Site Specific Weed Management (SSWM)

Table of Content:

Introduction

The loss in crop yield has to be taken into consideration for improving the return on investment for people depending on agriculture as well as for strengthen the economy of the country. The yield losses are caused by three major factors weed invasion, pests and pathogens, from which weeds are a major issue. Weeds are the unwanted plants that grow along with the crops e.g Chickweed, Scentless Mayweed etc. Weed control is the process of reducing or eliminating the loss in crop yield caused due to weeds

Objective: Our goal is to accurately differentiate weeds from crop seedlings based on their images in a reasonable time frame.

In this project, we would build a Convolutional Neural Network, as well as use several states of the art Convolutional Neural Networks and measure their respective performances on the pneumonia image dataset using several metrics, such as

- Accuracy: Calculates how often predictions equal labels.
- Precision: Computes the precision of the predictions for the labels
- Recall: Computes the recall of the predictions for the labels.
- True-Positive: Calculates the number of true positives.
- False-Positive: Outcome where the model incorrectly predicts the positive class.
- F1_score: the harmonic mean between precision and recall

The dataset consists of 6 classes of plants of RGB images, namely: **Blackgrass, Charlock, Cleavers, Fat Hen, Maize, and Wheat.**

Preprocessing: The image data were loaded into memory in a sorted manner. While the mask set was loaded and binarized to reflect classes of zero and 1, using the OpenCV package in python. The datasets were divided into two parts, 80% for training and 20% for validation, respectively. All images were resized to the same size (128 × 128) and normalised to the range [0,1].

Data Augmentation: Due to the small dataset available for training, it was paramount to do Data Augmentation. Hence, data augmentation was employed to generate larger training images and their corresponding masks.

Models Architecture

Several CNN architecture was built using the Tensorflow open-source framework:
Custom CNN: Convolutional Neural Network was created using Conv2D layers, MaxPooling2D layer, BatchNormalization, Dropout layers, and Fully Connected layers.

MobileVNet_v2: This is a convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. The intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity. As a whole, the architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers [1].

Resnet50: Resnet50 is a residual network with 50 layers. Residual networks use skip connections to add the output from an earlier layer to a later layer. This architecture enables the network to mitigate the vanishing gradient problem. The Resnet model won the 2015 ImageNet challenge. It has a top-5 error rate of 7.02 % and the number of parameters reaches 25.6M [2]. It outputs a 2048 dimension feature vector.

Xception: Xception was proposed by the creator of the Keras library. It is an extension of the Inception network which replaces the standard Inception modules with depthwise separable convolutions. It reached 5.5 % top-5 error rate on the ImageNet Dataset [3]. It outputs a 2048 dimension feature vector. And two classifiers were compared.

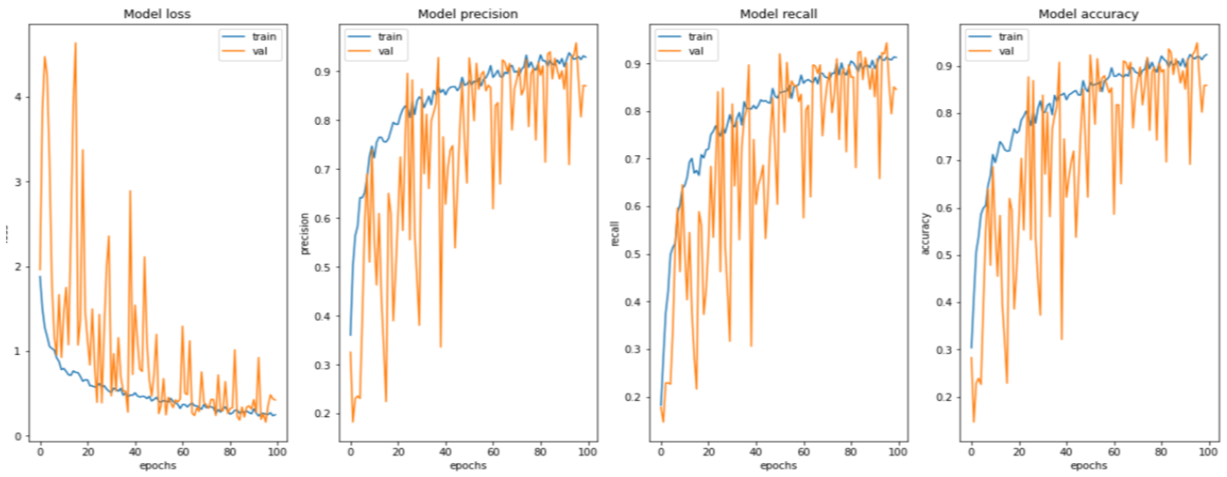
DenseNet: DenseNet is a type of convolutional neural network that utilises dense convolutions between layers, through Dense Blocks, where we connect *all layers* (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its feature maps to all subsequent layers[4].

DISCUSSION

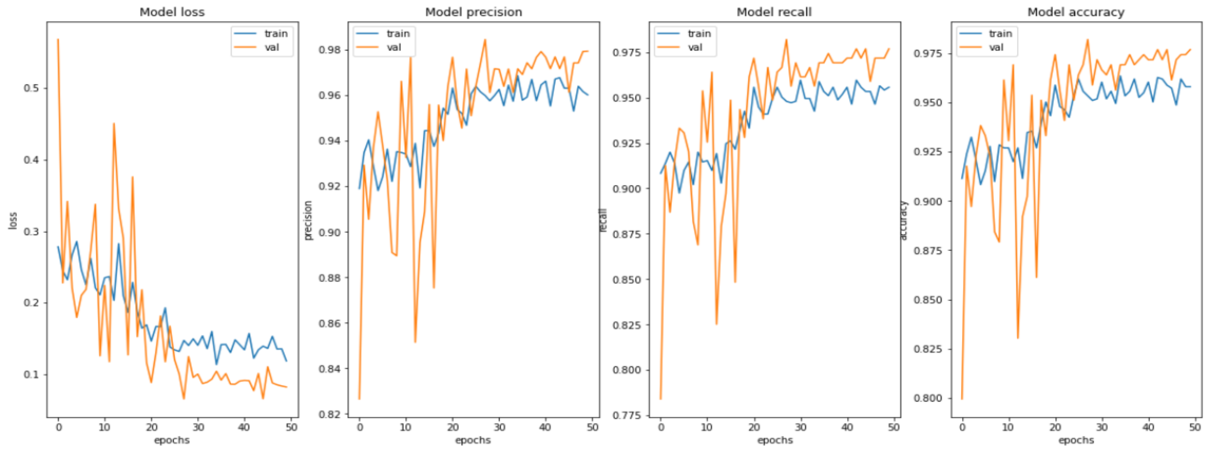
In this project, we have a training set with 1287 images, a 389 validation set, and a test set of 412 images. We split the training set into 80 percent for training and 20 percent for validation. Our experiment is divided into 3 parts. We used these models to extract features from the input image and then used the extracted features as input for the Neural Network classifier. Each of the classifiers was trained for a total of 150 epochs (100 epochs using default Adams[6] optimizer learning rate of 0.01, then fine-tune the models by further training for 50 epochs using Kera “Reduce on Plateau). To prevent overfitting, we first set the function to randomly change the image characteristics during fitting (such as rotation, translation, flipping and zooming) using the Keras Image generator class.

Custom CNN: the model learned the patterns and generalized pretty well on the validations set. we see the cost function of the training set as well as the validation set as shown in the plot below:

Training - 100 epochs

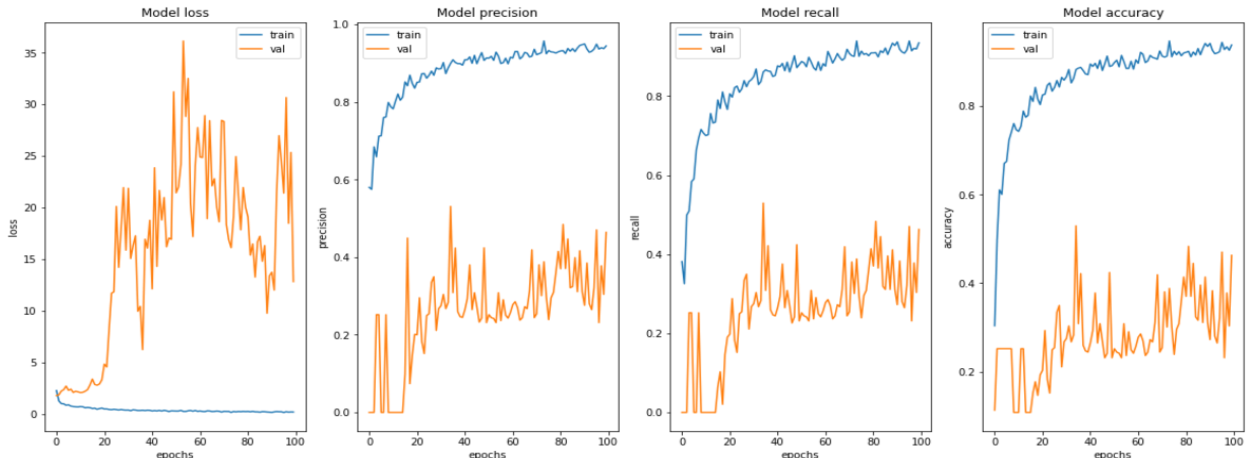


Additional 50 epochs

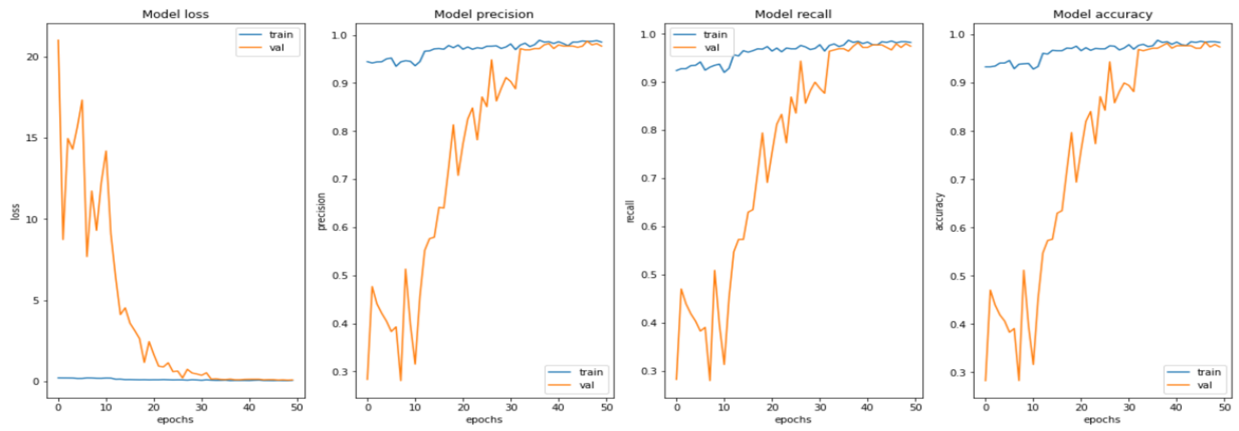


MobileVNet_v2: training with the default adam optimiser, the mobilenet_v2 had a problem generalizing on the validation set (overfitting). But by continuing the training process, the MobileNet_V2 got better at learning patterns and generalizing on the validation set as the learning rate reduces [on plateau], shown on the plots below:

Training - 100 epochs

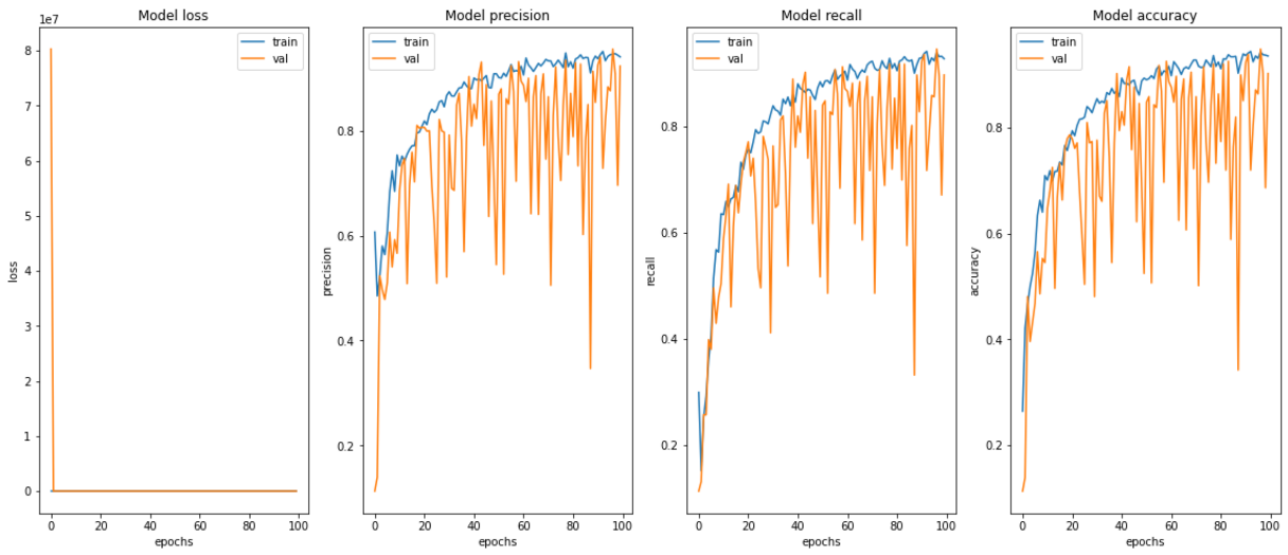


Additional 50 epochs

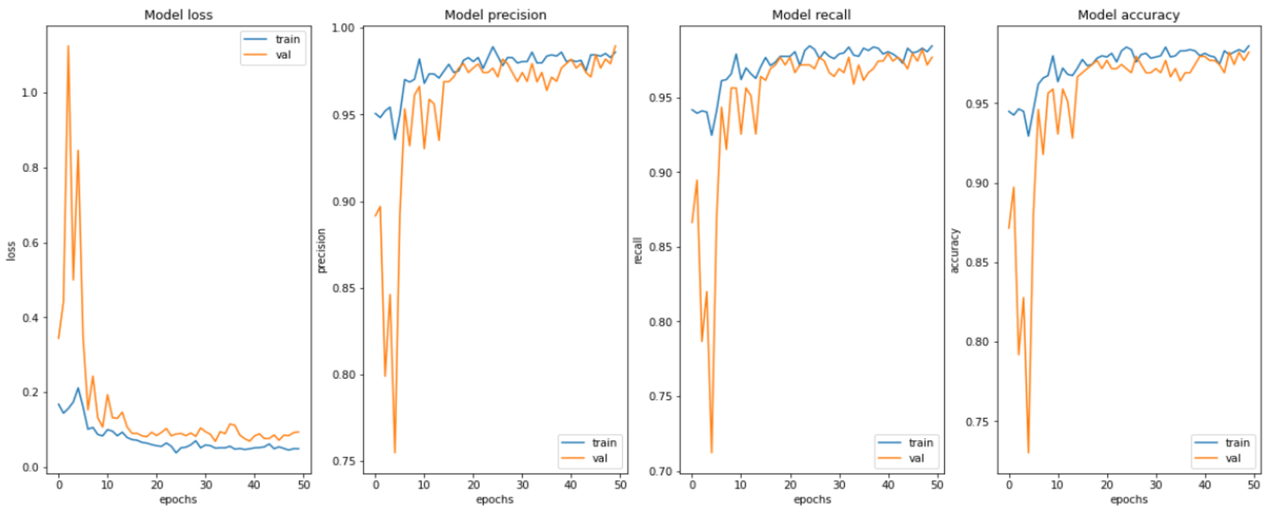


RestNet50: the RestNet50 learned the underlying patterns and trends of the dataset, as well as oscillating but generalizing on the validation set. And as the learning rate reduces on a plateau, we can see that the validation curve oscillates gradually but steadily downwards, as shown:

Training - 100 epochs

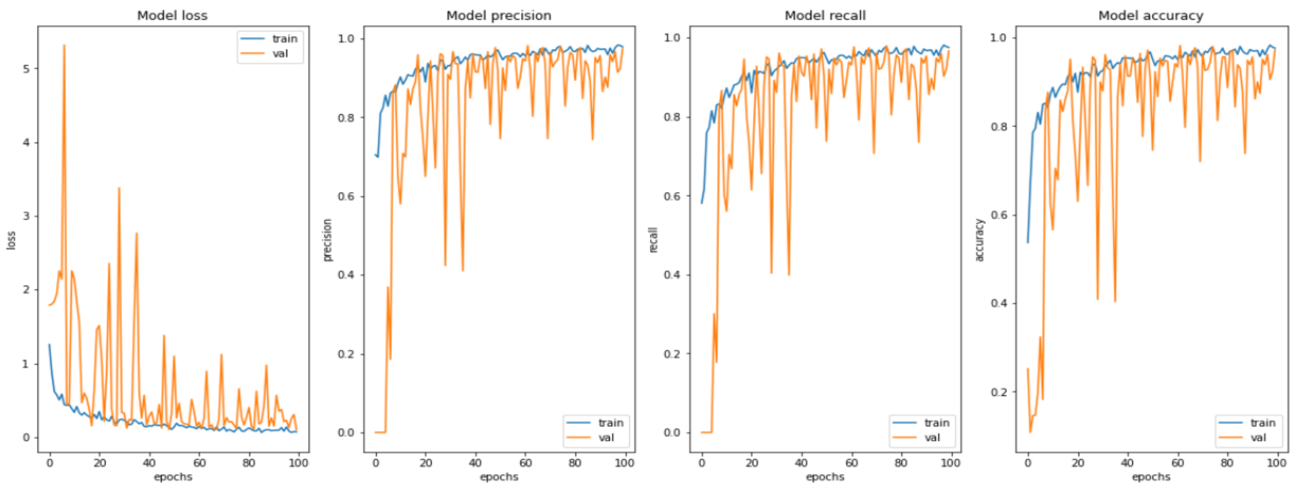


Additional 50 epochs

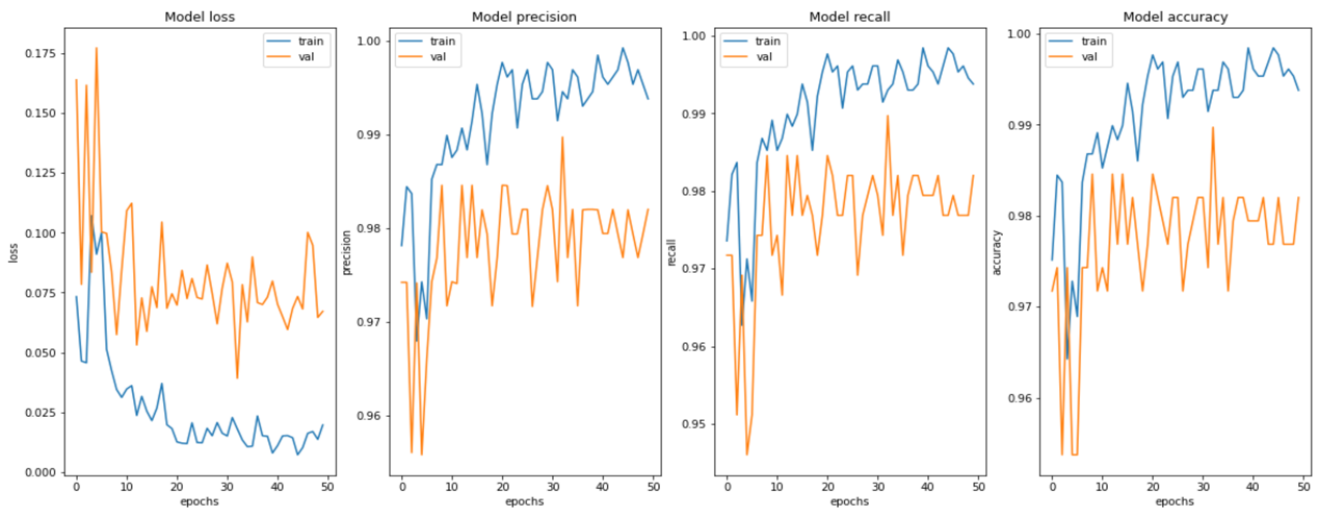


Xception: fitting the Xception model to the training dataset, within few epochs, the loss oscillates downwards. While the validation curve oscillates up and down, we can notice a general reduction in the loss curve, as shown:

Training - 100 epochs

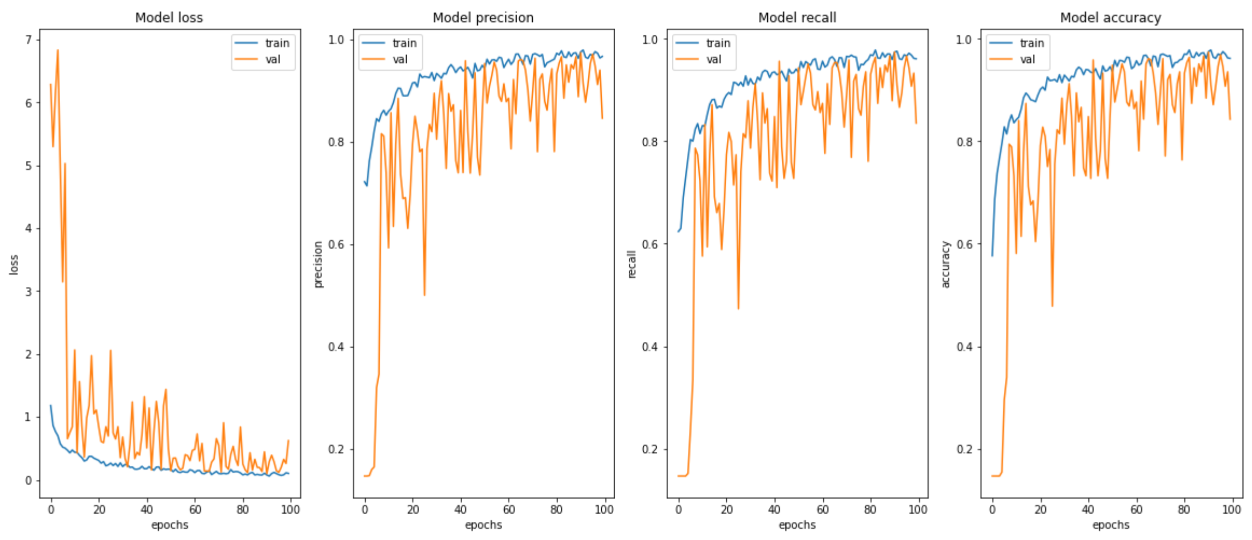


Additional 50 epochs

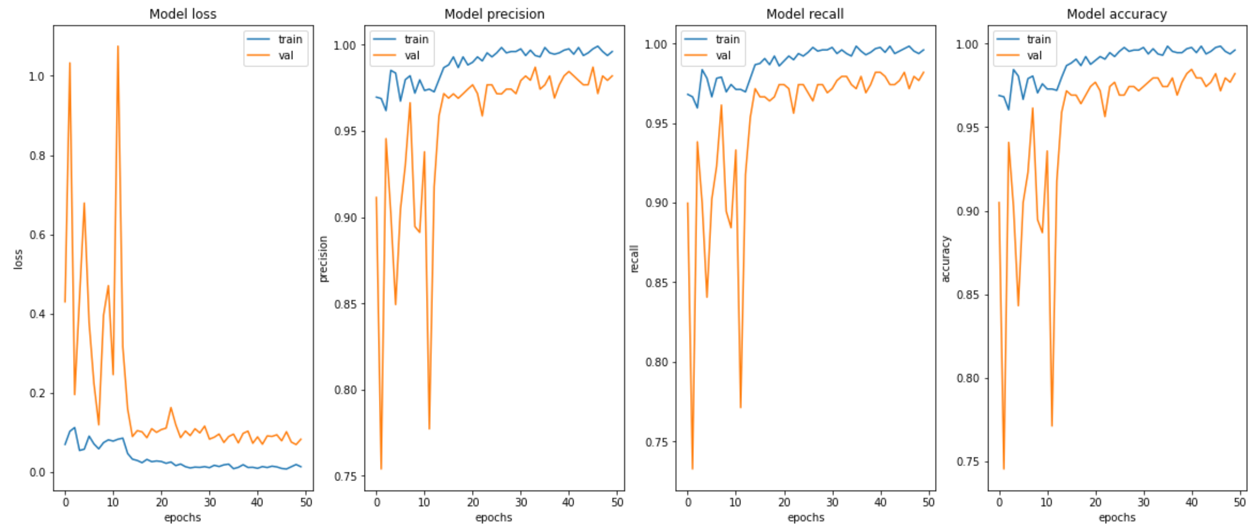


DenseNet: fitting the DenseNet model to the training dataset saw a steady decrease in the cost function on the training and validation loss curve.

Training - 100 epochs

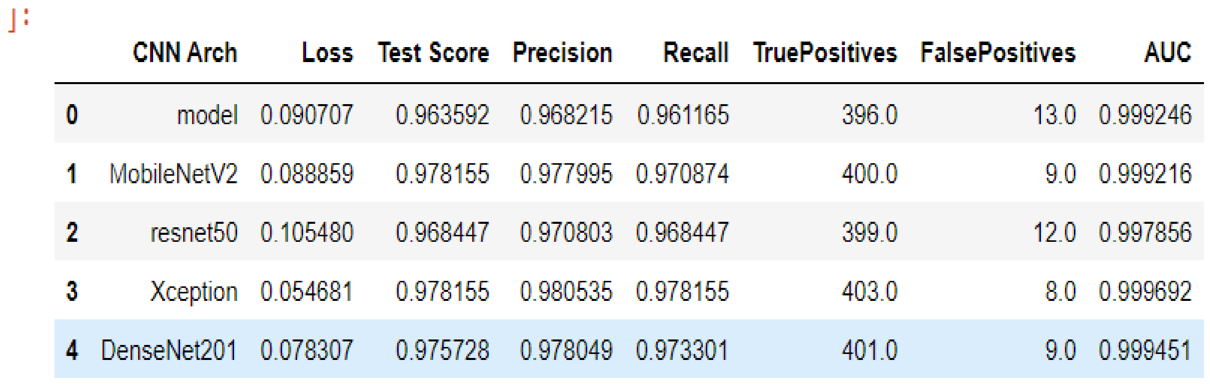


Additional 50 epochs



RESULTS

After successful training of the models on the training set, and evaluating performances on the test set, the result on the test set is summarized below:


]:

	CNN Arch	Loss	Test Score	Precision	Recall	TruePositives	FalsePositives	AUC
0	model	0.090707	0.963592	0.968215	0.961165	396.0	13.0	0.999246
1	MobileNetV2	0.088859	0.978155	0.977995	0.970874	400.0	9.0	0.999216
2	resnet50	0.105480	0.968447	0.970803	0.968447	399.0	12.0	0.997856
3	Xception	0.054681	0.978155	0.980535	0.978155	403.0	8.0	0.999692
4	DenseNet201	0.078307	0.975728	0.978049	0.973301	401.0	9.0	0.999451

CONCLUSION

In this work, we intended to differentiate weeds seedlings from crop seedlings based on their images. We started with different data preprocessing methods which include image segmentation, data augmentation, and label conversion for a highly usable training set. And successfully built and trained 5 neural network classifiers which have shown to be highly effective in classifying each of the images in the dataset with high accuracy and other relevant metrics used to evaluate performance. Although the result of the classifiers was relatively similar, we saw that the **Xception** model architecture performed slightly better than all other models.

References

[1] Menglong Zhu et al.., “MobileNetV2: Inverted Residuals and Linear bottleneck,” arXiv:1801.04381 (2018)

[2] Karen Simonyan and Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,”
arXiv preprint arXiv:1409.1556 (2014).

[3] Kaiming He et al., “Deep Residual Learning for Image Recognition,” arXiv preprint arXiv:1512:03385 (2015).

[4] Yadav, P., Menon, N., Ravi, V., Vishvanathan, S. and Pham, T., 2022. EfficientNet convolutional neural networks-based Android malware detection. *Computers &amp; Security*, 115, p.102622.

[5] Volodymyr Mnih et al., “Human-Level Control Through Deep Reinforcement Learning,” Nature 518 (2015): 529–533.

[6] Diederik P. Kingma, Jimmy Ba, “Adam: A Method for Stochastic Optimization,” arXiv:1412.6980 (2015)

[7] François Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” arXiv preprint arXiv: 1610.02357 (2016).