Multi-Object Detection using Deep Learning

Team: The Inquisitive

Arthur Hsieh	Abhishek Kandoi	David Lu	Louis Lu
A53298402	A53315077	A53308491	A53307505

Github Link: multi-object-detection

Abstract

Two multi-object methods are investigated in this report, YOLO and Faster R-CNN. Both models were implemented with two different pre-trained backbones, VGG16 and ResNet-50, for comparison purposes. The methods, experimental settings, and procedures that led to the results produced are discussed. With the Faster-RCNN architecture utilizing the VGG16 backend, a mean Average Precision of 0.7002 was achieved.

1 Introduction

In this project exploring multi-object detection using deep learning, two object detection methods are investigated. The You Only Look Once (YOLO) and Faster Region-based Convolutional Neural Network (Faster R-CNN) were selected for comparison. The goal of this report is to outline the implementation of these two architectures, justify the parameters used in the training process, and examine the validation and testing procedures. The performance characteristics of both models will be also discussed, with the key evaluation metric used for measuring performance being mean Average Precision (mAP).

2 Description of Methods

There are numerous different pre-trained backbone models available for the purpose of multi-object detection. In this project, several different backbone models were implemented and compared, with efforts eventually focusing on the model that performed best. In this section, the various backbone models that were implemented will be discussed.

2.1 YOLO

2.1.1 Architecture

Numerous different backbone architectures were realized for YOLO model; their details will be described in depth in the following sections.

2.1.1.1 VGG16 with Batch Normalization

VGG16 is a convolutional neural net architecture which was used to win the ImageNet competition in 2014. The most unique thing about VGG16 is that instead of having a large number of hyper-parameters, the model utilizes convolution layers of 3x3 filter with stride 1, uses consistent padding,

and a maxpool layer of 2x2 filter of stride 2. At the end, it has two fully connected layers followed by a softmax for the output. The 16 in VGG16 refers to its 16 layers that have weights. This network is a fairly large and has approximately 138 million parameters. See Figure 1 for the network architecture.

In addition, the batch normalization technique mentioned in class was used since it was believed that this technique would produce superior results.

2.1.1.2 ResNet-50

ResNet-50 is a convolutional neural network that is trained on more than one million images from the ImageNet database. The network is 50 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and a diverse set of animals. Therefore, this network has learned rich feature representations for a wide range of images. The network has an image input size of 224x224.

The ResNet-50 model consists of five stages, each with a convolution and identity block. Every convolution and identity block each have three convolution layers. ResNet-50 has over 23 million trainable parameters; see Figure 2 for network architecture.

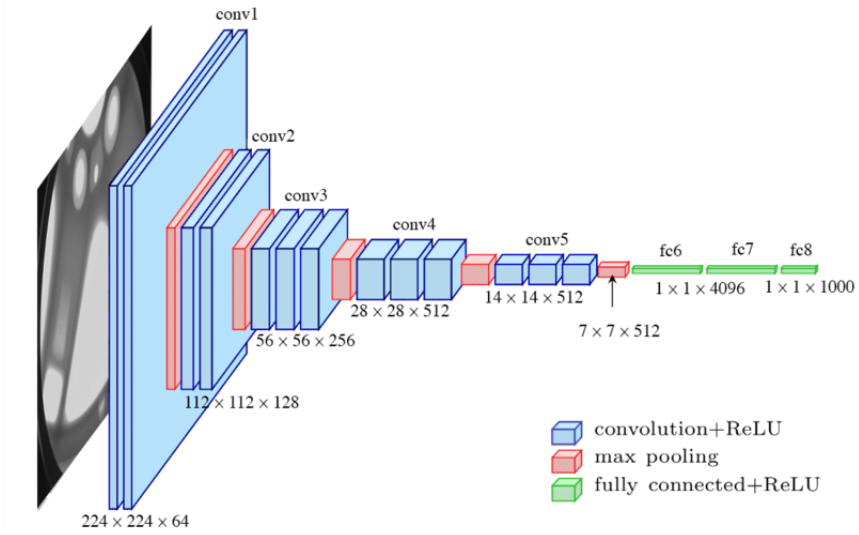


Figure 1: VGG16 Network Architecture

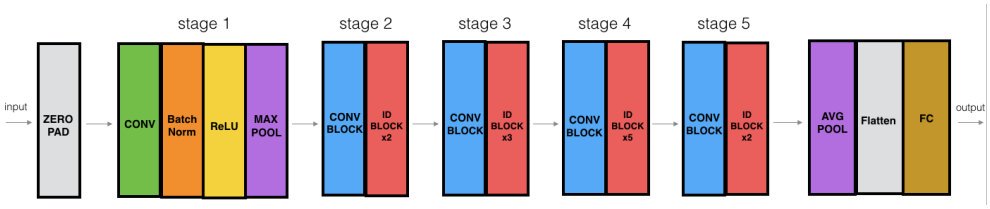


Figure 2: ResNet-50 Network Architecture

2.1.2 Algorithm

Torch optimizer was used for training the model while the YoloLoss function was utilized to evaluate the accuracy in this project. As in most neural networks, the training process is accomplished using stochastic gradient descent (SGD).

2.1.3 Equation

Intersection over Union(IoU):

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

We calculate the YOLO loss using formula below:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2]$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} [(\sqrt{w_i} - \sqrt{\hat{w_i}})^2 + (\sqrt{h_i} - \sqrt{\hat{h_i}})^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C_i})^2 + \lambda_{noobj} 1_{ij}^{noobj} (C_i - \hat{C_i})^2$$

$$+ \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p_i}(c))^2$$

where,

- x_i, y_i, the location of the centroid of the anchor box
- w_i, h_i, the width and height of the anchor box
- C_i, the box confidence score of the box j in cell i
- $p_i(c)$, the classification loss

The following are further explanations for the YoloLoss function:

- Loss from bounding box for coordinate(x, y)

$$\begin{cases} \lambda_{coord} \sum_{i=0}^{S^2} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2], & \text{responsible bounding box} \\ 0, & \text{others} \end{cases}$$

- Loss from bounding box with width w and height h

$$\begin{cases} \lambda_{coord} \sum_{i=0}^{S^2} [(\sqrt{w_i} - \sqrt{\hat{w_i}})^2 + (\sqrt{h_i} - \sqrt{\hat{h_i}})^2], & \text{responsible bounding box} \\ 0, & \text{others} \end{cases}$$

- Loss from the confidence in each bound box

$$\begin{cases} \sum_{i=0}^{S^2} (C_i - \hat{C_i})^2, & \text{object in grid cell and responsible bounding box} \\ \lambda_{noobj} \sum_{i=0}^{S^2} (C_i - \hat{C_i})^2, & \text{object not in grid cell and responsible bounding box} \\ 0, & \text{others} \end{cases}$$

- Loss from the class probability of grid cell, only when object is in the grid cell as ground truth

$$\begin{cases} \sum_{i=0}^{S^2} \sum_{c \in classes} (p_i(c) - \hat{p_i}(c))^2, & \text{object in grid cell} \\ 0, & \text{others} \end{cases}$$

2.2 Faster R-CNN

2.2.1 Architecture

For the Faster-RCNN model, the same two backbone architectures implemented in YOLO were used. While ResNet-50 was applied in Faster-RCNN in the same manner as in YOLO, the VGG16 backend was implemented without batch normalization unlike in YOLO, where batch normalization was used.

2.2.2 Algorithm

Torch optimizer was used for training this model as with YOLO. The loss function is a combination of classification loss and bounding box regression loss while evaluation was performed using the mAP metric from PASCAL VOC 2007 benchmark. As in most neural networks, the training process is accomplished using stochastic gradient descent (SGD).

2.2.3 Equation

Faster-RCNN loss function:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}}$$

Where $\mathcal{L}_{\text{cls}}$ is the log loss function over two classes (object vs. not object) and $\mathcal{L}_{\text{box}}$ is the bounding box smooth L1 loss

The loss function for an image is defined as:

$$\mathcal{L}(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}} \sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*)$$

where,

- p_i, Predicted probability that anchor i is an object
- p_i^*, Ground truth label (binary) of whether anchor i is an object
- t_i, Predicted four parameterized coordinates
- t_i^*, Ground truth coordinates
- N_{cls}, Normalization term, set to be mini-batch size (=8)
- N_{box}, Normalization term, set to the number of anchor locations (approx. 2400) in the paper
- λ, A balancing parameter, set to be 10 in the paper (so that both $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{box}}$ terms are roughly equally weighted).

and,

$$\mathcal{L}_{\text{cls}}(p_i, p_i^*) = -p_i^* \log p_i - (1 - p_i^*) \log(1 - p_i)$$

and the smooth L1 loss is given as

$$L_{1;smooth}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1; \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

3 Experimental Setting

In this section, the dataset, environment, and parameters of the training will be discussed.

3.1 Dataset

Both object detection methods (YOLO and Faster R-CNN) were comprehensively evaluated on the PASCAL VOC 2007 detection benchmark. For training, a nine to one (9:1) split of the PASCAL VOC 2007 and 2012 dataset was used. Overall, the dataset contains 14,896 images for training, out of which 4,510 are from the PASCAL VOC 2007 dataset and 10,386 are from the PASCAL VOC 2012 dataset (only training and validation sets were used), and 1,655 images for validation. In Figure 3, a histogram showing the number of samples used for each class is displayed for both the PASCAL VOC 2007 and 2012 datasets, as well as the the number of samples across both years.

Furthermore, data augmentation techniques and horizontally flipped images were both utilized in the training of YOLO and Faster R-CNN, thus effectively doubling the size of the dataset. During testing and validation, the datasets were not augmented and horizontal flipping was not performed.

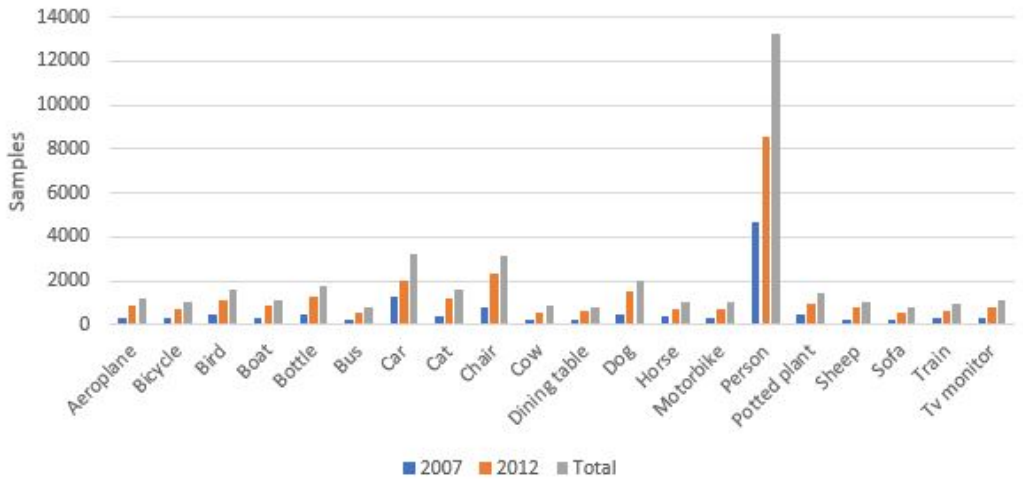


Figure 3: Histogram of Samples per Class

3.1.1 Training Parameters

3.1.1.1 YOLO

YOLO was trained with two different backends. The first was trained with a pre-trained VGG16 with Batch Normalization and the second with a pre-trained ResNet-50. Both YOLO models were trained with a step decay learning rate starting from 1e-3 and ending at 1e-5 using the `torch.optim.SGD` optimizer with a momentum of 0.9. See Table 1 for the parameters used during training.

3.1.1.2 Faster R-CNN

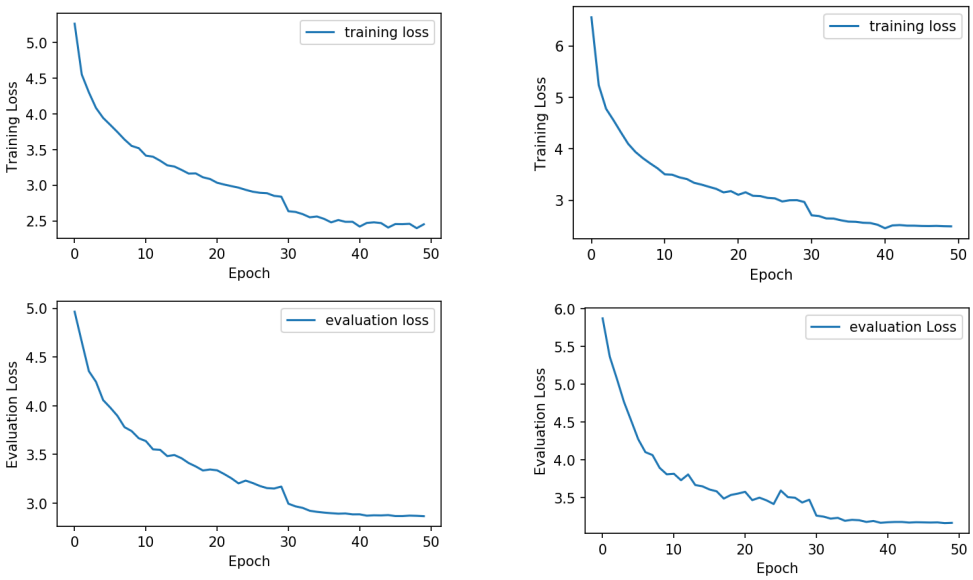
Similar to YOLO, Faster R-CNN was also trained with two different backends. The first was trained with a pre-trained VGG16 with 13 convolutional layers and three fully-connected layers as its backend, and the second with a pre-trained ResNet-50. Faster R-CNN with VGG16 backend was trained with a constant learning rate of 4e-3 using the `torch.optim.SGD` optimizer with a momentum of 0.9, while Faster R-CNN with ResNet-50 backend was trained with a learning rate of 6e-4 (which decayed by a factor of 0.1 after every eight epochs) using the `torch.optim.SGD` optimizer with a momentum of 0.9. Both models used a batch size of eight to stay consistent with the trained YOLO models. See Table 1 for the parameters used during training.

Table 1: Training Parameters

	Models			
Method	Pre-trained Backbone	Learning Rate	Optimizer	Momentum
YOLO	VGG16 with Batch Norm	1e-3 (step decay)	SGD	0.9
YOLO	ResNet-50	1e-3 (step decay)	SGD	0.9
Faster R-CNN	VGG16	4e-3	SGD	0.9
Faster R-CNN	ResNet-50	6e-4	SGD	0.9

3.1.2 Validation and Testing Procedure

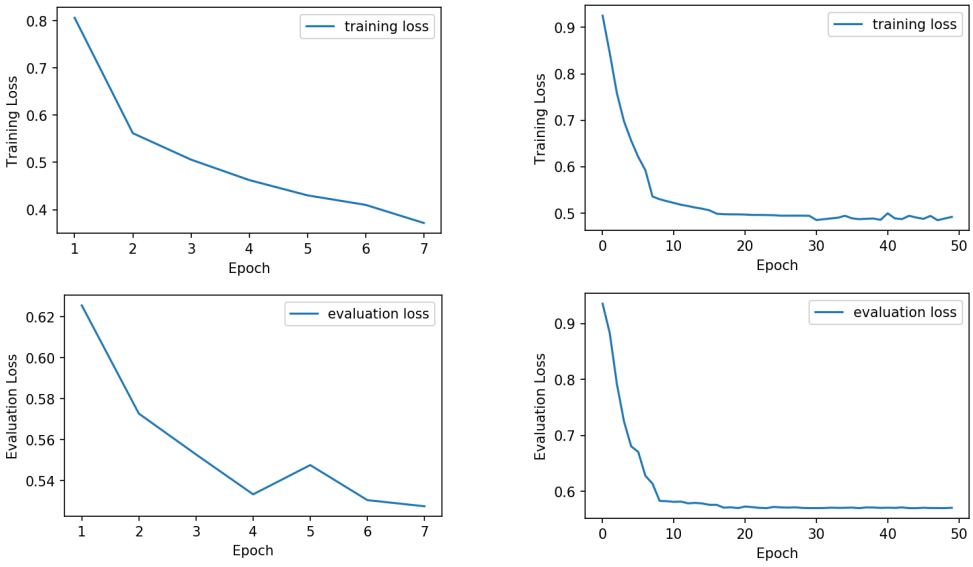
The training and validation data from the PASCAL VOC 2007 and PASCAL VOC 2012 datasets was split into a 9:1 ratio for use in training and validation, respectively. This was done to prevent overfitting of the model on the training dataset. Both object detection methods were tested on the PASCAL VOC 2007 dataset which has 4,952 test images with performance measured by the mean Average Precision (mAP) metric used by the PASCAL VOC 2007 object detection benchmark. Results are reported and examined in the next section. See Figure 4a and Figure 4b for loss evolution for the YOLO models, and Figure 5a and Figure 5b for loss evolution for the Faster R-CNN models.



(a) VGG16 with BN Backend

(b) ResNet50 Backend

Figure 4: Loss evolution of YOLO



(a) VGG16 Backend

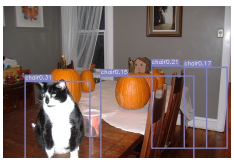
(b) ResNet50 Backend

Figure 5: Loss evolution of Faster R-CNN

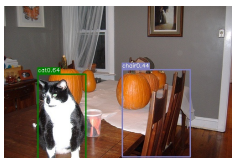
4 Results

4.1 Prediction Samples

Here the prediction of the same image from the four different models are shown. See Figure 6 for the YOLO prediction and Figure 7 for the Faster R-CNN prediction.

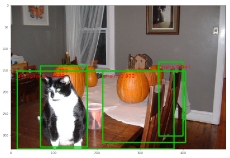


(a) VGG16 with BN

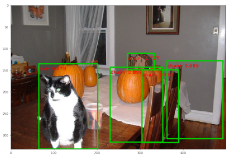


(b) ResNet50

Figure 6: Prediction from YOLO



(a) VGG16



(b) ResNet50

Figure 7: Prediction from Faster R-CNN

4.2 Model Comparison

After choosing optimal configurations and tuning the hyperparameters for each model, comparisons between the models can be made. Faster R-CNN + VGG16 was trained for seven epochs, and the rest for 50 epochs. The final results are shown in Figure 8 and Table 2. In Figure 8, the histogram shows the average precision of each class for each model. In Table 2, the mean Average Precision (mAP) for each of the models is displayed. In addition, the average running time of detecting objects in an image is shown in Table 3.

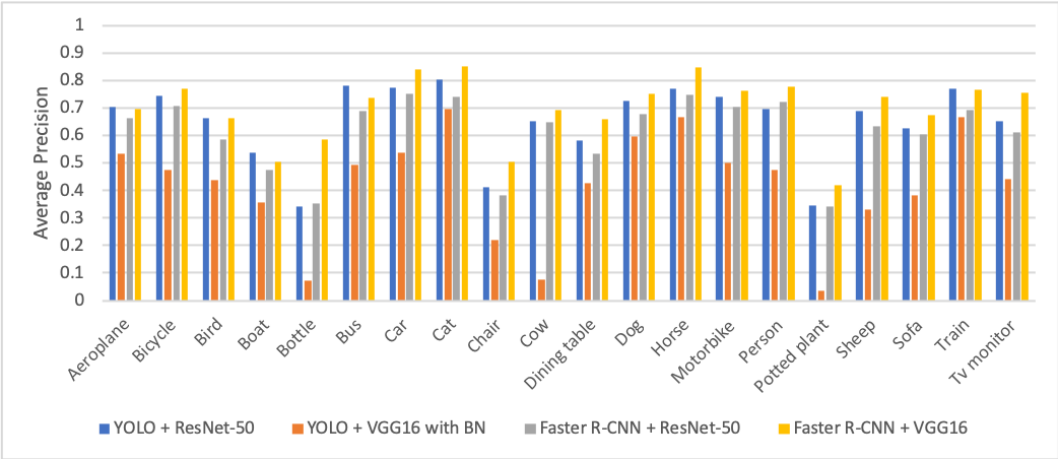


Figure 8: Histogram of Average Precision for Each Class

Table 2: mean Average Precision (mAP) Comparison

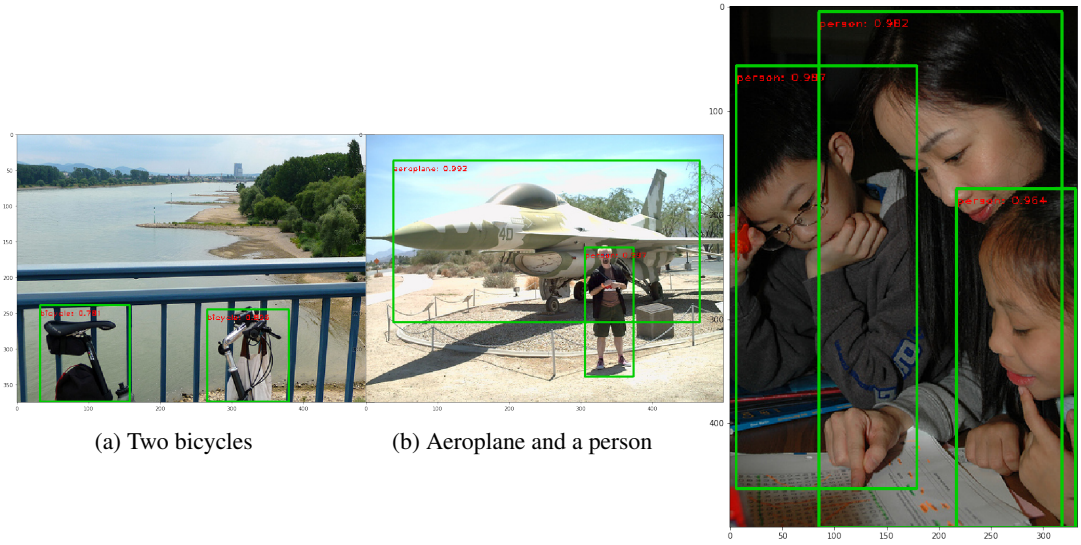
Models		
Method	Pretrained Backbone	mean Average Precision
YOLO	VGG16 with Batch Norm	0.4211
YOLO	ResNet-50	0.6509
Faster R-CNN	VGG16	0.7002
Faster R-CNN	ResNet-50	0.6137

Table 3: Inference Time Comparison

Models			
Method	Pretrained Backbone	Detection Time (per image)	FPS
YOLO	VGG16 with Batch Norm	0.0524 sec	19.08
YOLO	ResNet-50	0.0571 sec	17.51
Faster R-CNN	VGG16	0.0610 sec	16.39
Faster R-CNN	ResNet-50	0.0599 sec	16.69

4.3 Final Performance

As can be seen from Table 2, the Faster R-CNN model with the pre-trained VGG16 model outperforms the other models trained and experimented with a mAP (mean Average Precision) of 0.7002 achieved. One interesting thing to note is that it does so within seven epochs which is a relatively fast convergence when compared to the other three models that were implemented. Some examples of the predictions from this VGG16 backend based Faster R-CNN model are displayed in Figure 9. See Figure 10 for some failure cases for the Faster R-CNN model.



(a) Two bicycles (b) Aeroplane and a person

(c) Three persons

Figure 9: Predictions from Faster R-CNN (best performing model)

5 Discussion

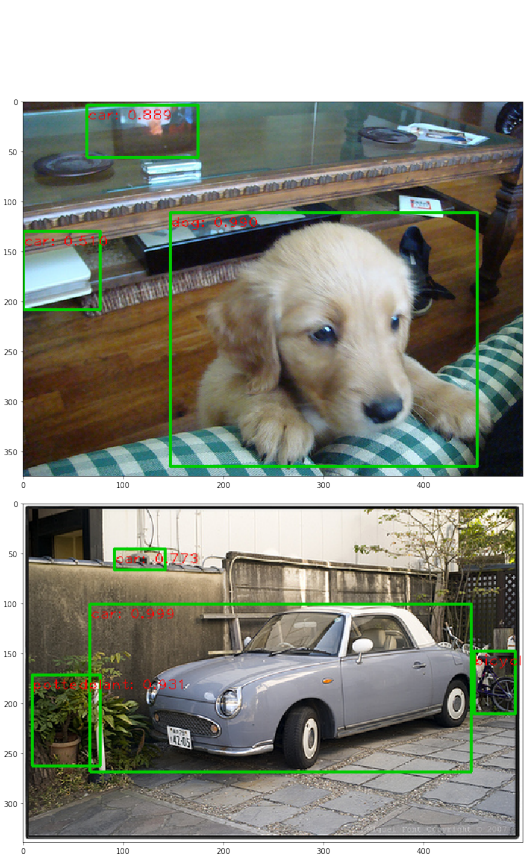
It was attempted to improve the pre-trained model VGG16 by utilizing the batch normalization technique. However, satisfactory results could not be obtained; surprisingly, it got the worst mAP (about 0.42) out of the four models. One factor that caused this result could be the small batch size (8 images) that was being used due to CUDA memory limitations. Thus, at least for these particular data/parameter settings, it appears that batch normalization had a negligible influence on the results.

The ResNet-101 pre-trained backbone was also applied for YOLO and produced an evaluation loss lower than the ResNet-50 backend. However, due to time constraints, the training was not completed. However, it can be concluded that by making the network deeper, the results will be superior for this specific task.

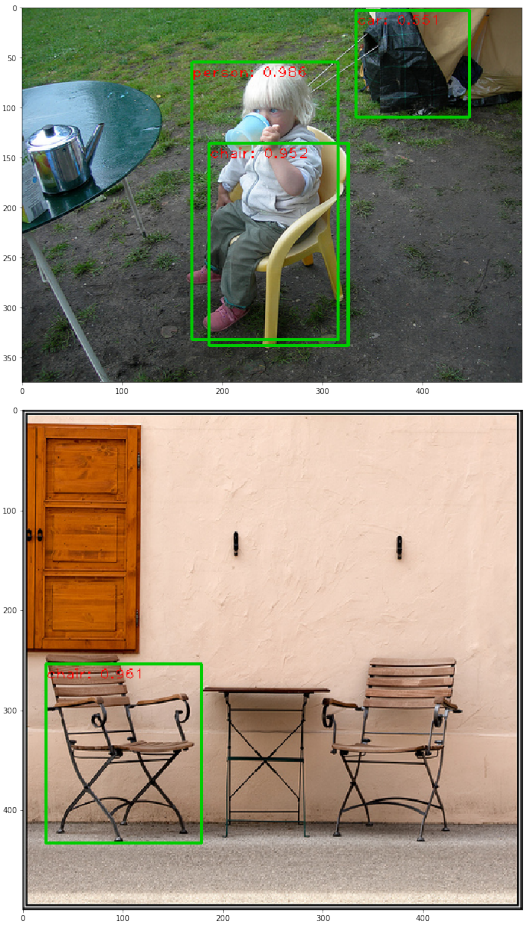
The model that obtained the best performance for this task tended to utilize larger batch size (16 or 24 images). Therefore, if more resources were available, a larger batch size would have been attempted, which is believed to help the training converge faster.

It is interesting to note that our Faster R-CNN model with VGG16 backbone converged within seven epochs. It is suspected that this could be because of the higher learning rate as compared to the ResNet-50 backend. This is because the ResNet-50 backend started diverging with a higher learning rate, so it was reduced in order to train the model. Further, using a larger batch size (batch size = 8 was used) was not possible due to resource constraints. It would have been interesting to observe if using a batch size of, for example, 32 or 64, would have led to different convergence rates.

Several low light photos were experimented with to see if the model could predict and localize objects in them. The results exceeded expectations for some photos, and were not so great for other images. It appears that small objects such as bottles were very hard for the network to detect. See Figure 11 for some of these sample predictions using the best model (Faster R-CNN with VGG16 backend).



(a) Car detected in unusual places (in both)



(b) Dining table not detected, car in unusual place (top) and identical chair not detected (bottom)

Figure 10: Failure cases for Faster R-CNN

Additionally, the best model (Faster R-CNN with VGG16 backend) was also used on some photos taken with personal smartphones. See Figure 12 for some of these predictions done on images taken in San Diego. Two of these images (MTS bus and campus area) were taken on the UC San Diego campus. The third image was taken in a room at La Jolla Crossroads, Judicial Drive.

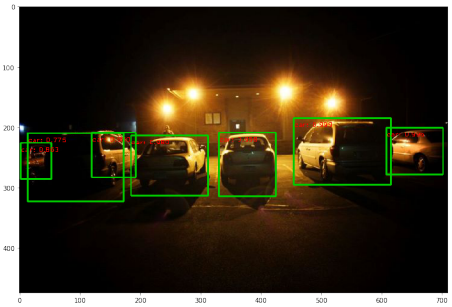
References

[1] Ren, S., He, K., Girshick, R., & Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In arXiv:1506.02640 [cs.CV]

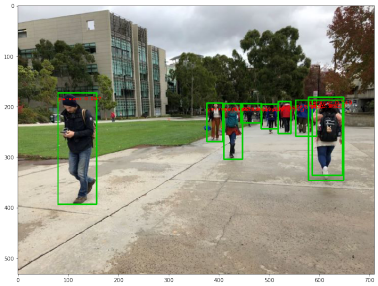
[3] Yang, J., Lu, J., Batra D., & Parikh D., 2017. A Faster Pytorch Implementation of Faster R-CNN. At https://github.com/jwyang/faster-rcnn.pytorch

(a) A vending machine with many different bottles in low light



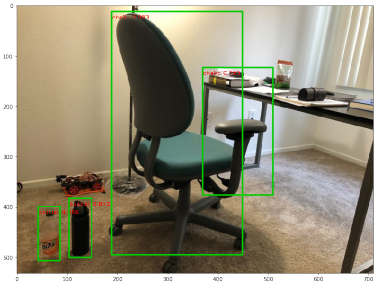
(b) A parking lot in low light

Figure 11: Predictions for low light photos using Faster R-CNN



(a) UC San Diego campus area

(b) Gilman Drive Transit Center



(c) Abhishek Kandoi's room at La Jolla Crossroads

Figure 12: Predictions on real-world photos using Faster R-CNN (taken on iPhone 8)