42028: Deep Learning and Convolutional Neural Network

Assignment -2

Student Name: Nivetha Anand

Student ID: 13663024

1. Introduction:

The main purpose of this report is to provide an insight on the basic understanding of the Image Classification and Object Detection algorithms. This report is basically divided into two parts such as first part tends to be Image classification and second part constitutes object detection. In case of first part, Image classification I used vgg16 as the baseline CNN architecture with transfer learning to create the customized CNN. Whereas, in the second part Object Detection I have used two methodologies such as Single Shot Detector and Faster RCNN.

2. Dataset:

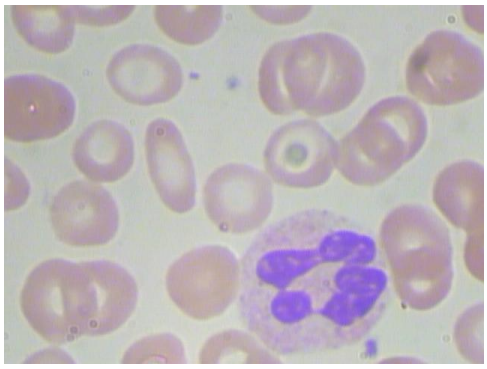
Image Classification:

The Dataset that was used for image classification was flower dataset with 17 different categories of flowers including Blue bell, Buttercup, Coltsfoot, Cowslip, Crocus, Daffodil, Daisy, Dandelion, Fritillary, Iris, Lilly Valley, Pansy, Snowdrops, Sunflower, Tigerlily, Tulip and Wind flower. So, the total data set constitutes to 1360 images where 680 images where used for training, 340 images were used for validation and 340 images where used for testing. Where the image classification data set was already provided with 3 set of separate text files with details for train, validation and test images. I choose first set for the assignment.(Train set 1, Test set 1, Validation set 1)

Object Detection:

The Data set that was provided for object detection consist of Red Blood Cell Images where totally there were 686 images with the xml data set and 9 images without xml. These 9 images without xml files were used for testing where objects are deducted with bounding boxes created around it. Training data set were separated into 274 images and validation set constitutes of 69 images. PASCAL VOC is the most commonly used data format. Training your model needs location of objects which is specified in XML file since tensor flow uses TF records instead of xml. So, we convert xml to csv and csv to TF records.

Sample data set image for Object Detection



3. Proposed CNN Architecture for Image Classification:

a. Baseline architecture used **VGG-16** without transfer learning

Baseline Architecture that was used for image classification was **vgg-16** which was created from scratch without using any transfer learning where we had 5 convolution layers, and each convolution layer was succeeded by pooling layer. We have 3 dense layers. Final dense layer was provided with **17 output neurons** and the activation function for the **output layer** was **SoftMax. Activation function** used for other layers are **'Relu'**. Input shape of the image tends to be (224,244). **Optimizer** used was **RMS prop**. **Loss** used was **categorical cross entropy.** Batch size was 20. Steps_per_epoch is 34(Train images/batch size (680/20)) and validation_steps is 17 (Validation images/ batch size (340/20). Image Generator is used to classify the images with the labels as well as it was used for image augmentation. Class mode was **categorical** since, it had 17 different classes.

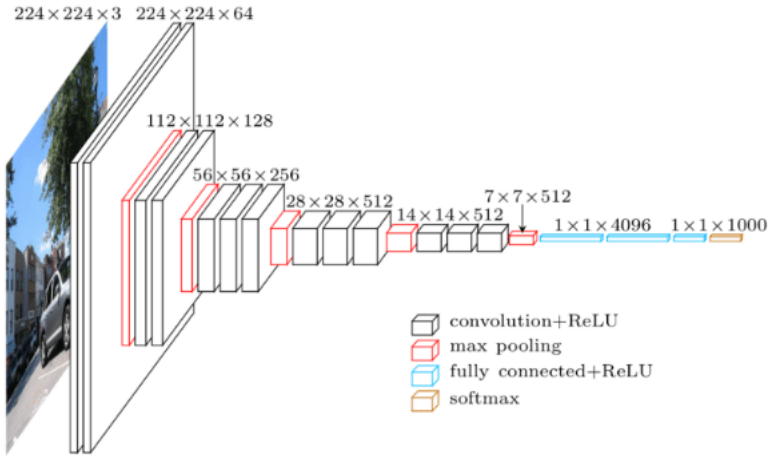


Fig: Baseline vgg-16 architecture

b. Customized architecture

Customized architecture is made on top of the baseline architecture of vgg-16 model. Customized Architecture contains 3 convolution layers in the first block and second block with filter size to be 64 with image augmentation and dropouts with 0.5 added at the end of convolution block 2, block 3, block 4, block 5 and final drop out added after the first dense layer.

c. Assumption/intuitions:

Drop out is added to reduce the overfitting of the model by using regularization technique. Convolution layer 1 generally provides low level features like edges from raw pixels. So, adding one more convolution layer with 64 filter size to the first convolution block can extract edges better from raw images. As well as dropouts with value 0.5 randomly dropouts 50% of nodes during training the model. Layer two of convolution block generally detects simple shapes from edges detected from first layer so I thought to add one more convolution layer to block 2 with 64 filter size to get more shapes in order to provide better accuracies.

d. Model Summary:

Baseline Architecture:

13 convolution layers

5 max pooling layers

3 dense layers: First 2 dense layers with 4096 intermediate nodes and last dense layer with 17 output nodes.

Total parameters: 134,330,193

Trainable parameters: 134,330,193

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 112, 112, 64)	0
conv2d_3 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_4 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 56, 56, 128)	0
conv2d_5 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_3 (MaxPooling2	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_4 (MaxPooling2	(None, 14, 14, 512)	0
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 4096)	102764544
dense_2 (Dense)	(None, 4096)	16781312
dense_3 (Dense)	(None, 17)	69649

Total params: 134,330,193
Trainable params: 134,330,193
Non-trainable params: 0

Customized Architecture:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 224, 224, 64)	36928
conv2d_3 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 112, 112, 64)	0
conv2d_4 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_5 (Conv2D)	(None, 112, 112, 128)	147584
conv2d_6 (Conv2D)	(None, 112, 112, 64)	73792
max_pooling2d_2 (MaxPooling2	(None, 56, 56, 64)	0
dropout_1 (Dropout)	(None, 56, 56, 64)	0
conv2d_7 (Conv2D)	(None, 56, 56, 256)	147712
conv2d_8 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_9 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_3 (MaxPooling2	(None, 28, 28, 256)	0
dropout_2 (Dropout)	(None, 28, 28, 256)	0
conv2d_10 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_11 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_12 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_4 (MaxPooling2	(None, 14, 14, 512)	0
dropout_3 (Dropout)	(None, 14, 14, 512)	0
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_14 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_15 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 512)	0
dropout_4 (Dropout)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 4096)	102764544
dropout_5 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dense_3 (Dense)	(None, 17)	69649

Total params: 134,293,457
Trainable params: 134,293,457
Non-trainable params: 0

4. CNN Architecture for Object Detection:

a. Faster RCNN

Faster RCNN pipeline provides information about the training model configuration such as:

Model used such as Faster_RCNN, number of classes: 1 , feature extractor type: **faster_rcnn_inception_v2, first stage feature stride:16** , **score converter** used is **SOFTMAX**, Major configurations include **fine_tune_checkpoint** which is used to save checkpoint for the model, **train_input_reader** which saves the path where the **train record** as well as **label map for training record** are present, **eval_input_reader** which saves the path of **test record** as well as **label map** of test record are present.

Convolution base used is **"faster_rcnn_inception_v2"**.

b. SDD (Single Shot detector):

SSD pipeline provides information about the training model configuration such as:
Model used such as SSD, number of classes: 1 (since we classify only the red blood cells), Major configurations include **fine_tune_checkpoint** which is used to save checkpoint for the model, **train_input_reader** which saves the path where the **train record** as well as **label map for training record** are present, **eval_input_reader** which saves the path of **test record** as well as **label map** of test record are present. similarity calculator, anchor generator, image Resizer, box predictor, feature extractor: Type – **ssd mobile net v2**.

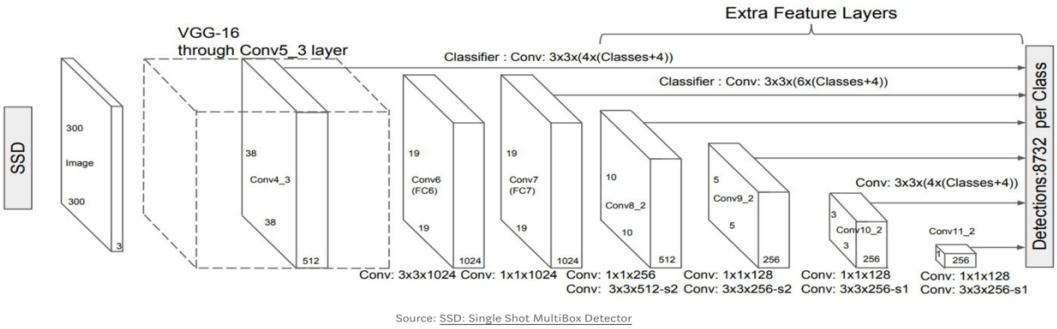
Convolution base used is **"ssd mobile net v2"**.

c. Assumption/intuitions:

Based on my assumption Faster RCNN provide better accuracy than SSD. Because when comparing loss steps final loss is calculated to be 0.5481 for Faster RCNN and for SSD the final loss is calculated to be 2.6982. Even though SSD takes a quick time to train the model still it needs 10,000 training steps to give a loss of 2.6982. Where as Faster RCNN takes a lot of time but still provides better accuracy with less loss of 0.5481.

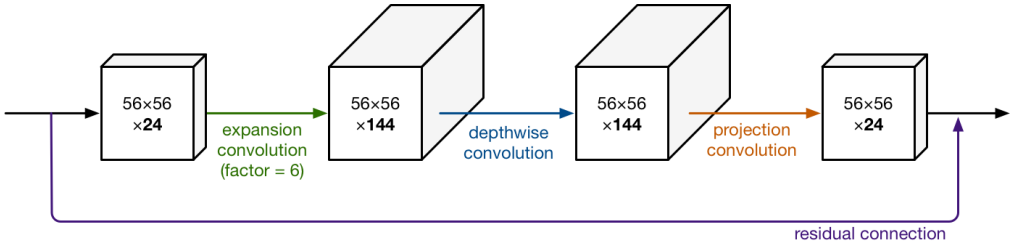
d. Model Summary:

Provided are the model summary of two different architectures used on object detection.

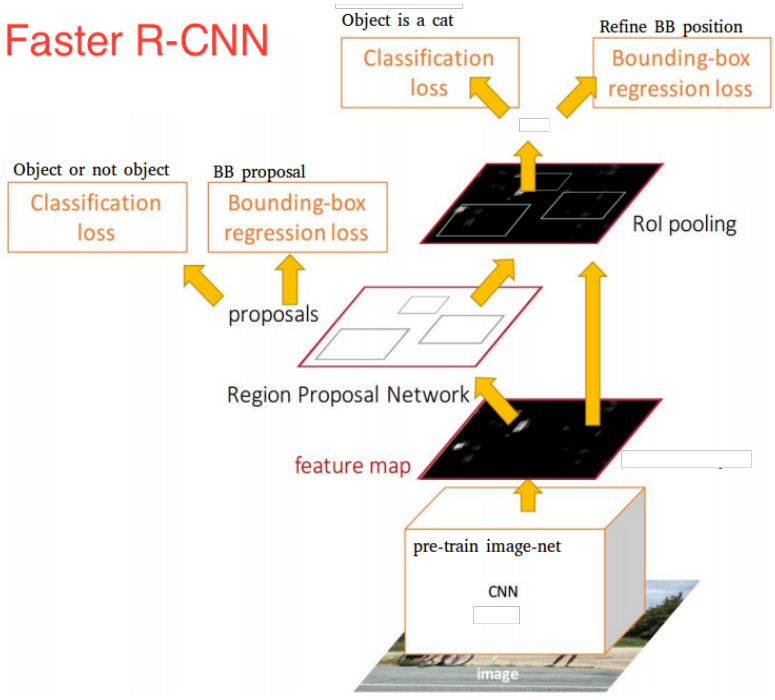


Source: SSD: Single Shot MultiBox Detector

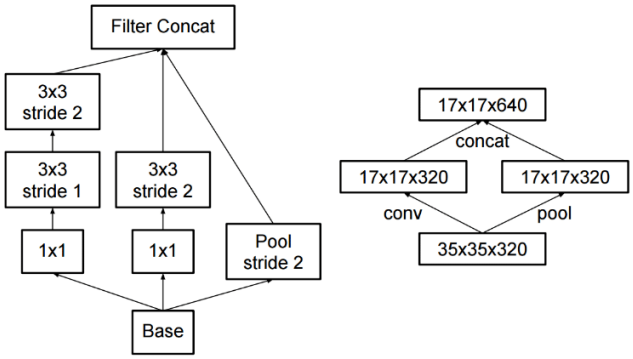
Mobile net version2: architecture used on ssd



Faster RCNN:



Inception v2 architecture used on Faster-RCNN:



5. Experimental results and discussion:

a. Experimental settings:

i. Image Classification:

Model used: vgg-16
Hyperparameter settings:

Image shape: 224*224
Activation function: Relu
Output layer Activation function: SoftMax
Loss fn: categorical_crossentroph Qy
Optimizer: RMS prop
Epochs: 100
Class mode: categorical
Steps_per_epoch: 34
Validation_steps: 17

1. Customization 1:
   Adding drop out 0.5 at the end of block 4, block 5 and after first dense layer:

	Accuracy	Loss
Train	**83.97%**	**58.95%**
Validation	**73.82%**	**1.2551**
Test	**81.17%**	**82.15%**

2. Customization 2:
   Customization 1 + Adding a convolution layer on the first block with filter sized 64 and a drop out after 3rd block of convolution.

	Accuracy	Loss
Train	**81.32%**	**55.14%**
Validation	**65.88%**	**1.4784**

Test	**76.47%**	**35%**

3. Customization 3: **(Final best customized model)**

   Customization 1 + customization 2 + adding a new convolution layer with 64bit filter size on the second block of convolution and a drop out after 2nd block of convolution layer

	Accuracy	**Loss**
Train	**81.18%**	**59.65%**
Validation	**71.18%**	**1.2038**
Test	**80.00%**	**45%**

4. Transfer Learning of vgg-16 using ImageNet data set:

   Epochs: 10

	Accuracy	**Loss**
Train	**100%**	**14.97%**
Validation	**87.06%**	**.08**

5. Customization 1- done on vgg-16 with transfer learning as base model

   Adding a convolution layer (64 filter size) with max pooling layer with a drop out, image augmentation and a dense layer with 512 intermediate nodes. Epochs = 20

	Accuracy	**Loss**
Train	**0.0456**	**83.34%**
Validation	**0.0588**	**83.32%**

6. Customization 2- done on vgg-16 with transfer learning as base model

   Image augmentation and a dense layer with 512 intermediate nodes. Epochs = 10

	Accuracy	**Loss**
Train	**79.71%**	**64.60%**
Validation	**82.94%**	**67.19%**

7. Customization 3- done on vgg-16 with transfer learning as base model

Adding a convolution layer (64 filter size) with max pooling layer and a dense layer with 512 intermediate nodes. Epochs= 10

	Accuracy	Loss
Train	**72.79%**	**86.70%**
Validation	**71.8%**	**1.0109**

Out of all the experimental settings base model which is created from scratch performed better when comparing it with transfer learning model created and base model also provided better accuracy with less loss percentage even after customization. So, 3rd experimental setting used was considered to be final best customized model because it provides better accuracy and less loss percentage when compared to other experiments.

i. Object Detection:

Faster RCNN:

Hyper parameters are **number of evaluation steps: 50, number of training steps: 1500**. Training configuration with **batch size** to be **12, optimizer** to be **momentum optimizer,** Regularizer to be **12 regularizer** and the weight to be 0.0. Only one data augmentation was used such as **random_horizontal_flip.**

SSD:

Hyper parameters are **number of evaluation steps: 50, number of training steps: 10,000**, **activation function: Relu_6**, Regularizer to be **12 regularizer** and the weight to be 0.00004. Training configuration with **batch size** to be **12** and **optimizer to be rms_prop optimizer**. Two data augmentation such as **random_horizontal_flip** and **ssd_random_crop.**

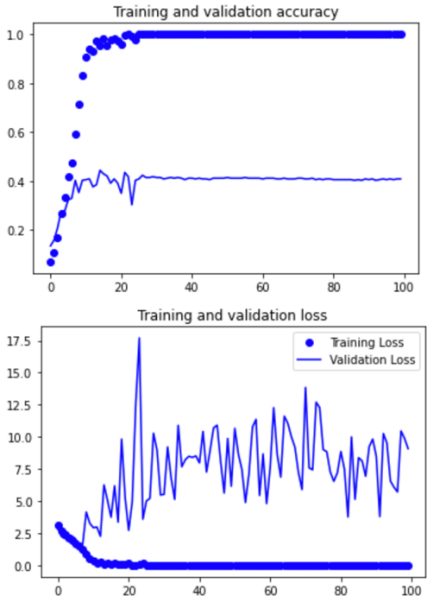
Transfer learning used was object Detection API from tensor flow
Pretrained model had 90 classes where the model was trained for pets' dataset. Pretrained model consists of information such as checkpoint, frozen_inference_graph, pipeline.config file, saved model as well as model checkpoint saved in 3 formats of data, index and meta.

b. Experimental results:

i. **Image classification:**

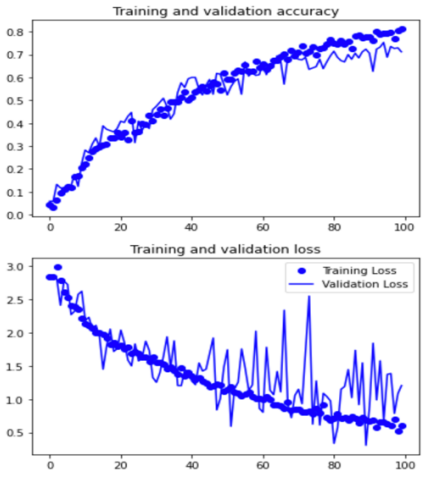
1. **Performance on baseline/standard architecture**

	Accuracy	Loss
Train	**100%**	**8.7654**
Validation	**40.88%**	**9.0811**
Test	**47.05%**	**5.4286**



2. **Performance on customized architecture**

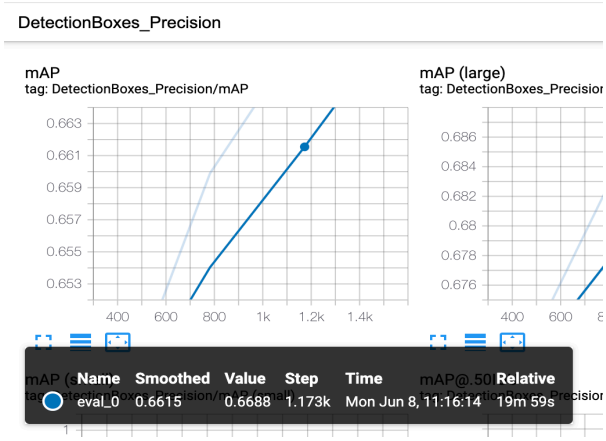
	Accuracy	Loss
Train	**81.18%**	**59.65%**
Validation	**71.18%**	**1.2038**
Test	**80.00%**	**45%**



ii. Object Detection:

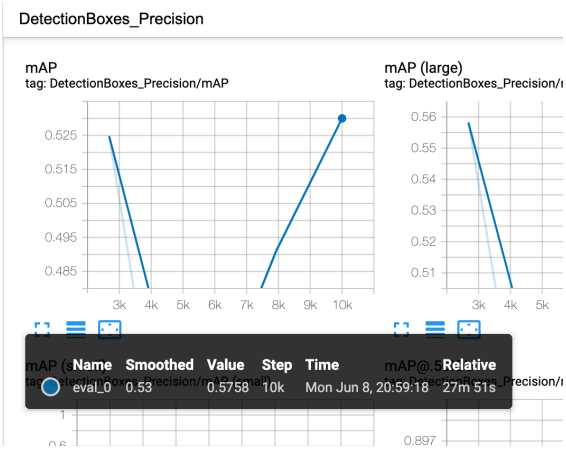
1. Performance on Faster-RCNN

Mean Average Precision for training model is 0.6
Final loss step: 0.5481



2. Performance on SDD or any object detector

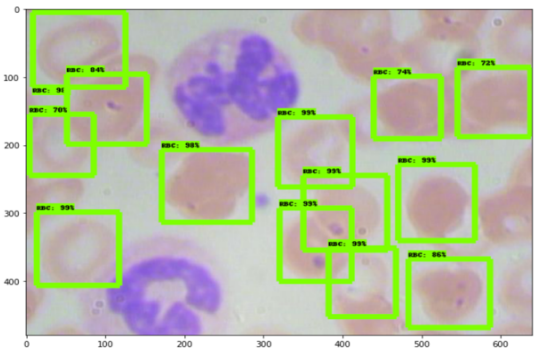
Mean Average Precision for training model: 0.57
Final Loss step: 2.6982



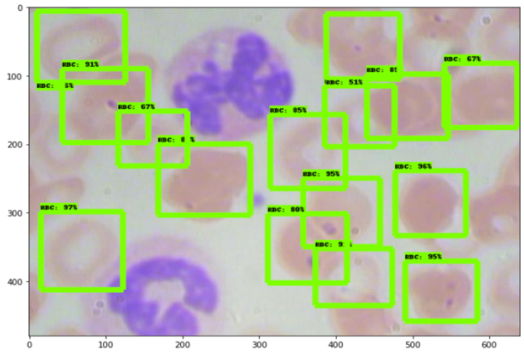
iii. Discussion:

Object Detection: Testing Sample

SSD:



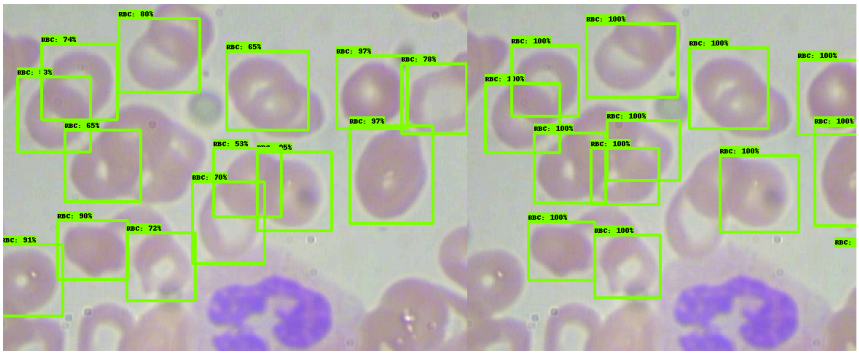
Faster Rcnn:



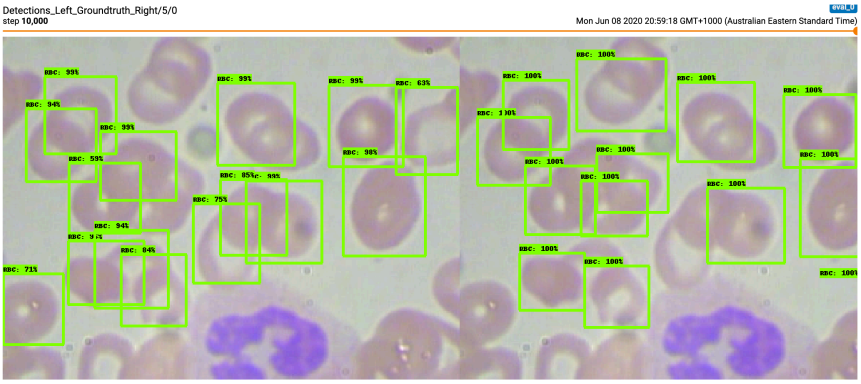
On comparing the testing samples provided by Faster RCNN and SSD it is clear that Faster RCNN provides larger bounding boxes and better accuracy than SSD.

Validation samples:

SSD:



Faster Rcnn:



6. Conclusion

Based on the experiments conducted on object detection and Image classification algorithms, first and fore most I learned how different CNN architecture can be used such as Alex Net and VGG-16. I learned how each layer works when I did the experimental settings of customizing the base model. Where I knew how to use check points to save the training model. Also how does each layer output affect succeeding layers feature extraction during customisation. Overfitting model was handled by data augmentation and regularisation technique such as dropouts. and the accuracies were measured. In case of object detection two architectures were used Faster RCNN and Single Shot Detector in order to locate object in an image. Where Faster RCNN provided better results with more bounding boxes when compared to Single short Detector which provided higher loss. I learned how to train a model with already pre-trained model which was available from TensorFlow. From the basic information of providing data to the model where the data is fed as TF records since we are using model provided by tensor flow to configurations on training model and obtaining frozen inference graph on the trained model with which we evaluate the testing model. On the experiments conducted gave me a clear understanding of how different CNN architecture and object detection algorithms work.