Multi-Object Detection with Single-Shot Detector

Jeffrey Yeung
c1yeung@eng.ucsd.edu

Abstract

Single Shot Detection, SSD, is a modern and popular method of multiple object detection. SSD is called a single shot process because it performs both the region proposal and the classification steps of previous multiple object detection algorithms in a single forward pass of the network, which results in a significantly faster computation time [1]. In this paper, our goal is to modify the architecture of an SSD model to increase performance for a specific dataset. We achieve this through increasing the data set of specific classes, adjusting optimizers, and changing parameters during model training.

1 Introduction

Prior to the introduction of the method of Single Shot Detection, state of the art object detection techniques consist of two main components, a region proposal network to localize the objects in the image, and a classifier that predicts the category of the objects in the localized regions. Though effective for their time, these methods are computationally expensive and take a significant amount of time to perform. SSD combines the two components in a single forward sweep of the network to improve computation time while maintaining, or even improving, the mean Average Precision score (mAP), an evaluation metric that we will discuss in much more detail.[2]

The low computation time of SSD allows it to perform multiple object detection tasks in near real time applications. It is commonly used to detect objects in not only images, but also videos, where each frame of the video can be considered an image. The method of SSD is the result of years of research, and in this project we explore a subset of its structure. We wish to explore three topics: 1) observe the performance models with different proportions of objects in their datasets, 2) the performance of different optimizers, and 3) the performance of changing parameters during training. Each of these topics promotes different ways of changing the model architecture to observe how it affects performance for a certain dataset.

It is important to understand how to train an accepted model towards specific data because every application is unique and there is no single algorithm that will be best for every application. For example, a model that's designed to detect cars, trucks and objects along a highway will be very different than a model designed to differentiate small differences in birds and classify their species.

2 Model

Single Shot Detection is a Convolutional Neural Network that can be split into three main components:

- Base convolutions - Obtained from existing architectures via transfer learning to provide low-level feature maps.
- Auxiliary convolutions - Appended to the base convolutions to provide high-level feature maps.
- Prediction convolutions - Predicts object classes in the select feature maps.

The original authors of SSD use VGG16 as their base architecture for transfer learning due to its simple and effective structure. VGG16 is a proven model that works well for image classification problems and is easy to manipulate for different applications. The authors [1] propose two main variants of the SSD model: SSD300 and SSD512 (the numbers represent the sizes of the input images, 300x300 or 512x512). For our project, we will be using the SSD300 model because it is computationally cheaper and we will also use a simpler model, SSD7, that the same authors proposed for quicker training and results. The SSD model begins with the base VGG16 architecture, but since we do not need the fully connected layers, the fc8 layer is discarded, and the fc6 and fc7 layers are converted to conv6 and conv7, respectively. This is allowable because for an input image of size (H,W) with I input channels, the fully connected layer of output size N is equivalent to a convolutional layer with equal kernel size of an image size (H,W) and N output channels. This means that an fc layer can be converted to a conv layer by reshaping its parameters. A second change to the VGG16 base is modifying the 5th pooling layer from a (2,2) kernel of stride=2 to (3,3) kernel of stride=1. This change is necessary so that the dimensions of the feature map are no longer halved from the previous convolutional layer [1].

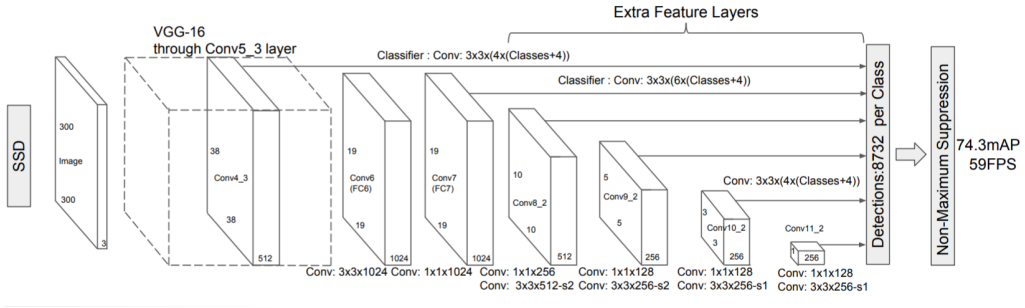

Figure 1: **Architecture of Single Shot Multibox Detector**

During the process of SSD, many localization boxes are predicted and the goal is to match these predictions to their ground truths. This is done by matching each localization box to the ground truth box with the highest overlap (typically a threshold of > 0.5). The result of this matching is that each prediction box now has a corresponding ground truth label that determines if it is a positive or a negative match; a positive match means the ground truth label is the type of the object, and a negative match means it is the background class. These matches will serve as the basis of the loss function that we will discuss later in the paper.

We used this SSD model and adjusted datasets, optimizers, and training parameters model to observe how each uniquely affected the performance of the our model.

For our data set, we use the PASCAL Visual Object Classes Challenge data sets 2007 and 2012 (VOC2007+VOC2012), and since we use the SSD300 model, our input images are of size 300x300. We add extra images of the bird class from Caltech-UCSD Birds-200-2011 to this dataset to observe the performance of the overall dataset with an influx of bird images. In total, we had 16551 images in the training set, and 5324 images in the validation set.

For the optimizer, the authors use SGD with learning rate=1e-3, momentum=0.9, and batch size=32, but we experiment three optimizers including Adam, RMSprop and SGD. For the training parameters, we use the SSD7 model with the Udacity driving data set. We change the prediction layers, amount of predictor boxes, and the usage of data augmentation to observe the affects on the training models.

3 Experimental Setting

In this section, we talk about the dataset used, training parameters for all our experiments, loss function and evaluation metric used for training our model.

3.1 Data set

- VOC2007+2012 train/val for training, and VOC2007 test for testing. VOC2012 test has incomplete annotations files when downloaded online.

- Caltech-UCSD Birds-200-2011 birds data set for modifications 4.1,4.2
- Udacity driving dataset for modifications 4.4.

3.2 Model parameters

We use the model parameters described from [1], except for each modification, we change only one parameter. For the additional birds added to our dataset, the new dataset is the parameter for training. For testing different optimizers, the optimizers are the parameter for training. Lastly, when we retrain the entire SSD7 model, each parameter that we change in comparison to the control is our parameter of interest.

3.3 Loss Function

Since SSD consists of two main components (bounding box localization and object classification), its loss function also consists of two components and the overall loss is the sum of the localization loss (loc) and the confidence loss (conf):

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g))$$

where N is the number of matched boxes, and alpha is the weight term, which is conveniently set to 1 by the authors. The confidence loss term measures how confident the model is of the type of object the matched box contains. The localization loss measures how distant the predicted box is from the ground truth.

3.4 Evaluation Metric

A common evaluation metric for measuring the accuracy of object detection algorithms is the mean Average Precision (mAP). The mAP for a set of Q queries and its average precision (AP) is as follows [2]:

$$mAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}$$

where

$$AP = \frac{TruePositives}{TruePositives + Falsepositives}$$

Obtaining the number of True Positives (TP) and False Positives (FP) depends on the area of overlap (AO) and the area of union (AU) between the predicted box and the ground truth box. AO corresponds to the area of overlap between the two boxes, and AU corresponds to the union of both boxes, and the Intersection over Union (IoU) is the ratio of AO/AU. Once IoU is calculated for each pair of boxes, a prediction is considered a TP if its IoU is greater than 0.5, and FP otherwise. Obtaining the final numbers of TP and FP allows us to calculate AP, which is then used to calculate mAP.

4 Modifications and Results

4.1 Fine-tuning with Added Birds Data Set

The VOC pre-trained model gives a decent result in detecting all 20 classes of objects. However, imagine the situation when either most of our data or most of what we are capturing contain a significant majority of one of the 20 classes, such as "person" or "birds". This means that we want our network to have a bias on this particular object so that it is more robust to detecting it. The "No Free Lunch Theorem" tells us that if we are better in one object, we will be worse in others, but this is acceptable to an extend since we assumed that the probability of this object occurring is much higher. We would like to explore the difference in the model performances between a specialized model and a generic model.
In this experiment, we continue fine-tuning the pre-trained VOC-2012 model with not only VOC-2012 train/val data set, but also the Caltech-UCSD Birds-200-2011 data set to make the network have a bias on the "birds" class. To achieve this, we first had to adjust the Caltech-UCSD-Birds ground truth CSV files so that they all belong to one "birds" class. To make a fair comparison, we train

the pre-trained model with and without the Birds data set for same number of epochs, and run the evaluation to see their performance in mAP in all 20 classes. Pre-trained Model weights can be found here.
The comparison can be seen in Figure 2. The overall mAP increased for the birds dataset, but that is in large part due to the high increase in the detection of birds. The other classes still perform well, so their accuracy is not drastically less. This is ideal for detecting a scenario with mostly birds, but also with the 19 other classes.

VOC2012,(5 epochs,20 steps/epoch)

aeroplane	AP	0.787
bicycle	AP	0.733
bird	AP	0.57
boat	AP	0.555
bottle	AP	0.38
bus	AP	0.896
car	AP	0.844
cat	AP	0.801
chair	AP	0.503
cow	AP	0.653
diningtable	AP	0.623
dog	AP	0.742
horse	AP	0.851
motorbike	AP	0.753
person	AP	0.747
pottedplant	AP	0.578
sheep	AP	0.807
sofa	AP	0.483
train	AP	0.717
tvmonitor	AP	0.621
	mAP	0.682

VOC2012+Birds,(5 epochs,20 steps/epoch)

aeroplane	AP	0.859
bicycle	AP	0.693
bird	AP	0.772
boat	AP	0.714
bottle	AP	0.383
bus	AP	0.97
car	AP	0.847
cat	AP	0.696
chair	AP	0.552
cow	AP	0.487
diningtable	AP	0.659
dog	AP	0.657
horse	AP	0.738
motorbike	AP	0.791
person	AP	0.747
pottedplant	AP	0.408
sheep	AP	0.772
sofa	AP	0.659
train	AP	0.825
tvmonitor	AP	0.551
	mAP	0.689

Figure 2: **Comparison between the performance of model trained on VOC2012 dataset vs VOC2012+Birds dataset.**

4.2 Fine-tuning with Species of Birds

We continue with the model in previous session. If our situation has changed and now we know that we will almost always be seeing birds, we may desire to have the model only classify birds based on their species. This fine-tuning is called transfer learning and is applicable to SSD networks. The reason for using transfer learning is that it is always better to start from pre-trained than from scratch because more useful information is contained in a pre-trained model than a random model.
In SSD, their are six classifying layers for object classification. If we want to replace all pre-trained classes with new classes, it would make sense to delete the original weight at the classification layers and re-initialize them. Then, we can re-train those weights with our own data set, which in this case, is the Caltech-UCSD Birds-200-2011 data set. We use only the first 20 species of birds in the dataset to match the number of classes in VOC. The ground truth CSV files has to be adjusted again to reflect the species.
For this experiment, we ran 30 epochs with 20 steps/epoch. It took about 4.5 hours on GTX 1050. The training summary of val-loss is shown in Figure 3. There is some unknown issue with the trained model that it cannot make prediction on images. This is left as future work to see the prediction results on images. The loss after 30 epochs is still relatively high compared to the ideal loss of around 4. This means more training time is required for this fine-tuning, and the learning rate has to be carefully chosen and tested. This method of transfer learning and only training the classification layers is much faster than training the entire model again from scratch. To save time and use publicly recognized models, it is useful to use fine-tuning instead of starting from scratch. From our experiment, we would still need to train the model longer though.

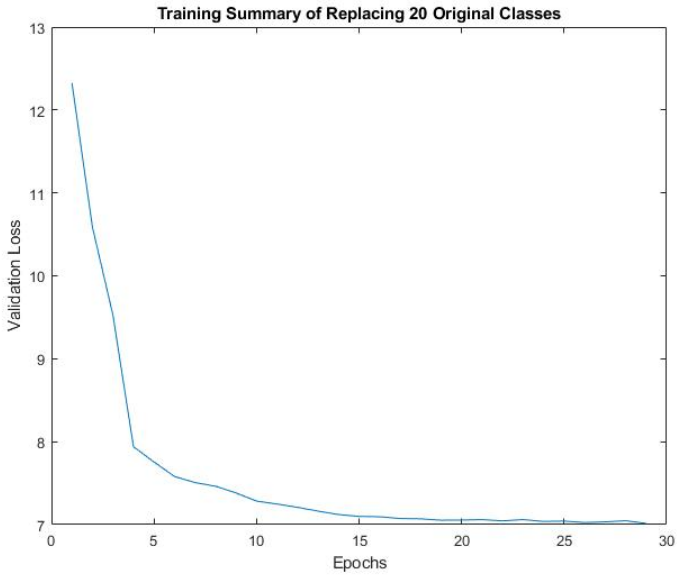

Figure 3: **Loss Plot**

4.3 Optimization Algorithm

Our main aim during the training phase of any model is to modify its parameters in such a way that the loss function is minimized. Optimization algorithms help perform this task by tweaking learnable parameters, like weights and bias, of the model. The significance of this step is that decreasing the loss function would improve our model's predictions. Hence, the motivation behind the following series of experiments was to find the optimization algorithm that would best suit our model.

In this section, we report the results of running our model with two different optimizers- namely Adam (Adaptive moment estimation) and RMSprop (Root Mean Square prop). We then compare our results with SGD (Stochastic Gradient Descent) with momentum, which was the optimization algorithm in the original paper [1]. We experiment with different hyper-parameters to optimize our model's learning process. For that, we have used Random search as opposed to Grid search because the latter is computationally expensive and quite time consuming.

Both the optimizers were evaluated for 10 epochs with 5 steps per epoch, for 3 sets of tuned hyper-parameters, to maintain consistent conditions through all experiments.

4.3.1 Adam Optimizer

The hyper-parameters used for tuning were the learning rate (used to adjust the model's learning), exponential decay rate (used to drop the learning rate every few epochs) and the stability constant (used to avoid dividing by zero in some operations). We first start with the default values and then in each subsequent experiment, change few of those values. Our results show that higher exponential decay rate results in lower losses. The reason for such a behavior could be that the learning rates adapt themselves in Adam optimizers. But, in order to ensure that each update step strictly decreases the learning rate, we increase the exponential decay rate. This is helpful for further reducing the losses during the training process. The results of this modification are summarized in Table 1.

4.3.2 RMSprop

Here, the hyper-parameter that we have tuned is the decay rate while keeping an optimal learning rate, fixed to 0.0001. Following similar reasoning as above, we find that the validation loss decreases as we increase the decay rate. The results of this modification are summarized in Table 2.

Learning Rate	Exponential Decay Rate	Stability Constant	Validation loss*
0.001	0.0	'None'	9.7167
0.001	0.97	0.1	3.1544
0.0001	0.5	0.1	3.1563

Table 1: **Experimental results of using Adam Optimizer.**

*Validation loss achieved after 10 epochs of training with 5 steps per epoch.

Learning Rate	Decay Rate	Validation loss*
0.0001	0.0	4.7087
0.0001	0.3	3.4787
0.0001	0.5	3.3540

Table 2: **Experimental results of using RMSprop Optimizer.**

*Validation loss achieved after 10 epochs of training with 5 steps per epoch.

4.3.3 Comparison with SGD

The original implementation of the model in [1] made use of Stochastic Gradient Descent with initial learning rate 1e-3, momentum 0.9 and weight decay of 0.0005. In order to compare the results of our experiments with different optimizers, we first recreate the learning conditions as specified in the original paper.

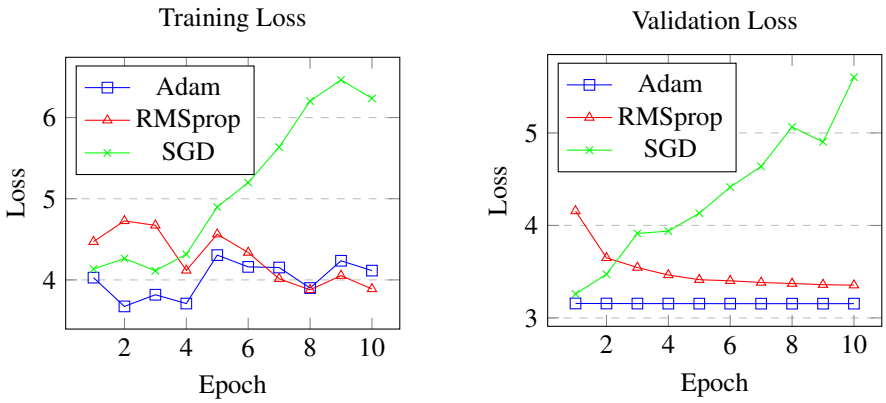

Figure 4: **Comparing Adam and RMSprop* with SGD.**

*Based on the results above, the parameters chosen for Adam optimizer is learning rate 0.001, decay 0.97 and stability constant 0.1. Similarly, for RMSprop, the chosen parameters are learning rate 0.0001 and decay 0.5. As for SGD, the parameters are learning rate 0.001, momentum 0.9 and weight decay 0.0005 as mentioned in [1].

Contrary to what we expected, the Stochastic Gradient Descent method did not work well for us as can be seen by the exploding loss values in Figure 4. There are several factors that could have caused this discrepancy in behavior. First, the batch size used in the original implementation was 32. However, in our experiments, we have fixed the batch size as 8 in order to avoid GPU memory issues. Second, we have run all our experiments in a controlled environment of 10 epochs with 5 steps each. Perhaps, the SGD model would have performed better had we run our model on a larger number of epochs. Moreover, there is no universal set of best hyper-parameters that would work well on all models. Fine tuning is largely a hit and trial process and it is unlikely that we can reproduce the results solely by recreating similar learning conditions based on parameter values. Therefore, an obvious choice for us was to use the Adam optimizer.

4.4 Training Modifications

We desire to explore improvements to the training process in order to perfect a working model. From [3], we use a simplified model called SSD7 with a 5 class dataset to detect road objects. To download the dataset used, please visit this link: Traffic Data. This model has 7 layers and uses 4 predictor layers, but it trains much faster than SSD300. By using this model, we can train the entire model on a specific dataset to see which training parameters increase/decrease the performance of the model. We will compare the validation error between our control model against trained models with one changed parameter at a time. All these models are trained from scratch from the architecture in [3]. For training time concerns, we limited each training to 10 epochs because this is long enough to see the behavior of the model. The first epoch is not graphed as it has much higher loss and distorts the shape of the graph.

4.4.1 SSD7 Control Set

The training loss and validation loss for the control sequence can be seen in Figure 5. This is the performance we will compare against different parameter changes to this model.

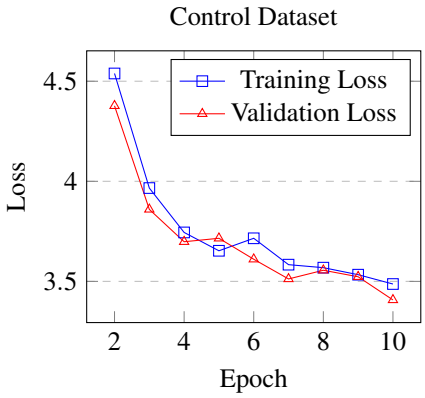

Figure 5: **Loss plot for the control dataset**

4.4.2 One Extra Layer

We modified the model by adding an extra convolution and maxpooling layer. This would decrease the size of the receptor field for the box predictions. Our hypothesis is that this extra layer may increase the ability of the network to detect larger objects as smaller receptor fields detect larger objects better [4].

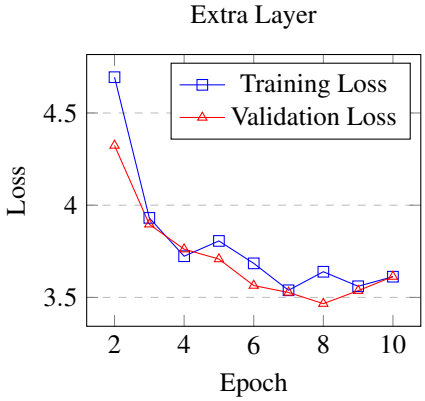

Figure 6: **Loss plot on adding an extra layer**

The validation loss is close to that of the control dataset, but the control set does perform better. Although the network is more complex, the deeper layers and smaller receptor fields do not add significant changes. This suggests that based on our dataset, smaller receptor fields will not improve our model. As mentioned earlier, this could suggest that the data does not have many large objects. This adjustment to our model could even hurt our performance because the extra layer changes the last predication layer to improve its performance on larger objects, which in turn decreases its performance on smaller objects.

4.4.3 More Box Dimensions

In the SSD7 model, the anchor box dimensions dictate the size and quantity of boxes to compare with the ground truth boxes at each prediction layer. The control model uses the same amount of boxes for each prediction layer, but with different sizes. We decided to add another box to each prediction layer, so that instead of box sizes being 0.5, 1.0 and 2.0, they were 0.5, 0.75, 1.0 and 2.0.

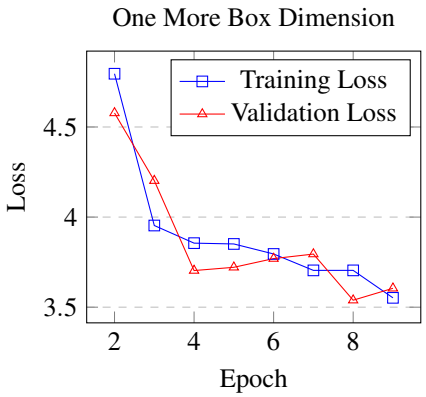

Figure 7: **Loss plot with one more box dimension**

The training of this model took twice as long as our previous models to train and did not show significant improvement over the 9 epochs that we tested it for (after 9 epoches, the program would timeout or crash). We expected that adding another size for anchor boxes would improve our model because if an object had a ground truth size of close 0.75 our anchor box, then it would have a higher IoU than a ground truth box of either 0.5 or 1.0 size. The main reason we believe why our results do not show this improvement is that there are more parameters in the model when we add an extra anchor box. These parameters would take longer to train and looking at the behavior of our graph, there was a large drop from epoch 7 to 8 and this could happen again if we extrapolate our results. From our test, we cannot conclude if adding more anchor boxes will increase our performance. For the future, We would need to make this test again on a powerful machine that would allow us to go farther than 9 epochs.

4.4.4 No Data Augmentation

In the training process for SSD7, the data is fed into a data augmentation train that randomly flips, scales and performs other operations on the training data. This would make the model more robust to changes in validation images and increase performance. We desire to see how the absence of augmentation affects the training and the importance of spending more resources on different data augmentations.

Our results show a slight increase in validation loss compared to our control model, but this slight change is not the drastic comparison we expected. One reason this could be is that the validation set came from the same dash camera that the training set came from. The variations created by the data augmentation did not significantly help in the validation of images from the same camera in the same background and similar objects. If we validated our model on a set that had different roads, cameras or camera positionings, we would expect the data augmentation to help the model be more robust to these factors.

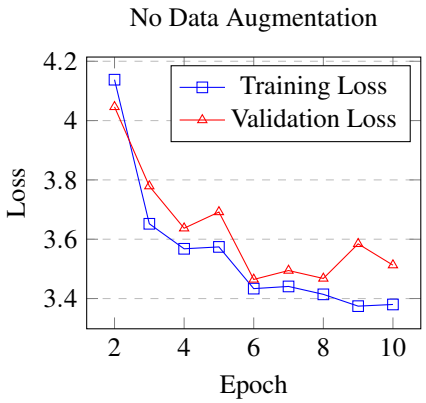

Figure 8: **Loss plot with no data augmentation**

5 Discussion

5.1 Observations

We learned about how our SSD model was created and the tradeoffs between datasets, optimizers and trainable parameters. We found that increasing the likeliness of one class in our training set can have a significant affect on the accuracy of that class without a drastic loss of performance among other classes. From this trained model, we can also drastically reduce training new models through transfer learning. We learned about the trade-offs between different optimizers and how a decaying learning rate can drastically help performance (we can also see one of the results of our trained VOC2012 Model in Figure 9). We learned about how different trainable parameters can affect training a model from scratch and how we should extend our experiments to longer training sessions on faster computers to observe more significant results.

Figure 9: **Evaluating our model on VOC2012 dataset**

5.2 Challenges

One of the main challenges we faced was that of exploding loss values and errors dropping to NaN while using the Stochastic Gradient Descent optimizer. Generally, this problem of extremely high training and validation losses signifies a high learning rate. But, even after lowering our learning rate, we found diverging costs.
Apart from this, we encountered a lot of connectivity issues with the university's GPU server since training an existing model with a few modifications, is a computationally expensive and time consuming process. And, since we had a lot of modifications, training the model from scratch was quite problematic for us. This is one reason why we pursued the SSD7 model because it was much less computationally expensive. But, even training this model for 10 epochs would take 2-3 hours, so an upgrade to a better GPU would be needed for better results.
Another challenges when we started working on the project was the implementation environment issue. The original paper used Caffe, but our team could not get this working. Other environments such as PyTorch and Keras are also available and popular, so we have to experiment to find out which is the environment that has the least compatibility issue. Finally, we decided to move on with Keras.

5.3 Future Scope

While we tried a lot of modifications on the existing architecture, one modification that we could try in the future would be an implementation of [5]. In this paper, the authors have combined the SSD model with other cutting edge classifiers like Residual-101, in place of VGG. Further, they have shown that augmenting the resulting model with deconvolution layers was able to introduce large scale context and improve accuracy.

Another interesting possibility would be to adjust where the prediction layers are drawn from. In section 4.4.2, we noticed that improving our detection of larger objects did not improve performance. It would be interesting to see if we changed the predication layers to come from shallower layers to detect smaller objects and compare the performance to our control.

References

[1] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[2] Christopher Williams John Winn Andrew Zisserman Mark Everingham, Luc Van Gool.

[3] Pierluigi Ferrari. Ssd: Single-shot multibox detector implementation in keras, May 2018.

[4] Hao Gao. Understand single shot multibox detector (ssd) and implement it in pytorch. 2018.

[5] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017.