2025

Deep Learning & CNN

Yingrong Zhang
University of Technology Sydney
30/05/2025

42028: Deep Learning and Convolutional Neural Network

Animal Species Classification

-GoogLeNet V3

Cabbage Growth Stage Detection

-Faster-RCNN

-Yolov8n

Student Name: Yingrong Zhang

Student ID: 25428842

1. Introduction:

In this report, two image classification models and two object detection models are implemented.

For Image classification, GoogleNetV3 is implemented as a baseline model, followed by a customised inception model (Mini GoogleNet).

In the experiment, four different customised mini GoogleNet Inception models are tested, the best result model is chosen to be shown in the report.

For Object Detection, the Faster R CNN model with a backbone of MobileNetV3 and the YOLOv8 model are implemented.

In the experiment, Faster R CNN with a Backbone of MobileNetV3 and ResNet50 is tested. For the Yolo model, YOLOv5 and YOLOv8 are tested. In the end, Faster R CNN with a backbone as MobileNetV3 and Yolov8n was chosen for the final result.

In this report, a dataset for image classification regarding animal species and object detection for leaves will be introduced. Afterwards, the proposed model architecture will be introduced with two models for object classification and two models for object detection, followed by a detailed experimental result analysis. Ultimately, a conclusion will walk through the overall image classification and object detection.

2. Dataset:

Image Classification:

Dataset: There are 3241 images, separated into 20 folders. The images contain all different animals. Some images have only one animal, some have more than one animal, the same species or different species. The images have different backgrounds. Each folder contains one type of animal species, and the folder name is the animal species name. The dataset is quite balanced; each folder contains images between 110 and 150.

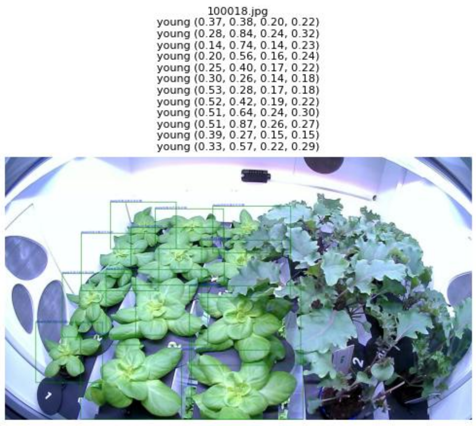
Since the image numbers are not big for deep learning and CNN, when the dataset is split, 80% of the data is in training, 10% in validation, and 10% in testing. In this case, a decent amount of data will be trained after data augmentation.

Object Detection:

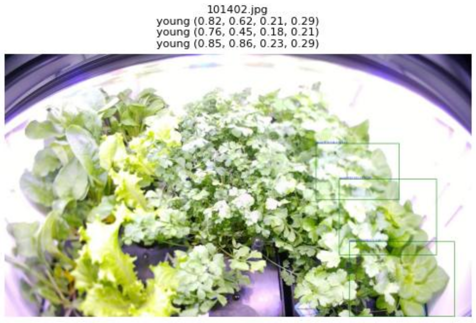
Dataset: Dataset is first separated into three different type of dataset which are Coco, pascal and Yolo. In each type in the dataset, there are 1509 pictures and 1509 labels in total. Images data and labels data are separated into different folders. The dataset already separated into training images 1057, validation images 226, and test images 226.

The Pascal and Yolo datasets' folders contain the separate image folder and the label folder. Pascal's label folder contains JPG and HTML files for each image's labelling information, and the label folder for the Yolo Dataset contains a TXT file for each label's labelling information.

The Coco dataset has a different folder structure. It does not separate the images and labels folders. Instead, one folder, the train folder, contains all images for training, together with a JSON file for all the training annotation information. The same structure is applied to the valid and test folders in the Coco dataset.



```
100018.jpg
young (0.37, 0.38, 0.20, 0.22)
young (0.28, 0.84, 0.24, 0.32)
young (0.14, 0.74, 0.14, 0.23)
young (0.20, 0.56, 0.16, 0.24)
young (0.25, 0.40, 0.17, 0.22)
young (0.30, 0.26, 0.14, 0.18)
young (0.53, 0.28, 0.17, 0.18)
young (0.52, 0.42, 0.19, 0.22)
young (0.51, 0.64, 0.24, 0.30)
young (0.51, 0.87, 0.26, 0.27)
young (0.39, 0.27, 0.15, 0.15)
young (0.33, 0.57, 0.22, 0.29)
```

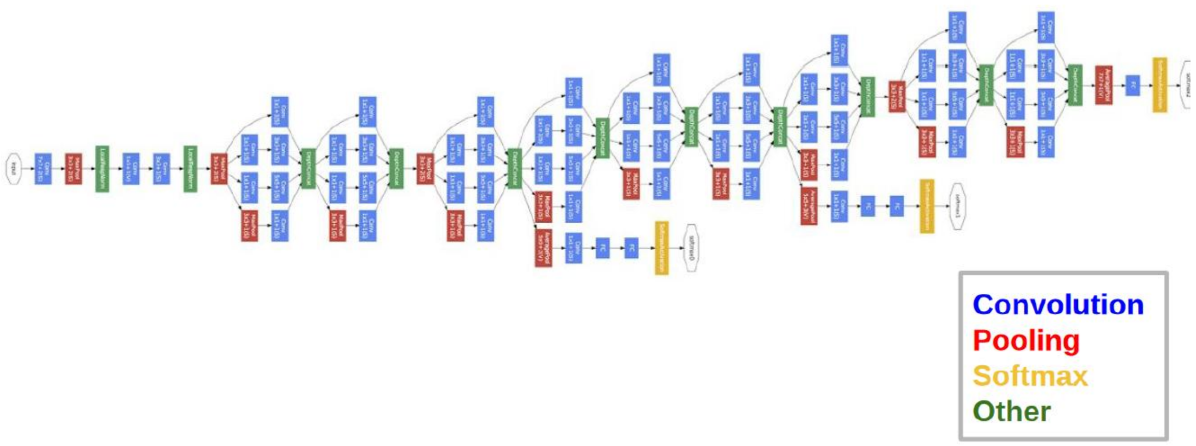


```
101402.jpg
young (0.82, 0.62, 0.21, 0.29)
young (0.76, 0.45, 0.18, 0.21)
young (0.85, 0.86, 0.23, 0.29)
```

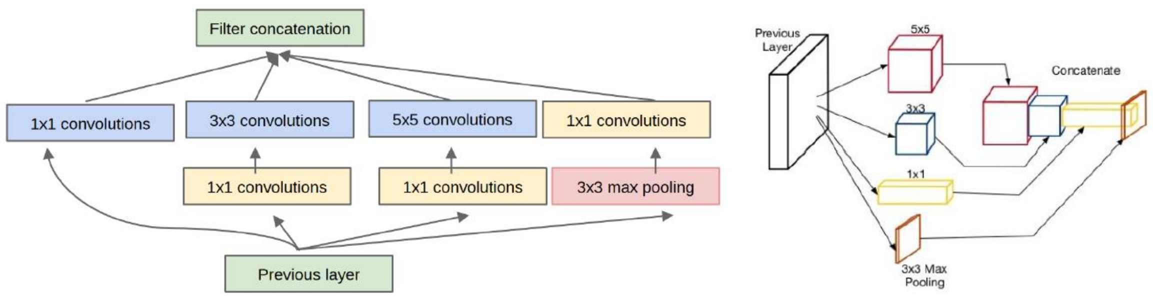
3. Proposed CNN Architecture for Image Classification:

a. Baseline architecture used

GoogLeNet InceptionV3

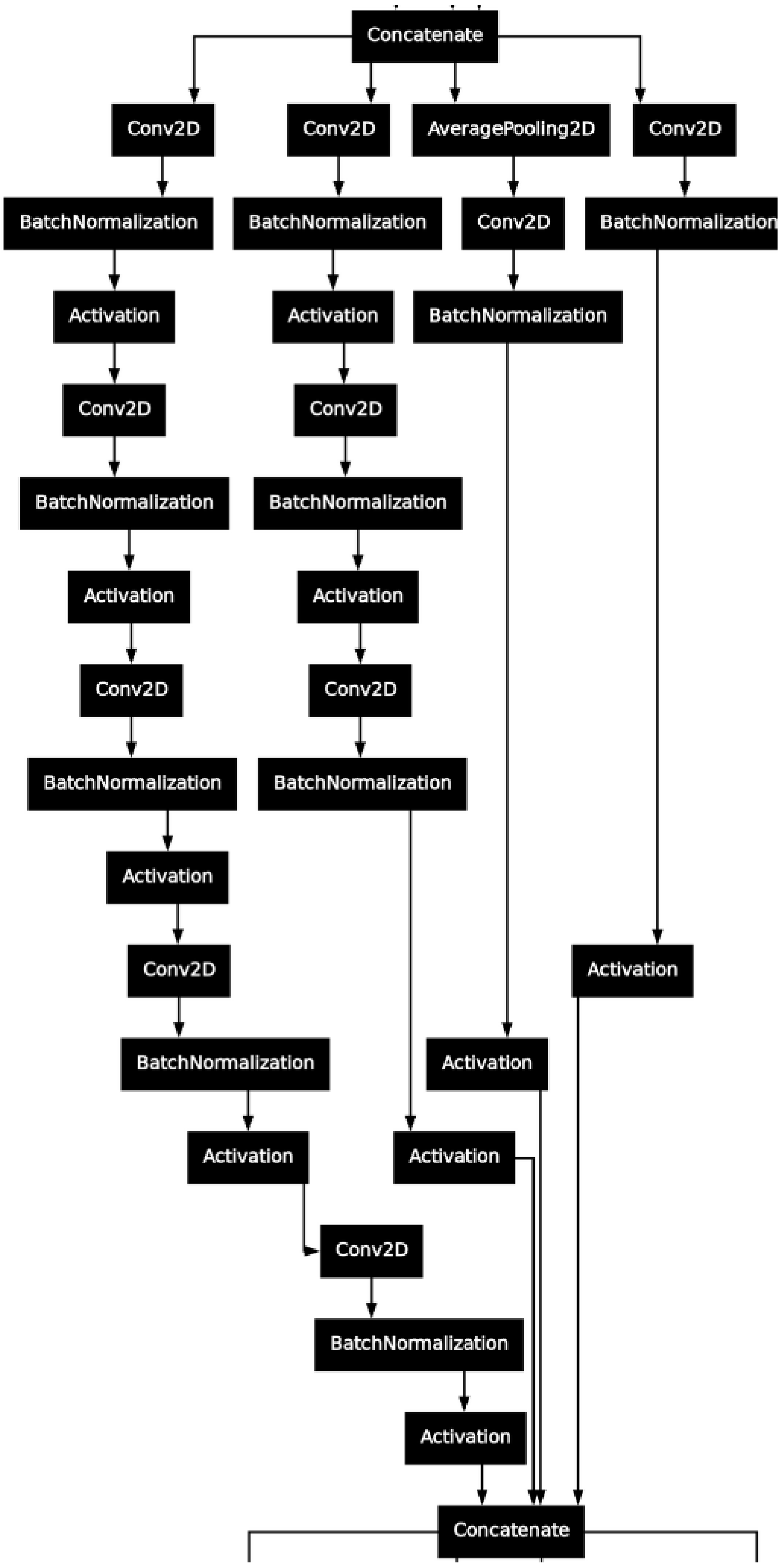


Inception Module



1) Input size (299,299,3)
2) Stem Layer
   In GoogLeNet Inception V3, stem lay is implemented before an initial set of layers, before the main inception model starts. It reduces input size, extracts low-level features and prepares feature maps. There are seven layers in the stem layer.

3) Inception Block
   After the Stem layer, there are 13 inception blocks, each with four branches of convolution layers.


Concatenate
Conv2D
Conv2D
AveragePooling2D
Conv2D
BatchNormalization
BatchNormalization
Conv2D
BatchNormalization
Activation
Activation
BatchNormalization
Conv2D
Conv2D
BatchNormalization
BatchNormalization
Activation
Activation
Conv2D
Conv2D
BatchNormalization
BatchNormalization
Activation
Conv2D
Activation
BatchNormalization
Activation
Activation
Activation
Conv2D
BatchNormalization
Activation
Concatenate

4) Using 13 Inception Blocks in total.
5) Final classification stages:
   After 13 blocks of inception block, it is followed by a global average pooling where the feature map is 1*1. A dropout layer with a 0.5 dropout rate. In the end, with a dense layer (fully connected) , followed by a softmax activation.

6) In total, there are 314 layers in GoogLetNet V3

```
print(f"Total number of layers: {len(model_best.layers)}")
```

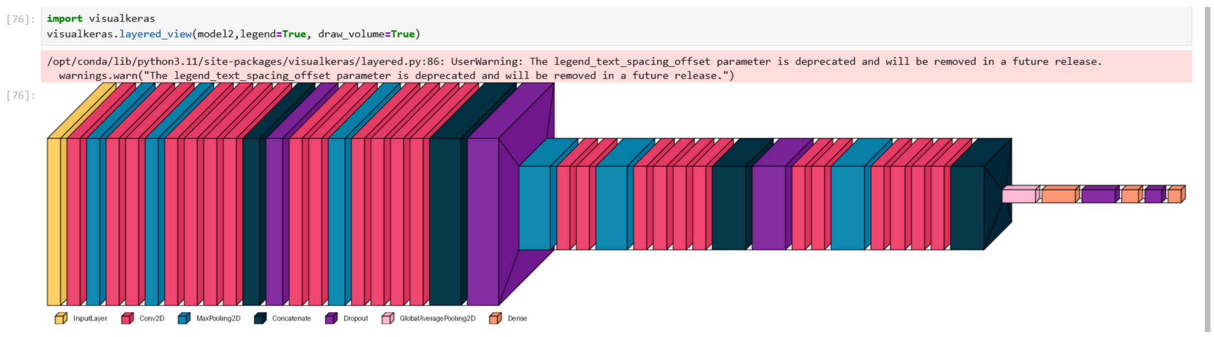
```
Total number of layers: 314
```

7) Activation function
   ReLu: After every Conv2D layer and after BatchNorm
   Softmax: At the end, output the probabilities over classes.

8) Optimizer:
   SGD: Stochastic Gradient Descent with Momentum
   Learning rate: 0.001
   Momentum: 0.9

9) For the whole GoogLetNet V3 architecture, please refer to the Jupiter notebook.

b. Customised architecture
   Customised MiniGoogleNet model
   - Input layer (64,64,3)
   - It started with one convolution layer and max pooling.
   - After maxpooling, the customised MiniGoogleNet is designed to have four inception blocks.
   - After each inception block, designed add a drop out layer.



```
[76]: import visualkeras
      visualkeras.layered_view(model2,legend=True, draw_volume=True)
```

```
/opt/conda/lib/python3.11/site-packages/visualkeras/layered.py:86: UserWarning: The legend_text_spacing_offset parameter is deprecated and will be removed in a future release.
  warnings.warn("The legend_text_spacing_offset parameter is deprecated and will be removed in a future release.")
```

- Because the output parameters are not high
  enough, another two fully connected dense layers with dropout layers are added to the end of the architecture.
- In the end, with the softmax activation function, it is classified into 20 classes.
- In total, there are 45 layers in the customised MiniGoogleNet Inception Model.

- Activation function
  ReLu: After every Conv2D layer and after BatchNorm
  Softmax: At the end, output the probabilities over classes.
- Optimizer:
  SGD: Stochastic Gradient Descent with Momentum
  Learning rate: 0.01
  Momentum: 0.9
- For the whole Customised MiniGoogleNet Inception Model architecture, please refer to the Jupiter notebook.

Customised Mini GoogleNet Inception Model:

GoogleNet V3's design is comprehensive, and the result is relatively high compared with other traditional deep learning models. In this case, the customised model is hard to outperform GoogleNetV3.

However, GoogleNet, due to its complex modelling, is resource-intensive and time-consuming . In this case, a lighter version of the Inception model is designed. Some accuracy is sacrificed compared with GoogleNet but fast training and less computational resources are required.

Customised Mini GoogleNet Architecture designing process:

As the architecture is designed, I tried four different customised inception model architectures and ran the experiment:

1. **Model 1 architecture:**

   With four inception blocks, it does not contain dropout layers after each inception block. Meanwhile, it does not add two more fully connected layers to increase the number of neurons and parameters.

2. **Model 2 architecture:**

   Based on Model 1 architecture, only added two fully connected layer in the end to increase the number of neurons and parameters.

3. **Model 3 architecture:**

   Based on Model 1 architecture, only drop-out layers were added after each inception model, but two fully connected layers were not added in the end.

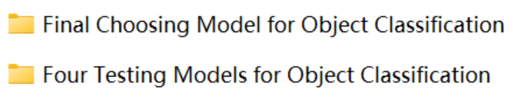
4. **Model 4 architecture:**

   Combine the designs of Model 2 and Model 3, and add both designs to the architecture. Add dropout layers after each inception block, and then at the end of the model architecture, add another two fully connected layers to increase the number of neurons and parameters.

Experiment result and Model choice: Model 4

After running four experiments, the accuracy rate of the model 4 architecture outperformed that of other models. In this case, Model 4 customised Mini GoogleNet Inception blocks were chosen as the final customised model.

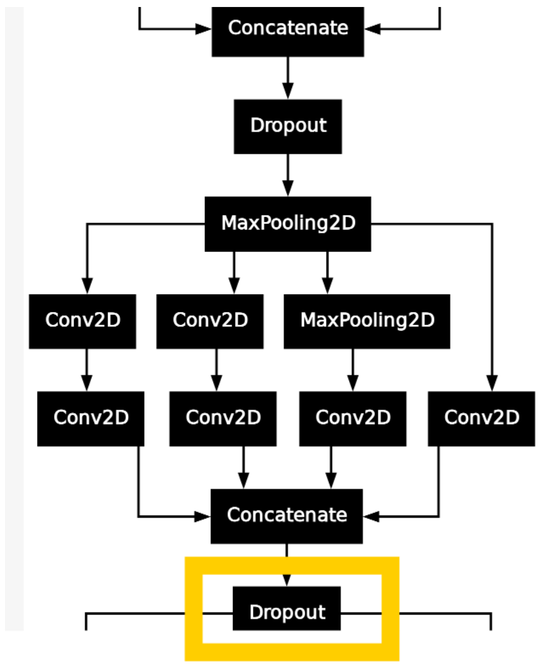
Note: The following report will only mention the Model 4 architecture design and experiment results. For the other models experiment, please refer to the the notebooks in the folder of Four Testing Models for Object Detection.



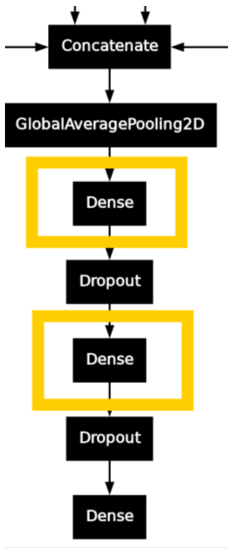
Final Choosing Model for Object Classification

Four Testing Models for Object Classification

a) Stem layers are designed to be simplified. In a stem layer, instead of several convolution layers, they are simplified into one convolutional layer.
b) Four inception blocks are designed in architecture. In the original GoogleNet, there were 13 inception blocks, so I simplified them into four in the customised one.
c) After every Inception block, I added a dropout layer. After each inception layer, I tested the architecture without a dropout layer (Inception_Image_classification_5th model) without dropout in each inception model; the accuracy rate is lower than that with dropout layers. My dataset is relatively small, with fewer than 3000 images, and it is easy to overfit. Furthermore, features learnt early can easily dominate the learning process. Dropout randomly deactivates neurons during training, also ensembles data, making the customised model more robust with a higher accuracy rate.



d) At the end of the customised Inception model, I added two more dense layers before softmax. I tested the architecture without a dense layer (Inception_Image_classification-4th Model), and the result is lower than if I added more dense layers. The reason is that the parameters are not high enough before sending to the softmax for classification. In this case, add two more dense layers allows the model to learn more, and enable deeper decision boundaries.



c. Assumption/intuitions

Customised MiniGoogleNet Inception Model

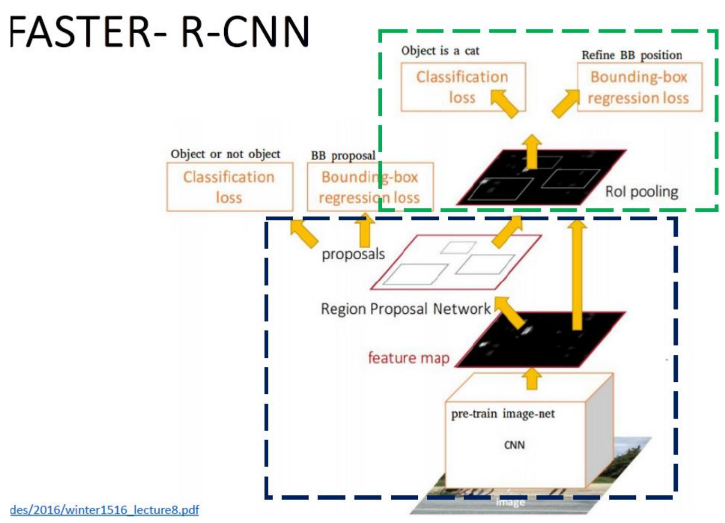
- Limited Dataset size
  Due to the small sample size, the model is prone to overfitting. Dropout can reduce the overfitting and randomise the training data, which might lead to a higher accuracy rate.
- Faster Training and deployment
  Due to lighter architecture with simplified stem layers and fewer inception blocks (four blocks), the model training will be faster than GoogleNetV3.
- More dense layers
  With a lighter convolutional backbone, deeper dense layers were added, compensated, and the parameters were increased before sending them to the softmax layer for classification tasks with 20 classes. It helps allow abstract, high-level reasoning, making the lighter model more robust.
- Bias
  The small dataset may cause the model to overfit. Simplified architecture may cause the model to underfit.And the layer design where features learned by shallow layers may dominate the model prediction.

d. Model Summary

1. GoogneNetV3 Summary – Please refer to GoogleNet_classification Notebook.
2. MiniGoogleNet -Inception Model- Please refer to the Notebook

4. <u>CNN Architecture for Object Detection:</u>

a. Faster RCNN

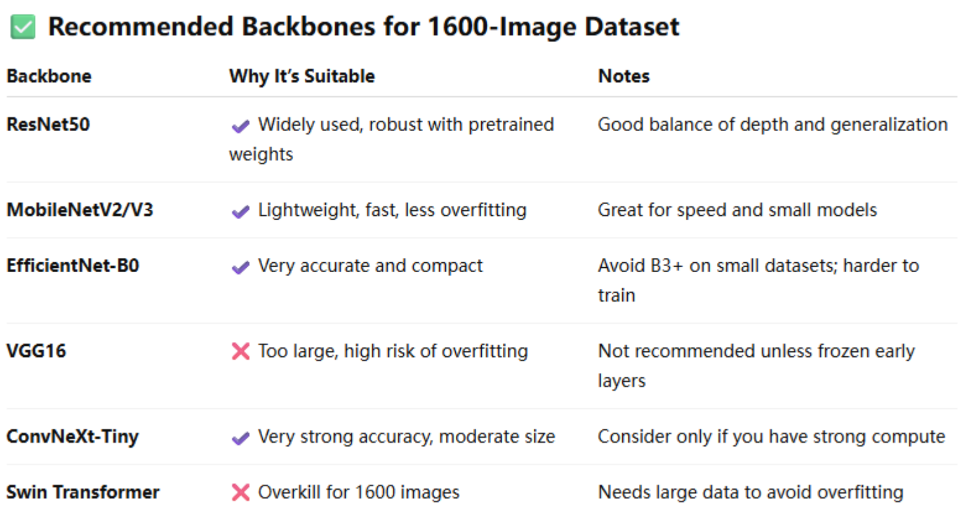


In architecture design, MobileNet V3 is used as a convolution base. The backbone of ResNet50 and MobileNetV3 was tested in the experiments.

I also tested MobileNet50 as a backbone; the speed is extremely slow compared with the backbone of MobileNetV3. During the training process, AWS experienced several issues that required running several times for one experiment.

In comparison, MobileNetV3 is much easier and quicker to implement and train. MobileNetV3 was decided to be implemented due to its high speed, and for small models due to the small dataset.

The following is a comparison of different backbones (generated by Chatgpt).



✅ **Recommended Backbones for 1600-Image Dataset**

Backbone	Why It's Suitable	Notes
ResNet50	✓ Widely used, robust with pretrained weights	Good balance of depth and generalization
MobileNetV2/V3	✓ Lightweight, fast, less overfitting	Great for speed and small models
EfficientNet-B0	✓ Very accurate and compact	Avoid B3+ on small datasets; harder to train
VGG16	✗ Too large, high risk of overfitting	Not recommended unless frozen early layers
ConvNeXt-Tiny	✓ Very strong accuracy, moderate size	Consider only if you have strong compute
Swin Transformer	✗ Overkill for 1600 images	Needs large data to avoid overfitting

For Faster R-CNN, it has two stages of object detection in its framework.
Stage 1: Regional Proposal Network (PRN)- propose candidate object regions (bounding boxes)
Stage2: Fast R-CNN head classifies each region and refines its bounding box

Faster R-CNN Pipeline

Stage1:

1. Input Image
2. Goes into the backbone
   a. ResNet-50/ MobileNetV3 extracts deep feature maps from the input image
   b. Deep features means high-level, abstract representations of the image, such as the whole object concept and relationships between parts.
   c. The feature map is used by both the RPN(region proposal network) and the second stage
3. Region Proposal Network (RPN)
   Goal: Suggest candidate object regions
   It operates on the deep feature maps, which were extracted by the last step, and slides a small network over them. Afterwards, it predicts objectness score(whether it is object or background) and bounding box refinements (4 values)
4. The output RPN is filtered by Non-Max Suppression (NMS) and object scores.
5. ROIAliggn (Region of Interest Pooling)

Stage2:

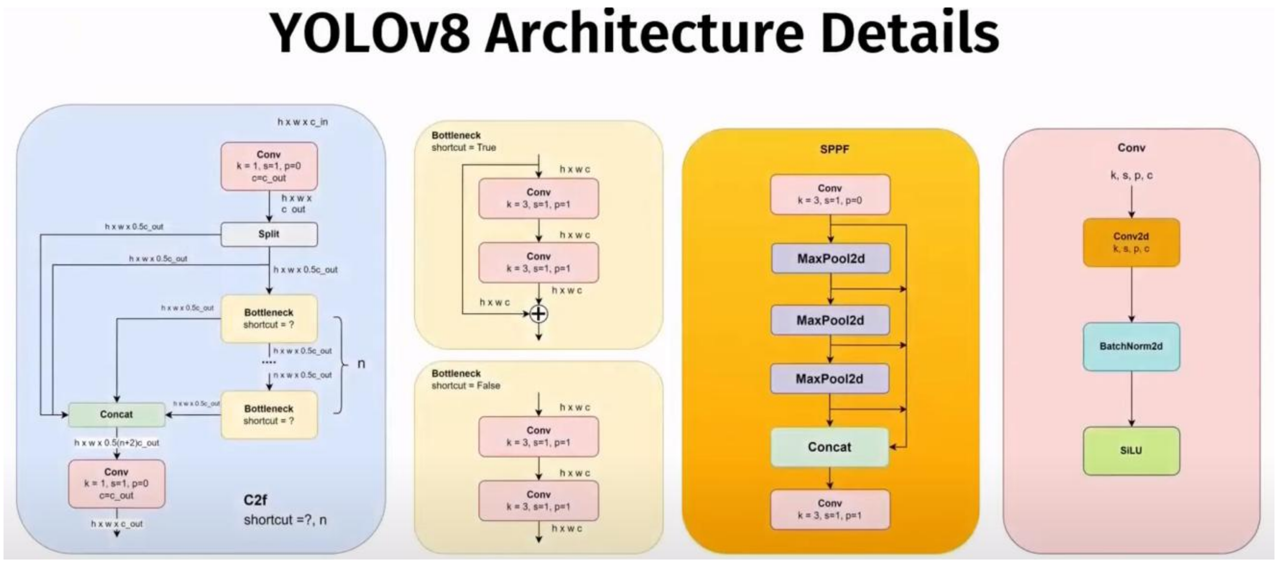
6. Fast R-CNN Head
   For each ROI, predict the class label (multi-class classification)
   Predict four bounding box coordinates
7. Final Predictions
   Outputs
   a. Class probabilities
   b. Bounding box coordinates
   Use NMS(Non-Max Suppression) again to filter overlapping predictions

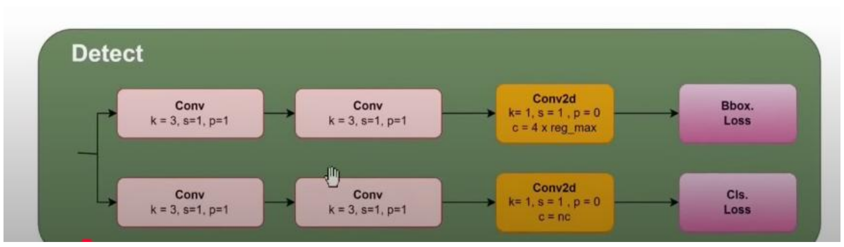
b. YOLO Model for Object Detection

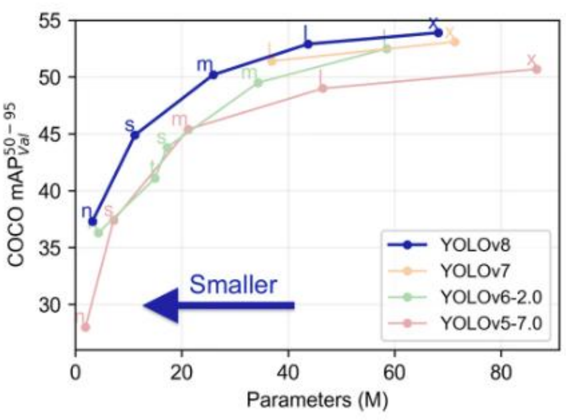
Two different Yolo models are tested: Yolov5 and Yolov8. Yolov8's Model performance is higher, and its implementation is easier than Yolov5's.

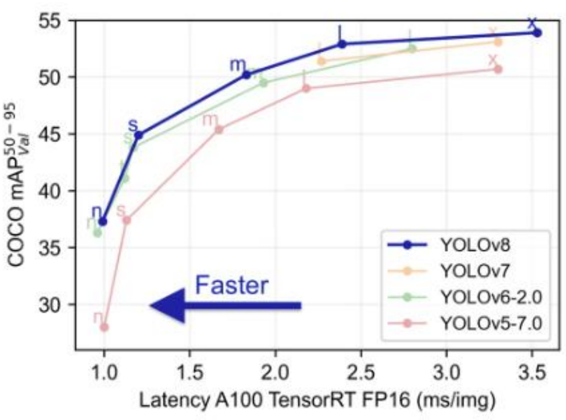
In this case, Yolov8 is decided to be implemented.

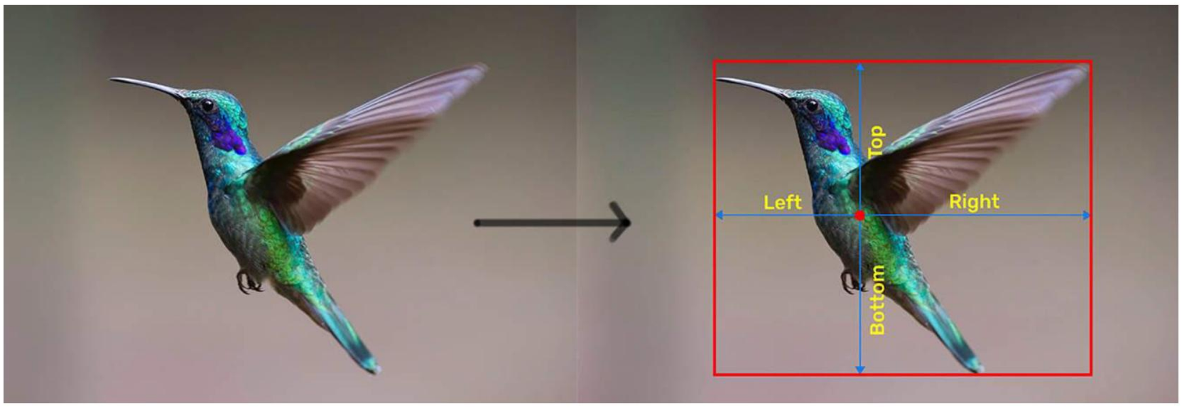
You only look once!




Detect
Conv k = 3, s=1, p=1
Conv2d k= 1, s = 1, p = 0 c = 4 x reg_max
Bbox. Loss
Conv2d k= 1, s = 1, p = 0 c = nc
Cls. Loss







Yolov8 is using centre-based anchor-free detection.

Backbone: Uses a custom CSPDarknet53 backbone, which employs cross-edge partial connections to improve information flow between layers and boost accuracy.

1) Conv Layers
2) BatchNorm
3) SiLu(Swish) activation
4) C2f models
   Note: The C2f model combines high-level semantic features with low-level special information, improving detection accuracy. It works exceptionally well on small objects.

Neck: It managed features from different scales to improve object detection at diverse sizes.It uses a simplified FPN/PANet-style structure with:

1) C2f layers (to improve cross-srage connections)
2) Upsampling and downsampling paths

Head: Yolov8 employs multiple detection modules that predict bounding boxes, object detection scores, and class probabilities for each grid in the feature map. These predictions are then aggregated to produce the final predictions.

1) Outputs each grid cell's final prediction
2) For each grid cell, it predicts directly
   A. Objectness score
   B. Class probabilities
   C. Bounding box offsets(center x, center y, width, height)

Note: During training, anchor boxes or matching are not required. It uses distribution focal loss(DFL) for more precise localisation.

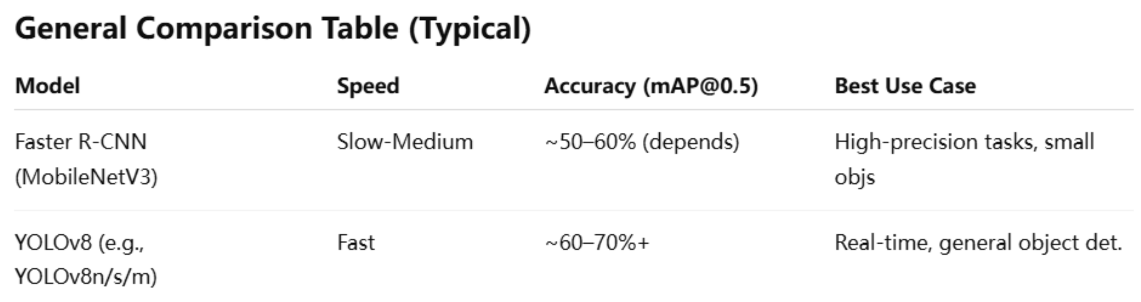
c. Assumption/intuitions

Speed:

Faster R-CNN is a two-stage detection that generates regional proposals and classifies and refines the bounding boxes in this proposed region.
Yolov8 is making a single-stage detection assumption, detecting and predicting in a single step.
In this case, using the Yolov8 model should be way faster than training with Faster R-CNN regarding object detection.

Accuracy:

Faster R-CNN will be more accurate in detecting small or overlapping objects based on the Anchor Box assumption. Yolov8 is a grid-based prediction that uses centre-based Anchor-Free detection and improved NMS (non-max suppression).
I implemented Yolov8n, a nano model, and it works well on small to medium-sized datasets.
In this case, Yolov8 Nano model accuracy on mAP should outperform Faster R-CNN.

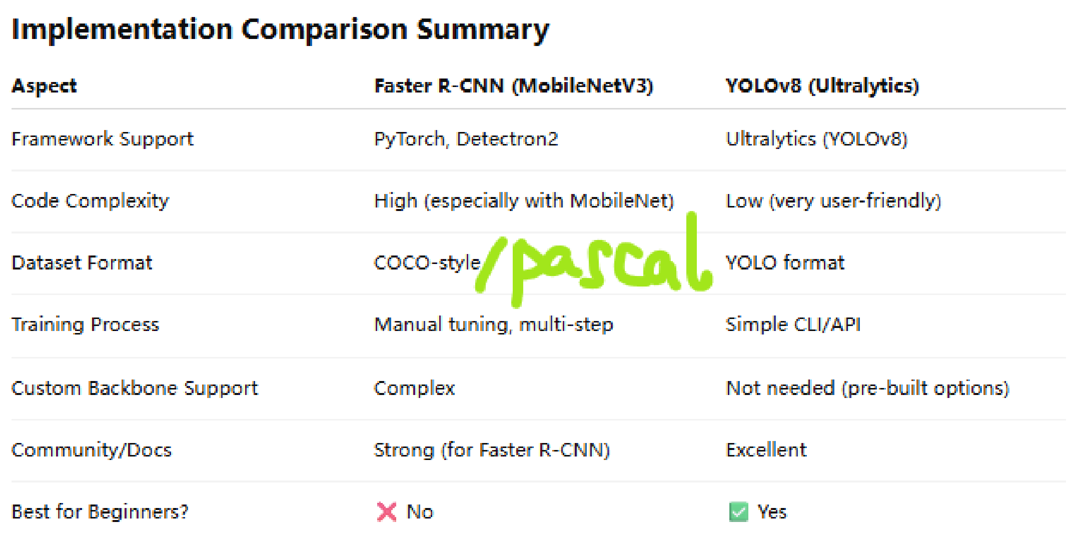


General Comparison Table (Typical)

Model	Speed	Accuracy (mAP@0.5)	Best Use Case
Faster R-CNN (MobileNetV3)	Slow-Medium	~50–60% (depends)	High-precision tasks, small objs
YOLOv8 (e.g., YOLOv8n/s/m)	Fast	~60–70%+	Real-time, general object det.

Ease of Implementation

Faster R-CNN with MobileNet: Moderate to High
Yolov8: Low
Yolov8 is easier to implement than Faster R-CNN. Yolov8, for the hyperparameter, yolov8 could automatically be turned if tune=true. For training, it is a simple CLI/API. All the measurements would be automatically saved.



Implementation Comparison Summary

Aspect	Faster R-CNN (MobileNetV3)	YOLOv8 (Ultralytics)
Framework Support	PyTorch, Detectron2	Ultralytics (YOLOv8)
Code Complexity	High (especially with MobileNet)	Low (very user-friendly)
Dataset Format	COCO-style/pascal	YOLO format
Training Process	Manual tuning, multi-step	Simple CLI/API
Custom Backbone Support	Complex	Not needed (pre-built options)
Community/Docs	Strong (for Faster R-CNN)	Excellent
Best for Beginners?	❌ No	✅ Yes

d. Model Summary
Please refer to the Jupiter notebook

5. Experimental results and discussion:

a. Experimental settings:

i. Image Classification:

A. **GoogLeNet V3 Baseline model**

1. Transfer Learning is used.
   Pre-trained model used: InceptionV3 from keras.applications
2. Top （fully connected） layer removed
   Setting include_top=False allowed me to add my own classification head, which predicts 20 classes.
3. The baseline model did not have frozen layers, which allowed me to train all the layers in it.

```
base_model = keras.applications.InceptionV3(weights='imagenet',
include_top=False)
```

4. Hyperparameter Setting:
   a. Optimizer- SDG
   b. Initial Learning Rate: 0.001
   c. Momentum: 0.9
   d. Learning Rate Decay: drop=0.96 every 8 epochs
   e. Epochs: Maximum 200, until it triggers early stop
   f. Batch size: 20
   g. Loss function: categorical_crossentropy
   h. Metrics: accuracy
5. Data Augmentation
```
Rescale=1./255  # Normalises pixels from [0,255] into [0,1]
rotation_range=40,
 width_shift_range=0.2,
 height_shift_range=0.2,
 shear_range=0.2,
 zoom_range=0.2,
 horizontal_flip=True,
 brightness_range=[0.8, 1.2],
 channel_shift_range=0.2,
 fill_mode='nearest')
```
6. Callbacks Used
   Earlystopping, patience = 20. Stops training if val_loss doesn’t improve for 20 epoches.
   ModelCheckPoint, saves the best model based on val_loss
   LearningRateScheduler, applies exponential decay to the learning rate

B. **Customised MiniGoogleNet - Inception model**

Three different design of MiniGooleNet Inception models has been tested and running experiment result.

A. Model without adding two fully connected layers at the end
B. Model without a dropout layer after each inception block
C. Model with adding two fully connected layers at the end. Meanwhile, every inception block adds a dropout layer in the end of every inception block

   After running three experiments, model C performed better on image classification. In this case, Model C is implemented and analysed in the report.

1. No transfer Learning used
2. Hyper Parameter setting
   a. Optimizer- SDG
   b. Initial Learning Rate: 0.01
   c. Momentum: 0.9
   d. Learning Rate Decay: drop=0.96 every 8 epochs
   e. Epochs: Maximum 500, until it triggers early stop
   f. Batch size: 20
   g. Loss function: categorical_crossentropy
   h. Metrics: accuracy
3. Data Augmentation

```
Rescale=1./255  # Normalises pixels from [0,255] into [0,1]
rotation_range=40,
 width_shift_range=0.2,
 height_shift_range=0.2,
 shear_range=0.2,
 zoom_range=0.2,
 horizontal_flip=True,
 brightness_range=[0.8, 1.2],
 channel_shift_range=0.2,
 fill_mode='nearest')
```

4. Callbacks Used
   Earlystopping, patience = 30. Stops training if val_loss doesn’t improve for 30 epoches.
   ModelCheckPoint, saves the best model based on val_loss
   LearningRateScheduler, applies exponential decay to the learning rate

ii. Object Detection:

Faster R-CNN

Two different Faster R-CNN model architecture was tested:

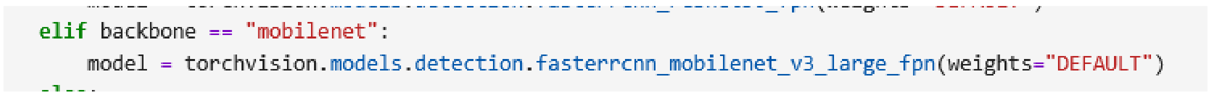
A. Backbone with ResNet 50
B. Backbone with MobileNetV3

For the model, running FasterRNN with a backbone ResNet50 is extremely slow. Due to too running time, AWS always automatically log out half way of the training.

When changed into a backbone like MobileNetV3, the speed is much faster compared with the backbone as MobileNetV3. The accuracy rate of the running experiment is relatively similar.

In this case, I choose MobileNetV3 as the backbone of Faster R-CNN to run the result.

Backbone: MobileNet V3
Transfer Learning: Yes
Pretrained Weights: CoCo pre-trained, head replace



```
elif backbone == "mobilenet":
    model = torchvision.models.detection.fasterrcnn_mobilenet_v3_large_fpn(weights="DEFAULT")
```

Optimizer: SDG
Learning Rate: 0.002
Momentum: 0.9
Weight Decay: 0.0005
Epochs: 300, until it trigger early stop
Batch Size: 2
Early Stop Patience: 10 epochs
Validation Frequency: every epochs
Device: CUDA
Seed: 25428842

Note: Since I use the Pascal dataset in my experiment, and the model loaded the Coco pretrained weight, classes are designed to add '_background_'as :
CLASSES = ['__background__','Ready', 'empty_pod', 'germination','pod','young']

Yolov8n

Two different Yolo models are tested. Yolov5 and Yolov8. Model performance is higher as Yolov8, and also implementation of Yolov8 is easier than yolov5.

In this case, I decided to choose Yolov8 instead of Yolov5.
And Yolov8n is the nano model which suitable for small to medium dataset. In this case, Yolov8n is chosen to be implemented.

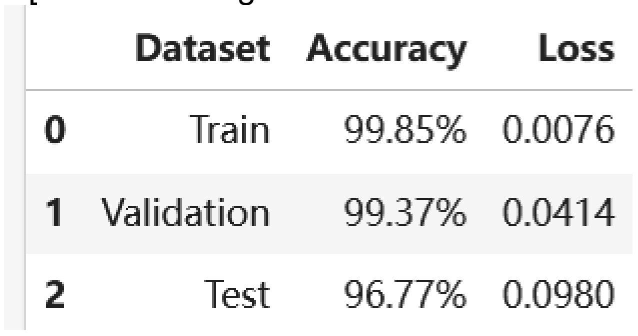
Transfer learning: Yes
Pretrained weight: yolov8n.pt
Pretrained on: Coco dataset
Ultralytics's built-in training Pipleline

Hyperparameter:
Batch: 16
Epochs: 400, until it trigger early stop
Image size: 640
Seed: 25428842

b. Experimental results:

i. Image classification:

1. Performance on baseline/standard architecture
[Baseline GoogLeNetV3 Performance:



	Dataset	Accuracy	Loss
0	Train	99.85%	0.0076
1	Validation	99.37%	0.0414
2	Test	96.77%	0.0980

Customised MiniGooLetNet Inception Model

Four different designs of MiniGooleNet Inception models have been tested.

The baseline GoogleNetV3 model performance is already high, but due to complex model design, the model training is slow. In this case, a lighter version of MiniGoogle Net is designed. With simpler stem layers, fewer inception blocks and fewer fully connected layers in the end. The model will be lighter, so it will train faster compared with the baseline model. However, the accuracy rate will be sacrificed. The accuracy rate of the customised inception model will be lower than that of the baseline model.

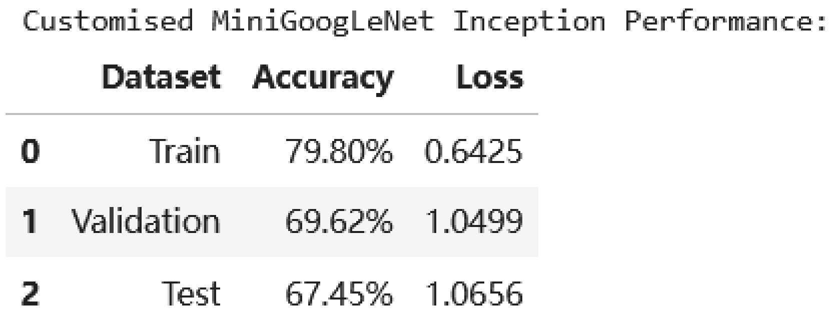
There are three model architectures designed and experiments running.

A. Model without adding two fully connected layers at the end
B. Model without a dropout layer after each inception block
C. Model with adding two fully connected layers at the end. Meanwhile, every inception block adds a dropout layer in the end of every inception block

   After running three experiments, model C performed better on image classification. In this case, Model C is implemented and analysed in the report.

The following result is from Model C, which outperformed Models A and B.

Customised MiniGoogleNet Inception Model

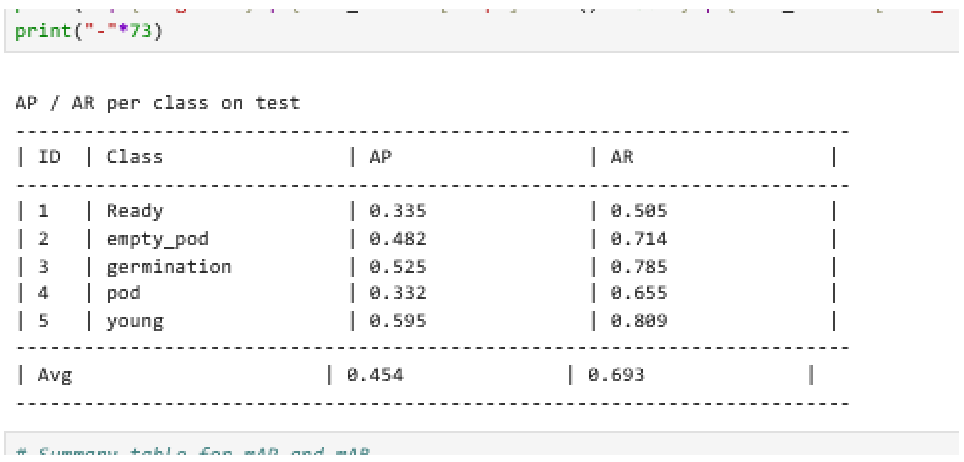


```
Customised MiniGoogLeNet Inception Performance:
```

	Dataset	Accuracy	Loss
0	Train	79.80%	0.6425
1	Validation	69.62%	1.0499
2	Test	67.45%	1.0656

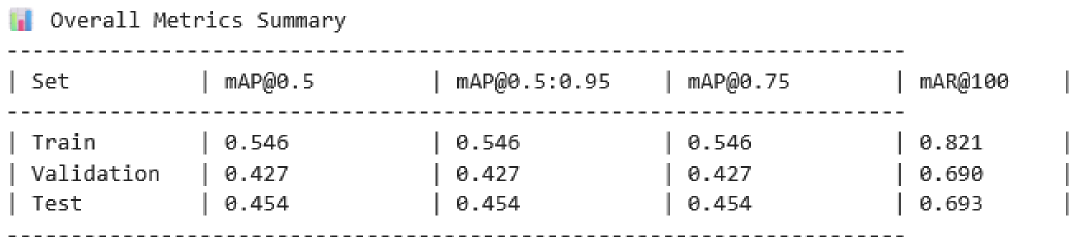
ii. Object Detection:

1. Performance on Faster-RCNN



```python
print("-"*73)
```

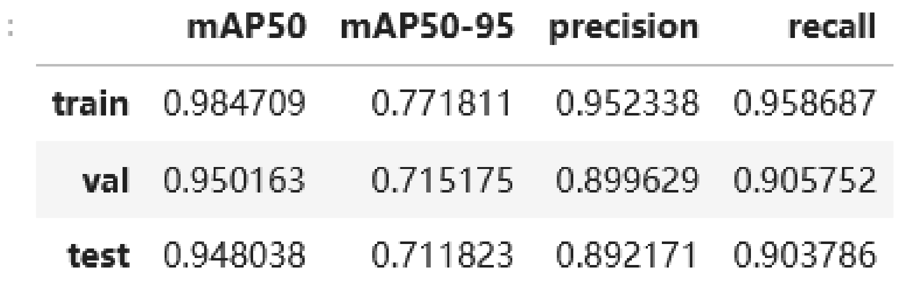
```
AP / AR per class on test
-------------------------------------------------------------------------
| ID  | Class                | AP                   | AR                   |
-------------------------------------------------------------------------
| 1   | Ready                | 0.335                | 0.505                |
| 2   | empty_pod            | 0.482                | 0.714                |
| 3   | germination          | 0.525                | 0.785                |
| 4   | pod                  | 0.332                | 0.655                |
| 5   | young                | 0.595                | 0.809                |
-------------------------------------------------------------------------
| Avg                      | 0.454                | 0.693                |
-------------------------------------------------------------------------
```



```
📊 Overall Metrics Summary
--------------------------------------------------------------------
| Set          | mAP@0.5          | mAP@0.5:0.95     | mAP@0.75         | mAR@100      |
--------------------------------------------------------------------
| Train        | 0.546            | 0.546            | 0.546            | 0.821        |
| Validation   | 0.427            | 0.427            | 0.427            | 0.690        |
| Test         | 0.454            | 0.454            | 0.454            | 0.693        |
--------------------------------------------------------------------
```

2. Performance on SDD/YOLO or any object detector

Yolov8 model Performance

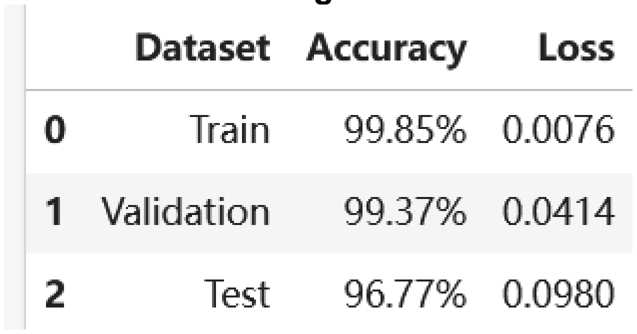


	mAP50	mAP50-95	precision	recall
train	0.984709	0.771811	0.952338	0.958687
val	0.950163	0.715175	0.899629	0.905752
test	0.948038	0.711823	0.892171	0.903786

iii. Discussion:

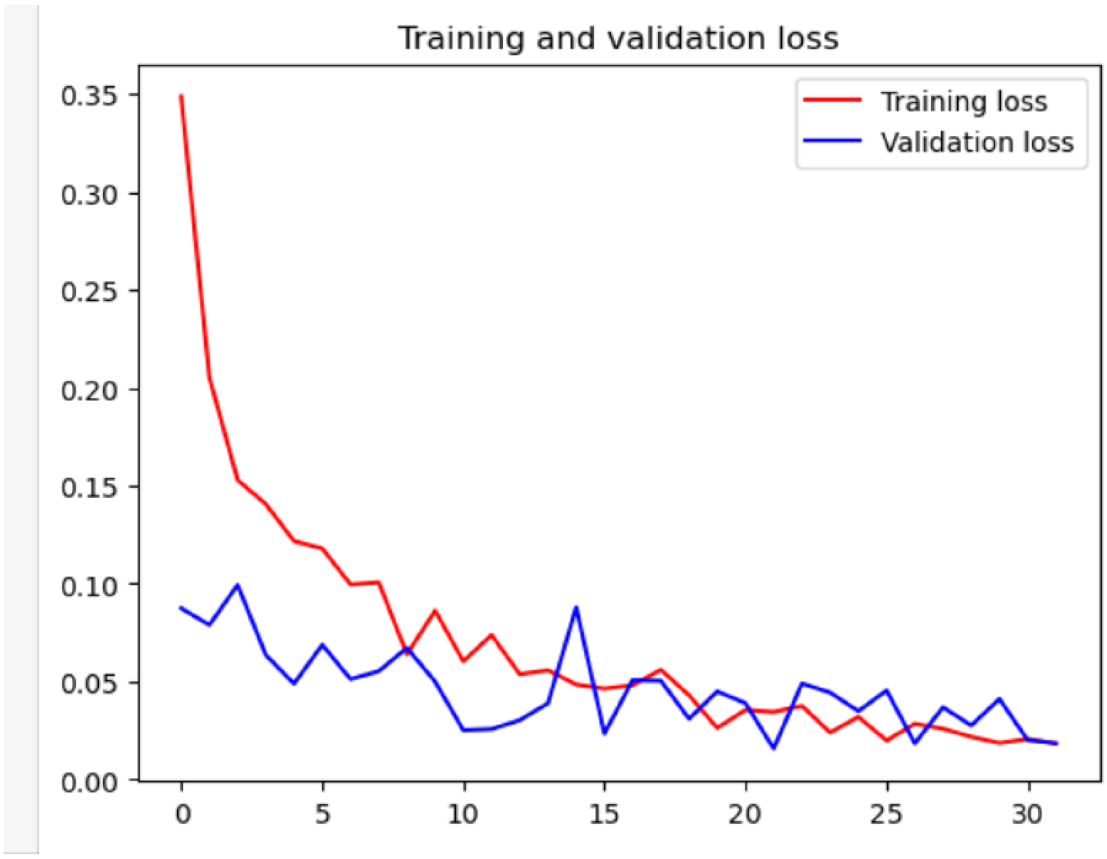
iv. Image classification:

Baseline GoogLeNetV3 Performance:

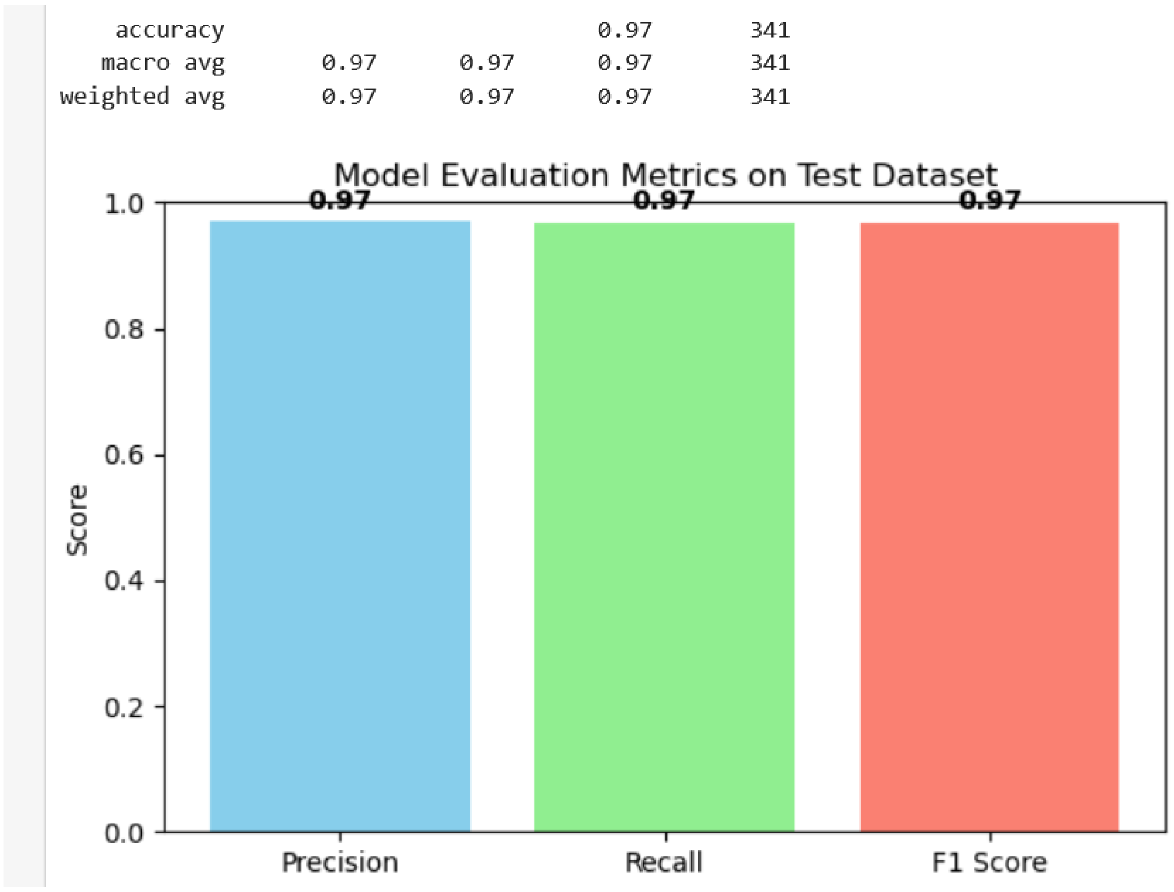


	Dataset	Accuracy	Loss
0	Train	99.85%	0.0076
1	Validation	99.37%	0.0414
2	Test	96.77%	0.0980

As experiment shown that the experiment result by GoogleNet V3 is relatively high. The accuracy rate on training and validation dataset is around 99%, and on the test dataset performance is also well which reach to 96.77%. In this case, our model is doing well classification task without overfitting.



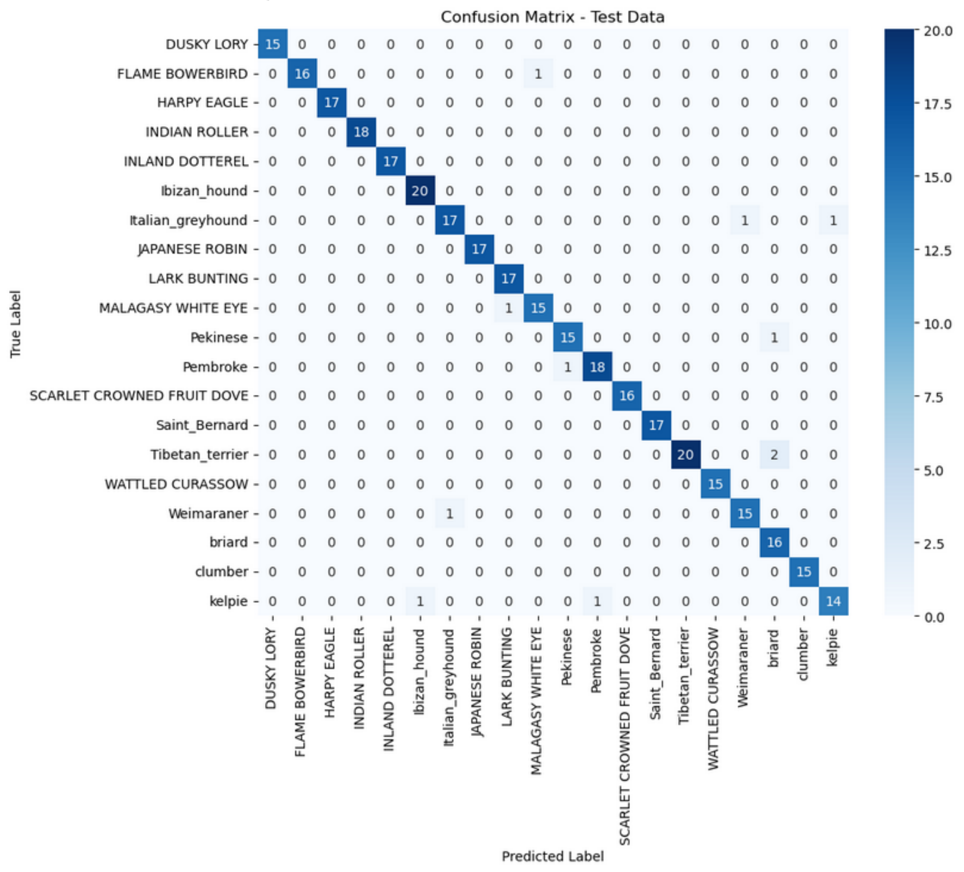
From the loss of training and validation, at beginning the loss is relatively high, after 15 epochs, it started to converge.And also the training loss continuously decrease from beginning until epochs 15, means model is learning the model well. And in the end the two loss line converged and relatively stable, which means the model is well trained.



```
    accuracy                           0.97       341
   macro avg       0.97      0.97      0.97       341
weighted avg       0.97      0.97      0.97       341
```

Furthermore, precision, recall and F1 score on the test dataset are shown here. It is relatively high as well. High perception 97% which means the model can correctly predict the positive instances out of all instances predicted as positive. Together, high recall 97% means model correctly predicts the positive instance out of all the positive instance. F1 balance recall and precision, our result is very good for GoogleNetV3 model.

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step
```



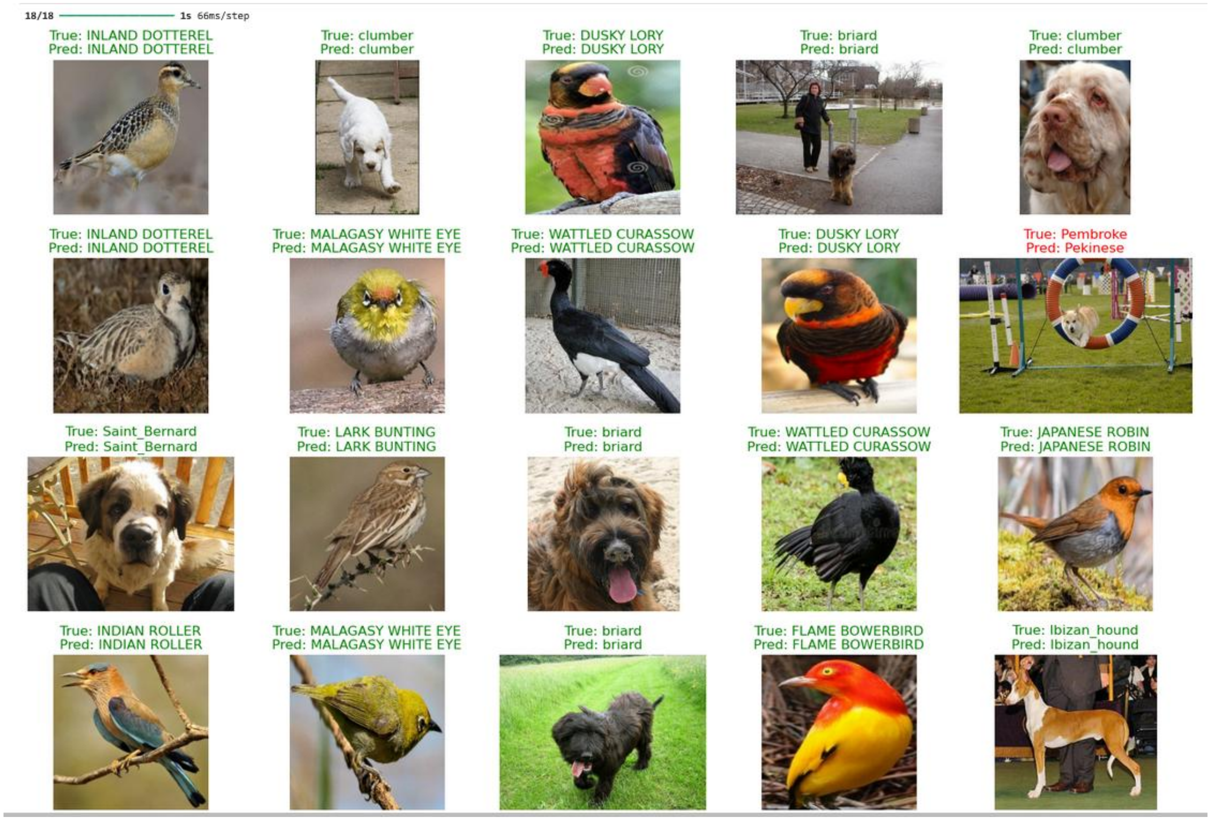
True \ Predicted	DUSKY LORY	FLAME BOWERBIRD	HARPY EAGLE	INDIAN ROLLER	INLAND DOTTEREL	Ibizan_hound	Italian_greyhound	JAPANESE ROBIN	LARK BUNTING	MALAGASY WHITE EYE	Pekinese	Pembroke	SCARLET CROWNED FRUIT DOVE	Saint_Bernard	Tibetan_terrier	WATTLED CURASSOW	Weimaraner	briard	clumber	kelpie
DUSKY LORY	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FLAME BOWERBIRD	0	16	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
HARPY EAGLE	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
INDIAN ROLLER	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
INLAND DOTTEREL	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ibizan_hound	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Italian_greyhound	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	1	0	0	1
JAPANESE ROBIN	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0
LARK BUNTING	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0
MALAGASY WHITE EYE	0	0	0	0	0	0	0	0	1	15	0	0	0	0	0	0	0	0	0	0
Pekinese	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	1	0	0
Pembroke	0	0	0	0	0	0	0	0	0	0	1	18	0	0	0	0	0	0	0	0
SCARLET CROWNED FRUIT DOVE	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0
Saint_Bernard	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0
Tibetan_terrier	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	2	0	0
WATTLED CURASSOW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0
Weimaraner	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	15	0	0	0
briard	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0
clumber	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0
kelpie	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	14

```
Classification Report:

                            precision    recall  f1-score   support

                DUSKY LORY       1.00      1.00      1.00        15
           FLAME BOWERBIRD       1.00      0.94      0.97        17
               HARPY EAGLE       1.00      1.00      1.00        17
             INDIAN ROLLER       1.00      1.00      1.00        18
           INLAND DOTTEREL       1.00      1.00      1.00        17
              Ibizan_hound       0.95      1.00      0.98        20
         Italian_greyhound       0.94      0.89      0.92        19
            JAPANESE ROBIN       1.00      1.00      1.00        17
              LARK BUNTING       0.94      1.00      0.97        17
        MALAGASY WHITE EYE       0.94      0.94      0.94        16
                  Pekinese       0.94      0.94      0.94        16
                  Pembroke       0.95      0.95      0.95        19
SCARLET CROWNED FRUIT DOVE       1.00      1.00      1.00        16
             Saint_Bernard       1.00      1.00      1.00        17
           Tibetan_terrier       1.00      0.91      0.95        22
          WATTLED CURASSOW       1.00      1.00      1.00        15
                Weimaraner       0.94      0.94      0.94        16
                    briard       0.84      1.00      0.91        16
                   clumber       1.00      1.00      1.00        15
                    kelpie       0.93      0.88      0.90        16

                  accuracy                           0.97       341
                 macro avg       0.97      0.97      0.97       341
              weighted avg       0.97      0.97      0.97       341
```

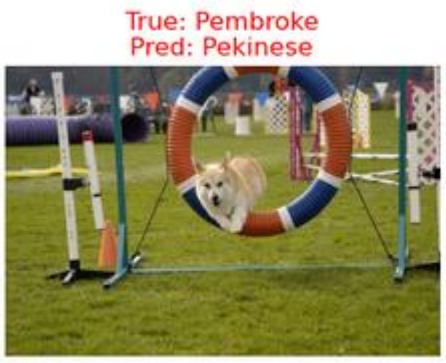
This confusion matrix compares true labels (on the y-axis) with predicted labels (on the x-axis). Each cell shows the number of predictions where a given true class (row) was predicted as another class (column). The diagonal values represent correct predictions, while off-diagonal values indicate misclassifications.

Most classes show high prediction accuracy with very few misclassifications.

For example, class *FLAME BOWERBIRD* had 16 correct predictions. One instance was misclassified as MALAGASY WHITE EYE.


```
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step
```

Wrongly Predicted Label:



1. **Performance on customized architecture**

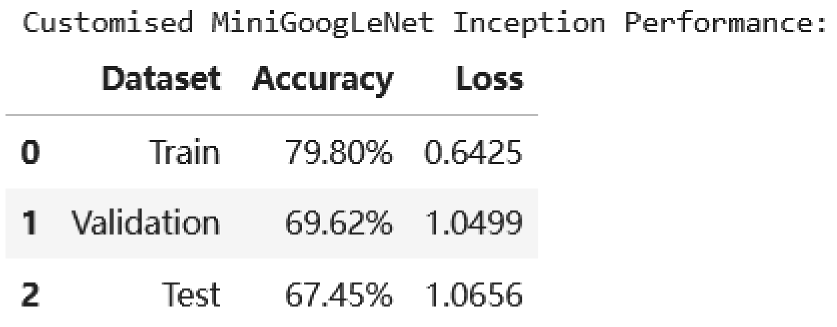
Customised MiniGooLetNet Inception Model

Four different designs of MiniGooleNet Inception models have been tested.

After running three experiments, model C performed better on image classification. In this case, Model C is implemented and analysed in the report.

The following result is from Model C, which outperformed Models A and B.

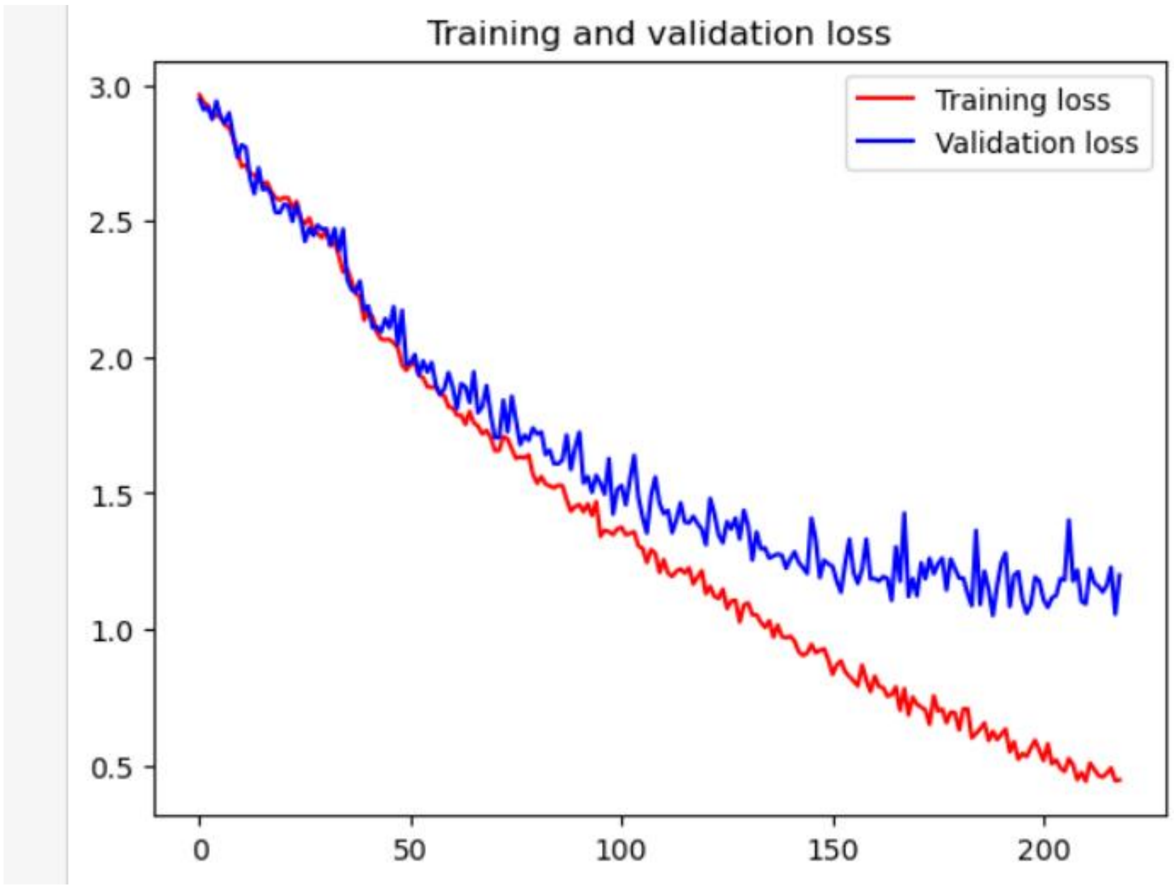
ModelC: Model with adding two fully connected layers at the end. Meanwhile, every inception block adds a dropout layer in the end of every inception block



Customised MiniGoogLeNet Inception Performance:

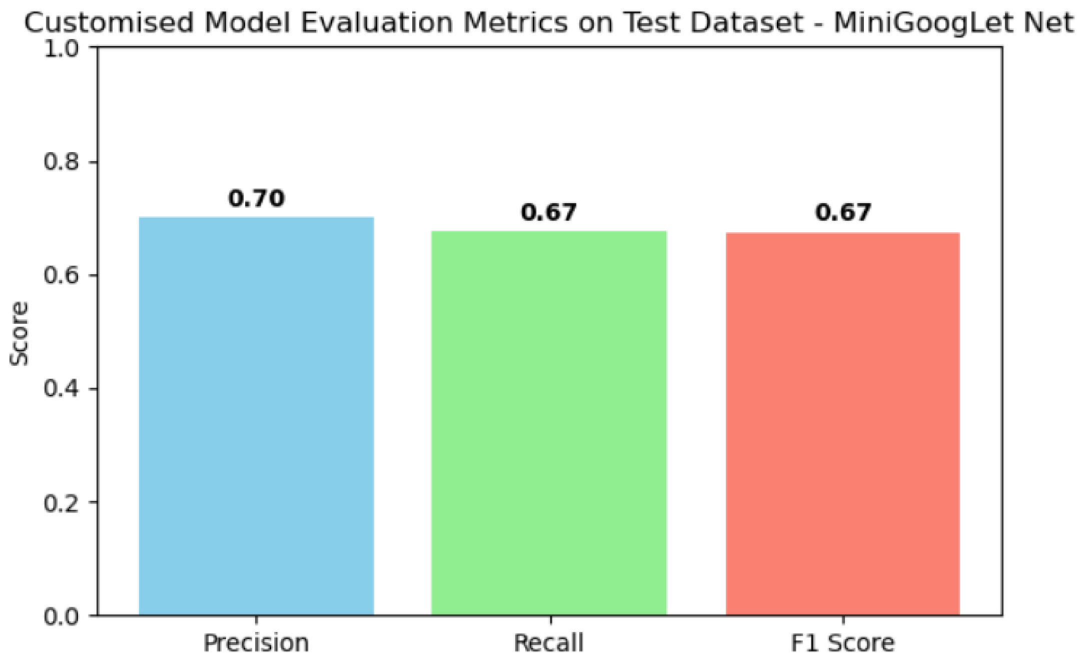
	Dataset	Accuracy	Loss
0	Train	79.80%	0.6425
1	Validation	69.62%	1.0499
2	Test	67.45%	1.0656

Accuracy rate on the training dataset is 79.8%, There is a 10% gap between training data accuracy with validation accuracy. It means the model learn well on known data, but it causes a slight overfitting, but still the generalisation performs alright on the validation dataset. Furthermore, on the test dataset, it is alright for 67, which is fair.



For the loss, as the plot shows,the training loss continuously decreases, which means the the model learns well. Validation loss also decreases, but it starts to plateau and become noisy after around epoch 100. Due to validation loss start to plateau, it means the model is finished training.

```
    accuracy                           0.67       341
   macro avg       0.70      0.68      0.68       341
weighted avg       0.70      0.67      0.67       341
```

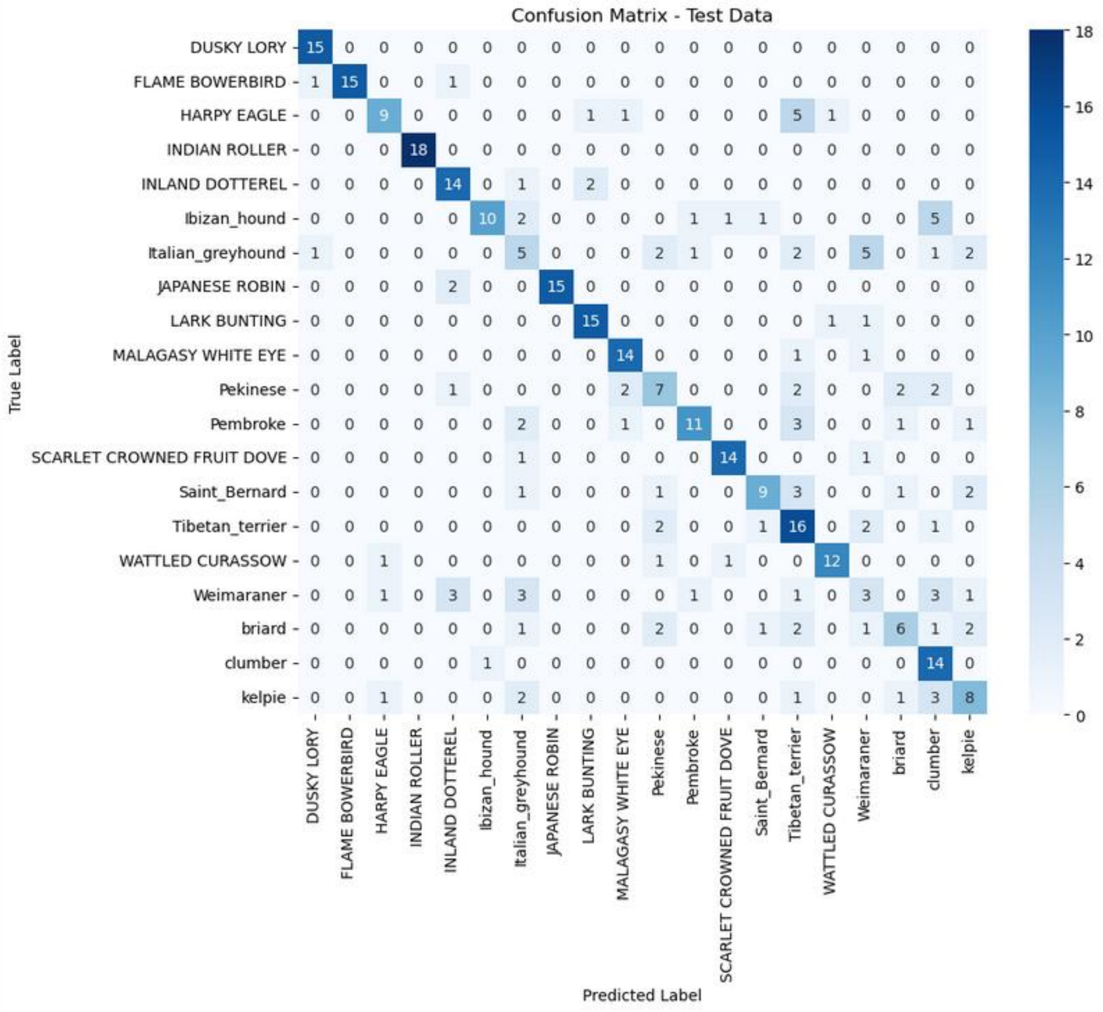


Precision of 0.70 means out of all the instances the model predicted as positive, 70% were actually correct.

Recall of 0.67 means out of all actual positive instances, the model correctly identified then 67%.

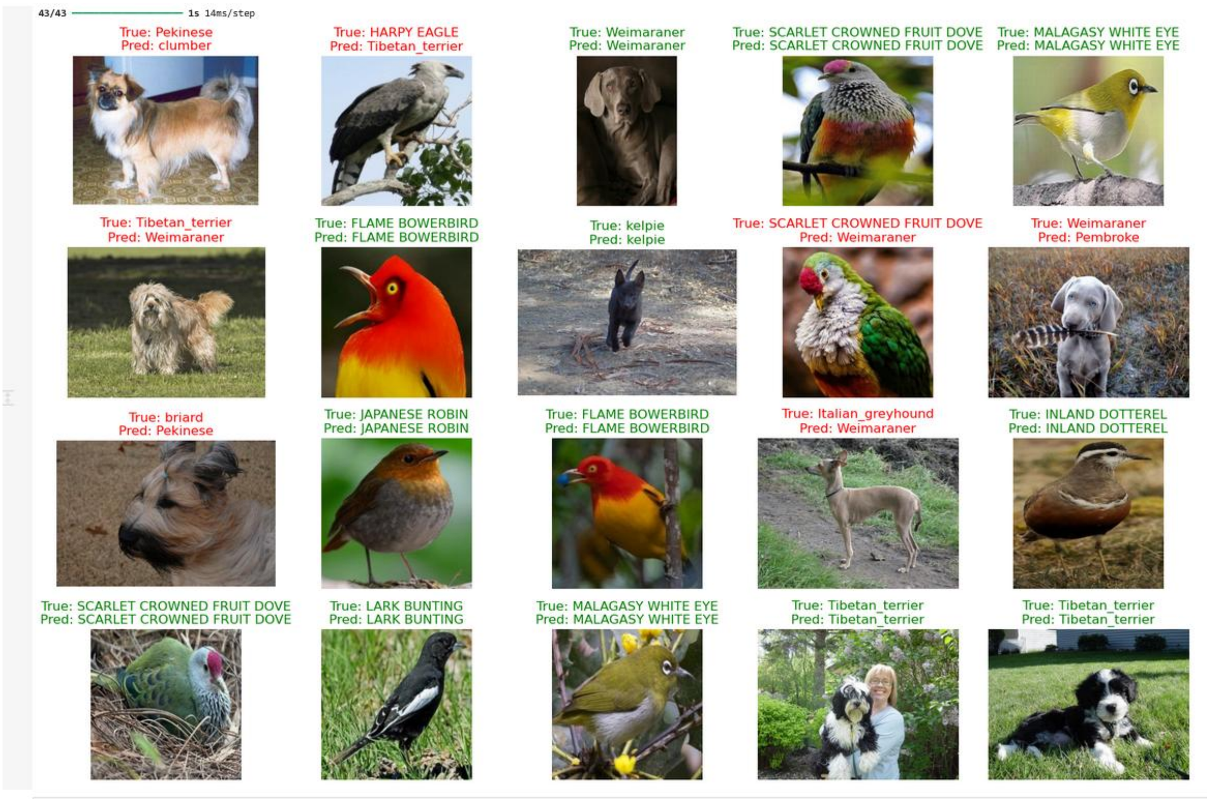
F1 is the balance between precision and recall which reaches to 0.67.

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step
```



```
Classification Report:

                            precision    recall  f1-score   support

                DUSKY LORY       0.88      1.00      0.94        15
           FLAME BOWERBIRD       1.00      0.88      0.94        17
               HARPY EAGLE       0.75      0.53      0.62        17
             INDIAN ROLLER       1.00      1.00      1.00        18
           INLAND DOTTEREL       0.67      0.82      0.74        17
              Ibizan_hound       0.91      0.50      0.65        20
         Italian_greyhound       0.28      0.26      0.27        19
            JAPANESE ROBIN       1.00      0.88      0.94        17
              LARK BUNTING       0.83      0.88      0.86        17
        MALAGASY WHITE EYE       0.78      0.88      0.82        16
                  Pekinese       0.47      0.44      0.45        16
                  Pembroke       0.79      0.58      0.67        19
SCARLET CROWNED FRUIT DOVE       0.88      0.88      0.88        16
             Saint_Bernard       0.75      0.53      0.62        17
           Tibetan_terrier       0.44      0.73      0.55        22
          WATTLED CURASSOW       0.86      0.80      0.83        15
                Weimaraner       0.21      0.19      0.20        16
                    briard       0.55      0.38      0.44        16
                   clumber       0.47      0.93      0.62        15
                    kelpie       0.50      0.50      0.50        16

                  accuracy                           0.67       341
                 macro avg       0.70      0.68      0.68       341
              weighted avg       0.70      0.67      0.67       341
```

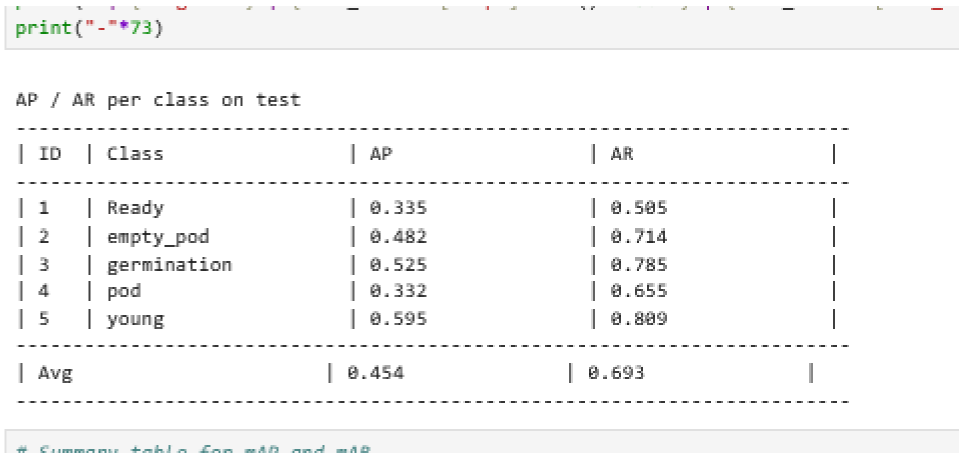
As the confusion matrix shown, some classes has misclassification. For example, Hyppy Eagle, classified correctly as 9 instances but misclassified 8 classes. Recall for this class Hyppy Eagle is only 0.53 and and precision is 0.75. However, class like Indian roller perform well, all classified correctly, which both precision and recall perform well. However, Class Weimaraner perform poorly, only predicting 3 correctly into the right class weimararner, where precision is 0.21 and recall is 0.19.



```
43/43 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step
```

v. Object Detection:

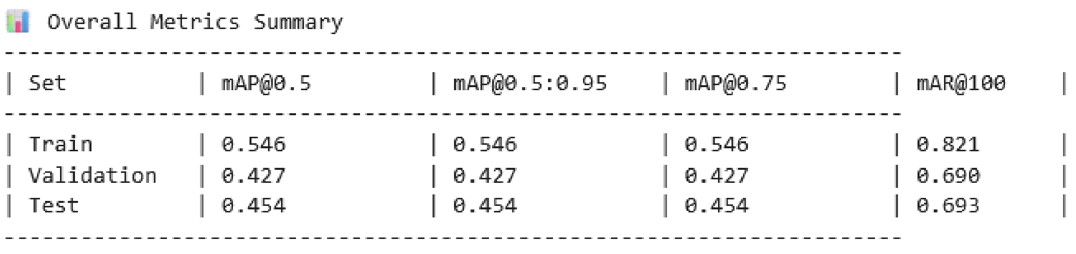
1. Performance on Faster-RCNN



```
print("-"*73)
```

```
AP / AR per class on test
-------------------------------------------------------------------------
| ID  | Class                 | AP                   | AR                 |
-------------------------------------------------------------------------
| 1   | Ready                 | 0.335                | 0.505              |
| 2   | empty_pod             | 0.482                | 0.714              |
| 3   | germination           | 0.525                | 0.785              |
| 4   | pod                   | 0.332                | 0.655              |
| 5   | young                 | 0.595                | 0.809              |
-------------------------------------------------------------------------
| Avg                       | 0.454                | 0.693              |
-------------------------------------------------------------------------
```

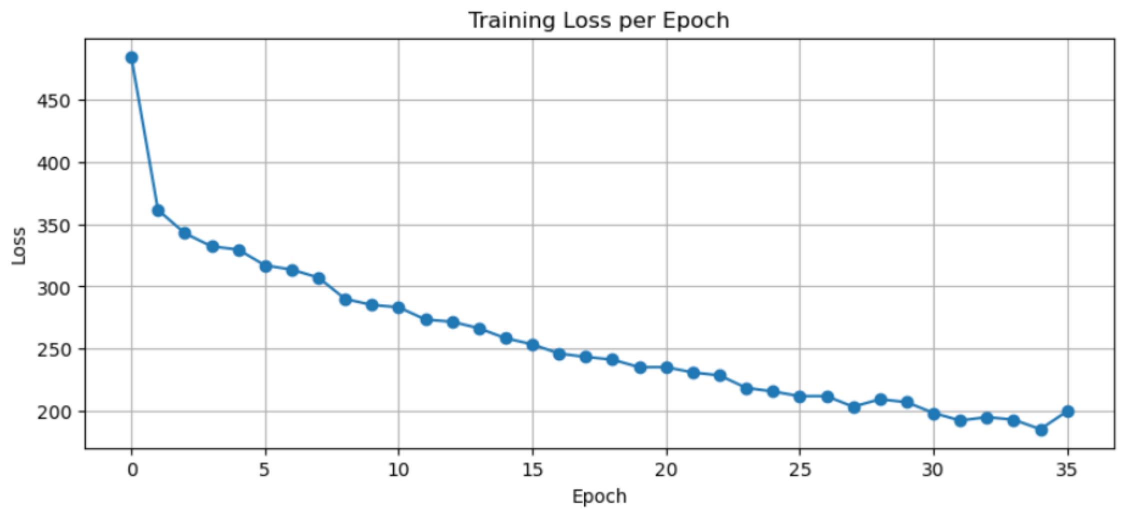
As the performance matrix shows, among all those classes, Class Young performed best , where average recall (AR) is around 0.81 andaverage precision(AP) is around 0.60. However, class ready does not perform well with AP is around 0.36 and AR is 0.5

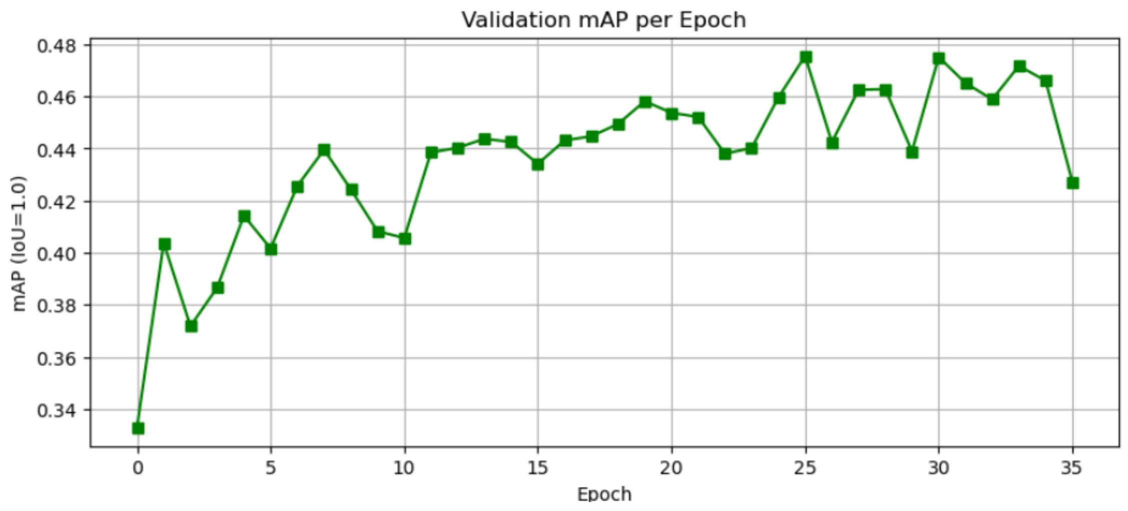


```
📊 Overall Metrics Summary
----------------------------------------------------------------------
| Set           | mAP@0.5         | mAP@0.5:0.95    | mAP@0.75        | mAR@100      |
----------------------------------------------------------------------
| Train         | 0.546           | 0.546           | 0.546           | 0.821        |
| Validation    | 0.427           | 0.427           | 0.427           | 0.690        |
| Test          | 0.454           | 0.454           | 0.454           | 0.693        |
----------------------------------------------------------------------
```

For the oeverall, mAP on the test data is acceptable.

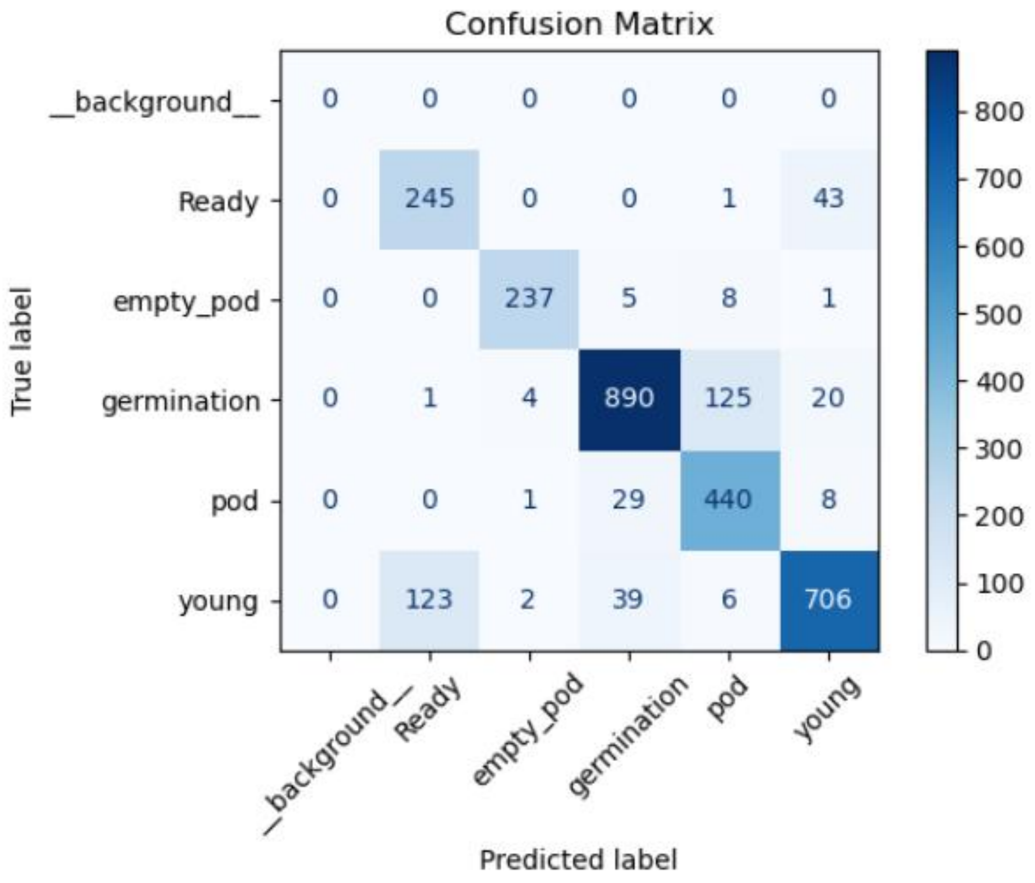
With the test dataset, the average precision(map) for each class is around 0.45 only when IoU(Intersection of Union) is over 50%, 50% to 95% or over 75%. The average recall when IoU = 100 is around 0.69.





Training Loss per Epoch shows a consistent downward trend, which means model is learning the training data well.

Validation mAP per epoch increases from 0.34 to 0.47 which is a healthy rise. And it start to slightly fluctuated after epchs 10, which could be start over fitting from middle training, and also maybe due to small dataset with only around 1500 training dataset.

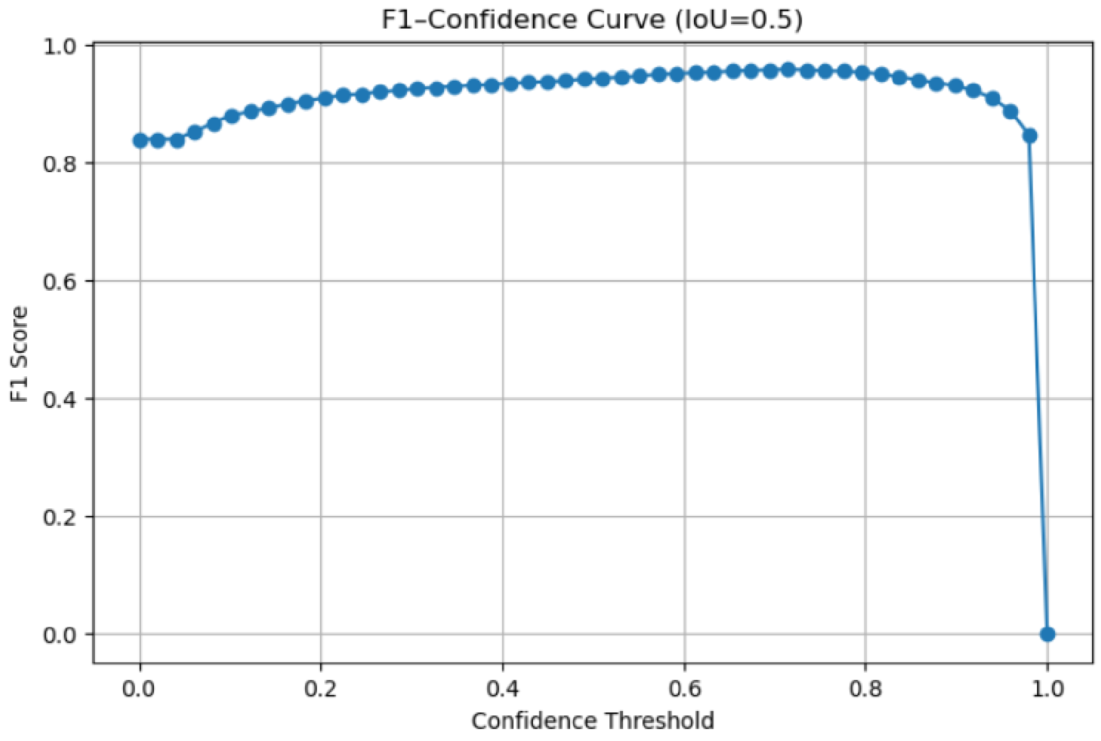


True label \ Predicted label	__background__	Ready	empty_pod	germination	pod	young
__background__	0	0	0	0	0	0
Ready	0	245	0	0	1	43
empty_pod	0	0	237	5	8	1
germination	0	1	4	890	125	20
pod	0	0	1	29	440	8
young	0	123	2	39	6	706

As confusion matrix showed, class germination predicted well, with 890 data correctly classified as germination, and 29 data was wrongly predicted into pod class and 39 was wrongly predicted into young class.

It also works well for the class young, with 706 correctly predicted as young; 43+1+20+8 was wrongly predicted into other classes.

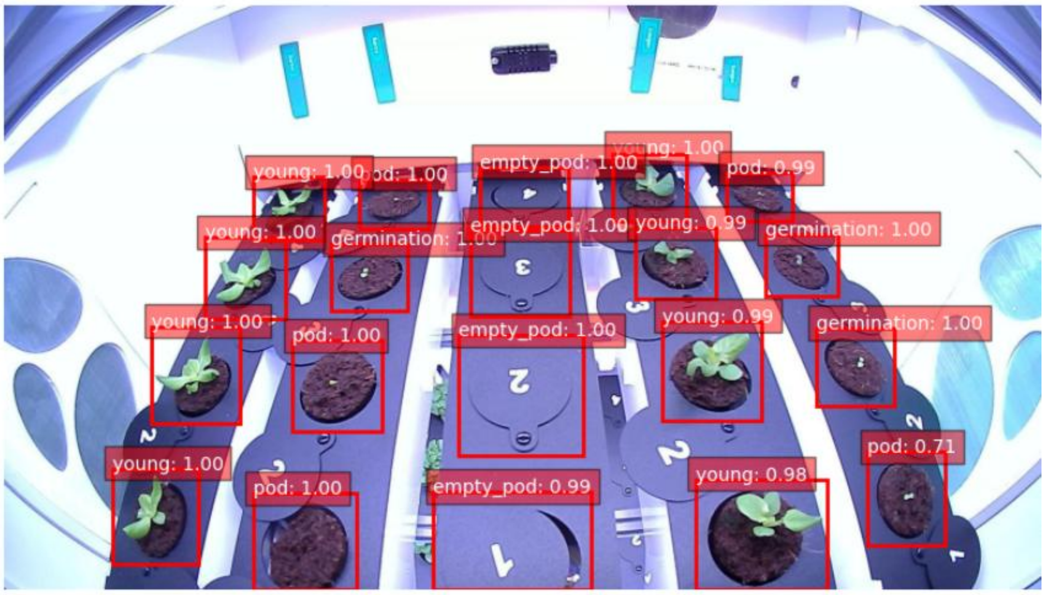
```
Collecting preds: 100%|██████████| 113/113 [00:12<00:00,  8.89it/s]
100%|██████████| 50/50 [00:05<00:00,  9.43it/s]
```

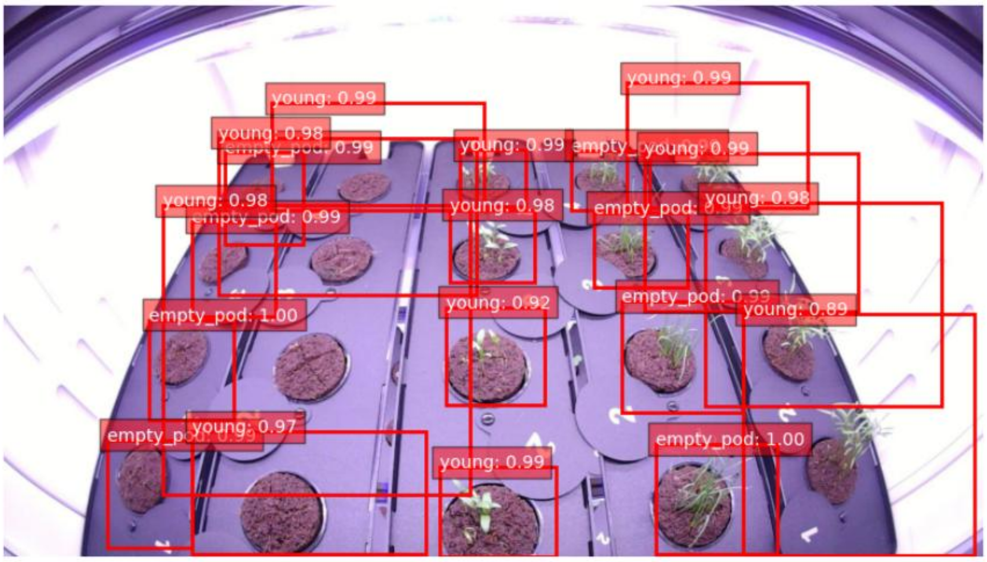


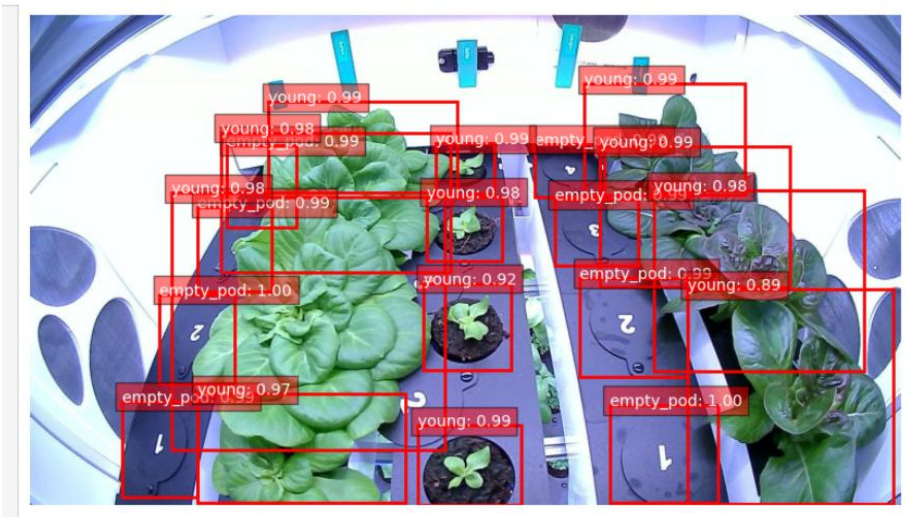
The F1 confidence curve(with an IoU threshold of 0.5) starts at 0.8 when the confidence threshold is as low as 0.0. Then, it increases steadily until it peaks around the 0.6 to 0.8 confidence threshold.

This means that model predictions with confidence levels in this range achieve a good balance between precision and recall.

When the confidence threshold is 1, the F1 score drops sharply to near 0. This may be because very few or no predictions meet this strict confidence requirement, resulting in low recall.

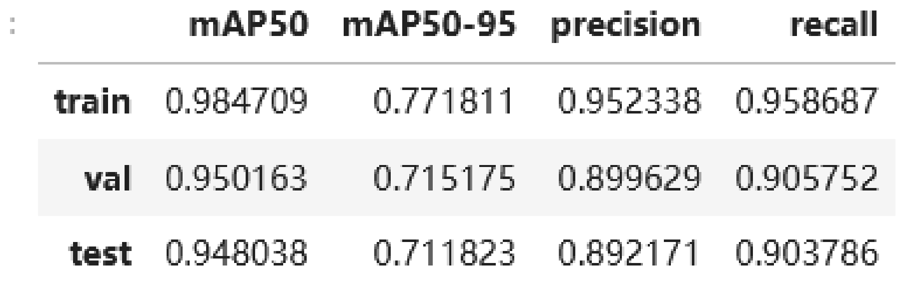

young: 1.00
pod: 1.00
empty_pod: 1.00
young: 1.00
pod: 0.99
young: 1.00
germination: 1.00
empty_pod: 1.00
young: 0.99
germination: 1.00
young: 1.00
pod: 1.00
empty_pod: 1.00
young: 0.99
germination: 1.00
pod: 0.71
young: 1.00
pod: 1.00
empty_pod: 0.99
young: 0.98


young: 0.99
young: 0.99
young: 0.98
empty_pod: 0.99
young: 0.99
young: 0.99
young: 0.98
empty_pod: 0.99
young: 0.98
young: 0.98
empty_pod: 1.00
young: 0.92
empty_pod: 0.99
young: 0.89
young: 0.97
young: 0.99
empty_pod: 1.00


young: 0.99
young: 0.99
young: 0.98
empty_pod: 0.99
young: 0.99
young: 0.99
young: 0.98
empty_pod: 0.99
young: 0.98
young: 0.98
empty_pod: 1.00
young: 0.92
empty_pod: 0.99
young: 0.89
young: 0.97
young: 0.99
empty_pod: 1.00

2. Performance on SDD/YOLO or any object detector

Yolov8 model Performance



	mAP50	mAP50-95	precision	recall
train	0.984709	0.771811	0.952338	0.958687
val	0.950163	0.715175	0.899629	0.905752
test	0.948038	0.711823	0.892171	0.903786

Overall, Yolov8 model performs well for this object detection tasks.

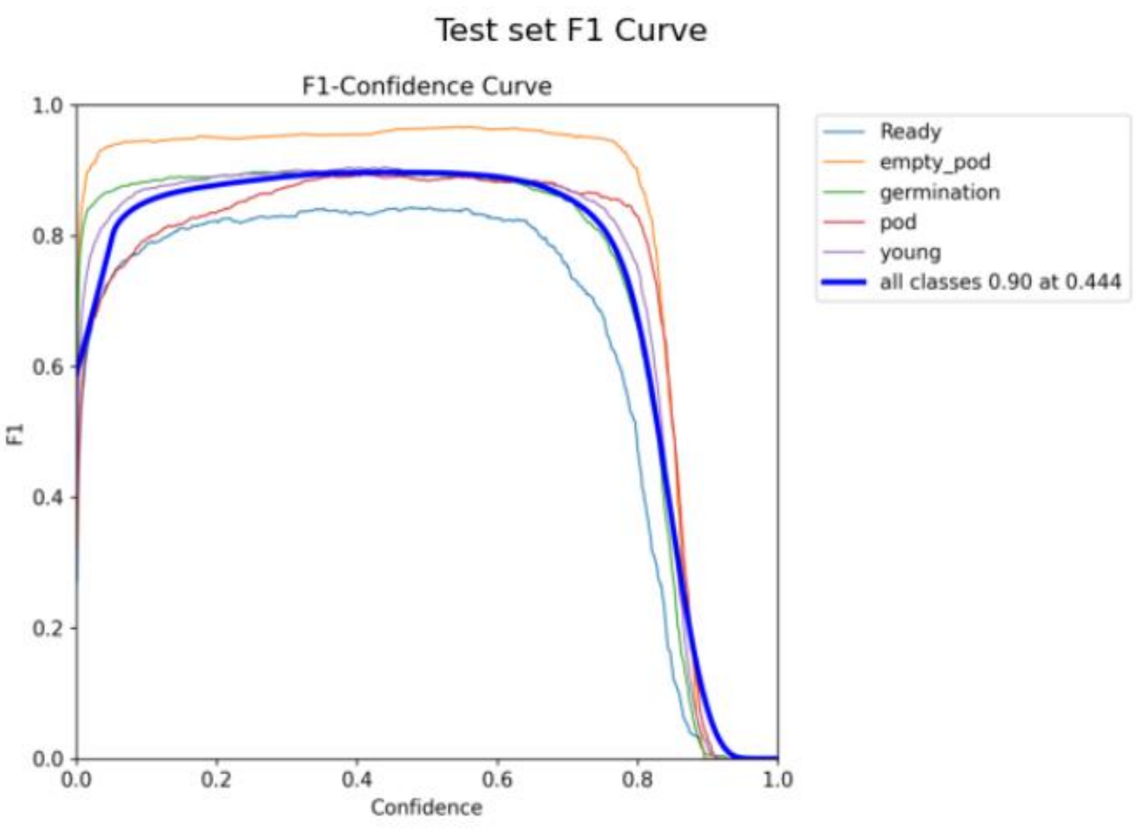
mAP50: mAP50 value is high above 0.94 when mAP50 (detecting objects at an IoU threshold at an IoU threshold of 0.5)

mAP50-95: mAP50-95 drops a bit to 0.71 is also consider model perform well due to mAP50-95 is much stricter requirement.

No overfitting: As the result shows difference between training, validation and testing is within 0.05 which is small, it shows no major overfitting.

Precision vs. Recall:

The model's precision and recall are both high across training, validation and testing, which means the model generates well, with low false positives and low false negatives. It has a goo balance between precision and recall.



The F1-confidence curve shows that the model performs very well.

Overall peak F1 score:

As it shows as dark blue lines, all classes 0.9 at 0.444

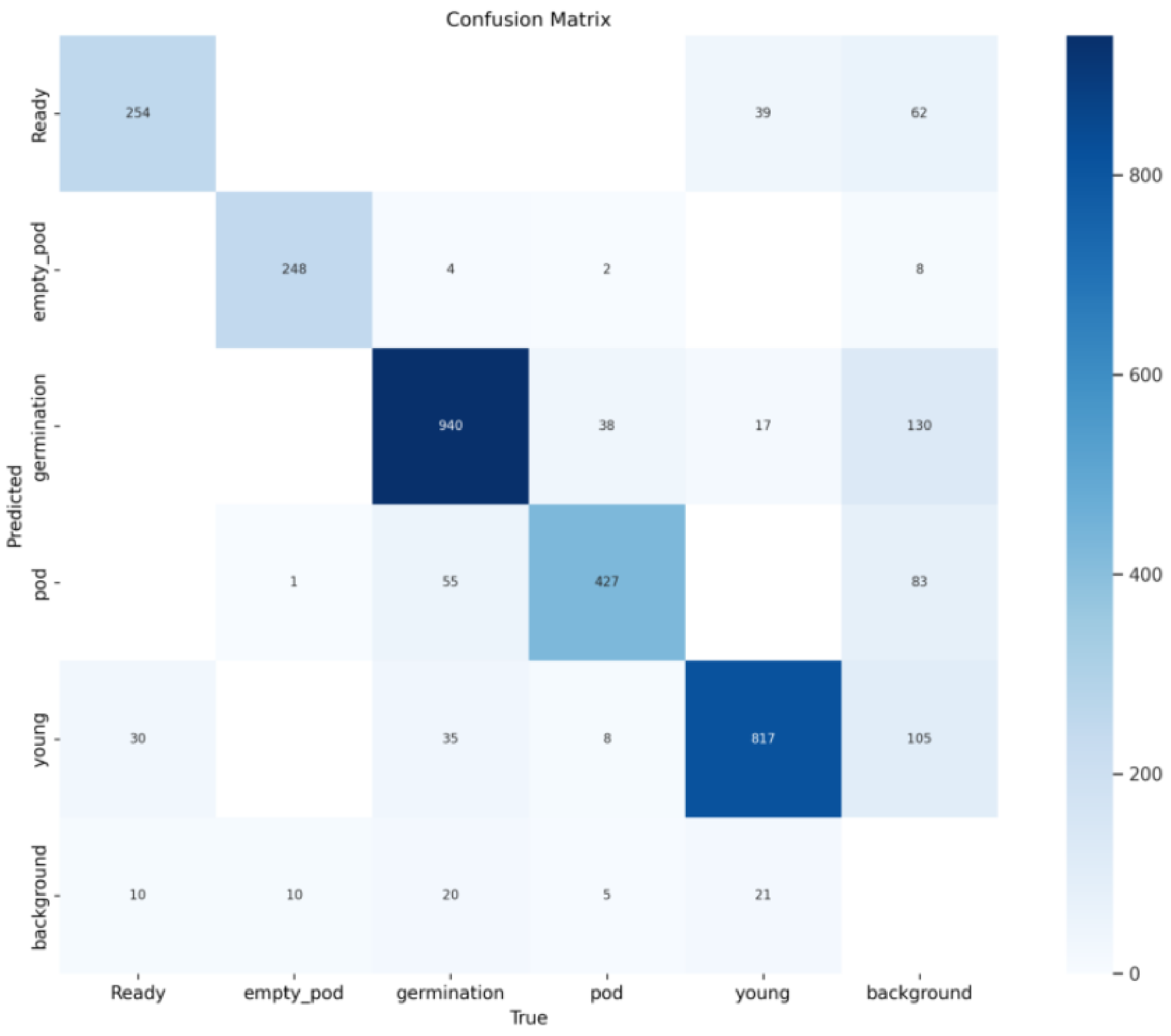
Which means when I set confidence thresholds to 0.444 during inference, I will likely get the best overall F1 score across all classes.

It indicated that the current Yolov8 model is robust, which has a good balance between recall and precision.

Sharp drop after ~0.8 confidence:

This sharp drop is a typical behaviour due to very high confidence leads to fewer predictions and lower recalls, so F1 drop sharply.

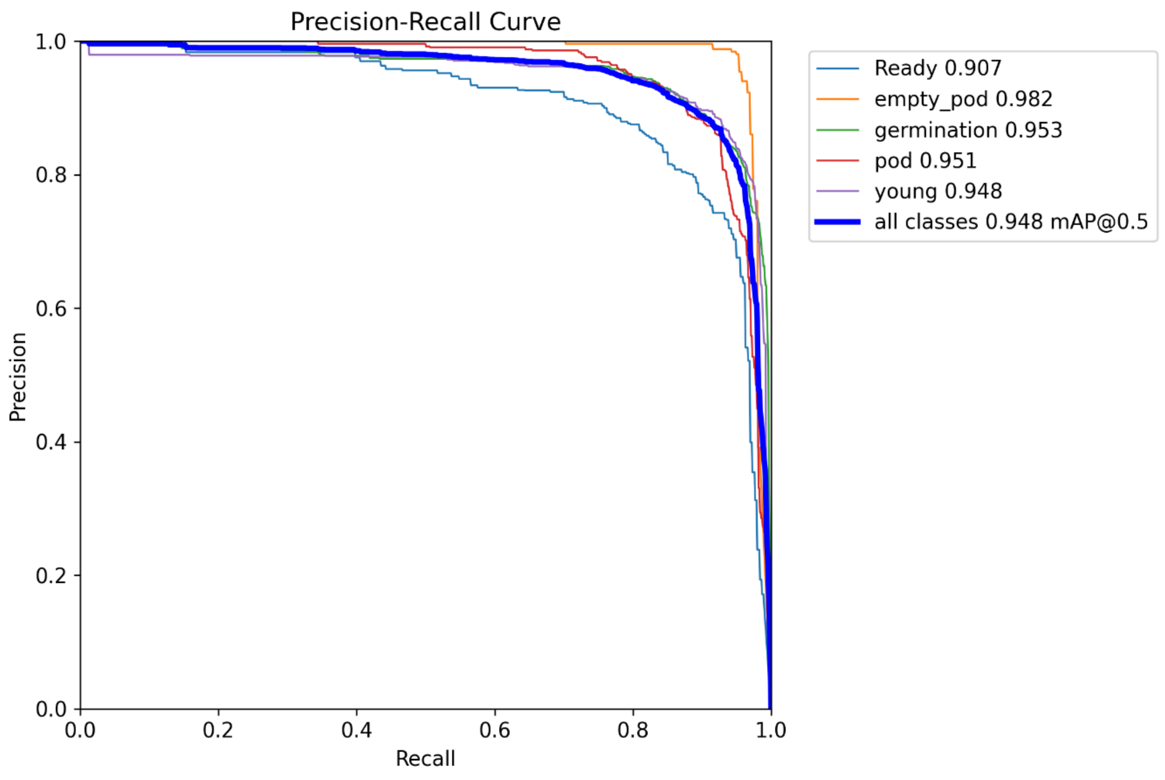
Test confusion matrix



As the confusion matrix on the test data shows, the model performs well.

Take the class ‘Ready’ as an example; among all the classes, ‘Ready’ performs the worst, but it is still at an acceptable rate.

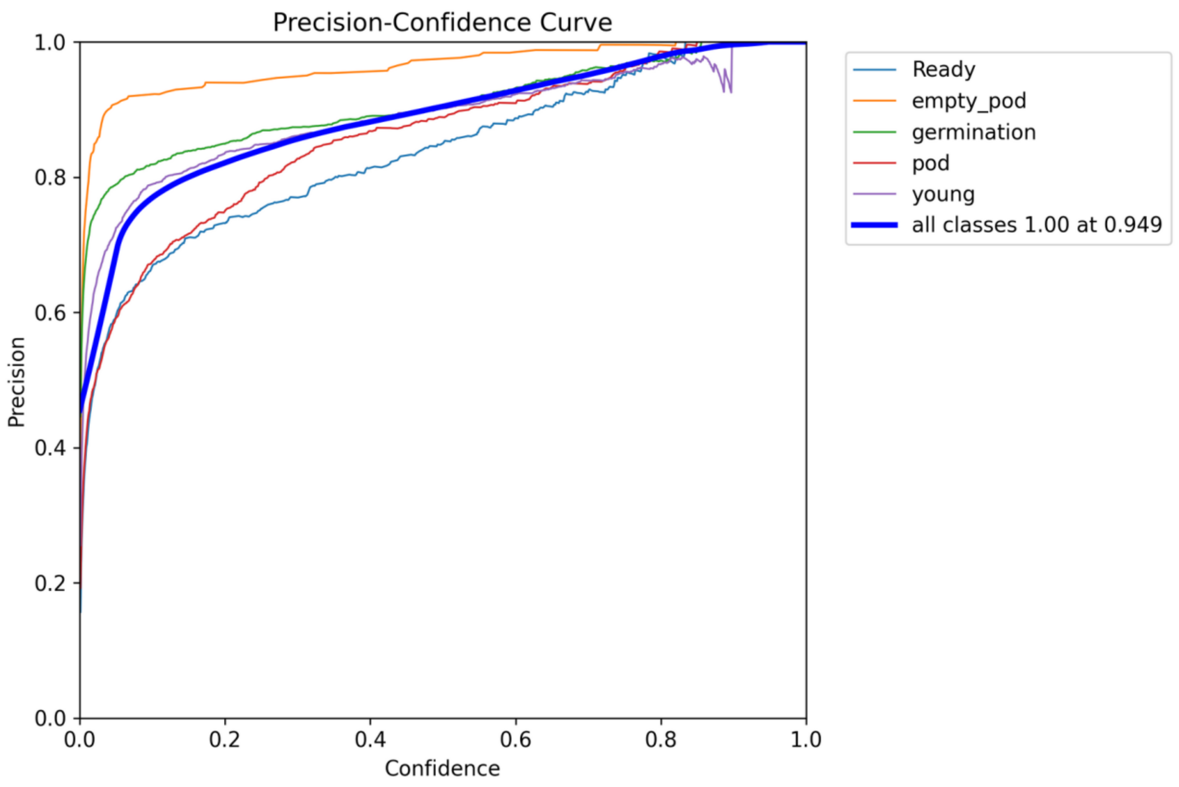
For the ‘ready ’ class on the test dataset, the model correctly predicted 254 instances as ‘ready’. For those true’ Ready’ leaves, 39 instances were wrongly predicted as young, and 62 were wrongly predicted as background.

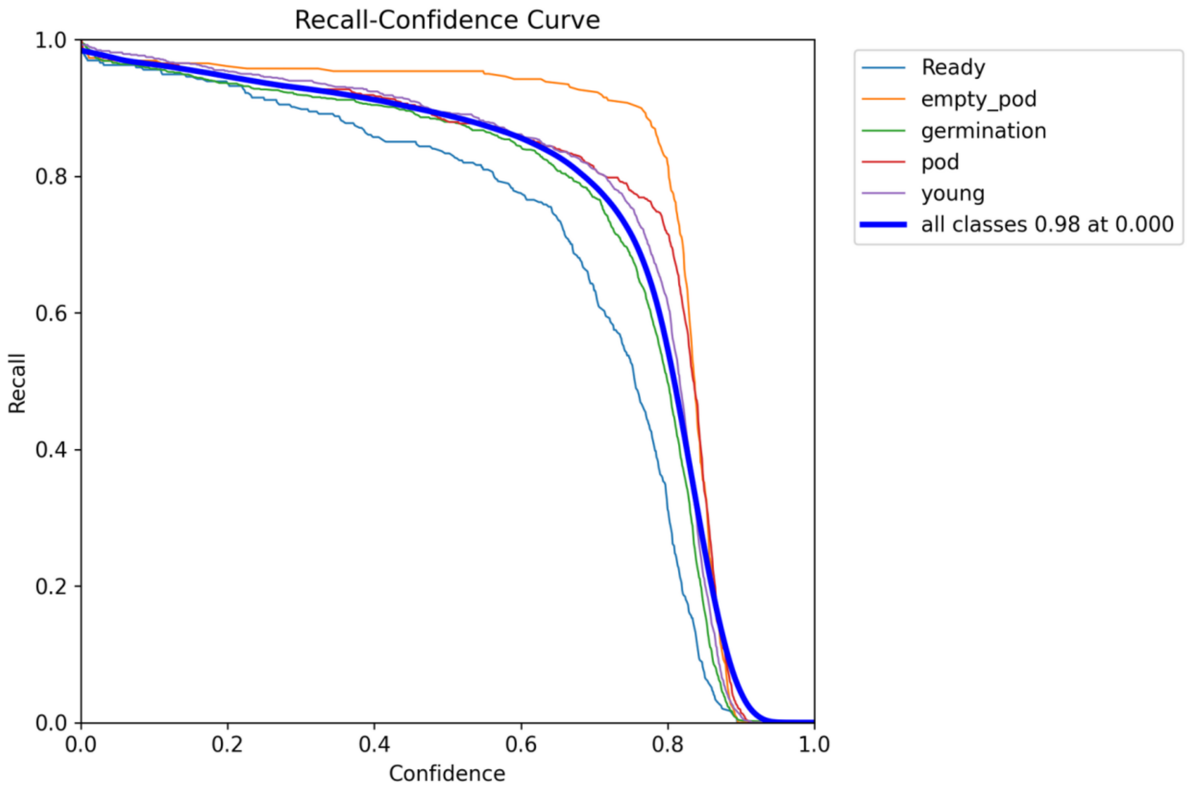


For the precision-recall curve, if it stays high and flat, it performs the best.

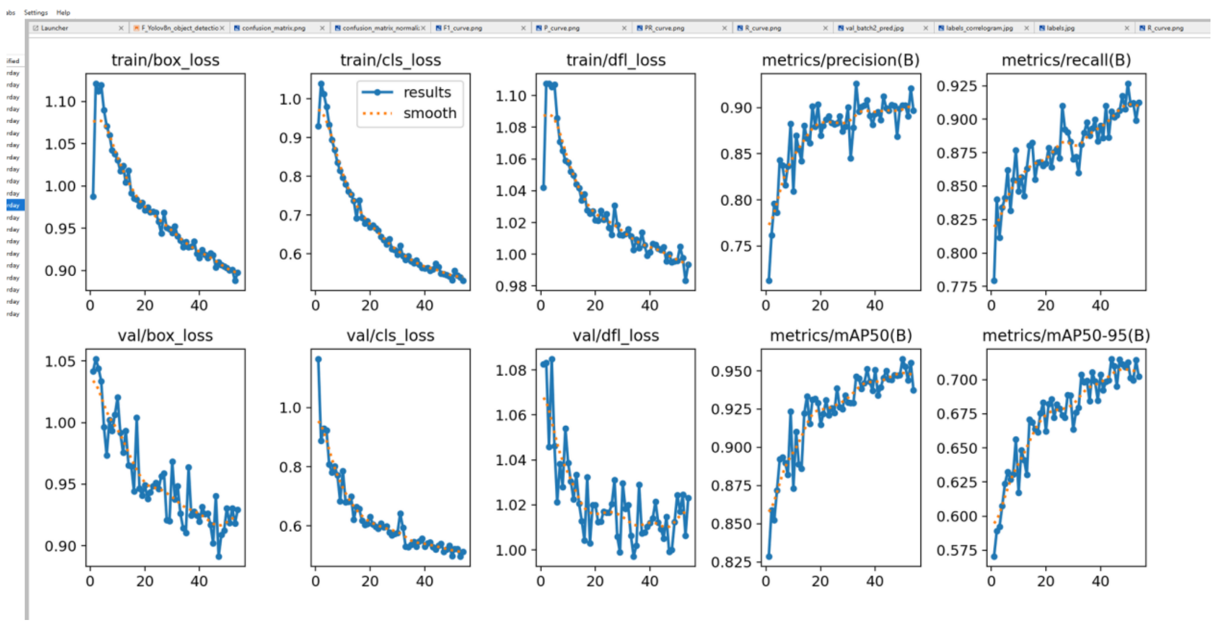
In this case, the class ‘ready’ performs the worst because it drops earlier than other models, which means precision falls faster as recall increases.

And the empty plot performs the best which it stays high and flat always.



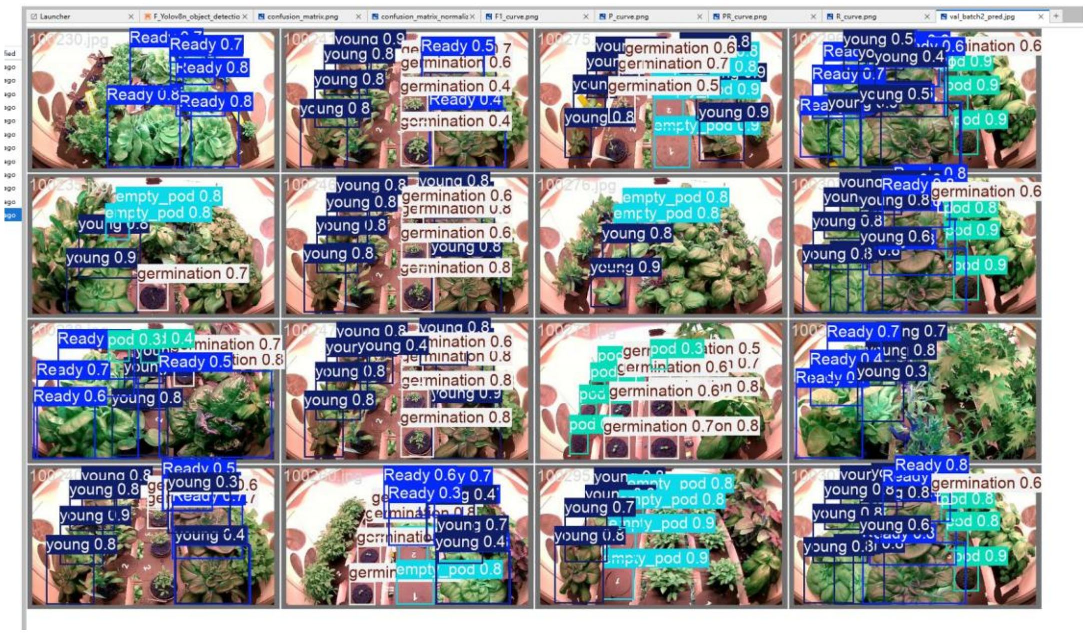


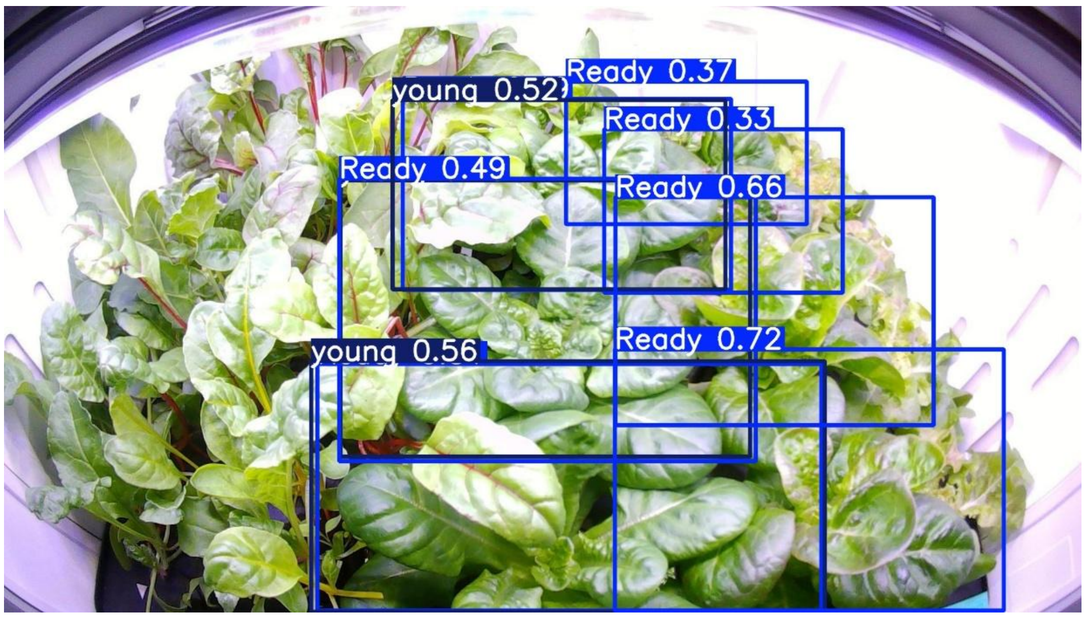
Above two curve is also shows the class of ready perform the worst, where empty_plot perform the best. Overall model performance is good.

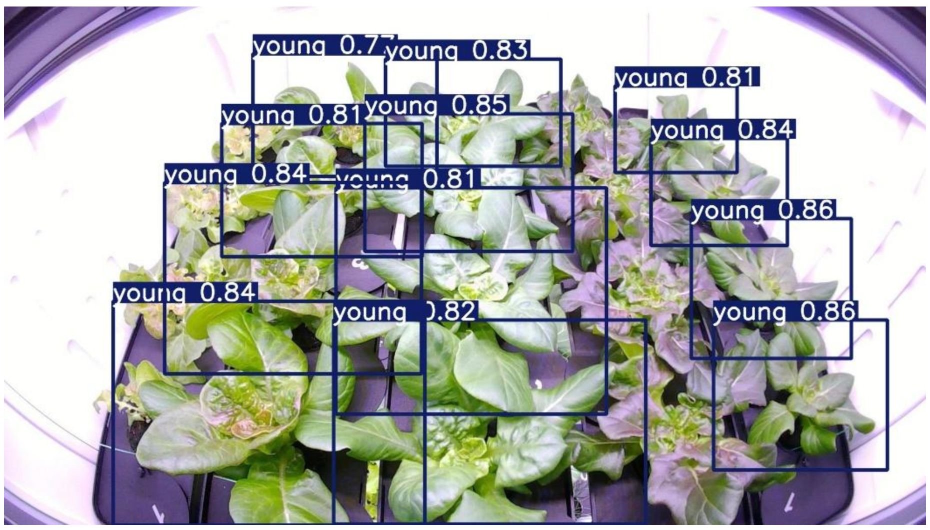


It is an overview of metrics and losses. All the losses decreased steadily, indicating that the model is effectively learning localization (box loss), classification (cls loss), and distribution (DFL loss). Although the val/box_losses and val/dfl_losses fluctuated in the latter half of the training/validation, they remained at an acceptable rate.

For precision(B) and recall(B), mAP50 and mAP50-95, all four plots trend upward, reflecting improved object detection performance in terms of detection rate and localisation accuracy. They fluctuate, but those fluctuations are within an acceptable range, and overall trends are an increase, meaning the model works well.


Launcher
F_Yolov8n_object_detectio
confusion_matrix.png
confusion_matrix_normaliz
F1_curve.png
P_curve.png
PR_curve.png
R_curve.png
val_batch2_pred.jpg
Ready 0.7
Ready 0.8
young 0.9
young 0.8
germination 0.6
germination 0.4
empty_pod 0.8
young 0.9
germination 0.7
pod 0.9


young 0.52
Ready 0.37
Ready 0.33
Ready 0.49
Ready 0.66
young 0.56
Ready 0.72


young 0.77
young 0.83
young 0.81
young 0.81
young 0.85
young 0.84
young 0.84
young 0.81
young 0.86
young 0.84
young 0.82
young 0.86

6. Conclusion

For object classification, the baseline model GoogLeNet V3 works exceptionally well. By using transfer learning, the model trains well. Its accuracy rate on test data is 96.77%, and the loss is 0.098. The accuracy difference between train, validate, and test is small, which indicates no overfitting. Recall, Precision and F1 score are equally high at 0.97 at the test dataset, which indicates the model has a good balance between recall and precision.

In this case, the further customised model cannot outperform the baseline model, GoogLeNet V3. However, GoogleNetV3 is a heavy model; the model training time could be long due to the heavy infrastructure. For customising model, I decided to design a lighter version of the inception model, which is a mini GoogLeNet with only four inception blocks. As a tradeoff, the accuracy will be less than the baseline model.

The customised Mini GoogleNet's accuracy rate on the test dataset is 67%, with a loss of 1.06. The precision is 0.7, the recall is 0.67, and the F1 score is 0.67. The customised performance is lower than

For Object detection, the Yolov8 model outperforms the Faster-R-CNN model regarding model performance, model training speed, and ease of implementation. On the Yolov8 test dataset, mAP50 is 0.948, and mAP 50-95 is 0.71, with precision at 0. 89 and recall at 0.9. Futhermore, Yolov8 model learns well during the training. And easy to implement.

On contract, Faster-R-CNN with a backbone of MobileNetV3 does not perform as well as Yolov8. For this test dataset, mAP@0.5 , mAP@0.5-0.95 and mAP@0.75 are all 0.45, with mAR@100 at 0.693, which is relatively lower than the Yolov8 model.

Overall, after all experiments with different AI models, the newest advanced model like GoogleNetV3 and Yolov8 enjoy a higher model performance with faster training rate and easier to implement. Together, each model suitable for different dataset. For example for Faster RCNN, it might perform better if detect small instances. But for our cases, our instances is big, so it might not perform well on it. And for Yolov8 nano model, it is more suitable for small to medium dataset, so for our cases, when we implement Yolov8n model, the result perform well. Futhermore, the running time difference a lot. Take Faster R CNN with ResNet 50 backbone as example, the training speed is extremely slow. In this case, we also need to consider the computation resource and training time as a model choice decition.

Declare of AI Usage:

1. Use Chatgpt to fix the code's bugs and explain the code.

2. For the code that lacks knowledge of building it, use ChatGPT to inspire and generate part of the code.

Reflection of GenAI on assignment

During the assignment, ChatGPT is extremely useful to understand deeply the code and function from the lab, and how to adjust and implement it into the assignments. I realise I need to understand the architecture deeply and function as well, so I can ask the right questions to adjust the model architecture. Also, I realise in the current new Genai world, instead of being able to write code quickly, I need to learn how to prompt correctly and ask the right questions, which requires a deep understanding of the model architecture and designing concept.

In the end, using ChatGPT in assignments helped me learn deep learning. In the future, I need to discover other advanced GenAI models and how to work best with the tool, which is extremely important.

Reference:

OpenAI. (2023). ChatGPT (March 14 version) [Large language model]. https://openai.com

Sharma, N. (2025). *Deep learning and convolutional neural network: UTS labs week 7,8 & week 9*. University of Technology Sydney

ChatGPT Saved Conversation for Assignment2

https://chatgpt.com/share/6825a23f-e9d0-8008-8da3-e7778aee55af

https://chatgpt.com/share/6825a260-24b8-8008-b2c8-f05762a059c9

https://chatgpt.com/share/6825a27a-6b64-8008-a30d-2a8b622a91d3

https://chatgpt.com/share/6825a289-4ed4-8008-ab61-5a98a3e77f7e

https://chatgpt.com/share/6825a29b-6c64-8008-9133-ac1b35a5686d

https://chatgpt.com/share/6825a2ac-cd58-8008-82d3-51282644c789

https://chatgpt.com/share/6825a2c1-736c-8008-bb64-26407929f2c7

https://chatgpt.com/share/6825a2d3-99e8-8008-b118-0cfb42b78e32

https://chatgpt.com/share/6825a2eb-30d4-8008-b862-39b484c1b28f

https://chatgpt.com/share/6825a2fd-4d14-8008-93e0-469079b78cfe

https://chatgpt.com/share/6825a310-f9c4-8008-9b12-68a79ae8c1cd

https://chatgpt.com/share/6825a335-8fd4-8008-a862-766526c4b15a

https://chatgpt.com/share/6825a34d-011c-8008-92d0-31d0aa9c488d

https://chatgpt.com/share/6825a35d-40d8-8008-b737-fc45006c8311

https://chatgpt.com/share/6825a373-5c28-8008-9880-da8cc1261227

https://chatgpt.com/share/6825a389-3a3c-8008-91e1-093f9d9e4732

https://chatgpt.com/share/6825a3e3-ce3c-8008-9a70-04f7996740b8