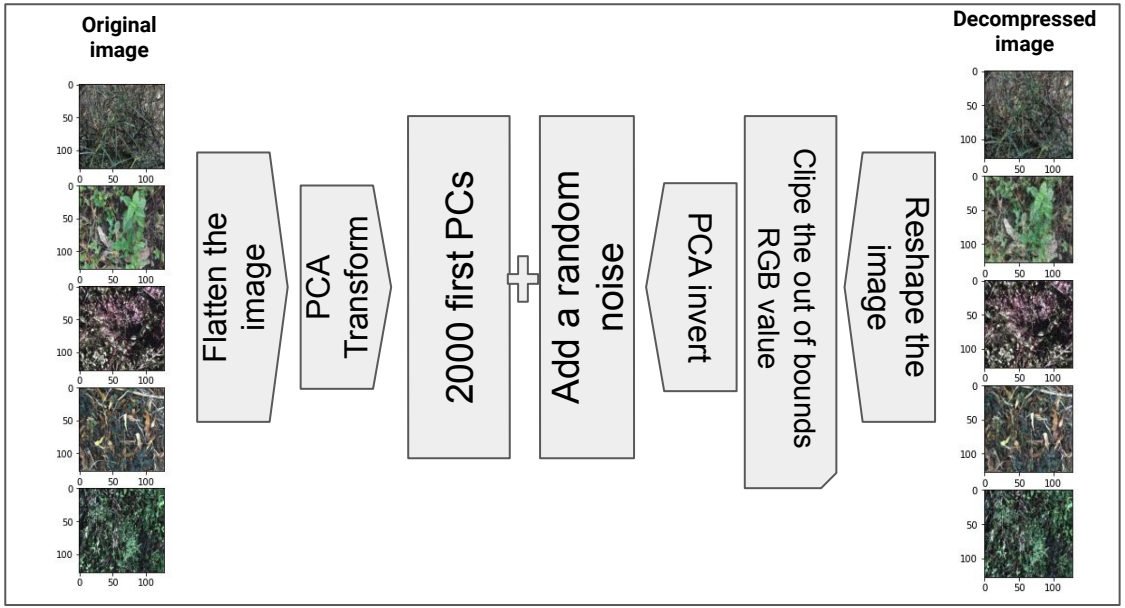
Detailed result



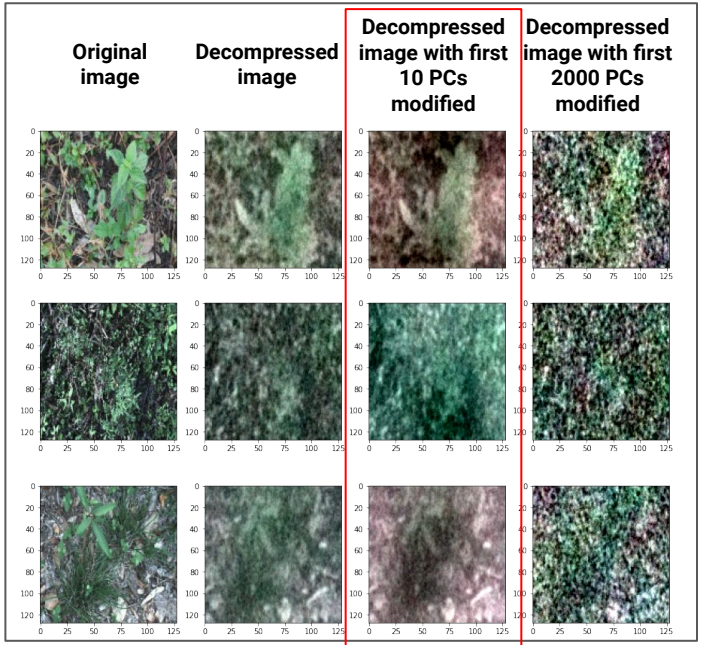
New Image generation with PCA

- Now adding a noise to the extracted components might generate a new image
- Result :
  - Adding small noise to the entire 2000 component did not work well -> just a noisy image
  - Adding larger noise to only the first 10 PCs produced a change in color with small noises

Compression/decompression with noise added architecture



Result



Models

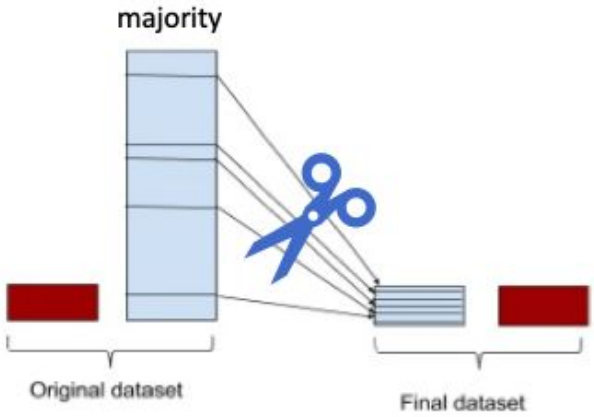
Linear Models

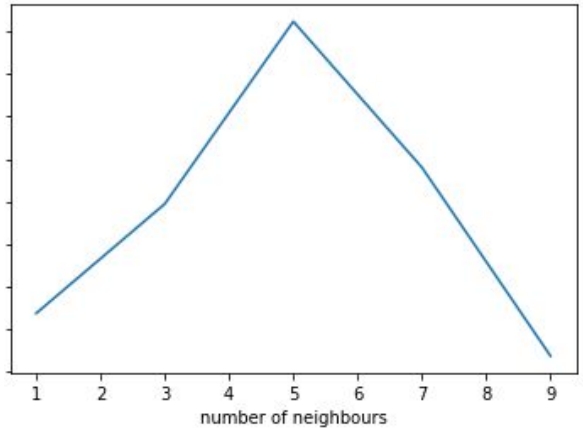
1. Use LDA on pixel values directly and train them
2. Engineer features from those pixel values then use LDA then train them

Linear Models

1. Use LDA on pixel values directly and train them
2. Engineer features from those pixel values then use LDA then train them

Usage of undersampling to mitigate imbalanced dataset and validation set to tune hyperparameters





Linear Models

1. **Use LDA on pixel values directly and train them**
2. **Engineer features from those pixel values then use LDA then train them**

Scores for model

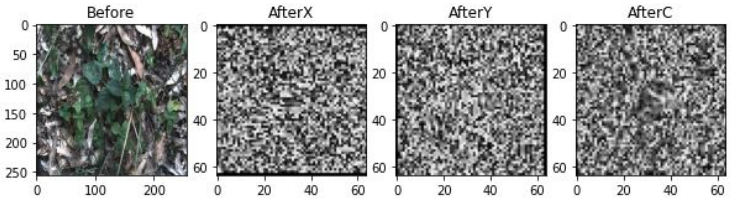
Model		f1-score	precision	recall
SVM	Weighted	0.24	0.43	0.20
	Macro	0.16	0.17	0.20
Random forest	Weighted	0.24	0.44	0.21
	Macro	0.17	0.18	0.20
KNN	Weighted	0.23	0.42	0.20
	Macro	0.15	0.17	0.19
Logistic regression	Weighted	0.26	0.43	0.23
	Macro	0.18	0.19	0.22

Model		f1-score	precision	recall
Dummy that randomly classifies	Weighted	0.13	0.32	0.11
	Macro	0.09	0.12	0.11

Linear Models

1. Use LDA on pixel values directly and train them
2. **Engineer features from those pixel values then use LDA then train them**

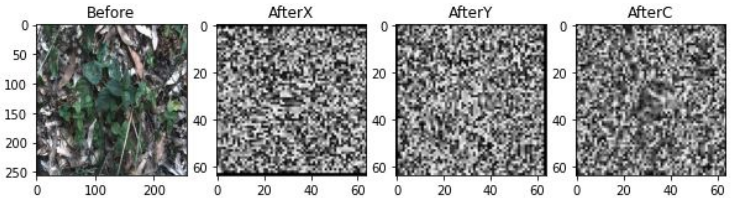
1. Edge detection Kernel



Linear Models

1. Use LDA on pixel values directly and train them
2. **Engineer features from those pixel values then use LDA then train them**

1. Edge detection Kernel



2. Other features

RGB mean, average intensity, laplace mean, edge mean, standard deviation, gabor filters

Linear Models

1. Use LDA on pixel values directly and train them
2. **Engineer features from those pixel values then use LDA then train them**

1. Edge detection Kernel

Scores for model

Model		f1-score	precision	recall
SVM	Weighted	0.15	0.33	0.13
	Macro	0.10	0.13	0.13
Random forest	Weighted	0.16	0.34	0.13
	Macro	0.11	0.13	0.14
KNN	Weighted	0.14	0.36	0.12
	Macro	0.10	0.13	0.12
Logistic regression	Weighted	0.15	0.33	0.13
	Macro	0.11	0.13	0.15

Model		f1-score	precision	recall
Dummy that randomly classifies	Weighted	0.13	0.32	0.11
	Macro	0.09	0.12	0.11

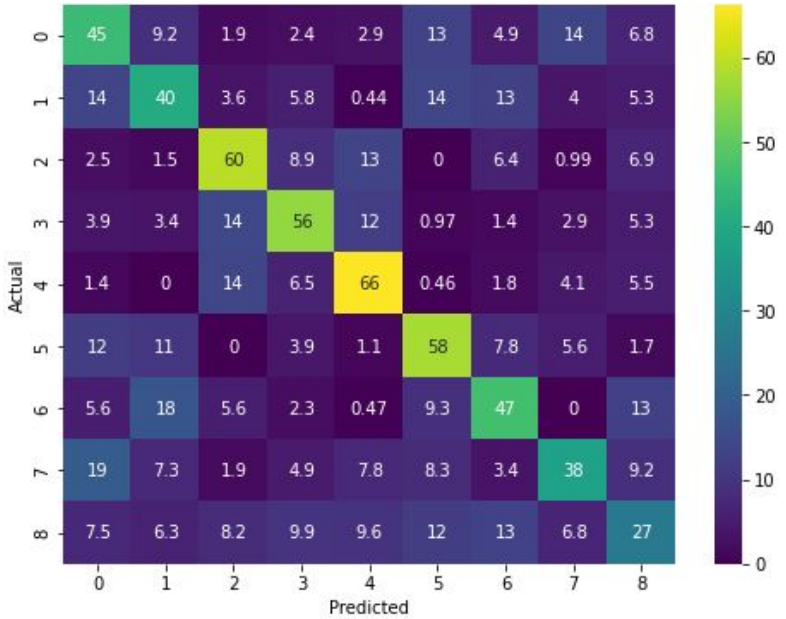
Linear Models

1. Use LDA on pixel values directly and train them
2. **Engineer features from those pixel values then use LDA then train them**

2. Other miscellaneous features

Scores for model

Model		f1-score	precision	recall
SVM	Weighted	0.35	0.56	0.36
	Macro	0.36	0.36	0.47
Random forest	Weighted	0.38	0.56	0.37
	Macro	0.36	0.34	0.46
KNN	Weighted	0.30	0.56	0.30
	Macro	0.31	0.31	0.41
Logistic regression	Weighted	0.29	0.47	0.28
	Macro	0.26	0.26	0.33



Possible reasons why it did not work well

- Features engineered were not good enough
    - Edge detection actually detected a lot of the edges from the background, not much meaningful information gained
    - Other engineered features did raise the performance of the linear models, but still far from being optimal

Possible reasons why it did not work well

- Features engineered were not good enough
    - Edge detection actually detected a lot of the edges from the background, not much meaningful information gained
    - Other engineered features did raise the performance of the linear models, but still far from being optimal
- Maybe linear classifiers are not powerful enough for this task?

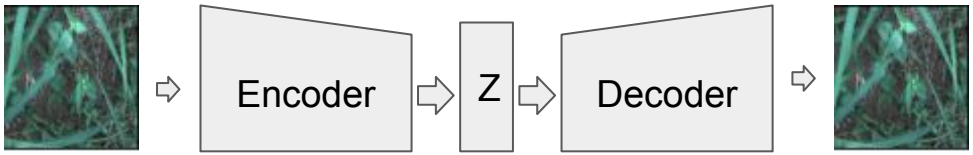
Possible reasons why it did not work well

- Features engineered were not good enough
    - Edge detection actually detected a lot of the edges from the background, not much meaningful information gained
    - Other engineered features did raise the performance of the linear models, but still far from being optimal
- Maybe linear classifiers are not powerful enough for this task?
- Maybe we should learn the features automatically?

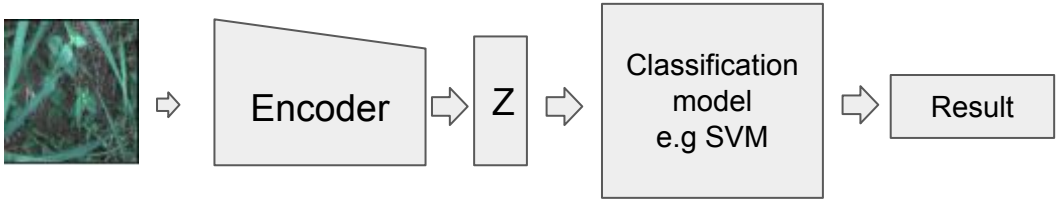
Auto Encoder

IDEA: Use auto encoder as a feature extractor and improve linear model performance

1. Train Autoencoder to reproduce the exact same image

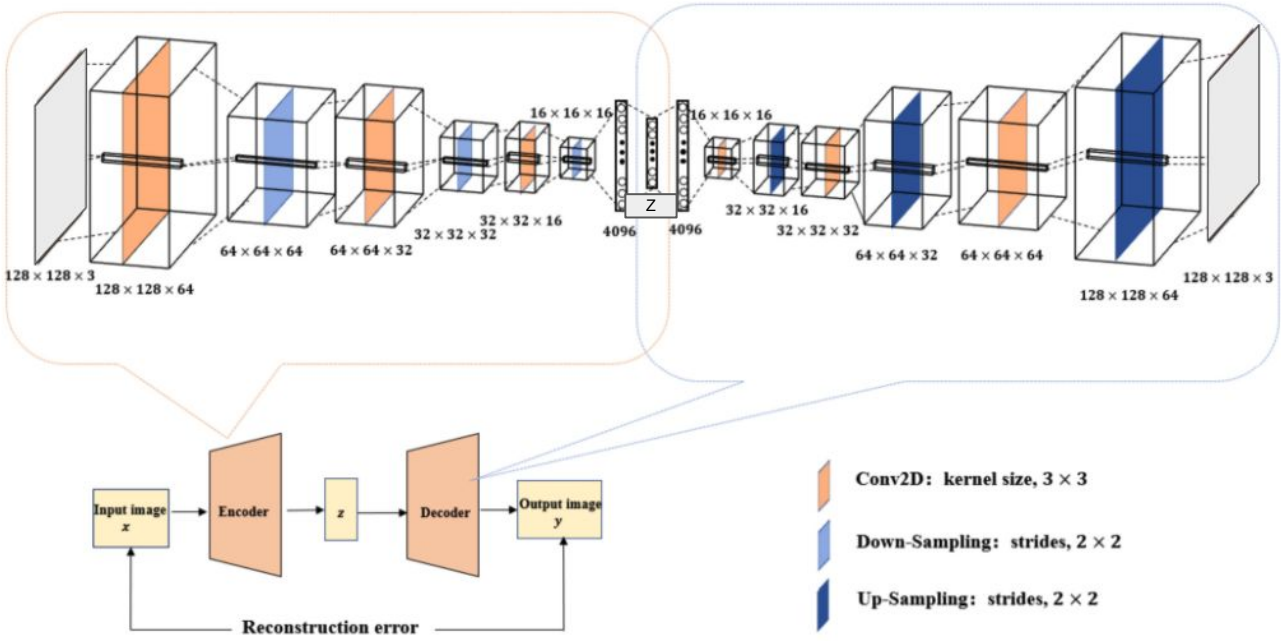


2. Use trained AE as feature extractor and use new latent vector z for classification model



Autoencoder Architecture

The architecture was based on a research paper done by researchers from North China electric power university where they performed well on a similar task.



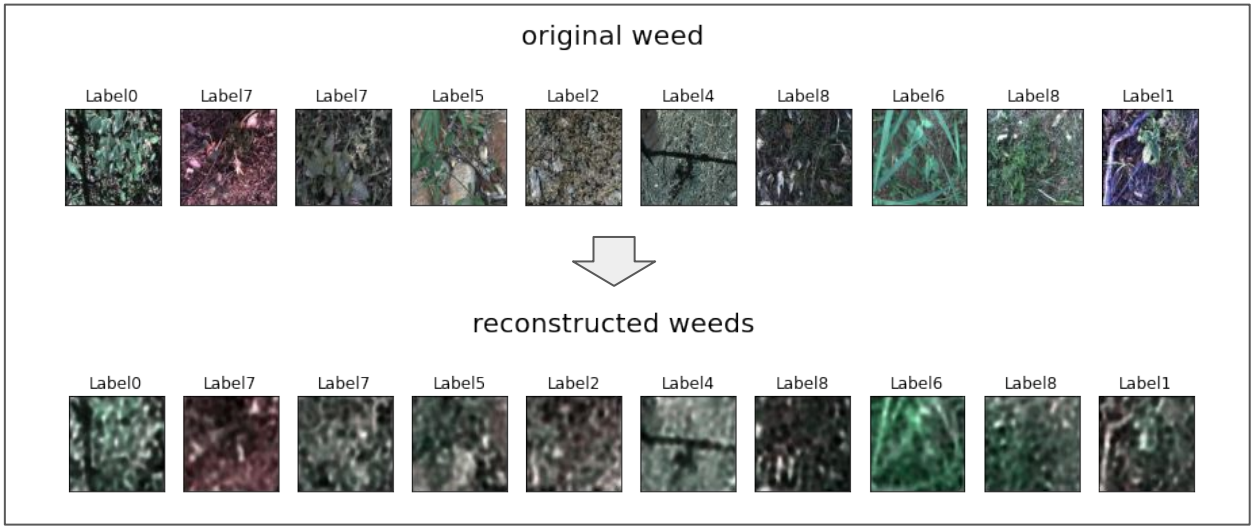
Auto Encoder Training result

- Trained with different z space (2, 256, 512, 1024)

Validation MSE for different z dimensions

Dimension of Z space	MSE loss
2	0.053
256	0.028
512	0.021
1024	0.021

Result after 20 epochs with z space of 512 dimensions



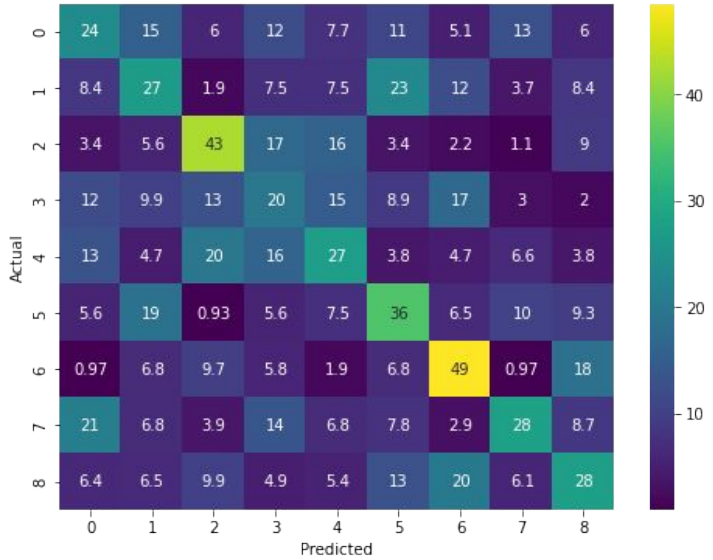
Linear classifier with the new feature space Z

- Now train the classifier model with the new self-learnt feature
- Result : Outperformed the raw pixels but did not do as well compared to the engineered features from pixels

Scores for model

Model		f1-score	precision	recall
SVM	Weighted	0.32	0.50	0.29
	Macro	0.25	0.25	0.31
Random forest	Weighted	0.31	0.49	0.29
	Macro	0.25	0.49	0.29
KNN	Weighted	0.49	0.26	0.24
	Macro	0.24	0.24	0.30
Logistic regression	Weighted	0.30	0.46	0.28
	Macro	0.24	0.24	0.30

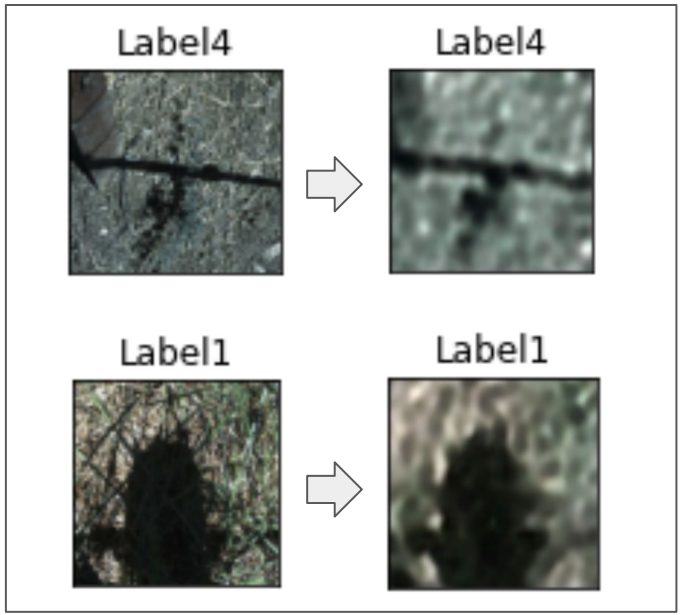
confusion matrix for SVM model



Possible reasons why it did not work well

- Auto Encoder might be learning a feature to represent the unnecessary information in the original image

AE learning to represent the shadow rather than the weed itself

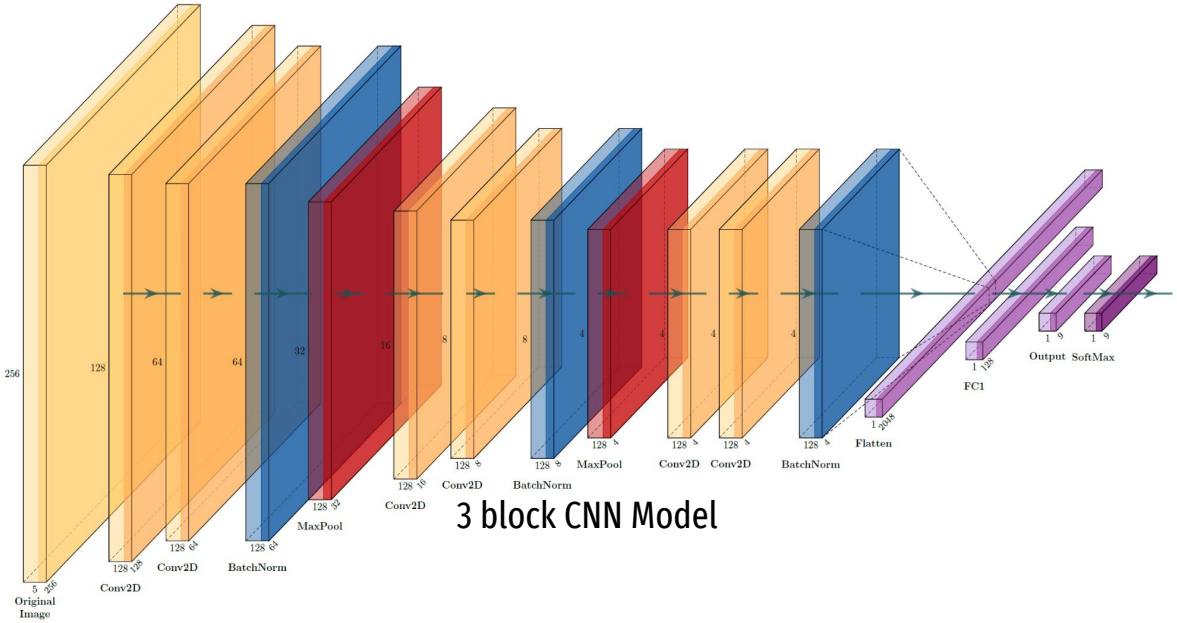


AE learning to represent the non-weed object rather than the weed itself

Convolutional Neural Networks

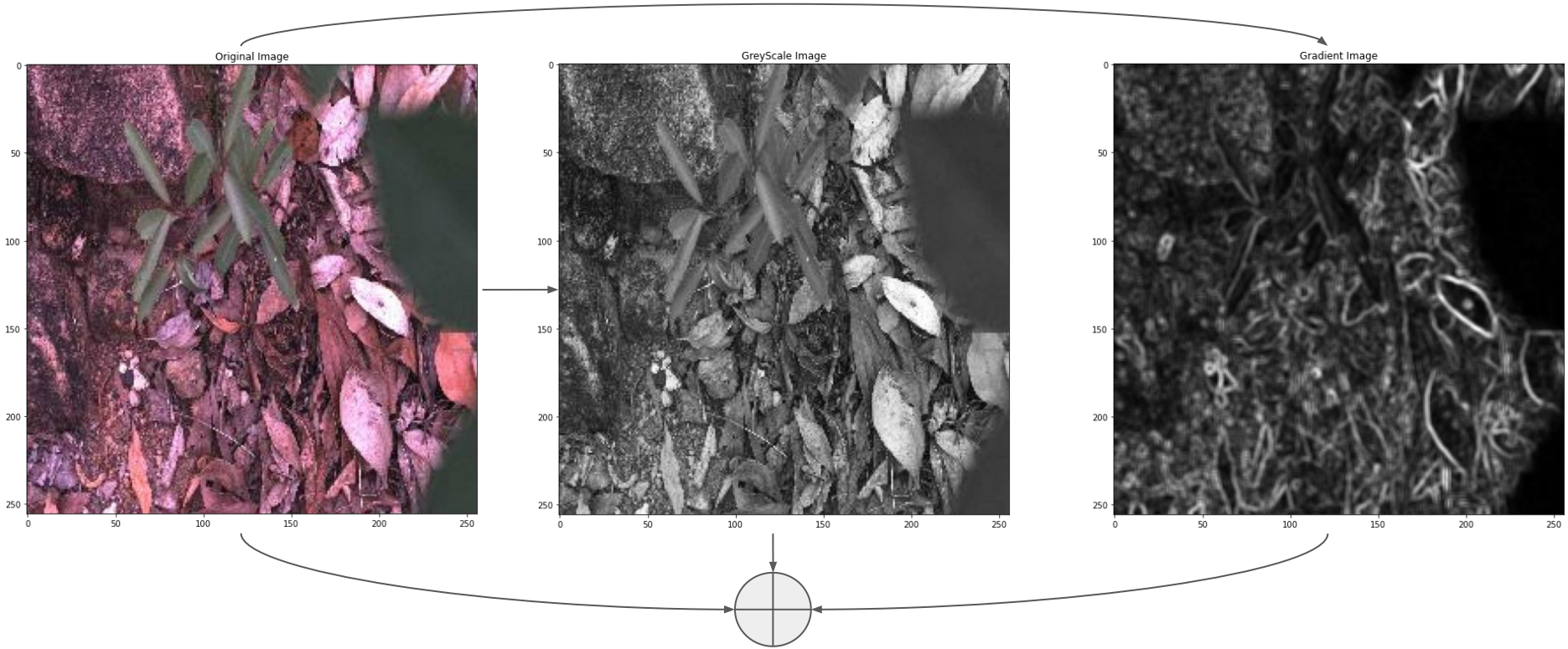
CNN are a deep learning algorithm which work especially well with images and other types of data by using Kernels and convolving.

They work well as they are able to correlate features with one another and learn to understand the relationship between the features as well.

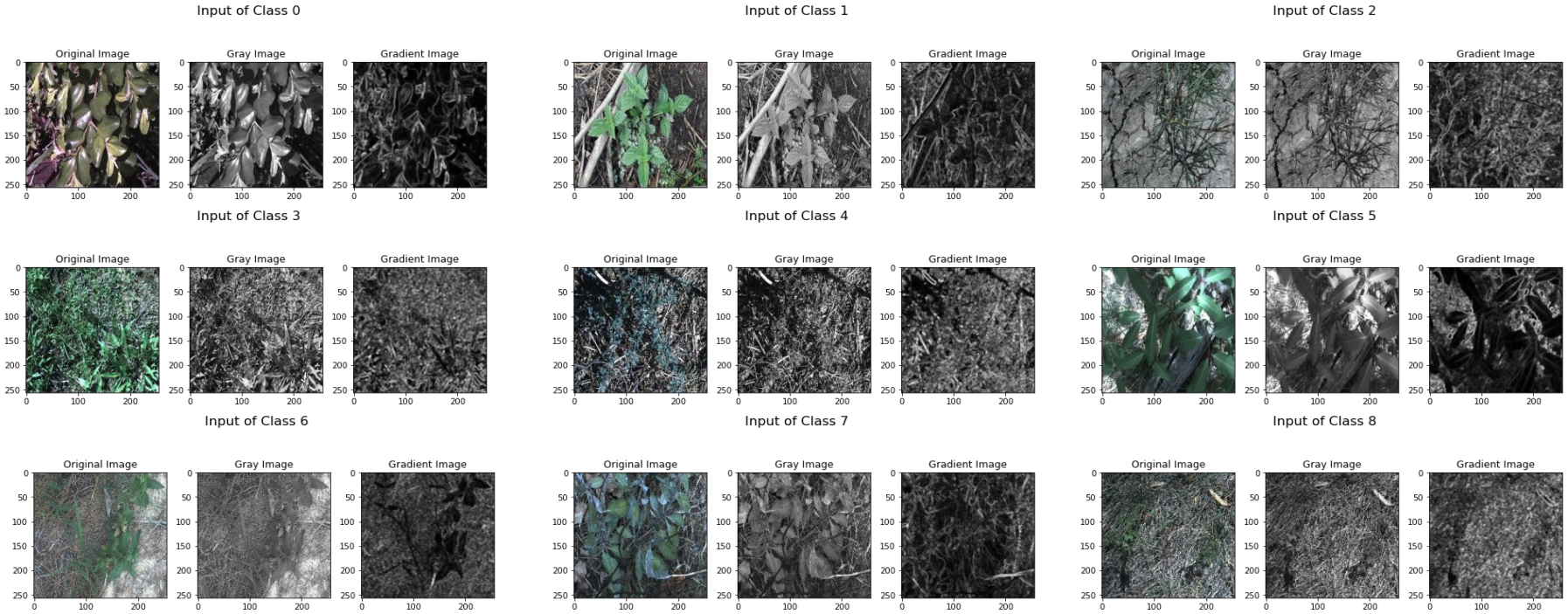


3 block CNN Model

Pre-processing Steps for CNN's



Pre-processing Steps for CNNs

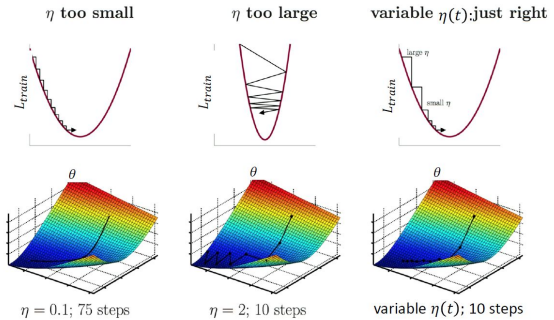


- Following which they are randomly
  - Rotated
  - Flipped along the x-axis and y-axis
- Lastly the Output Label is One-hot encoded

This helps reduce overfitting as during training the same image is always perturbed differently

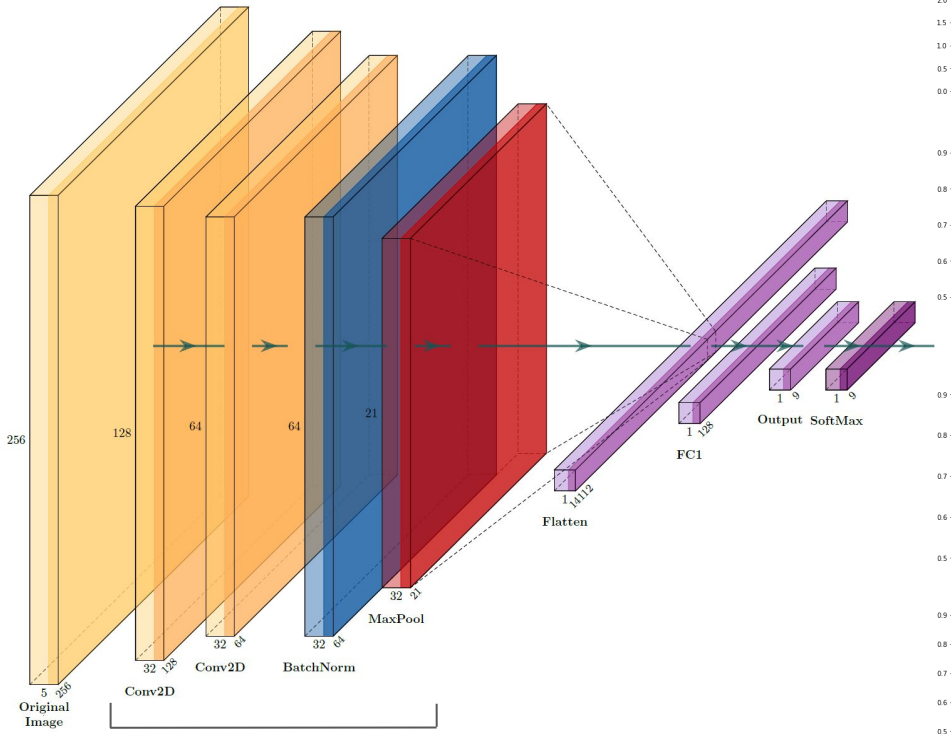
Error Function and Optimizer Used

- **Categorical Cross Entropy** was used as the loss function as there are in total 9 labels and the output of the CNN is a [9 x 1] vector indicating the probability that the input image is of class i where i ranges from 1 to 9.
- The optimizer used here was **Adam** which is used instead of Stochastic Gradient Descent as it makes use of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) to help determine the learning rate.
- On top of this an **early stopping** criterion was established which stops training if the validation loss does not improve after a fixed amount of epochs ***m*** which is also called patience.
- Lastly we utilised a function which reduces the learning rate if there isn't a improvement in the validation loss for ***n*** epochs, this give us some freedom for error in setting the initial learning rate, and it helps in convergence as well as it allows us to use a higher learning rate at the start and over the course of the training it will reduce to allow for fine-tuning.

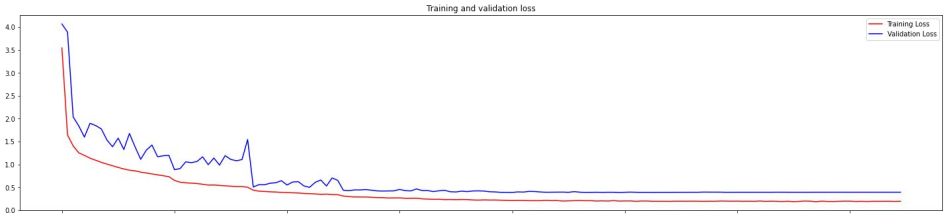


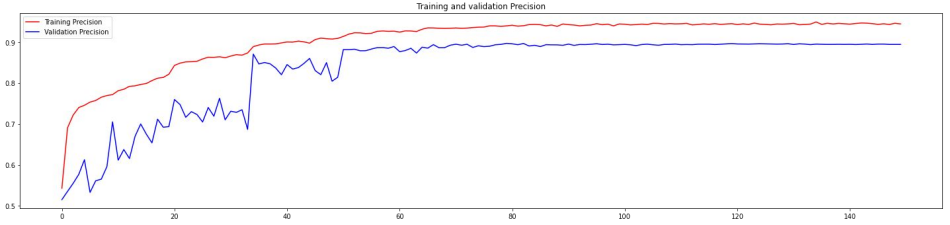
Problem with incorrect learning rate

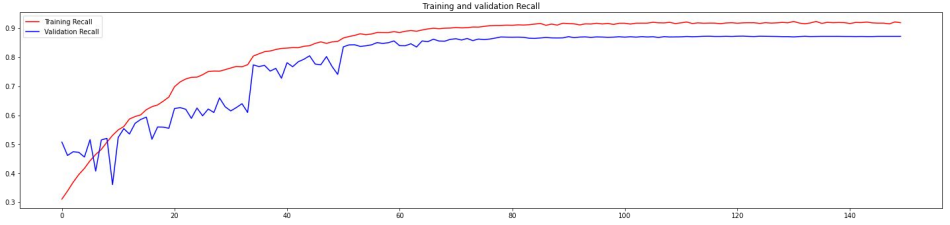
Base Model



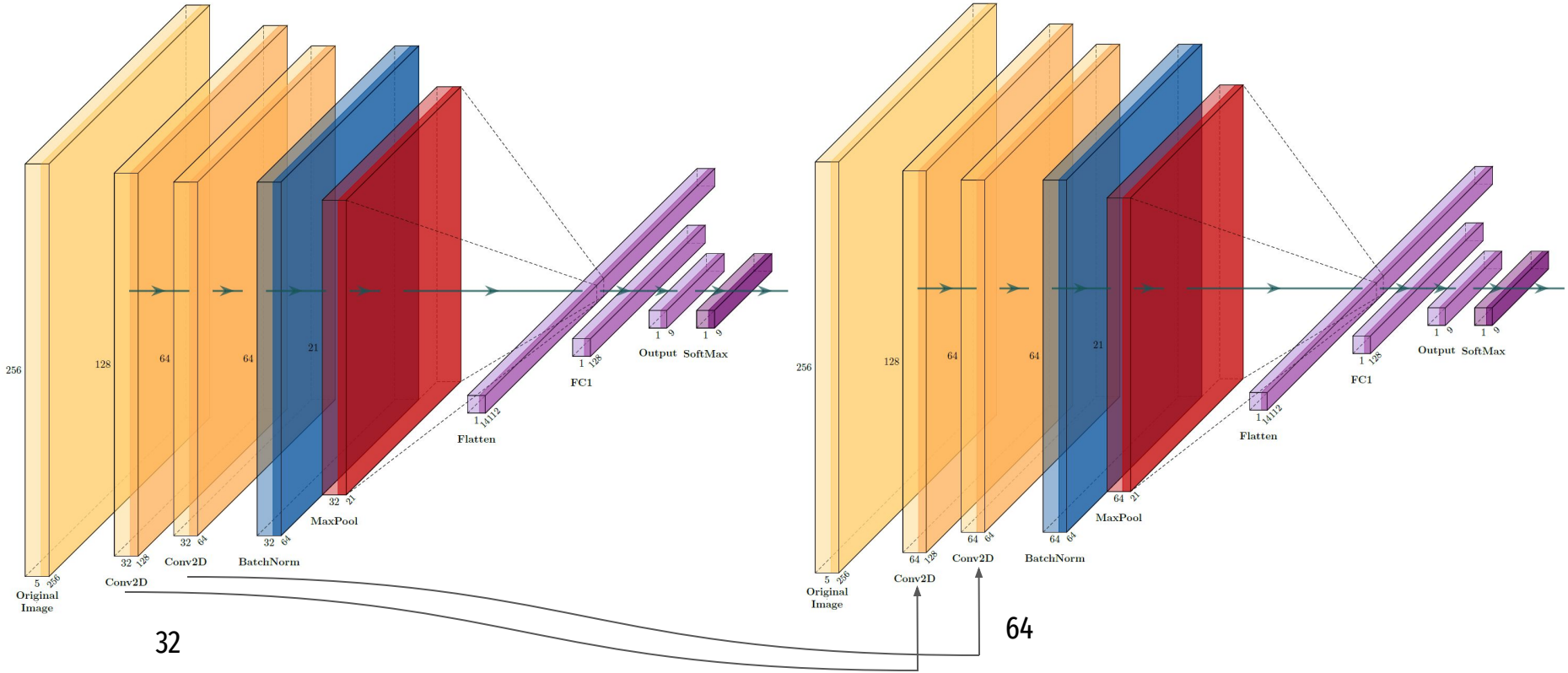
1 Convolution Block



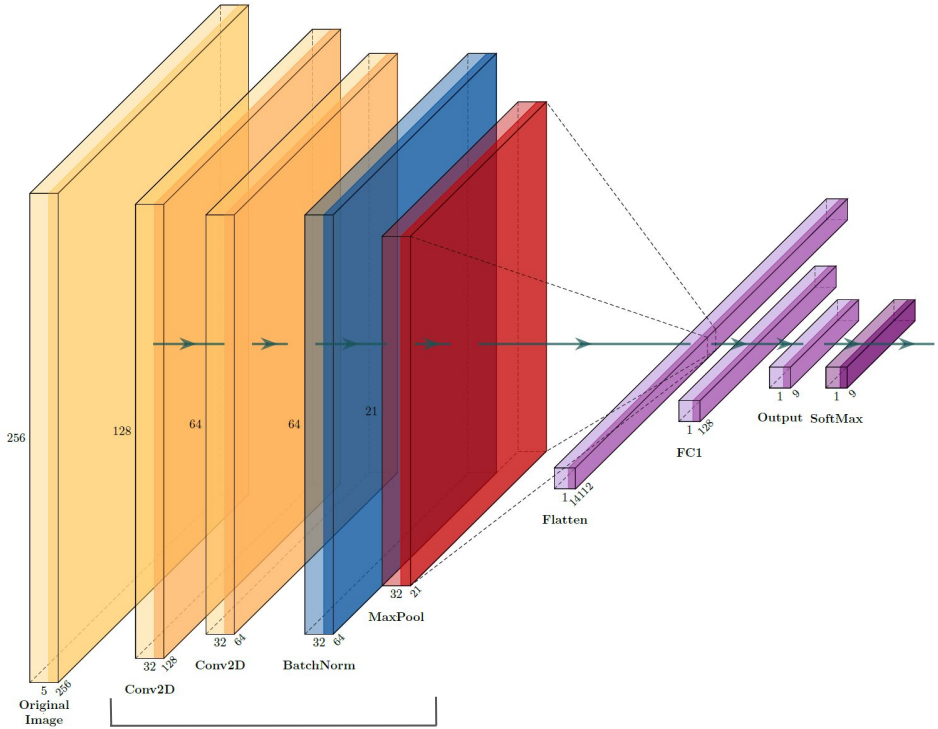


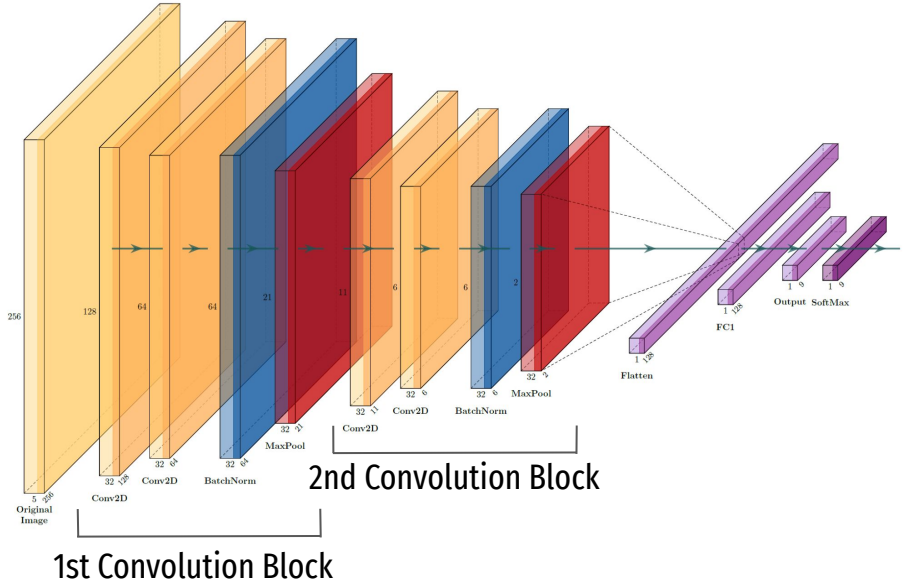


Increasing Number of Filters (Doubling)

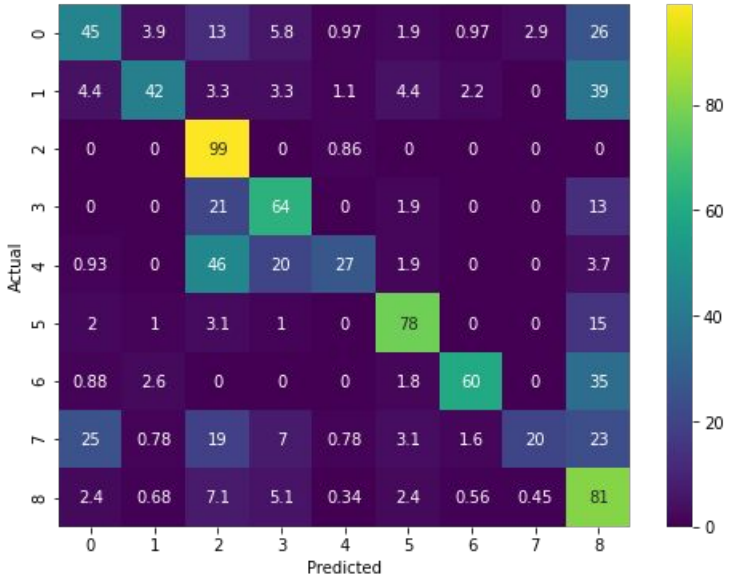


Increasing the number of Convolutional Blocks



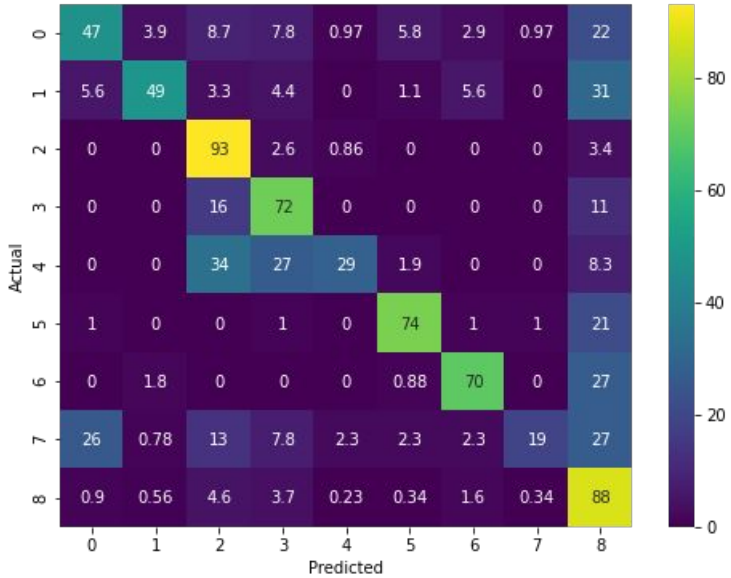


Results - 1 block CNN



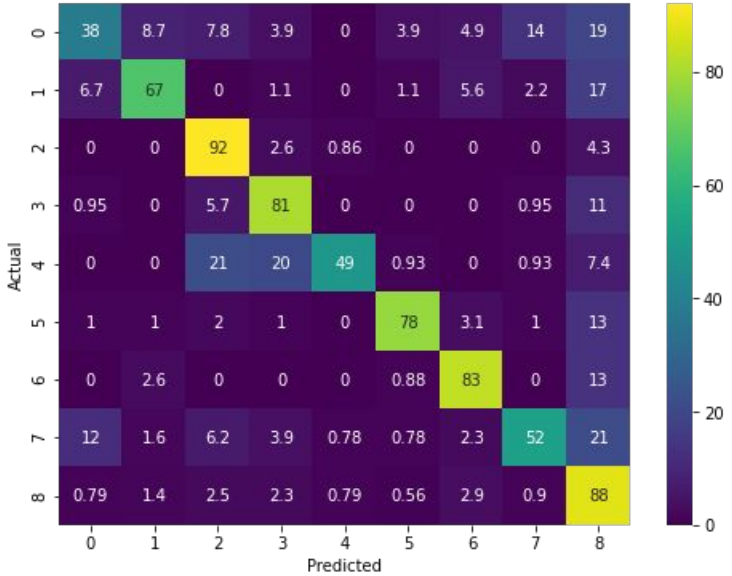
	precision	recall	f1-score	support
0	0.43	0.45	0.44	103
1	0.73	0.42	0.54	90
2	0.39	0.99	0.56	116
3	0.44	0.64	0.52	105
4	0.81	0.27	0.40	108
5	0.67	0.78	0.72	98
6	0.87	0.59	0.70	114
7	0.79	0.20	0.32	129
8	0.81	0.81	0.81	888
accuracy			0.68	1751
macro avg	0.66	0.57	0.56	1751
weighted avg	0.73	0.68	0.67	1751

Effect of Increasing Number of Filters →



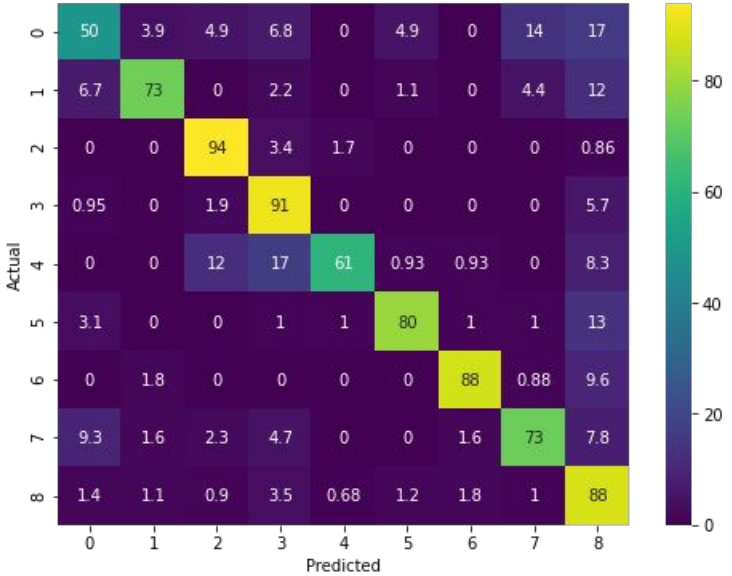
	precision	recall	f1-score	support
0	0.51	0.47	0.49	103
1	0.77	0.49	0.60	90
2	0.46	0.93	0.62	116
3	0.46	0.72	0.56	105
4	0.82	0.30	0.44	108
5	0.81	0.74	0.78	98
6	0.75	0.70	0.73	114
7	0.83	0.19	0.30	129
8	0.83	0.87	0.85	888
accuracy			0.72	1751
macro avg	0.69	0.60	0.60	1751
weighted avg	0.75	0.72	0.71	1751

Results - 2 block CNN



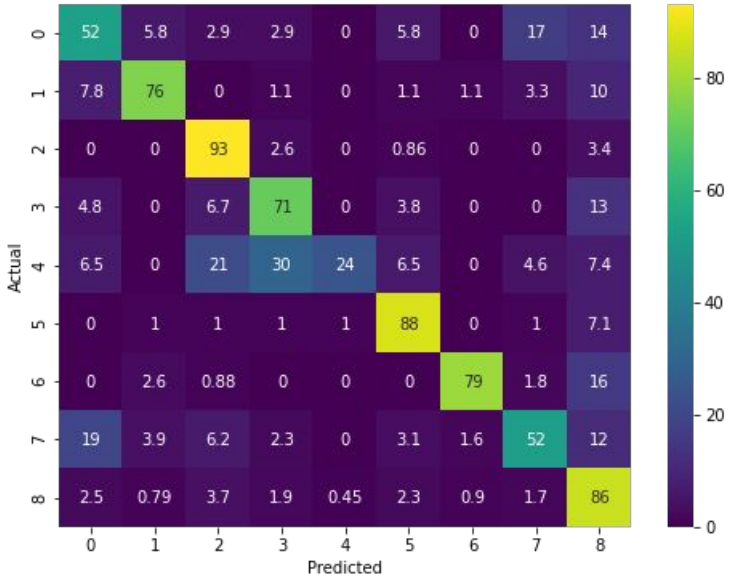
	precision	recall	f1-score	support
0	0.56	0.39	0.46	103
1	0.68	0.67	0.67	90
2	0.60	0.92	0.73	116
3	0.61	0.81	0.69	105
4	0.85	0.49	0.62	108
5	0.85	0.78	0.81	98
6	0.69	0.83	0.76	114
7	0.71	0.52	0.60	129
8	0.87	0.88	0.88	888
accuracy			0.78	1751
macro avg	0.72	0.70	0.69	1751
weighted avg	0.79	0.78	0.77	1751

Effect of Increasing Number of Filters →



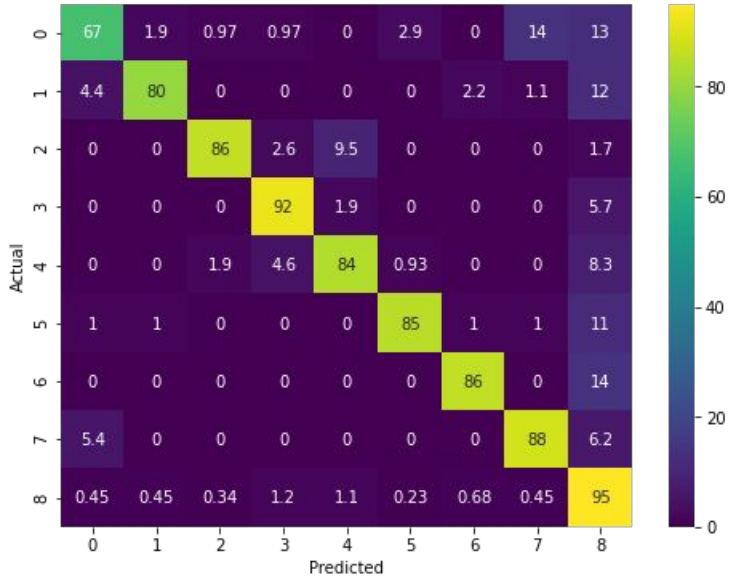
	precision	recall	f1-score	support
0	0.61	0.50	0.55	103
1	0.80	0.73	0.76	90
2	0.77	0.94	0.85	116
3	0.58	0.91	0.71	105
4	0.89	0.61	0.73	108
5	0.81	0.80	0.80	98
6	0.83	0.89	0.86	114
7	0.77	0.74	0.75	129
8	0.91	0.89	0.90	888
accuracy			0.83	1751
macro avg	0.78	0.78	0.77	1751
weighted avg	0.84	0.83	0.83	1751

Results - 3 block CNN



	precision	recall	f1-score	support
0	0.81	0.67	0.73	103
1	0.91	0.81	0.86	90
2	0.94	0.85	0.90	116
3	0.81	0.91	0.86	105
4	0.80	0.83	0.82	108
5	0.93	0.85	0.89	98
6	0.92	0.86	0.89	114
7	0.85	0.88	0.87	129
8	0.92	0.95	0.93	888
accuracy			0.89	1751
macro avg	0.88	0.85	0.86	1751
weighted avg	0.89	0.89	0.89	1751

Effect of Increasing Number of Filters



	precision	recall	f1-score	support
0	0.81	0.67	0.73	103
1	0.91	0.80	0.85	90
2	0.94	0.85	0.90	116
3	0.82	0.92	0.87	105
4	0.81	0.84	0.82	108
5	0.93	0.87	0.90	98
6	0.92	0.86	0.89	114
7	0.85	0.88	0.87	129
8	0.92	0.95	0.94	888
accuracy			0.90	1751
macro avg	0.88	0.85	0.86	1751
weighted avg	0.90	0.90	0.90	1751

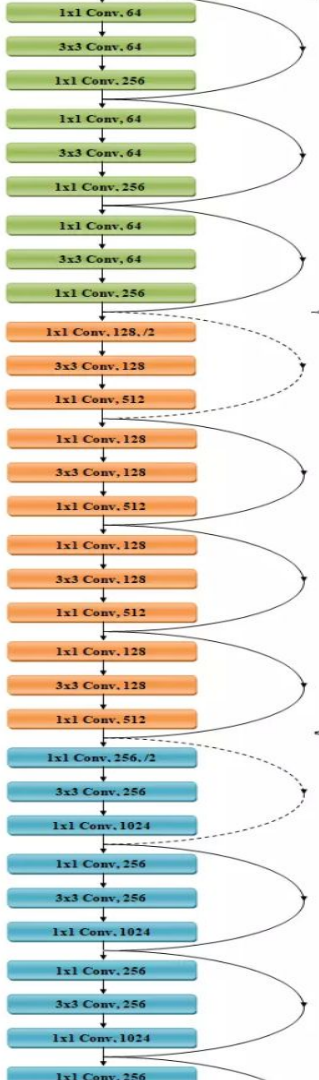
Transfer Learning using ResNet-50

Model - Resnet-50

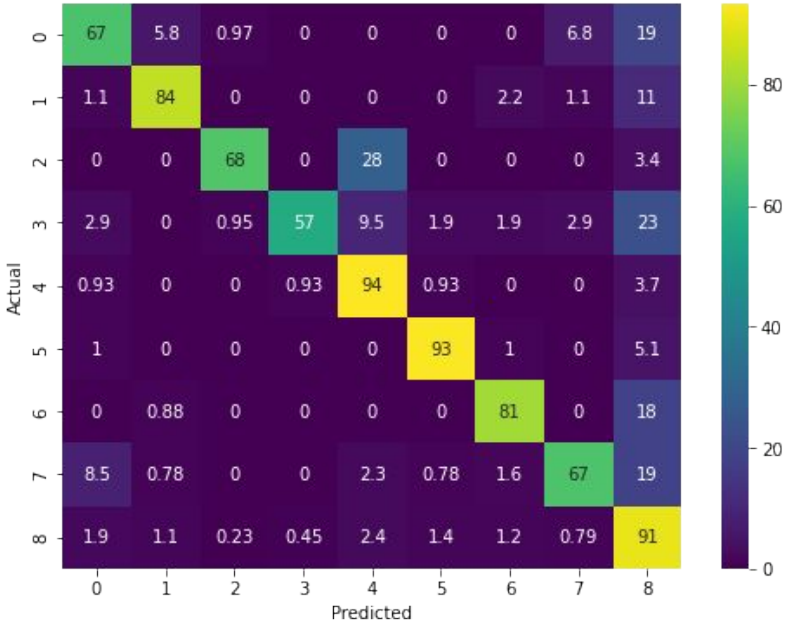
- 50-layer variant of the ResNet architecture, which adds shortcut connections → enables training of deeper models by mitigating the **vanishing gradients problem**
- Pre-trained weights on a subset of the ImageNet dataset (~1.3 million training examples and 1000 classes) → speed up training and improves performance on our small dataset

Approach

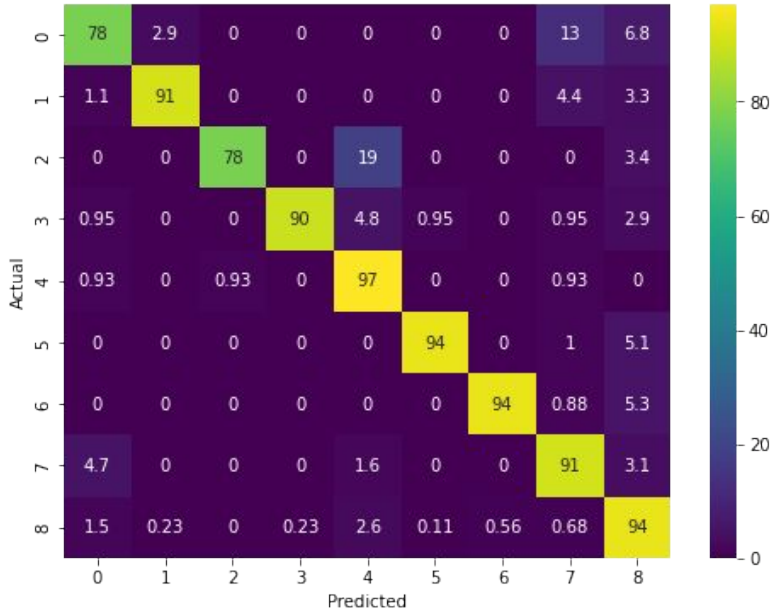
- Removed original classification head for ImageNet model
- Added a dropout layer + fully-connected dense layer for 9 classes with softmax activation function
- Trained only the new classification layer to convergence (froze pre-trained weights)
- **Fine-tuned model** by unfreezing all weights and training to convergence with a 50x smaller learning rate



Findings - Effects of Fine-tuning



Performance before fine-tuning



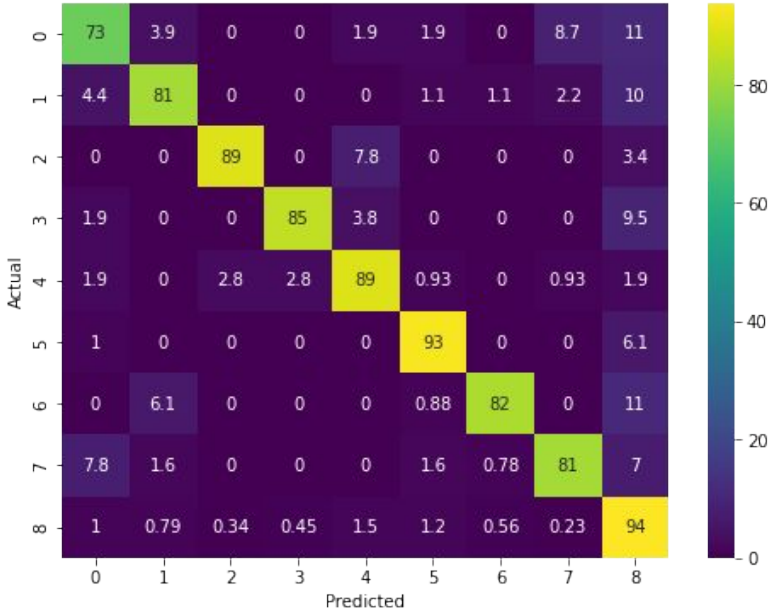
Performance after fine-tuning

Outcomes

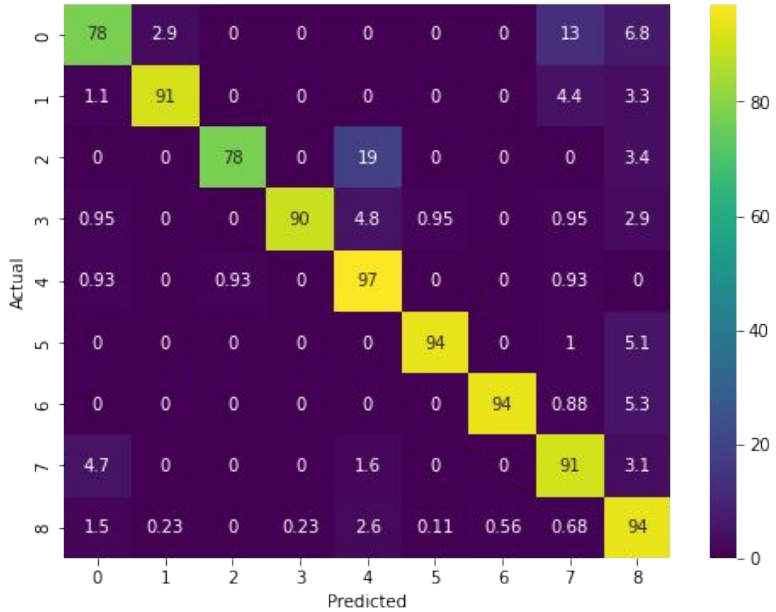
Much better performance with fine-tuning (Macro-average F1 Score - 0.79 vs 0.89, Micro-Avg F1 Score - 0.83 vs 0.92)

Longer training time (Additional 2x training time to finetune i.e. 1.5 hours on Colab)

Findings - Benefits of Transfer Learning



Random weight Initialization



ImageNet weight Initialization

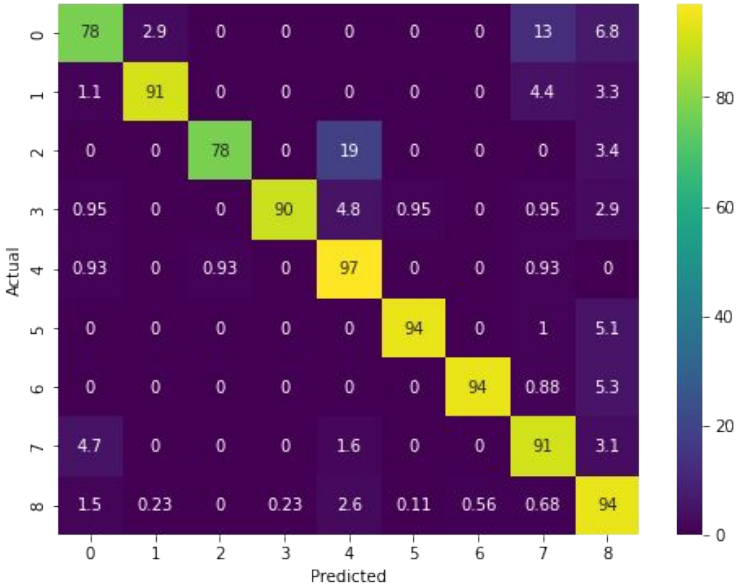
Outcomes

Better performance for transfer learning (Macro-Average F1 Score - 0.85 vs 0.89, Micro-Avg F1 - 0.89 vs 0.92)

Faster convergence for transfer learning! (~25% less training time needed i.e. 1.33 hours on Colab)

Discussion - CNN Models

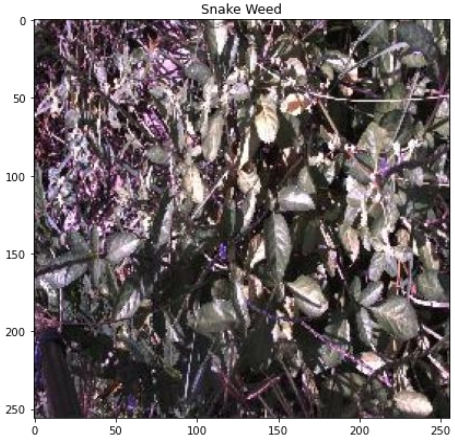
- Largely reproduces the findings from the original paper, which attained ~95% accuracy using the same Resnet50 model
- Many weeds misclassified as non-weeds (class 8)
- Some plants are often misidentified for each other
    - **13%** of Class 0 (Chinee apple) misidentified as Class 7 (Snake weed)
    - **19%** of Class 2 (Parkinsonia) misidentified as Class 4 (Prickly acacia)

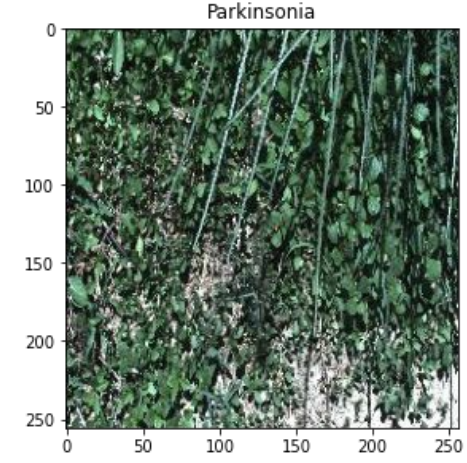


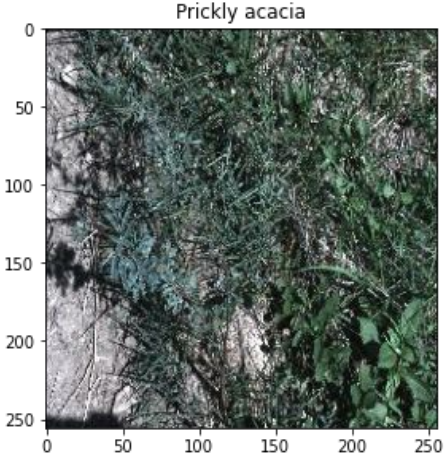
Resnet50 Model Results

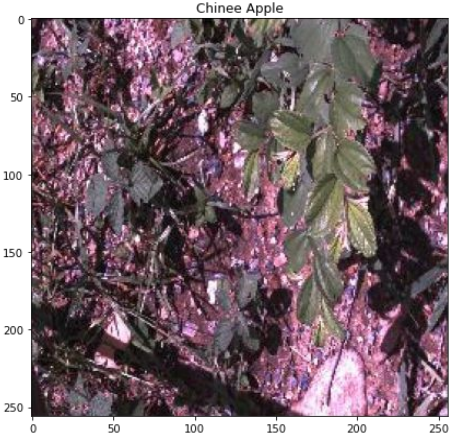
Visual Similarity between Classes

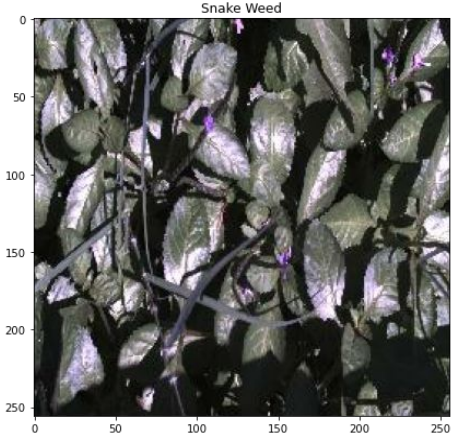
- Plants misidentified for each other are often **visually similar**
  - Class 0 (Chinee apple) and Class 7 (Snake weed)
  - Class 2 (Parkinsonia) and Class 4 (Prickly acacia)



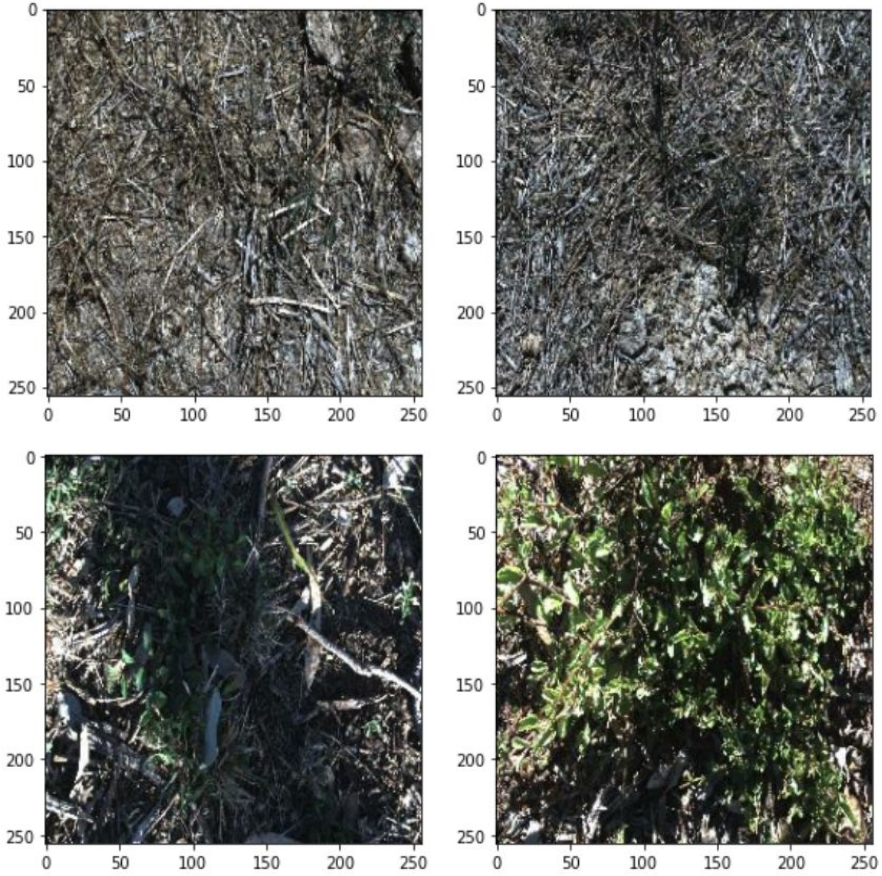






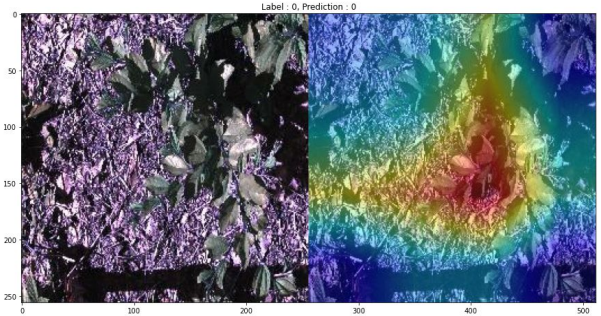


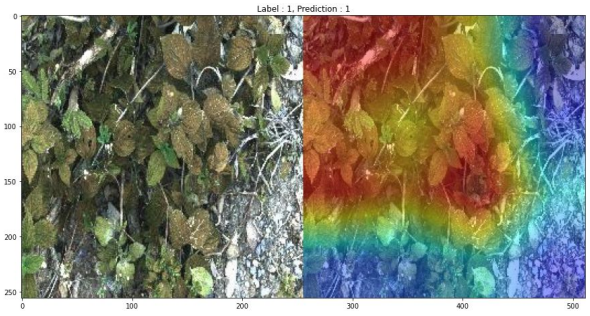
Misidentified Images

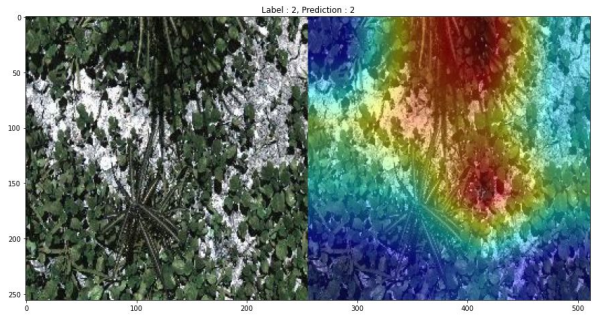


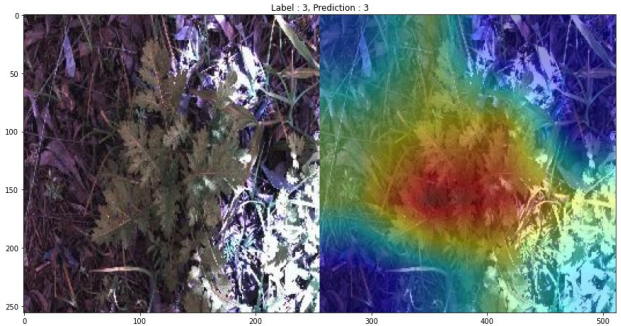
- Bad lighting
- Prominent shadows
- Details of plant unclear

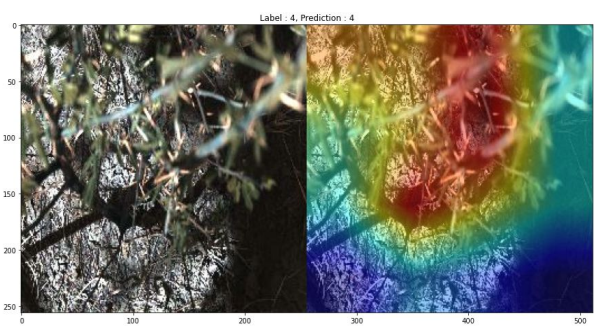
GRADCAM as a Sanity Check

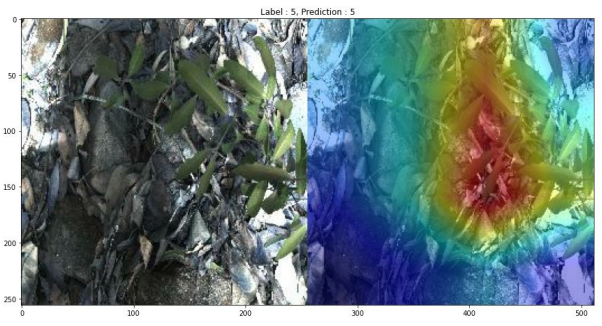


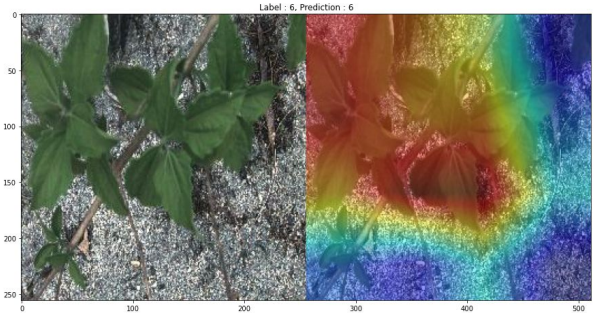


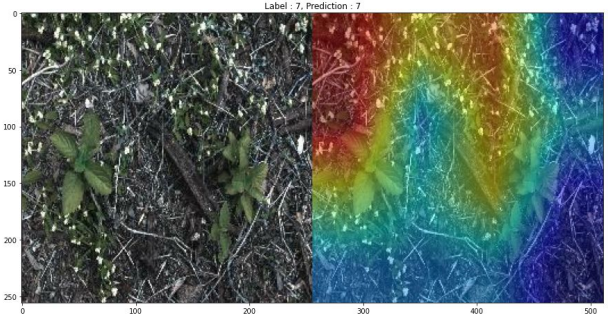


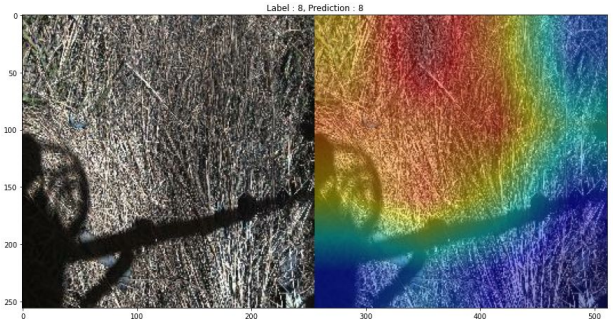






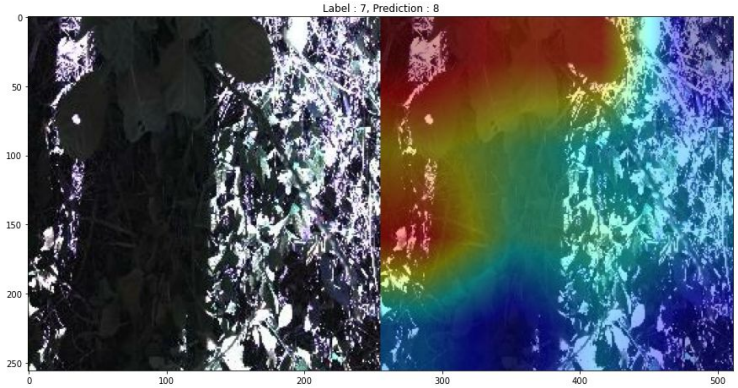


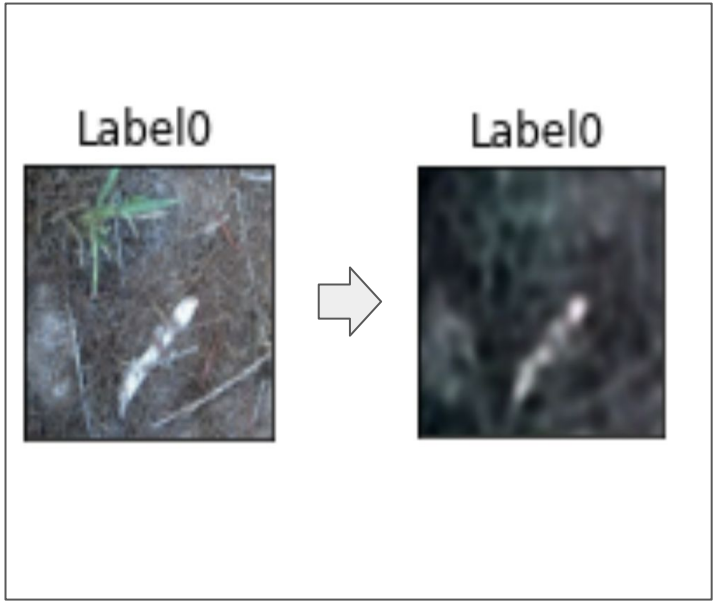




Future Improvements

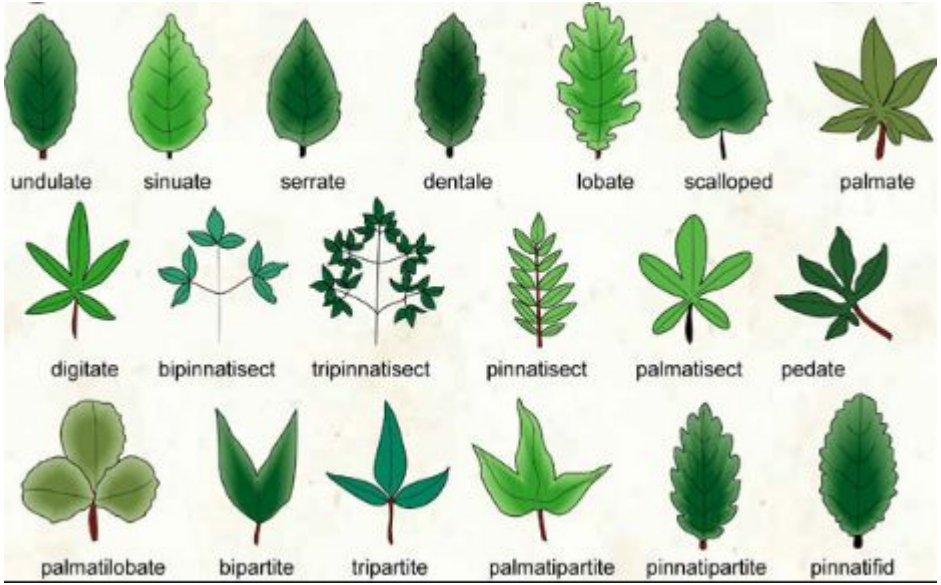
- Background/shadow removal using image segmentation
  - So the model actually looks at the weeds





Future Improvements

- Training a model to recognise specific plant characteristics (e.g. lead shapes, venation types) then using a classification key/rule-based model to predict
    - Similar to how humans classify plants



References

Scientific Papers

- Olsen, A., Konovalov, D. A., Philippa, B., Ridd, P., Wood, J. C., Johns, J., ... & White, R. D. (2019). DeepWeeds: A multiclass weed species image dataset for deep learning. Scientific reports, 9(1), 1-12.
- Hasan, A. M., Sohel, F., Diepeveen, D., Laga, H., & Jones, M. G. (2021). A survey of deep learning techniques for weed detection from images. Computers and Electronics in Agriculture, 184, 106067.
- Tian, Q, Minjian, L., Guiping Z, Wang L.(2019) An Unsupervised Classification Method for Flame Image of Pulverized Coal Combustion Based on Convolutional Auto-Encoder and Hidden Markov Model

Misc. Resources

- Brownlee, J. (2021, January 12). Gentle introduction to the adam optimization algorithm for deep learning. Machine Learning Mastery. Retrieved November 20, 2021, from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.
- GRADCAM reference code (https://www.pyimagesearch.com/2020/03/09/grad-cam-visualize-class-activation-maps-with-keras-tensorflow-and-deep-learning/)
- Keras Developer Guides (https://keras.io/guides/) and Code Examples (https://keras.io/examples/)
- Guide to Keras Data Generators (https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly)
- CS3244 Lecture Materials
- CS4243 Lecture Materials

Thank You!