COMP3029 Computer Vision

Module Convenor: Dr Iman Yi Liao

Group 13

Coursework Report

Name	Student ID	OWA
Lim Hong Leong	20081419	hcyhl2
Kevin Chee Shao Yang	20113454	hcykc1
Wong Kah Yan	20128822	hcykw1

Word count: 2048

1. Description of key features of the implementation

Image Segregation

The image segregation is where we pre-process the images, which is completely separable and independent from CNN. After the images have been processed, they will be used for training and validation of the CNN. Since the CNN is trained by the processed images, therefore when we want to use the CNN for classification, the test images must be processed the same way before being classified by the CNN.

The image segregation algorithm takes the original image and identifies the bounding box of each seed. The algorithm will return the exact coordinate along with the height and width of each seed which is known as ‘roi’, region of interest. We used these coordinates to crop the roi from the original image to produce an image of each cropped individual seed. These roi will then be stored into Google Drive. They are named as <original_filename>_<count_roi>.

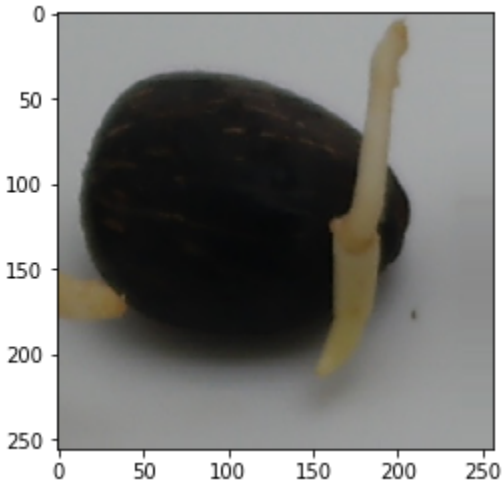


Figure 1.1 Image of segregated seed

For example, from the GoodSeed0 image we will obtain 10 roi. The naming for these roi will be as follow:

GoodSeed0_0

GoodSeed0_1

GoodSeed0_2

…

GoodSeed0_9

We used this naming convention in our trainingdata.csv and validate.csv. The total number of roi is about 1750. The total original images used for training and validation is

110(GoodSeed) + 105(BadSeed) - 20 - 20(reserved for test data)
= 175

Assuming each original image contains 10 seeds, the total individual seed images are..

175 * 10 = 1750

Therefore, we know that the algorithm was able to create a proper amount of roi from the original images. This is a crucial step before training the CNN with these images to ensure the CNN was trained with sufficient amounts of training data. With the roi stored into Google Drive, we do not need to run the image segregation algorithm again when we want to train the CNN.

The **strength** of this algorithm is that it can be used independently as object detection. We can adjust the padding to be added on the bounding box created by the algorithm. This allows the cropped image to include details that could not be picked up by the algorithm. The parameters can be fine tuned for different sets of images. With the images we were given, we were able to achieve zero noise and 99% accuracy on identifying and cropping each individual seed. Since these roi are cropped images, we

can modify them using any image processing method. In our case, we have made the roi to appear red or green to indicate the prediction of the CNN.

The **weakness** of this algorithm is it is solely for object detection and not object identification. Although it can filter out some noise, it is impossible to filter bigger foreign objects.

Additionally, our image segregation method is likely only able to work with images similar to the ones provided in the dataset, that is clearly separated seeds with a distinguishable white background. Even then we do still have faulty segregations where 2 seeds are picked up instead of one when the seeds were too close and the algorithm treated them as a single object. However, it happens too rarely (1% of total images) for us to consider a factor that affects the training of the CNN.

Loading the data

Because there are over 1000 images of individual seeds, storing the images directly into a variable will require more RAM than is provided by Google Colab. Therefore, our solution is to use an ImageDataGenerator and its 'flow_from_dataframe' function to read the filenames and labels from a csv file and then mapping these to the images from our file directory. This allows the model to read images from our directory instead of loading all the images into the RAM.

The advantage of using these functions would allow less powerful computers to be able to run our code. By implementing this feature, it was possible for us to train the CNN with the amount of training data we have. Without it, we would need to subscribe to Colab Pro for the extra ram allowance it provides.

The ImageDataGenerator also allows us to apply augmentations to our images. In our case, we have applied horizontal flipping and width and height shifts of 0.1 units. This allows us to artificially increase the size of our dataset to avoid overfitting.

CNN

Our CNN uses a Sequential model since the layout is linear, where each layer has only one input and one output.

We have 3 convolutional layers, with the number of filters starting at 32, then 64 and 128. As our image is large (256x256x3) we use a 5x5 kernel size for each layer. We add dropout after each convolutional layer to act as noise and encourage the network to generalise, layer 1 and layer 2 have dropouts of 40%, while layer 3 has a dropout of 30%. We also added max pooling layers after each convolutional layer with size 2x2.

After the convolutional layers, we flatten the outputs and input them into a dense neural network layer of 128 units. We then add another 20% dropout on this output and feed it into another network layer with 128 units and the output is fed into our final output layer which is a dense neural network of 1 unit as our output is a binary class. We used ‘relu’ as our activation function for all layers except the final one as it avoids the gradient vanishing problem, and our final layer uses ‘sigmoid’ as it’s output range is (0, 1) which can represent the probability of our binary class.

We used the Adam optimiser as it is generally considered the best. We have tuned the learning rate to 0.0001 (default is 0.001) for the best results to our model. We have also added weights to our classes since our training dataset actually has more GoodSeeds (716) compared to BadSeeds (680). We apply a weight of 716/680 to our BadSeed class so that it will have equal weighting to GoodSeed.

e.g,

GoodSeed = 716*1 = 716

BadSeed = ~~680~~*(716/~~680~~) = 716

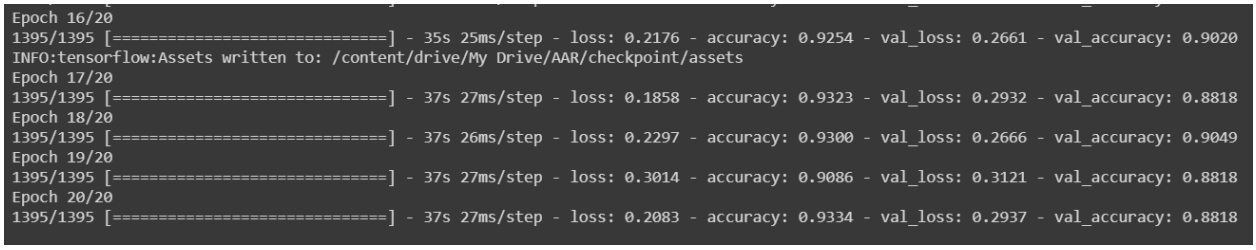
Training

We added a callback to our model.fit function to monitor the ‘val_loss’ parameter for each epoch during training. Everytime ‘val_loss’ reaches the lowest recorded score, a model checkpoint will be saved. This model would then be used for prediction on test data.

Here again, we faced an issue of limited RAM provided by Colab. When we wanted to train for more epochs, we had insufficient RAM leading to Colab terminating all processes and removing all progress. Colab also monitors the running time and throws CAPTCHA after a certain amount of time. That limits the amount of time the model can be trained.

However, with limited training time and epochs, the CNN was able to obtain a moderate result. On average, the model is able to achieve 95% accuracy on test data with around 14 minutes of training time.

We believe that with more training the result would improve as the model does not show signs of overfitting or underfitting, such as shown in the screenshot below. Note the epoch where the checkpoint was saved as it had the lowest recorded ‘val_loss’. The subsequent epochs seem worse due to the fluctuations during training.



2. Quantitative evaluation of the results against the ground truth

We decided to compute the confusion matrix values as a performance measurement since it is very suitable for a binary classification. The confusion matrix counts 4 different outcomes which are True Positive, True Negative, False Positive and False Negative. The outputs are defined as following:

TP – pixels that belong to good seeds and were correctly classified
FN – pixels that belong to good seeds and were incorrectly classified
TN – pixels that belong to bad seeds and were correctly classified
FP – pixels that belong to bad seeds and were incorrectly classified

The following are the values obtained for our trained model:

```
TP:  191
FN:  9
TN:  188
FP:  12
```

Using the values above, we can calculate the **accuracy and precision** with the formulas given below.

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

Precision = TP/(TP + FP)

For our results, the accuracy and precision value obtained is 0.9475 and 0.9409 respectively. Due to our accuracy value being greater than the precision value, the results suggest that the accuracy of our model is slightly better at classifying the bad seed than the good seed. A low precision score indicates that we are mistakenly

classifying more good seeds as bad seeds. Since our precision is at 95%, we are correctly predicting good seeds and having a small number of good seeds classified wrongly.

Another metric that was used was the **IoU (Intersection over Union)** which penalises the incorrect classification of pixels as lesions (FP) and background (FN). IoU measures the amount of overlap between the predicted and ground truth bounding box. The range of IoU is 0 to 1, where 0 indicates that there is no overlap between the boxes and 1 indicates that the union is the same result as the overlap.

IoU = (Lesion + Background)/2
where Lesion = TP/(TP+FN+FP) and Background = TN/(TN+FN+FP)

After calculating the IoU, the value obtained was 0.9002. Since we want the predicted bounding boxes to heavily overlap with ground truth bounding boxes, the higher the score is the better the classification performance.

We also computed the **ROC curve** and the **Area Under the ROC curve(AUC)**. The ROC curve plots the TP rate with respect to FP rate. The AUC value is a desirable performance measurement as it measures the quality of predictions irrespective to the classification group chosen.

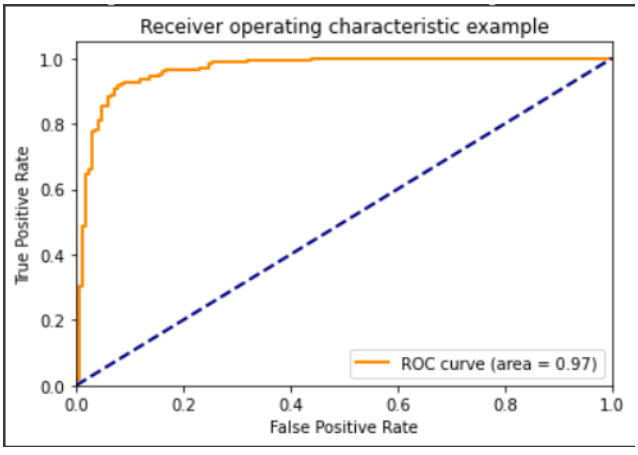


Figure 2.1 Plotting graph of ROC curve

Overall, based on the results of the accuracy and precision, we believe that the model has a very accurate classification for the seeds. Based on the graph of the ROC curve, the steep learning curve shows that the model is very accurate at the beginning.

3. Qualitative evaluation of the results obtained

To evaluate qualitative analysis, we can probe the network's visual knowledge, such as analysing the feature activations from an image. In our case, the feature activations of these images were the size of the seed and the colour and length of the stems.

For our dataset, we used the **labels 1 for good seeds** and **0 for bad seeds** Since our classification problem only has 2 labels, the outcomes of the classification has to be more centric to good or bad seed. Therefore, there is more ambiguity about the intended focus on the seeds in the image.

Based on the resulting images of seeds with bounding boxes, we noticed some of the seedling buds are too long that they exceed the bounding box. This was an important factor when deciding the padding of the bounding boxes.

As we can see in the images below, quite a few of the bounding boxes did not fully capture the stem of the seed. This implies that there could have been a loss of information during testing/training (e.g, color and length of the stem). In this process, the true positive pixels can be lost, resulting in smooth contours in the edge detection and ideally we want the network to have more detailed contours.

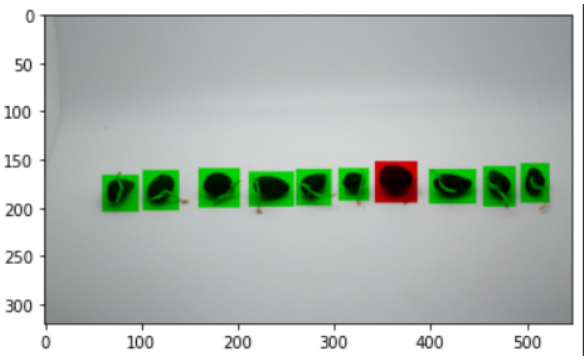


Figure 3.1 Example of good seed image

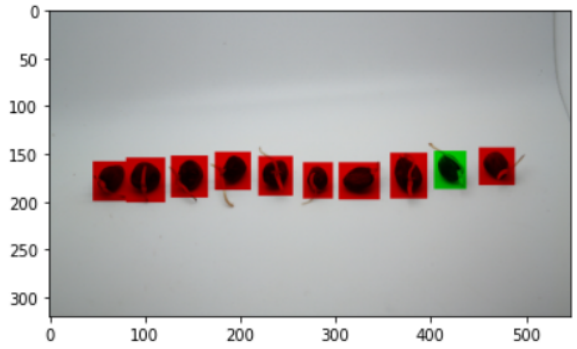


Figure 3.2 Example of bad seed image

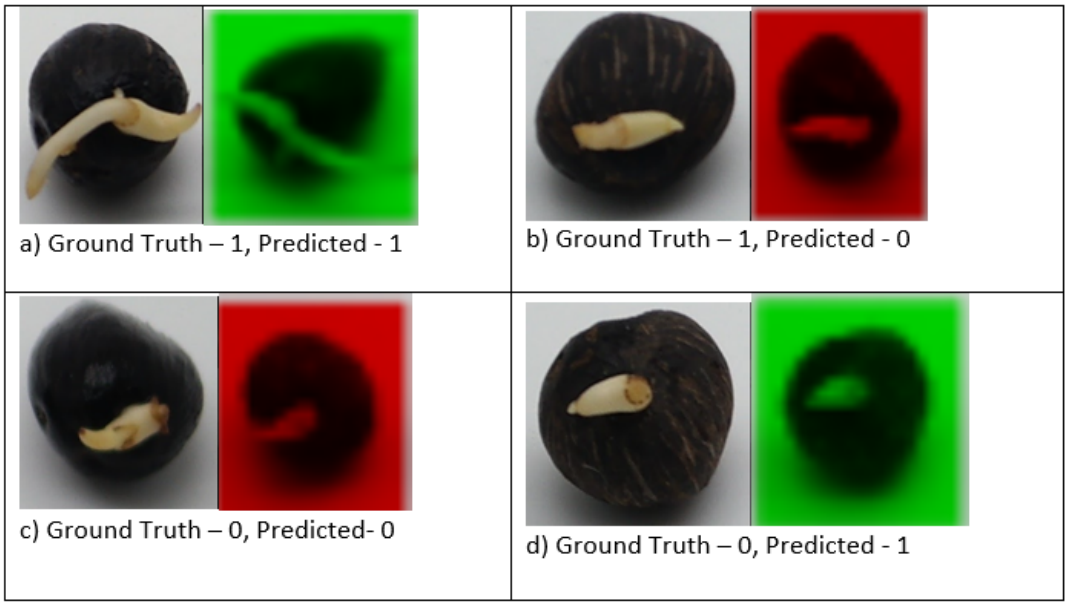


Figure 3.3 Sample images with predicted and ground truth values

The **strength** of our model is we used supervised pre-training to create a larger network of ground truth images. We can also deduce that the model has good time complexity because it is able to achieve a high accuracy within 14 minutes. Although this might be subjected to overfitting, we didn't notice any big changes in the validation loss during training. The primary **weakness** in our object detection algorithm is that the seeds images in the training data are taken with minimal noise and background. With the sample images above, we noticed a lot of the seeds had similar structures in the stems that make the model harder to classify. If we were to detect seeds with many other objects in the background, then the accuracy might not be as good. We believe that testing the model with other seed datasets can prove the performance of our model.

References

1. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, University of Toronto, NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, (2012)

2. Costa, M.G.F., Campos, J.P.M., de Aquino e Aquino, G. *et al.* Evaluating the performance of convolutional neural networks with direct acyclic graph architectures in automatic segmentation of breast lesion in US images. *BMC Med Imaging* 19, 85 (2019). https://doi.org/10.1186/s12880-019-0389-2

3. Zhuwei Qin, Fuxun Yu, Chenchen Liu, Xiang Chen, How Convolutional Neural Networks See The World - A Survey of Convolutional Neural Network Visualization Methods, Mathematical Foundations of Computing (2018)

4. G.S.Jayalakshmi, V.Sathiesh Kumar, Performance analysis of Convolutional Neural Network (CNN) based Cancerous Skin Lesion Detection System, Second International Conference on Computational Intelligence in Data Science (ICCIDS-2019)