COS801 - Assignment 1

Kampamba Chanda*
Department of Computer Science
University of Pretoria
u14366798@tuks.co.za

Abstract

This report addresses a 12 class plant species image classification task using three approaches: an end to end Convolutional Neural Network (CNN), and two classical models—Support Vector Machines (SVM) and XGBoost (XGB)—trained on features extracted by a compact CNN backbone. A tuned CNN (Leaky ReLU, 5×5 kernels, 48 filters) improved validation accuracy from 65% to 80% with lower loss, without materially increasing epoch time. On validation, SVM achieved 87% accuracy, while XGB reached 86% accuracy. Test set prediction distributions for all three models are reported for all classes. The pipeline was implemented on Google Colab (T4 GPU) with caching and version control for reproducibility.

1 Introduction

Modern visual recognition tasks are dominated by CNNs, which learn hierarchical features directly from pixels and routinely deliver reliable performance in image classification [1]. At the same time, margin based methods such as SVMs remain competitive when supplied with strong features, while gradient boosted trees (e.g., XGBoost) excel on structured/tabular representations [2], [3]. In this work, we compare: (i) an end to end CNN classifier and (ii) end-to-end CNN with improved hyperparameters (iii) and classical SVM/XGB models trained on vector features obtained from a lightweight, three layer CNN backbone with global average pooling.

2 Related work

A robust line of work shows that "off the shelf" CNN features, without task specific fine tuning, serve as powerful descriptors for downstream classifiers such as linear models or SVMs, providing strong baselines across recognition tasks [4]. This motivates our two stage pipeline where a compact CNN backbone produces fixed features for an RBF SVM and XGB. XGBoost is a scalable, regularized gradient boosting framework that often achieves state of the art results on tabular data via depth controlled trees, shrinkage, column/row subsampling, and early stopping [3]. Beyond compact custom CNNs, contemporary practice frequently leverages pretrained backbones with principled scaling followed by staged fine tuning to improve accuracy and efficiency [5].

3 Dataset and Features

Data sets were provided by means of two .zip files (Training data and Test Data)

1. The training dataset is made of 4750 images

*Kampamba Chanda is a UKZN Mechanical Engineering Graduate, with 7 years of engineering experience. He is currently a System Engineer at Camden Power Station Eskom and Graduate Student in MIT Big Data and Machine Learning ,

2. The test dataset is made of 794 images
3. All images in both datasets are Portable Network Graphics images
4. Strong majority of training images are sized in the 0-500kb class
5. 1.5% of training images are not square in dimension
6. Class size disparities exist within the training dataset
7. Image sizes in the test set are fairly distributed
8. All images in the test set are square
9. The smallest image by dimension in the train set is 49x49 and the largest is 3991x3991
10. The smallest image by dimension in the test set is 200x200 and the largest is 349x349

3.1 Pre-processing

Based the explorations in the above, the below could was selected:

- **RGB Conversion** - To ensure consistency in the input to the convolution neural network, all images were standardized to three-channel RGB format with an intensity range of 0–255 per channel (i.e., a 255 × 255 × 255 color space).
- **Resolution Reduction** - To prevent distortion caused by overstretching the smallest images in the training dataset, a target resolution of 96 × 96 pixels was selected during preprocessing.
- **Padding non-square images** - To preserve the aspect ratio of non-square images during resizing, padding with a pure white background was applied to generate square images. Each image was centered at coordinates (0.5, 0.5), and high-quality LANCZOS resampling was employed to maintain visual fidelity.

3.2 Augmentation

To enhance dataset diversity and ensure that the network was exposed to all possible variations of the images, random augmentation techniques were applied. This approach increased the robustness of the model by enabling it to learn features invariant to changes in lighting, orientation, and positioning, while also mitigating the risks of overfitting and memorization. Augmentation steps are listed below:

- **Pixel Rescaling** – Pixel values were normalized to the range 0–1 to reduce input complexity and facilitate efficient training.
- **Random Brightness Adjustment** – Image brightness was randomly varied within a range of $\pm 10\%$ to improve robustness to lighting conditions.
- **Random Contrast Adjustment** – Image contrast was randomly varied within a range of $\pm 10\%$ to enhance resilience against illumination differences.
- **Random Flipping** – Images were randomly flipped horizontally and vertically to prevent overfitting and to ensure invariance to mirror symmetries and viewpoint reversals.
- **Random Rotation** – Images were randomly rotated by up to $\pm 36^\circ$ (10% of 2π) to account for natural camera misalignment and object tilt, ensuring feature stability under moderate angular changes.

It's important to note that augmentation steps were applied to training and validation dataset only.

4 Methods

The steps described in **Section 3** above were done locally using VS Code. For the duration of the project, the work was done on Google Colab using a T4 GPU. The images were zipped and uploaded to a temporary Google drive folder for processing. A git repository was established to house different pushes of the program, and a python notebook was started in Google Colab using T4 GPU runtime.

4.1 Processing

For processing the data, packages such as Google Colab, TensorFlow, SKLearn, NumPy, Seaborn. A detailed list is provided in appendix A. The data was cached to a local storage on google colab for fast access. Input hyperparameters such as image size, batch size, random seed, and validation split were defined here as well.
Two datasets were produced using the defined validation split, and the test dataset was also defined and loaded with the cached data.
Furthermore, the CNN hyperparameters were also defined at this stage, namely Learning Rate, Weight Decay, Dropout Rate, Max number of epochs, Kernel Size, Activation function, Number of Kernels.

4.2 Feature Extraction

The steps followed to extract numeric vectors containing feature data for each image are highlighted in this section.

The back bone of the Feature extractor CNN was chosen to be a three layer TensorFlow keras architecture, with 2D MaxPooling of stride = 1 after each layer. The number of Kernals were allowed to follow a geometric doubling pattern, and maxpooling ensured dimension reduction. This waws done to ensure that an easy to work with 12x12x128 output from the third layer. The head of the Feature extractor CNN consisted of four additional processing blocks (i.e, Global Average Pooling, Batch Nortmalisation; Dropout; Dense Softmax.

4.3 Training

4.3.1 CNN

Convolutional Neural Networks learn by progressively detecting more complex patterns in inputs. Each layer of the network applies small filters (convolutions) that scan across the image and respond strongly when a particular feature is present. This operation is called convolution, and it is the mathematical way of expressing the idea of mixing one function (input data x) with another function (weighting balance w) to produce a new, smoother function (s).

These activations are then pooled and passed to deeper layers. During training, the network adjusts its filter weights to minimize error between its predictions and the true labels. CNNs excel in vision tasks because they automatically learn spatial hierarchies of features, reducing the need for manual feature engineering.

Training the CNN proved to be a straight forward exercise, since the results from the feature extractor above allowed for direct feed into the neural network. Results of this training model during validation and testing are provided in 5

4.3.2 SVM

Support Vector Machines approach learning by drawing the “best separating boundary” between classes. SVMs focus on identifying hyperplanes that maximizes the margin between the closest points of different classes, known as support vectors. SVMs use kernels—mathematical functions that map the original features into higher-dimensional spaces where classes are more easily separable. Training involves solving an optimization problem to find the hyperplane that achieves the widest possible margin while balancing misclassifications through a regularization parameter.

Before training of the SVM, some preparatory work had to be done to ensure that the model has the best chance of performing optimally. The layer before GAP modifications was chosen as this represented pure output of the feature extraction in 4.2. A single pass over the feature extractor was done since there was no need for multi-pass weight optimisation and through cost function reduction. A two stage SVM classificatioan wrapped in Sklean pipeline was chosen, with Standardisation and Radial Bias Function Kernel SVM.

With the class imbalances in mind, class weight balancing was chosen to adjust learning weights according to class size, as a parameter to mitigate bias in the model. In the training pipeline. Probability estimations were disabled during the search pipeline, however were later enabled in the final model.

Hyperparameter tuning occurred in two steps(1) Step 1 was a course grid search; (2) Step 2 was a refined grid search with tighter centres. During each stage, the macro-averaged F1 score was favored as an evaluation metric. Each C and γ reading was stored and kept as hyperparameter. Finally a fresh pipeline was retrained on the full dataset. At this stage, probability estimations were turned on again such that the model could calibrate accordingly. The final model therefore benefited from optimized hyperparameter, balanced weights, and standardised input features. Results of this training model during validation and testing are provided in 5

4.3.3 XGB

Extreme Gradient Boosting (XGBoost) learns by combining the strengths of many "weak learners," typically shallow decision trees, into a single strong model. Over many iterations, the ensemble converges into a powerful predictor. Regularization terms are added to control complexity, preventing overfitting and ensuring generalization to new data. XGBoost is designed for speed and efficiency, leveraging parallelization and, when available, GPU acceleration.

The same feature extractions used in the SVM model were also used in the XGB model, however they were converted to *float32* data type on account of the architecture of the model. Balanced class weights were also introduced. With the availability of GPU runtime, the model was able to run using GPUhist tree method which greatly accelerated the training, and still used the CPU as a fallback for graphing resources.

5 Results and Discussion

5.1 CNN Results

Table 1: Justification for CNN Hyperparameters

Hyperparameter	**Justification**
Learning Rate	0.001 – Common safe rate for Adam optimiser, used when compiling the model.
No. Kernels	32 – Generally accepted as a safe starting point for few-layer, less complex models.
Kernel Size	3×3 – Allows sufficient network depth while balancing computational cost.

The CNN model was able to achieve the below scores when checked with validation data:

Metric	**Type**	**Value(s)**
Keras	val_loss, val_acc	1.0148, 0.6531
sklearn	acc	0.6531
sklearn	macro (P, R, F1)	P=0.6811, R=0.6318, F1=0.6161
ROC-AUC	micro, macro	micro=0.9562, macro=0.9624

Table 2: Performance Metrics Summary

The CNN model under shows a balanced performance across class distribution, precision, and recall. From the Count plot, the dataset appears relatively well-distributed, avoiding the pitfalls of class imbalance that often bias training. The Precision and Recall plots indicate the model maintains consistent predictive reliability, identifying true positives effectively while minimizing false alarms. Together, these metrics suggest the CNN generalizes well across categories, though minor variations in recall may hint at room for refinement. Additionally, hyperparameters were changed for the CNN to see if any improvements would be noted.

- Activation function changed from ReLU to Leaky ReLU.
- Convolutional kernel size increased from 3×3 to 5×5.
- Number of convolutional filters increased to 48.

The tuned CNN delivered a clear performance boost. By switching from ReLU to Leaky ReLU, expanding the kernel size from 3×3 to 5×5, and increasing filters to 48, the model achieved higher validation accuracy (from 65% to 80%) and lower loss (from 1.01 $\rightarrow$ 0.69). ROC-AUC scores also improved, reflecting stronger class separation. Training time per epoch remained stable, meaning gains did not come at a significant computational cost. Overall, the tuning was worthwhile: the changes enhanced feature extraction depth and reduced vanishing activation issues, leading to more robust learning without sacrificing efficiency.

5.2 SVM Results

Table 3: Justification for SVM Hyperparameters

Hyperparameter	Justification
Regularisation (C)	Values [0.1, 1, 10] – In line with the two-step tuning process discussed in Section 4.3.
Gamma	Values [0.001, 0.01, 0.1] – In line with the two-step tuning process discussed in Section 4.3.
Kernel Choice	RBF – Suitable for high-dimensional spaces, as discussed in Section 4.3.

The SVM model was able to achieve the below scores when checked with validation data: The

Metric	Type	Value(s)
sklearn	acc	0.8708
sklearn	macro (P, R, F1)	P=0.8647, R=0.8689, F1=0.8634
ROC-AUC	micro, macro	micro=0.9933, macro=0.9915

Table 4: Performance Metrics Summary

SVM model demonstrates strong classification capability, achieving an overall accuracy of 87% with balanced performance across classes. Both precision and recall values are consistently high, with many categories exceeding 80%, reflecting the model's ability to minimize false positives and false negatives. The ROC-AUC scores (micro 0.993, macro 0.992) highlight excellent class separability, confirming the model's robustness.

5.3 XGB Results

Table 5: Justification for XGB Hyperparameters

Hyperparameter	Justification
No. Trees	12 – Same as number of classes.
Max Depth	5 – Provides balance between complexity and risk of overfitting.
Sampling Strategy	0.9 subsampling, 0.8 column subsampling per tree – promotes diversity and reduces overfitting.

The SVM model was able to achieve the below scores when checked with validation data:

Metric	Type	Value(s)
sklearn	acc	0.8624
sklearn	F1 (micro, macro)	micro=0.8624, macro=0.8533
ROC-AUC	micro, macro	micro=0.9916, macro=0.9892

Table 6: Performance Metrics Summary

The XGBoost model demonstrates strong overall performance, achieving 86.2% . Precision values also remain high, typically above 80%. The ROC-AUC scores (micro 0.992, macro 0.989) underscore the model's excellent discriminative capability. While minor misclassifications persist, particularly for visually similar classes like Black-grass and Loose Silky-bent.

5.4 Test Results

Class	CNN (samples %)	SVM (samples %)	XGBoost (samples %)
Black-grass	13 (1.6%)	41 (5.2%)	36 (4.5%)
Charlock	73 (9.2%)	61 (7.7%)	63 (7.9%)
Cleavers	27 (3.4%)	51 (6.4%)	48 (6.0%)
Common Chickweed	95 (12.0%)	102 (12.8%)	107 (13.5%)
Common wheat	40 (5.0%)	33 (4.2%)	33 (4.2%)
Fat Hen	66 (8.3%)	69 (8.7%)	73 (9.2%)
Loose Silky-bent	147 (18.5%)	113 (14.2%)	115 (14.5%)
Maize	69 (8.7%)	37 (4.7%)	32 (4.0%)
Scentless Mayweed	36 (4.5%)	75 (9.4%)	85 (10.7%)
Shepherds Purse	14 (1.8%)	53 (6.7%)	39 (4.9%)
Small-flowered Cranesbill	43 (5.4%)	87 (11.0%)	90 (11.3%)
Sugar beet	171 (21.5%)	72 (9.1%)	74 (9.3%)

Table 7: Prediction distribution of CNN, SVM, and XGBoost models across test dataset (794 samples).

6 Recommendations

Some recommendations to improve on the above work are:

1. **Transfer learning + staged fine-tuning.** Replace the custom CNN with EfficientNetV2B0 (ImageNet weights), train head-only then unfreeze last blocks with a lower LR and weight decay:
2. **Stronger augmentation & sample mixing.** Add RandAugment/AutoAugment and MixUp/CutMix to improve margins and calibration:
3. **Multi-scale features & progressive resizing.** Train at 160 to 224 to 288 px to learn coarse-to-fine features.

7 Conclusion

This report contrasts modern deep learning with classical methods by evaluating CNNs, SVMs, and XGBoost on image recognition. Results highlight the strengths of end-to-end CNNs, the impact of optimized hyperparameters, and the continued competitiveness of SVM and XGBoost when paired with features from a compact CNN backbone.

8 GIT Repository

GIT Repo location is linked here https://github.com/MrChanda/COS_801_Assignment1_CNNvsSVMvsXGB.git

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.