Ernest Ilustre 12763239

Assignment2-42028-DL-CNN-AUT2020

1. Introduction

In this assignment I was tasked to create a base CNN model and a customized model wherein I used transfer learning and added a few layers. The choices for my CNN models are VGG16, InceptionV3, and ResNet50. These are the base CNN models I have chosen to use for the weather dataset wherein I have to classify multiple classes, and, in this dataset, there are 4 different classes. I will continue to describe the dataset in the next section of this report.

For the object detection section of the assignment I chose to use Faster-RCNN and Single Shot Detector models in order to complete the task. The task is to identify red blood cells in images and compare the performance of both Faster-RCNN and SSD. For the Faster-RCNN model I chose inceptionV2 as the backbone and for SSD I chose mobilenetv2.

This report will contain explanations, findings, hypotheses, and such to show my learnings from this assignment.

2. Datasets

2.1 Weather Dataset (Multi-class Image Classification)

The weather dataset consists of 1,125 images in total and is split into four different classes. In order for me to use this dataset for multi-class image classification, I split the dataset into a training, testing, and validation set, and each set consists of the four classes which are cloudy, rain, shine, and sunrise. Training set consists of 80% of the total images and both testing, and validation have 10% each, respectively.

Figure 1. hot encoded vectors as labels by using keras

Each class is automatically labeled by *keras* as hot-encoded vectors this allows the model to recognize the classes of each image. In Figure 1 we can have an idea that [0,0,0,1] is sunrise, [0,1,0,0] is rain, and [0,0,1,0] is shine. From this we can conclude that [1,0,0,0] would be the hot encoded vector for cloudy.

2.2 Red Blood Cell Dataset (Object Detection)

For the object detection dataset, we were given 352 images of red blood cells with their corresponding .xml files which consists of the image's labels and bounding boxes. While splitting the dataset into a training and test set, I noticed that some images didn't have their corresponding .xml files so I used *labelimg* to annotate and create bounding boxes in the Pascal-VOC format for the images that were missing .xml files. I split the dataset into an 80-20 ratio with 80% of the images for training and 20% for testing.

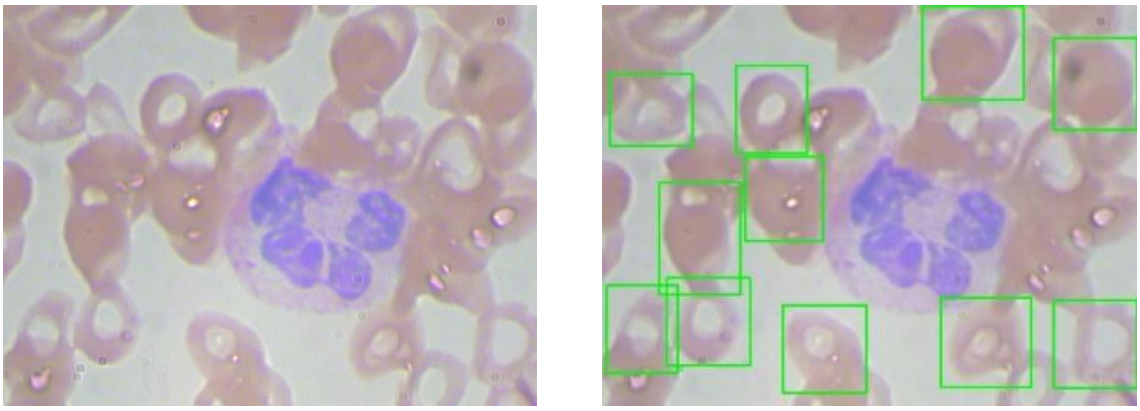

Figure 2. image from the RBC dataset with and without bounding boxes

3. Proposed CNN architectures

I decided to use three different CNN models for this project just to be able to compare each one and have a greater insight about the architecture of each, I used the same hyperparameters for each model to have a comparison.

3.1 VGG16

The baseline architecture of the VGG network is the simplest one I used for this project; it contains 3x3 convolutional layers which are stacked on top of each other. This architecture was a "pioneer" in deep learning back when it was first created.

In order for me to customize this architecture I used a pre-trained model which had weights from *Imagenet* and froze all the layers and used the last 4 layers, also I added a fully-connected layer which I will explain further in the experimental settings and results section of this report

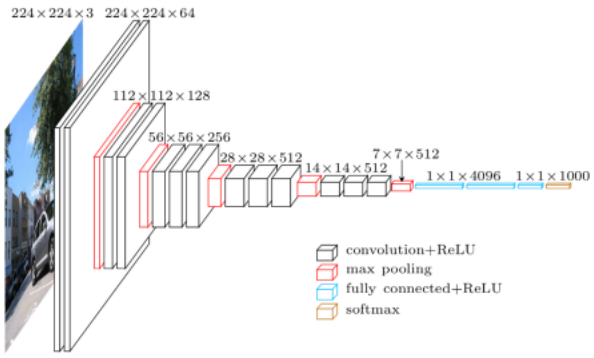

Figure 3. VGG architecture

As for my assumptions for the VGG architecture as compared to the other two models are that it is going to be training much slower than both just due to the parameters it uses, for the VGG16 models uses 138,357,544 parameters.

3.2 ResNet50

ResNet's basic architecture is very state of the art due to the fact that it doesn't have a problem with vanishing gradients unlike the traditional sequential neural networks. ResNet can also overcome the depth of deep learning by the use of *identity shortcut connection*, which create blocks used to create the network.

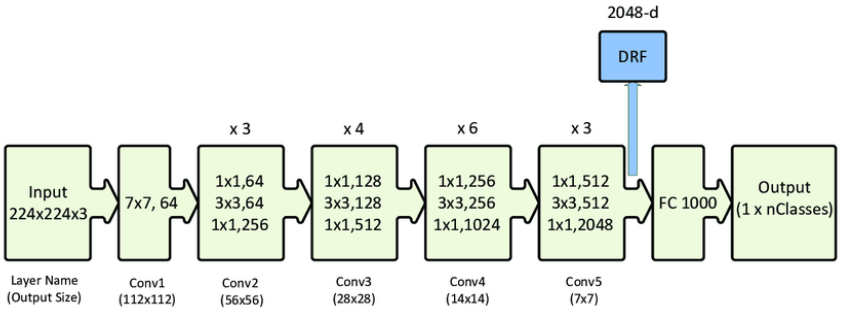

Figure 4. ResNet architecture

For customizing the ResNet50 architecture, I used transfer learning, by using *imagenet* as weights, freezing all the first layers and using the last 7 layers as well as adding a fully-connected layer. ResNet50 has a total of 25,636,712 parameters which is way less than VGG16.

3.3 InceptionV3

Inceptionv3 was introduced by GoogLeNet, it is known to factorize convolutions by reducing the number of parameters and has deep network of 42 layers. The objective of inception models is to be a "multi-level feature extractor" this is possible by inception models being able to calculate 1x1, 3x3, and 5x5 convolutional layers in the same network.

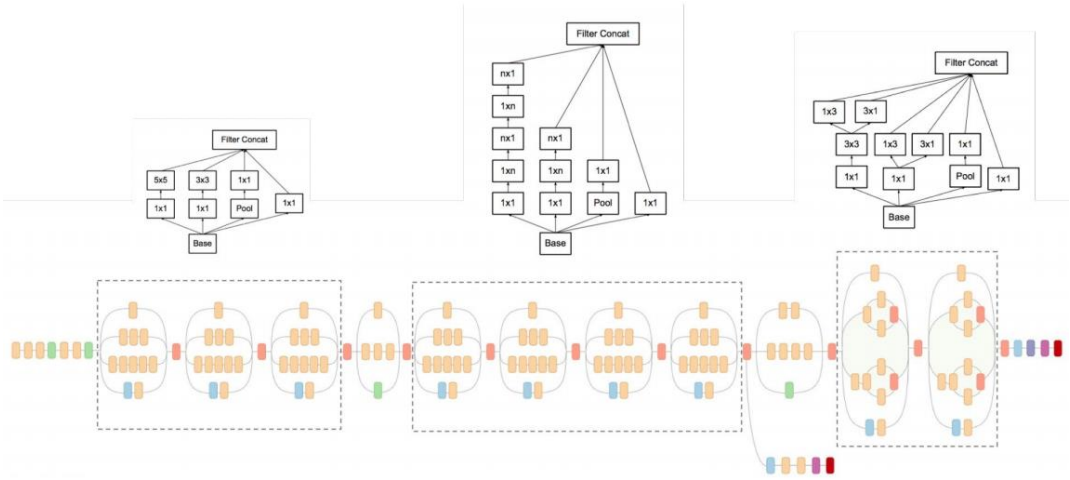

Figure 5. InceptionV3 architecture

For the customization of the InceptionV3 model I again used transfer learning, by using *imagenet* as weights, but this time freezing the first few layers and using the last 249 layers, and lastly adding a fully

connected layer. InceptionV3 architecture has a total of 23,851,784 parameters. It has the least number of parameters as of the 3 CNN models I am using for this project.

4. CNN architectures for Object Detection

Both Faster-RCNN and Single Shot Detector are known models for object detection, in this section I will be describing both. As mentioned earlier I have used inceptionV2 as the backbone for my Faster-RCNN model and mobilenetV2 for Single Shot Detector.

4.1 Faster-RCNN

Faster-RCNN comprises of three parts; the first part is where images are represented as tensors and pass through a pre-trained CNN where the image's features are extracted, the second part is known as the regional proposed network, this is where objects and bounding boxes are detected in the image or not, and the last part is finally where classes and bounding boxes are predicted that came from the region proposal network.

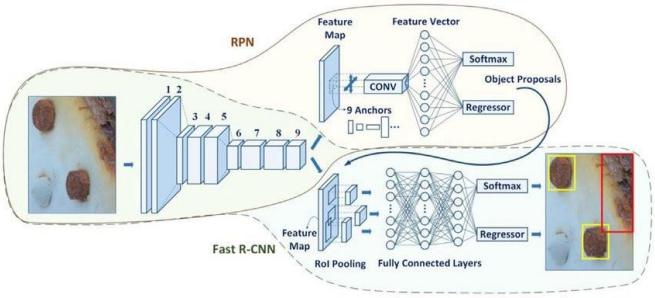

Figure 6. Faster-RCNN architecture

4.2 Single Shot Detector

Single Shot Detector on the other hand has 2 parts instead of three, it is quicker when it comes to detection as compared to Faster-RCNN due to the fact of removing the region proposal network present in the Faster-RCNN architecture. The first part of SSD is where features are extracted from the images and the second part is wherein convolutional filters are applied.

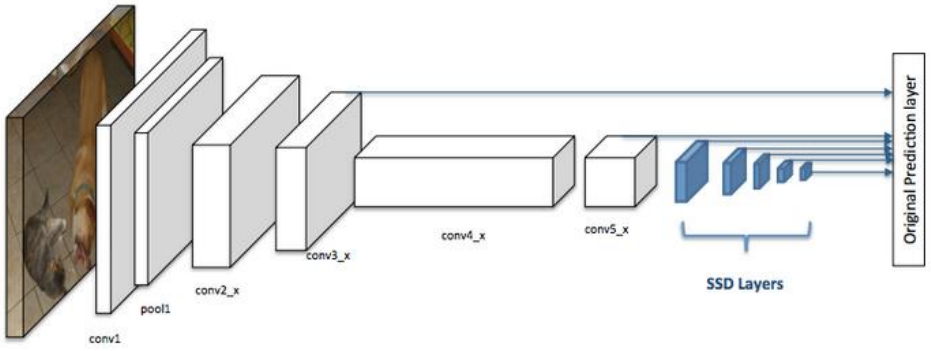

Figure 7. Single Shot Detector architecture

4.3 Assumptions

I believe that training using Faster-RCNN will takes less time to gain desired results due to the nature of its architecture as compared to SSD. Also, the accuracy of Faster-RCNN will be higher than SSD. SSD will only have an advantage over Faster-RCNN when it comes to real-time object detection, since the project requires detection of regular sized objects in images, though both models will provide good results, I believe Faster-RCNN will be more efficient for this project.

5. Experimental Settings and Results

In this section I will explain in detail the modeling settings I have used and corresponding results for multi-class image classification and object detection as well as my results from my experiments.

5.1 Experimental Settings

5.1.1 Multi-class Image Classification

For the image classification I set the same hyperparameters for all my models, baseline and customized. I used Image Data Generator, resized the images to 224x224 and used a batch size of 8 due to the size of the dataset.

```
#ImageGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
#datagen flow for multi-class
train_generator = train_datagen.flow_from_directory(train_path, target_size=(224,224),
    class_mode='categorical', batch_size=8)
valid_generator = test_datagen.flow_from_directory(valid_path, target_size=(224,224),
    class_mode='categorical', batch_size=8)
test_generator = test_datagen.flow_from_directory(test_path, target_size=(224,224),
    class_mode='categorical', batch_size=8)
```

Figure 8. Hyperparameters for my modeling

The loss function I chose for compiling the model was categorical crossentropy since it is a multi-class image classification problem. I also used Adam as my optimizer with a learning rate of le-5 for all the models for this optimizer gave better results as compared to RMSprop and SGD. Steps per epoch were set at 100, validation steps at 10 due to the size of the dataset, and 25 epochs while training.

For the baseline models of this assignment, I just used their bare architecture without adding any weights or layers to the models. As compared to the three customized models wherein I chose which layers to freeze and use, I also did transfer learning and fine-tuning by using *imagenet* as weights and added different layers per customized models. For inceptionV3 I added a dense layer with 1024 neurons with ReLu as the layer's activation, set dropout at 0.8 to avoid overfitting and underfitting, and used the last 249 layers. For my customized ResNet model I added two dense layers both having 512 neurons and ReLu

as activation, two batch normalization layers for speed and performance of the network, and two dropout layers set at 0.3. I also chose to freeze all layers except for the batch normalization layers for if it were kept frozen the model would overfit and finally chose last 7 layers to use. Finally, for my VGG16 model I only added one dense layer with 256 neurons and used ReLu as its activation and froze all the layers except for the last four. All three customized models each had an extra last dense layer with 4 neurons and softmax set as the activation.

5.1.2 Object Detection

While setting up the Google Object Detection API, instead of cloning the given Github repository from the tutorial, I decided to directly install the API directly to my Google Drive wherein the folders from the API, Red Blood Cell dataset, the generate_tfrecord.py file, xml_to_csv.py file, and all the required packages and proto files are found.

For both the Faster-RCNN and Single Shot Detector models I set the number of training steps to 10,000 and number of evaluation steps to 50. As mentioned in the earlier sections, for the Faster-RCNN model it has inceptionV2 as its backbone and the fine-tune checkpoint is a model that was pretrained with the COCO dataset. For the SSD model it has a mobilenetV2 as its backbone and the same as the Faster-RCNN model set the fine-tune checkpoint with a pretrained model with the COCO dataset. Both models only have 1 class to detect which is red blood cells.

5.2 Experimental Results

5.2.1 Performance of Baseline Architecture for Multi-class Image Classification

	Train acc/loss	Val acc/loss	Test acc
VGG16	**Acc**: 0.9296 **Loss**: 0.1954	**Acc**: 0.9125 **Loss**: 0.2529	**Acc**: 0.9017
ResNet50	**Acc**: 0.8550 **Loss**: 0.3825	**Acc**: 0.8500 **Loss**: 0.1719	**Acc**: 0.8303
InceptionV3	**Acc**: 0.8807 **Loss**: 0.3629	**Acc**: 0.9625 **Loss**: 1.2238	**Acc**: 0.8750

5.2.2 Performance of Customized Architecture Multi-class Image Classification

	Train acc/loss	Val acc/loss	Test acc
VGG16	**Acc**: 0.9600 **Loss**: 0.1464	**Acc**: 0.9125 **Loss**: 0.1316	**Acc**: 0.9196
ResNet50	**Acc**: 0.9146 **Loss**: 0.2114	**Acc**: 0.9375 **Loss**: 0.0147	**Acc**: 0.9553
InceptionV3	**Acc**: 0.9146 **Loss**: 0.2701	**Acc**: 0.9000 **Loss**: 0.3600	**Acc**: 0.8839

5.2.3 Graph Comparisons Customized vs Baseline Multi-class Image Classification

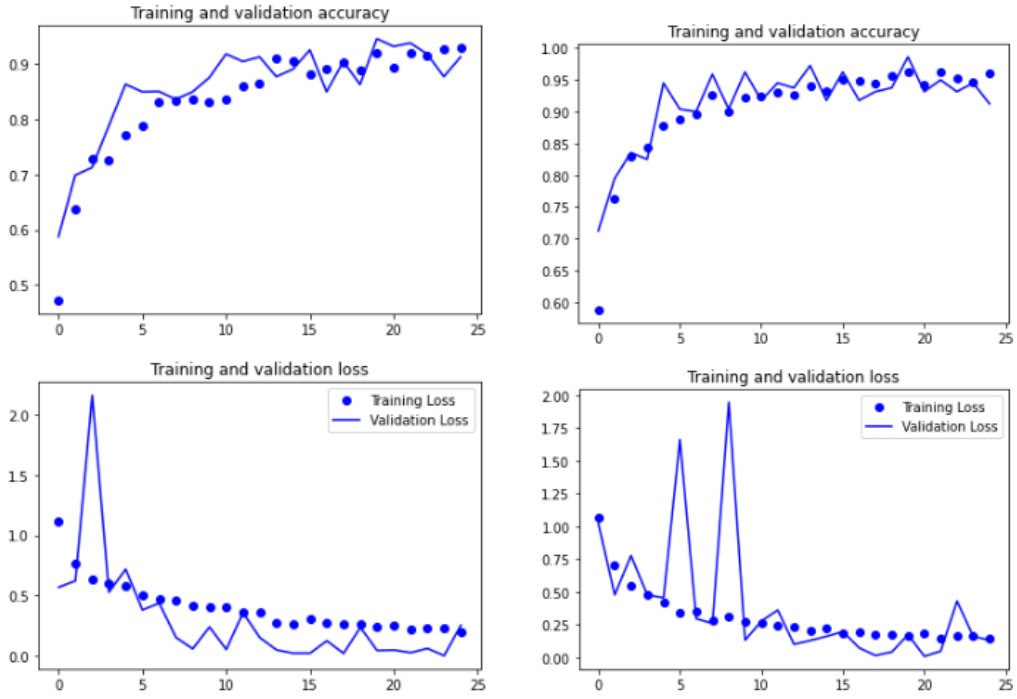

Figure 9. Baseline VGG16 vs Customized VGG16

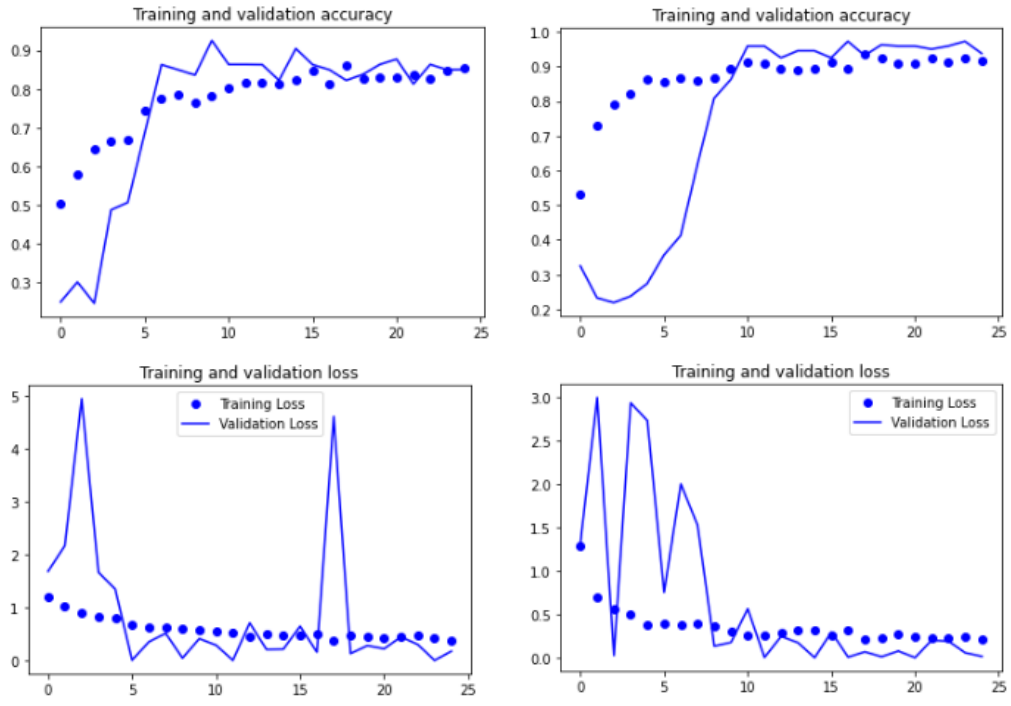

Figure 10. Baseline ResNet50 vs Customized ResNet50

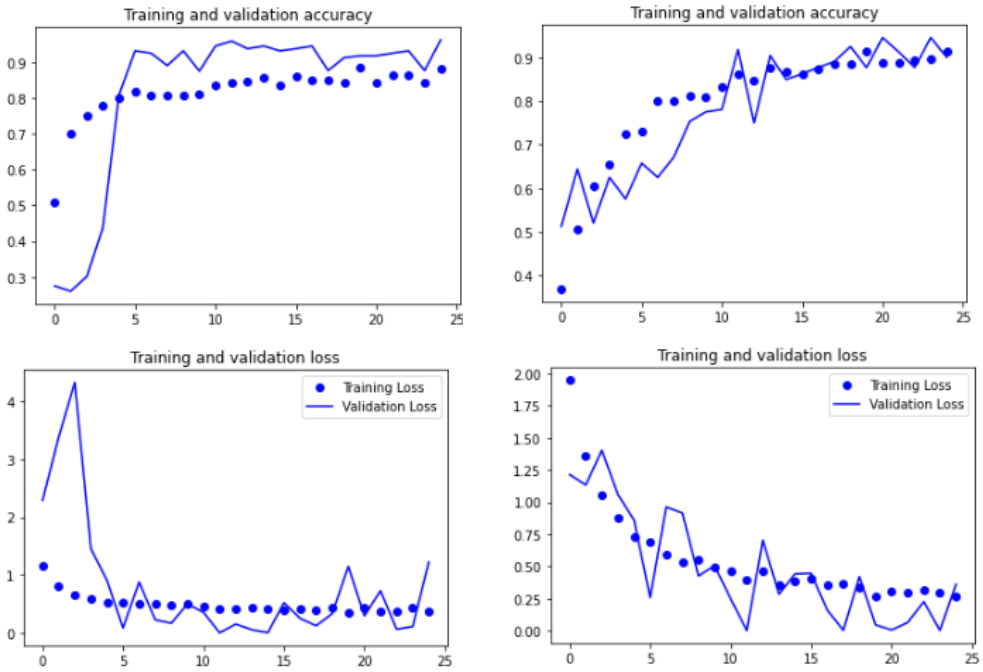

Figure 11. Baseline InceptionV3 vs Customized InceptionV3

Based on the results of my experiments it is evident that there is an increase in accuracy from the baseline model to the customized models, especially for the ResNet50 model. With regard to the accuracy and loss, the customized models seem to be more stable towards the end of training as compared to the baseline models.

5.2.4 Performance of Faster-RCNN and SSD for Object Detection

Based on the initial training of both models in Google-Colabotory Faster-RCNN immediately had a smaller loss as compared to SSD, the first 100 steps for Faster-RCNN was 1.0155 and at 674 steps its went down to .803 where I ended my training, while SSD started at a loss of 5.258 at the first 100 steps and at the end of training which was 4363 steps, the SSD model's loss was still at 3.987. While training both models simultaneously, Faster-RCNN had an average of 240 secs to train as compared to the 40 secs average to train the SSD model. I decided when to stop the models based on the loss graphs since the mean average precision graphs were not coming out on Tensorboard.

Figure 12. Faster-RCNN and SSD graphs from Tensorboard

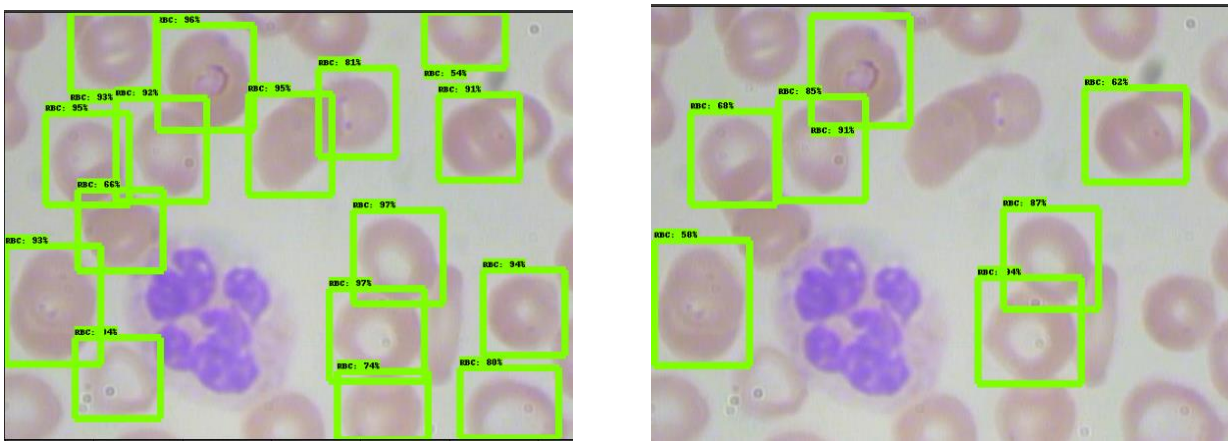

Figure 13. Results of the same image from Faster-RCNN vs Single Shot Detector

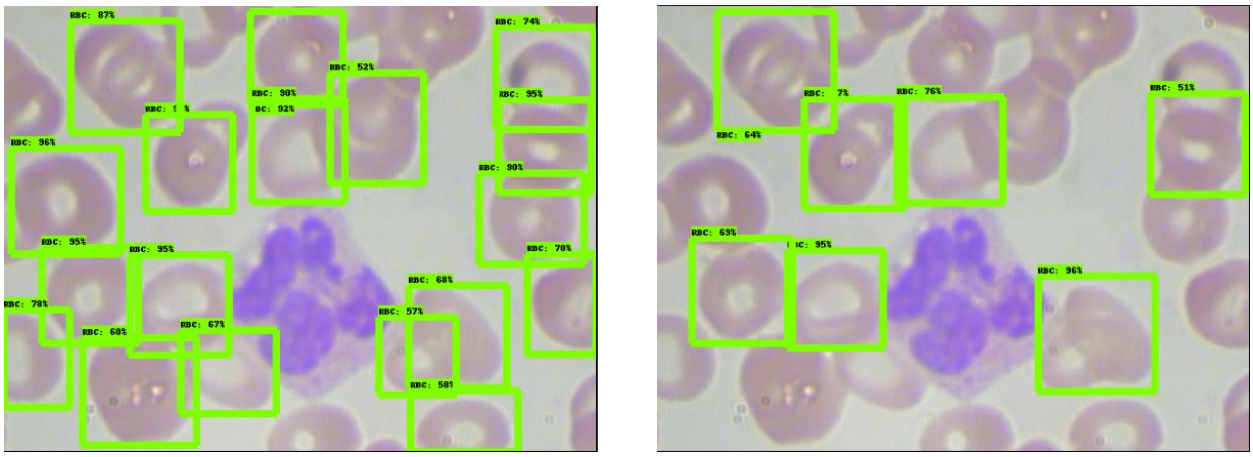

Figure 14. Results of the same image from Faster-RCNN vs Single Shot Detector

As seen in the figures above it is evident that Faster-RCNN has a higher accuracy and can detect more Red Blood Cells than SSD even if the SSD model was trained with more steps as compared to the Faster-RCNN model and also to note as seen in Figure 14 SSD misclassifies lumps of RBCs as one Red Blood Cell while Faster-RCNN is able to distinguish Red Blood Cells better.

5.5 Discussion

5.5.1 Multi-class Image Classification

I believe that there is some gap between train, validation, and test accuracies and losses because of the distribution of data and due to augmentation. Baseline models took much longer in training due to the fact of using all the layers and parameters unlike when I trained the customized models which trained faster. Adjusting overfitting and underfitting was also possible due to being able to customize fully connected layers. The accuracy for customized models were generally high as well because of the use of fine tuning, transfer learning, and again the customization of layers. Being able to customize layers allowed me to be more flexible and get what I was looking for in a sense. Though, there was not much difference in the results for both customized VGG16 and InceptionV3, there was a huge gap in performance in my baseline ResNet50 vs customized ResNet50 models. All customized models seemed to

perform much quicker and better right off the bat as compared to their baseline counterparts after figuring out the best parameters, layers, optimizers, and such.

5.5.2 Object detection

Comparing the performance and accuracy can easily be hypothesized based on each of their architectures and familiarity with both. Faster-RCNN's architecture sacrifices speed for accuracy, but for this project of building an object detector works great. Single Shot Detector on the other hand, takes more time to train, but for training time and real time detection speed is way faster than Faster-RCNN. The accuracy of Faster-RCNN proved to be way better than the SSD model even if it took less steps to finish, this may also be due to the backbone or feature extractor I chose for the SSD model.

6. Conclusion

With regard to multi-class image classification or any type of machine learning problem, understanding the dataset is the most crucial part, after understanding what you want to get or do you can than start picking different algorithms or models to solve your problem. In my case, being able to run and compare 3 different models gave me more of an understanding of each models' architecture and more ways to tackle different machine learning and deep learning problems.

The same case goes for object detection, having to use a model for real-time object detection is different from detecting objects in images. It depends solely on the use case and data you are using. Accuracy, though important, is not always what should be of priority in deep learning. Not because a certain model or architecture is proven to be faster or more accurate that that certain model should be used as there are a lot of things to consider from computational cost, time, and such.

In conclusion, aside from accuracy, performance, and different architectures, I have learned from this project are the essentials and fundamentals in machine learning and deep learning in general.

7. References

Hui, J. 2018, *'SSD object detection: Single Shot MultiBox Detector for real-time processing',* viewed April 4, 2020, <https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06>

Javier, 2018,*'Faster R-CNN: Down the rabbit hole of modern object detection'*, viewed April 3, 2020, .<https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/>

Rosebrock, A. 2017, *'ImageNet:VGGNet, ResNet, Inception, and Xception with Keras'*, viewed April 4, 2020,< https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/>