Inference Development for Classification and Object Detection.

Chuong Dao
riochuong@gmail.com

Abstract

In the areas of computer vision, object classification and object detection has been revolutionized by various type of state-of-the-art deep learning architectures especially Convolutional Neural Network. Training time and size of labelled data for a custom object detection can be reduced dramatically by leveraging transfer learning from pretrained model. This paper represents two experiences: building an object classification network that have inference time below 10 millisecond with provided training data from Udacity Robotics Nanodegree program, and building a customized object detection for stuffed toys using DetectNet architecture from NVIDIA. Both of the networks are builded and trained using NVIDIA GPU DIGITS platform provided by Udacity. The detection network is also deployed on the hardware NVIDIA Jetson TX2 for real-time detection using Logictech webcam C920.

Introduction

In recent years, convolutional neural networks (CNN) have transformed the field of computer vision in many different ways. Many state of the arts CNN architecture for image classification task like AlexNet, GoogLeNet, VGG16 have been designed and improved to achieve unbelievable accuracy on large corpus like ImageNet. On the other hand, YOLO and DetectNet architectures also prove that CNN can work great with detection task also. One of the fascinating idea is that we can leverage both the architecture and already-trained (pretrained) weights of these state-of-the-art networks to significantly reduce training time for a smaller customized tasks. This technique is called transfer learning and it can also help reduce the time for collecting data if you only need the networks to perform well on a smaller group of targets. This can be tremendously helps as users can evaluate various types of pretrained networks and pick the one that meet their specific requirements for latency and performance. The first experiment will illustrate how we can quickly accomplish more than 75% accuracy for image classification task using AlexNet with very little training time on the DIGITS platform. Similarly, the second experiment demonstrate how rapidly we can build a custom detection network to detect some specific stuffed animals with just around 416 labelled images using NVIDIA DetectNet architecture [4]. I also successfully deployed this detection network on Jetson TX2 and test it with real-time data from the Logitech C920 webcam and the result is very intriguing.

Background

1. Image Classification Task

Provided Dataset

The custom dataset for this classification task is supplied by Udacity Robotics NanoDegree program. The dataset has three classes of 10000 pictures which will be divided into three sets: train set (80%), validation set (15%), and test set (5%) (Figure 1).

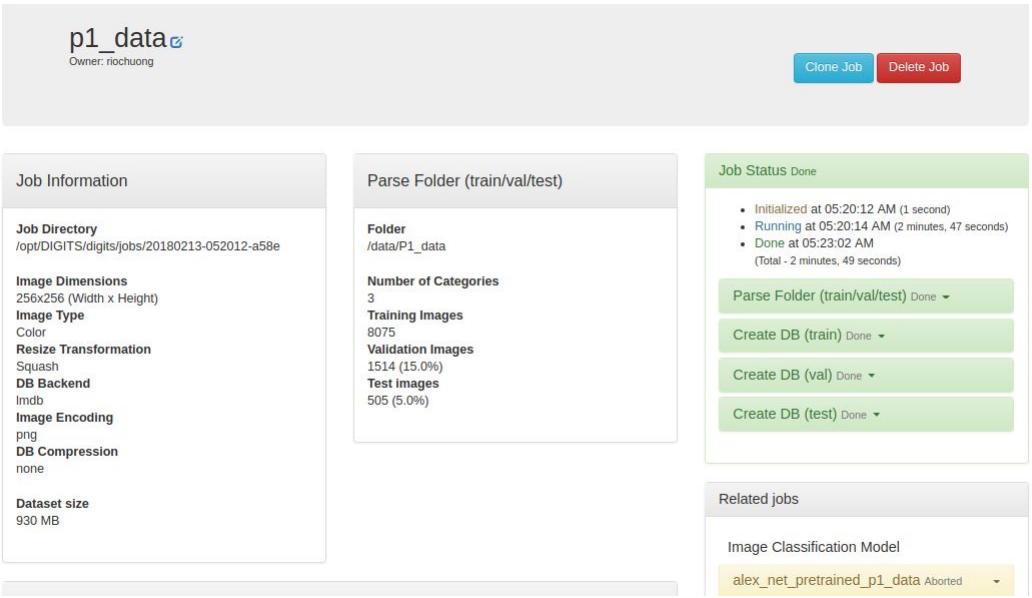

Figure 1: Supplied Data Set Configuration

Convolutional Network Architecture and Training Parameters

The Udacity DIGITS platform has various kinds of pretrained networks for classification task like. However,the decision is made after considering between AlexNet [1], GoogLeNet [2] and VGG16[3] to try network with architecture that are not too complex and have enough features to meet the accuracy requirements. AlexNet stands out because it has smaller number of layers and smaller number of parameters compared to the others so inference time can be fast. The “evaluate” program provided by Udacity Robotic can also help to check if the network meet the above-75-percent-accuracy performance. In addition, the original alexnet architecture also has a small modification to freeze all the weights of all the convolutional layers that are pre-trained with big dataset to help speed up training.
The training process on custom data only apply on all the fully-connected layers. As the provided dataset has three categories (candy, bottle, and nothing), the aboved custom

alexnet architecture is also added one more fully-connected layer (fc9 as in Figure 1) which has the output size of three to support the classification task.

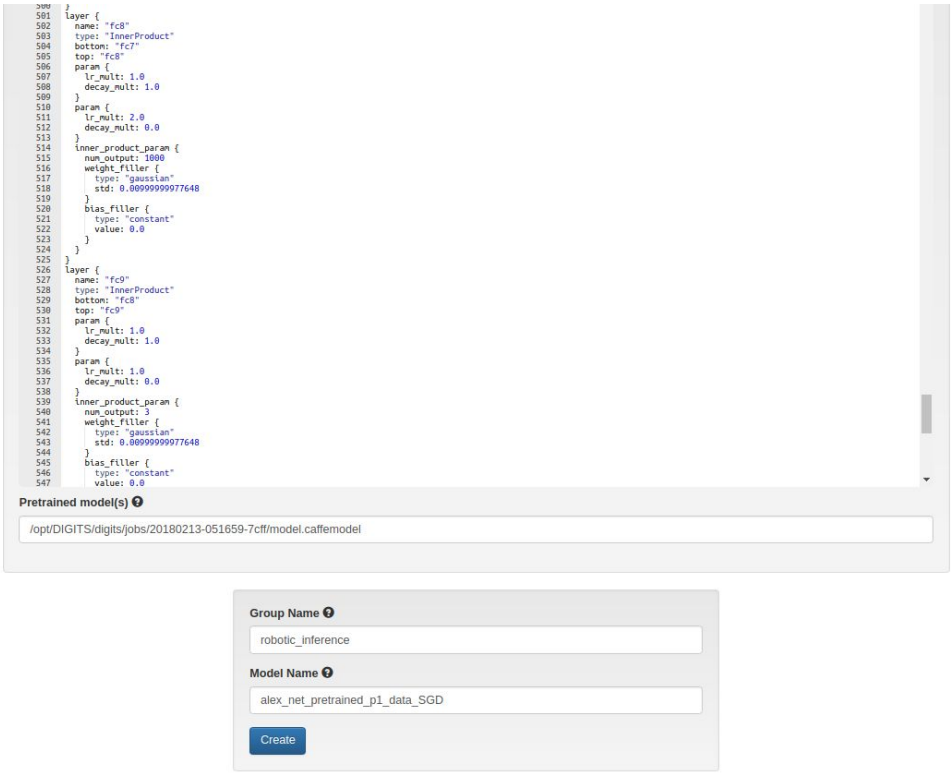

Figure 2: Adding Fully Connected Layer to the end

The training parameters are configured with learning rate of 0.005 and Stochastic Gradient Descent(SGD) as optimization method (Figure 2). Number of Epochs is selected at 30 but the process can be stopped in the middle if there is no improvement on accuracy and validation loss.

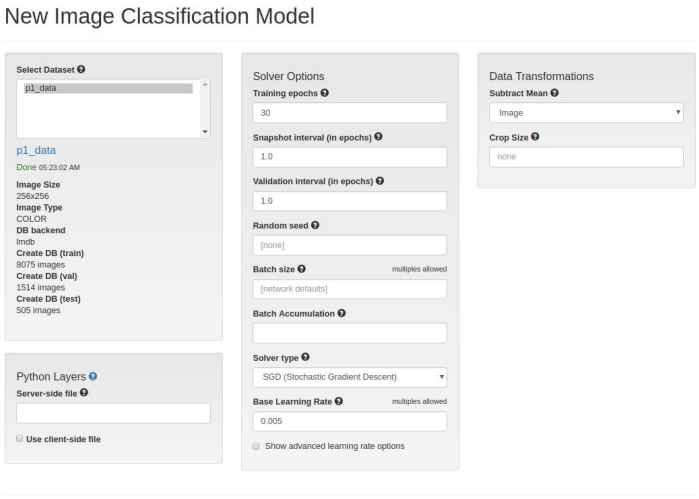

Figure 3: Training Configuration

Results

The training process reach the accuracy 100% and validation loss very close to 0 in only 3 epochs (Figure 4). The process is stopped at epoch seventh because of no dramatic improvement in accuracy and validation loss after third epoch.

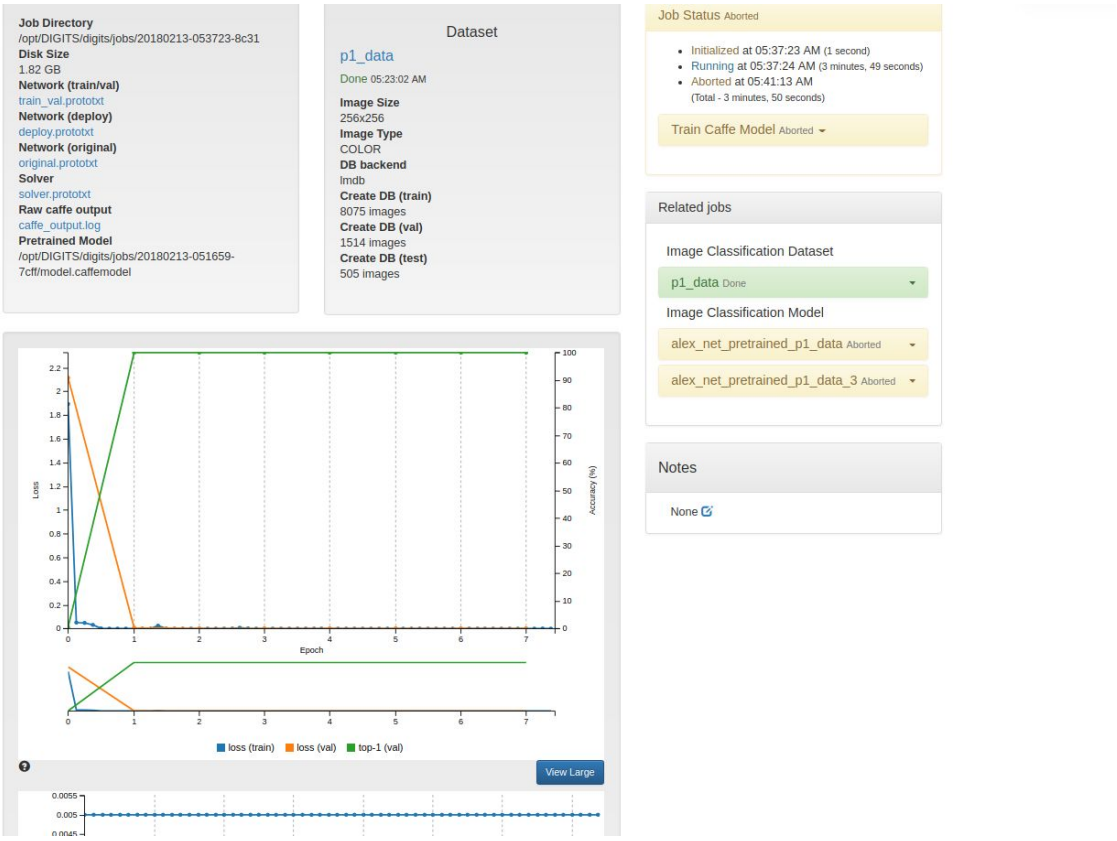

Figure 4: Alexnet Training Results

The newly trained network perform really well on the test set as it did not make any mistake on the test set based on the confusion matrix from the test set (Figure 5).

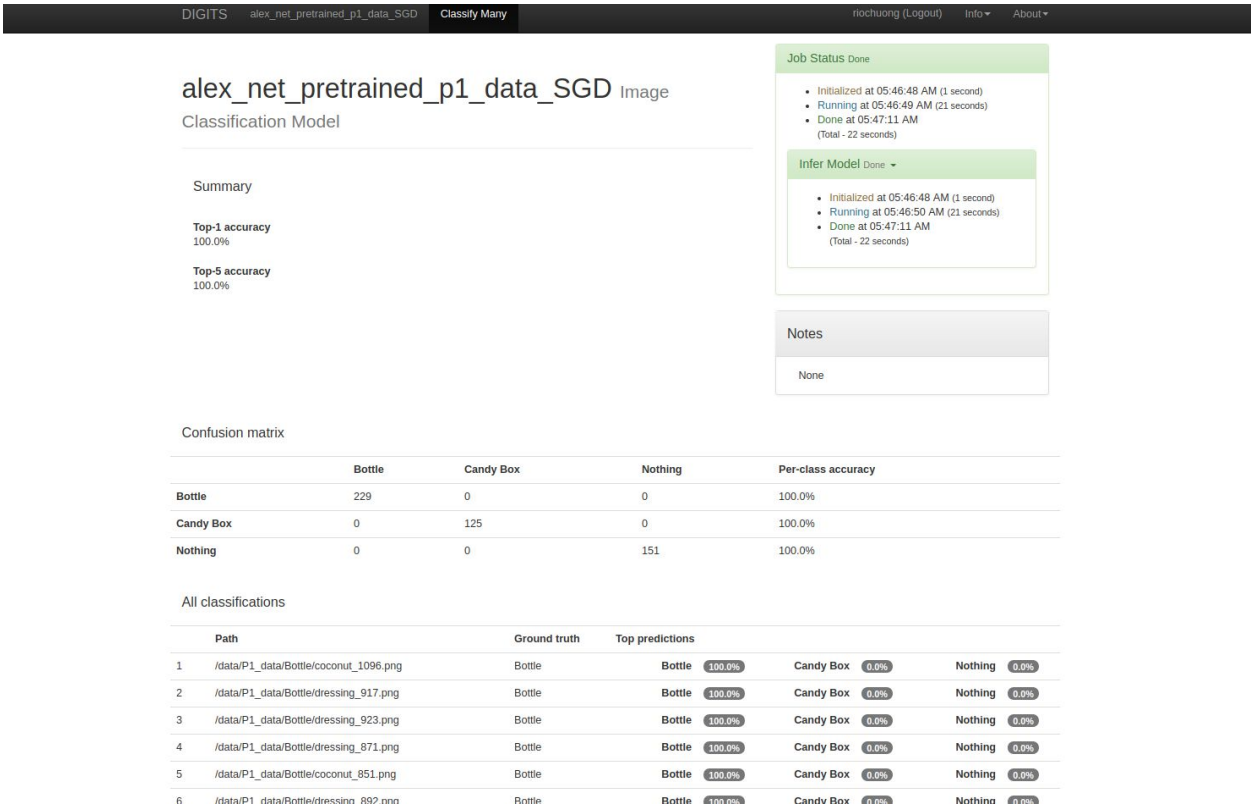

Figure 5: Custom Network Result on Test Set

Similarly, the “evaluate” command from Udacity also confirm the newly trained network achieve both latency inference requirement (less than 10 millisecond) and performance requirement (above 75%) as shown in Figure 6.

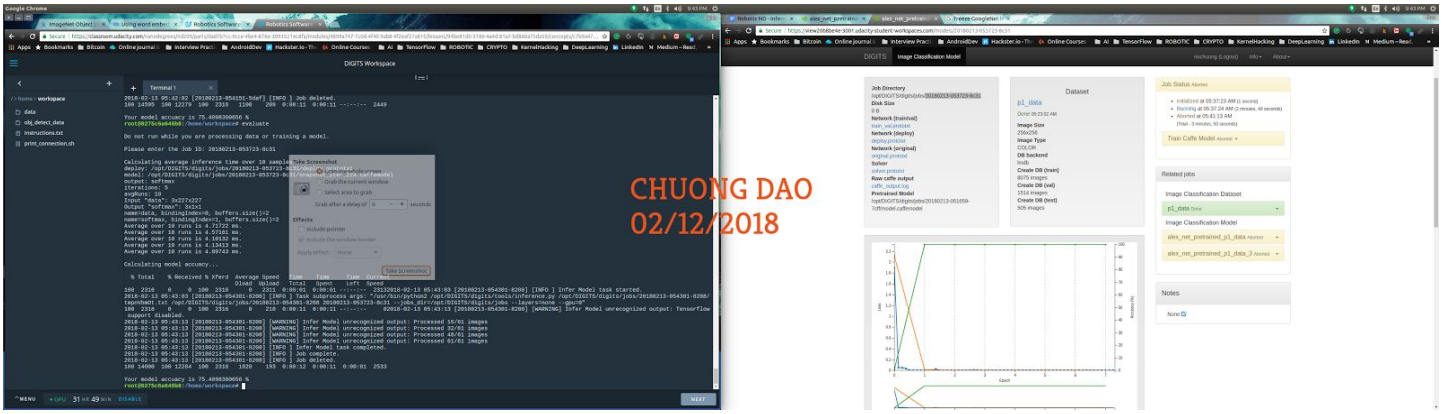

Figure 6: Evaluate Performance and Inference Latency

Discussion:

In my opinion, the newly trained model perform perfectly on the test set based on the confusion matrix information. However, it only achieves approximately 75.4% accuracy based on the “evaluate” program. It looks like the “evaluate” program might have a bigger data set that has some data the network might not have seen during training and validation. On the other hand, inference time is averaged around less than 4 millisecond so it meets the project requirements of under 10 milliseconds. I did some more experiments with more complicated type of network like GoogLeNet and VGG16 although I did not record the numbers for showing here. The training of these networks also reach the 100% accuracy very early. However, It looks like they did not outperform the pretrained AlexNet based on the “evaluate” program. Also, VGG16 inference time is about 22 milliseconds which also does not meet the requirements.

2. Object Detection Task

Data Collection

Images were taken using the camera of the Google Nexus 6. There are total of 416 pictures taken of 4 different types of stuffed animals (Figure 8). All images are resized to 640x640. Each image is labelled with KITTI format using the software *“AIp’s Labelling Tools for Deep Learning”* so it can be compatible with the DetectNet on DIGITS platform. The dataset is splitted up to 90% for training set and 10% for validation set. Also, about six images are set aside for manual verification of the network performance at the end.

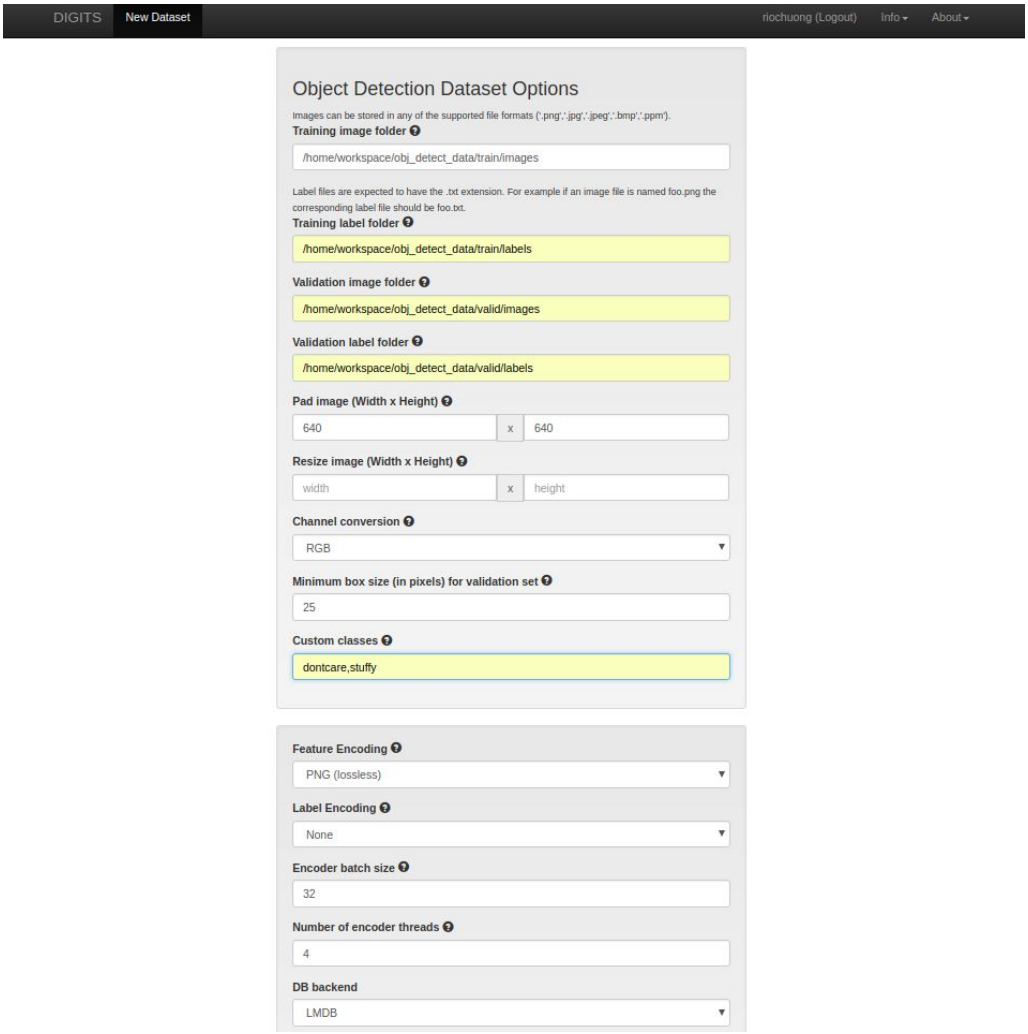

Figure 7: DetectNet Data Configuration

Figure 8 - Stuffed Animals as Detection Targets

Convolutional Network Architecture

According to NVIDIA [4], DetecNet has 5 parts:

1. Images and Labels will be divided in batch and fed to an augmentation layer that will feed the augmented images to the fully-convolutional network.
2. Detectnet leverage the fully-Convolutional network (FCN) architecture of GoogLeNet to extract feature from augmented training images and labels.
3. Loss functions are calculated for both object coverage and object bounding box corners per grid square.
4. At validation, images and labels are fed directly to the FCN without transforming and a set of predicted bounding boxes are generated by a clustering function
5. The mean Average Precision (mAP) is calculated to compare the the difference between model result and validation data.

Similar to classification task, a pre-trained model of DetectNet is used in this detection task to speed up training and help the network to only focus learning the feature of the new dataset only.

Training

Based on NVIDIA recommendation [4], the training process will use Adaptive Moment Estimation (ADAM) with exponential learning rate decay starting at 0.0001. Number of epochs are 200. It takes about three hours to finish the training process.

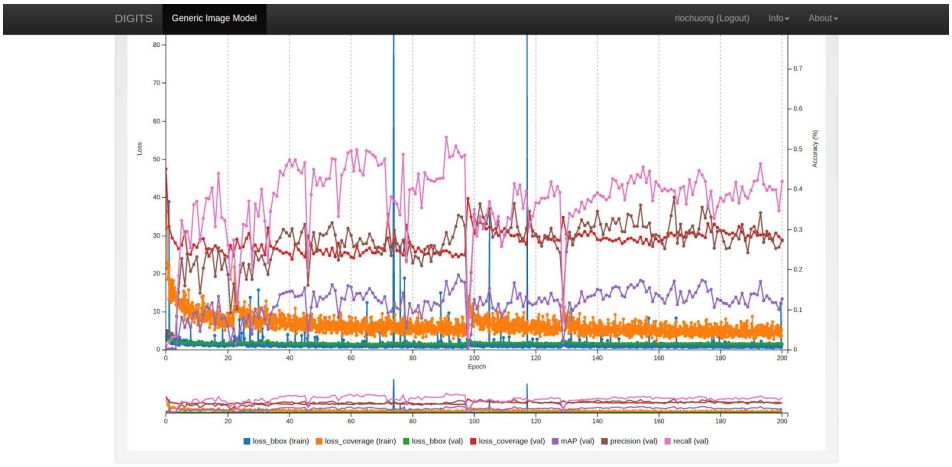

Figure 7: DetectNet Training Diagram

Results

The result of each epoch is compared based on mAP and precision to select the best model for deployment. The best result happened at epoch 154 with precision of 37.97 and mAP of 18.2.

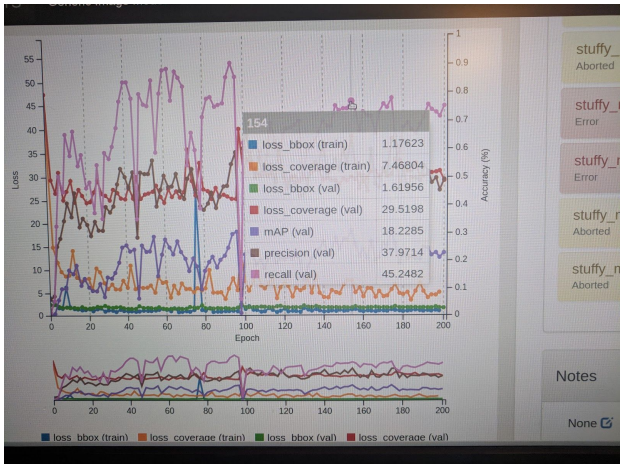

Figure 8: Epoch 154 Result.

Running this model on images of the test set proved that the network can detect the stuffed animals although not perfect (Figure 9 - 10).

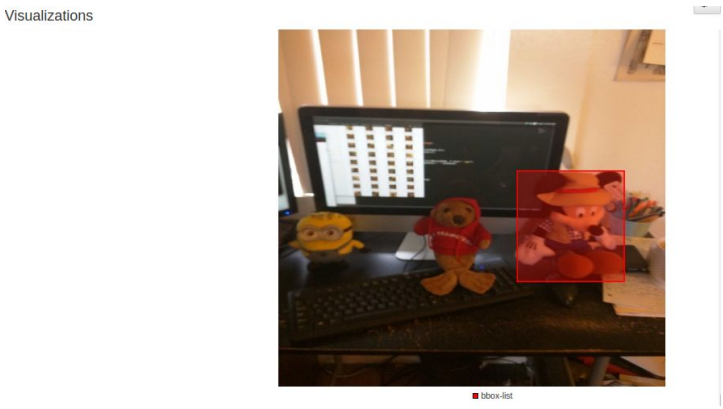

Figure 9: DetectNet on Test Image

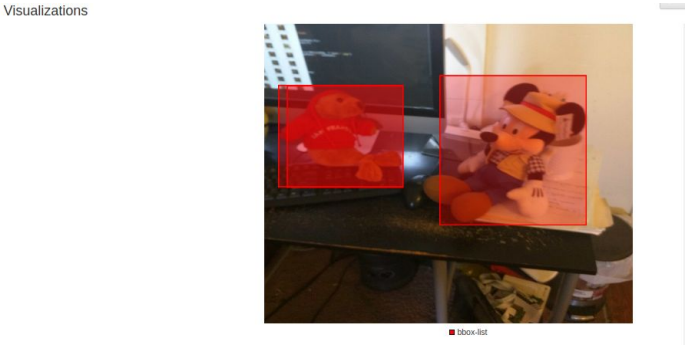

Figure 10: DetectNet on Test Image

Deployment On Jetson TX2 with real-time detection

The epoch-154 configuration of the model is downloaded and deployed on Jetson TX2 which has a Logitech C920 webcam connected for real-time detection task. The Jetson is configured with jetson-inference which is downloaded and compiled based on NVIDIA's tutorial [5]. The result from real-time detection is very intriguing as the Jetson is able to drawn the blue box around the correct stuffed animals (Fig 10-12). The real-time detection video can also be accessed at Youtube through this link: https://youtu.be/zzd9VbHGme4

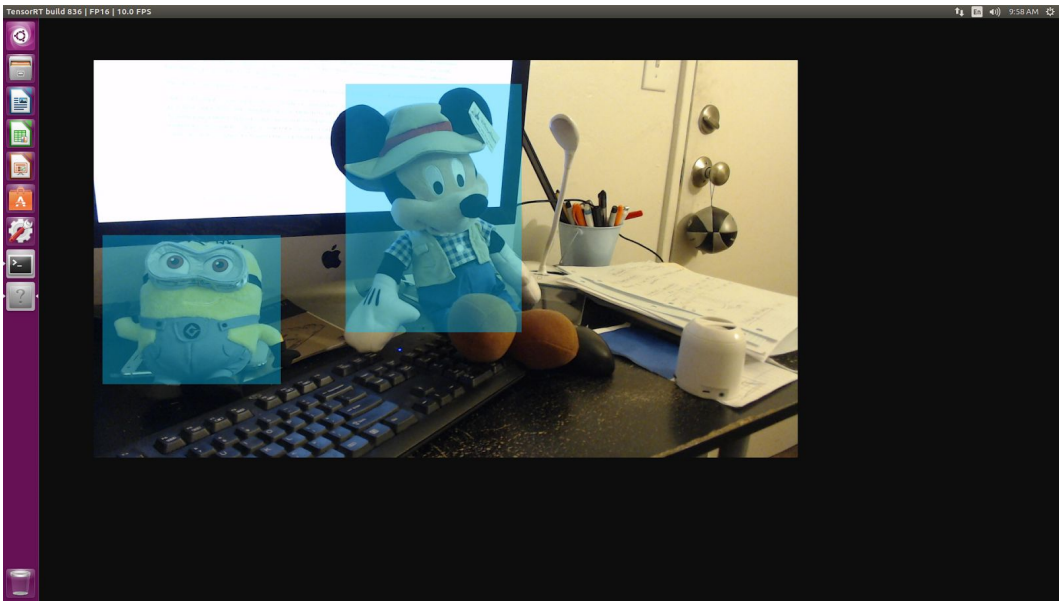

Figure 11 - Jetson Inference 1

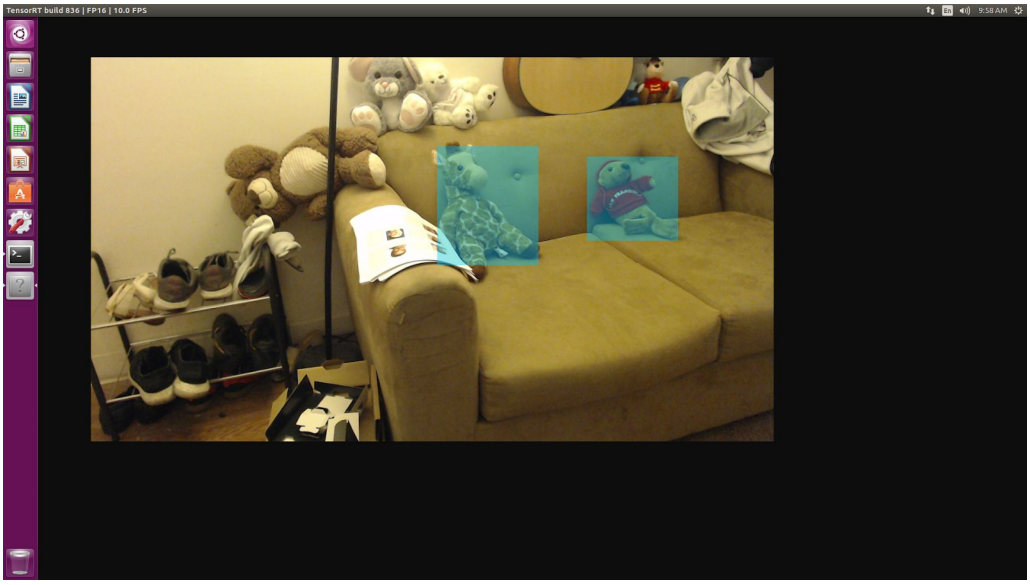

Figure 12 - Jetson Inference 2

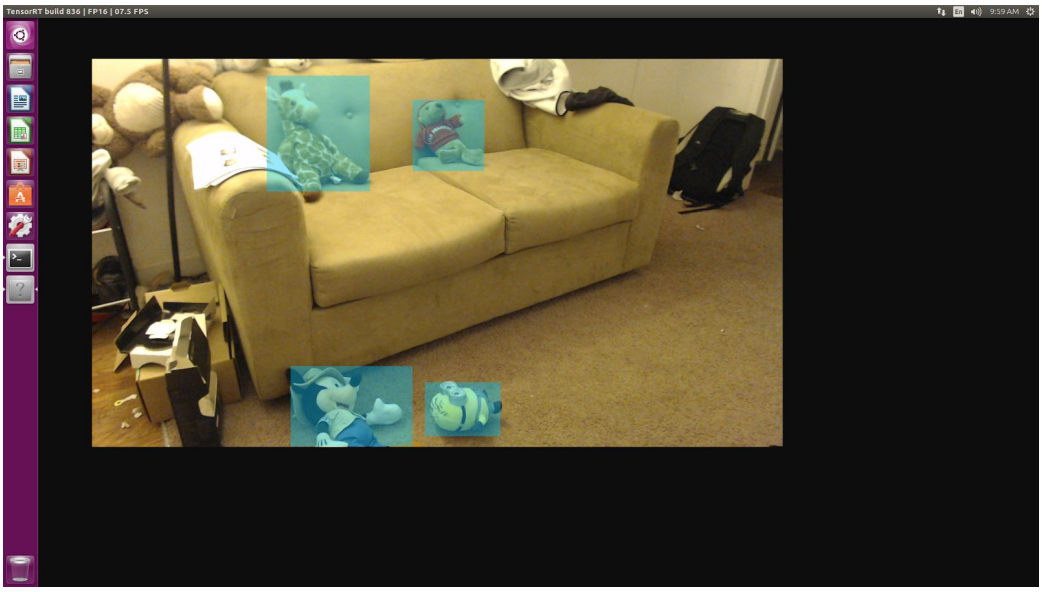

Figure 13 - Jetson Inference 3

Discussion

The result looks really fascinating for one first time going through the whole process of implementing a object detection network using state-of-the-art CNN. The network performs fine on real-time detection although it still has some jittering when moving the webcam around. On the other hand, the network still have weakness in detecting all objects available in slightly blurry image (Figure 10). Also, it draws multiple bounding boxes on same object.

Conclusion

The result of the classification task achieves the project requirements. We can try augmented the dataset to generate more data to help the network to generalize better. That might help improve the final performance using the “evaluate” program. On the other hand, due to time consuming of labelling images, only 416 images are collected for the custom detection task. Therefore, more data will definitely help improve the accuracy of the detection network. Moreover, data need to be collected at a more variety of conditions: angles, lightnings, object orientations.

References

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. “Going Deeper with Convolutions”. URL: https://arxiv.org/abs/1409.4842

[3] Karen Simonyan, Andrew Zisserman. “Very Deep Convolutional Network for Large-Scale Image Recognition”. URL: https://arxiv.org/abs/1409.1556

[4] Andrew Tao, Jon Barker, Sriya Sarathy. “DetectNet: Deep Neural Network for Object Detection in DIGITS”. URL: https://devblogs.nvidia.com/detectnet-deep-neural-network-object-detection-digits

[5] Dustin Franklin. “Deploying Deep Learning”. URL: https://github.com/dusty-nv/jetson-inference