Traffic Sign Detection using You Only Look Once Framework

Chung Yu Wang
Stanford University
chungyuw@stanford.edu

Royce Cheng-Yue
Stanford University
roycecy@stanford.edu

Abstract

Traffic sign detection is one of the most important problems in autonomous driving. Among the object detection models, Fast R-CNN [7] and Faster R-CNN [8] have recently become popular. This paper is focused on analyzing the costs and benefits of using YOLO as an alternative model in detecting traffic signs. In the end, after looking into multiple architectures and trying different hyper-parameters on the Belgium Traffic Sign Dataset, we used an ensemble of two of our models to achieve 33.8% mAP on the test data. Unfortunately, this accuracy pales in comparison to Faster R-CNN with similar classes in VOC 2012 [9]. After analyzing the error rates and visualizing saliency maps, we found that YOLO uses global features instead of local features to predict labels and bounding boxes, which is different from other region proposal CNN architectures. In general, YOLO's approach provides a fast alternative for object detection but has difficulty discriminating between spatially small classes if they are not correlated with their environment, as in the case of traffic sign detection.

1. Introduction

Over the past decade, autonomous cars have generated a large amount of interest within academia and the industry, specifically in urban environments. One of the core pieces of this technology is traffic sign detection and recognition. Detecting traffic signs is challenging due to a number of issues, including occlusion, color variation, rotation, and skew that arises from camera setup or environmental influences. Furthermore, there could be multiple signs in an image with varying shapes, colors, and sizes.

To solve this problem, most state of the art approaches require a large amount of GPU resources and do not take into account test time latency. This approach is not possible in an autonomous car system, as latency is extremely crucial in making split second decisions based on the environment. Among the recent advances in object detection, Fast R-CNN and Faster R-CNN provide predictions on the order of hundreds of milliseconds to seconds. As a means of searching for a faster solution, we decide to look into YOLO, which is about six times faster than Faster R-CNN. In this paper, we will analyze the costs and benefits of using YOLO to detect traffic signs.

Our problem is traffic sign detection. The input to our model is an image. We use multiple variations of YOLO architectures and hyper-parameters to predict traffic sign bounding boxes and their respective classes.

2. Related Work

Before the rise of CNNs in computer vision, SVM with HOG features [2] and deformable parts models (DPMs), which uses root and part templates to detect bounding boxes [3], were the state of the art in object detection. HOG-based DPM models on VOC 2007 dataset achieved mAP of 33.7 [4].

After 2013, CNNs started to become the standard for object detection tasks. Some examples of these CNNs include OverFeat [5], which uses a scalable sliding window and a greedy merge procedure to produce bounding boxes and scores, and R-CNN, which introduces a two-step process of a region proposal method and a regression problem within the proposed regions. Despite the overall success of R-CNN, training an R-CNN model is expensive in both memory and time and processing a test image takes about 47 seconds. By sharing computation and replacing Selective Search with a neural network [6], Fast R-CNN and Faster R-CNN are able to achieve higher accuracies and better latencies overall. Specifically, these two models achieve 70.0 and 73.2 mAP and 2000 ms and 142 ms latency on the PASCAL VOC 2012 test set respectively.

Instead of having a sequential pipeline of region proposals and object classification, YOLO provides an alternative by treating the overall problem as a regression problem. This leads to a much lower latency at the cost of some accuracy. On the PASCAL VOC 2012 test set, YOLO is able to achieve 63.4% mAP at 22 ms latency.

In the space of traffic sign detection specifically, HOG-based SVMs and HOG-based CNNs have been popular, with state of the art HOG based SVM models achieving 100% AUC and HOG based CNNs achieving 99.73% AUC [10][11].

Since many models were achieving over 95% AUC in VOC 2006, the standard metric for future VOC challenges was changed to mean Average Precision, mAP, to improve interpretability and visibility in models [12].

Because most of the traffic sign detection papers do not report mAP, we decided to look at baseline results for the "bottle" and "plant" classes in VOC 2012 due to their structural and spatial similarities to traffic signs. Faster R-CNN achieved 49.8% mAP for the "bottle" class and 40.1% mAP for the "plant" class in VOC 2012.

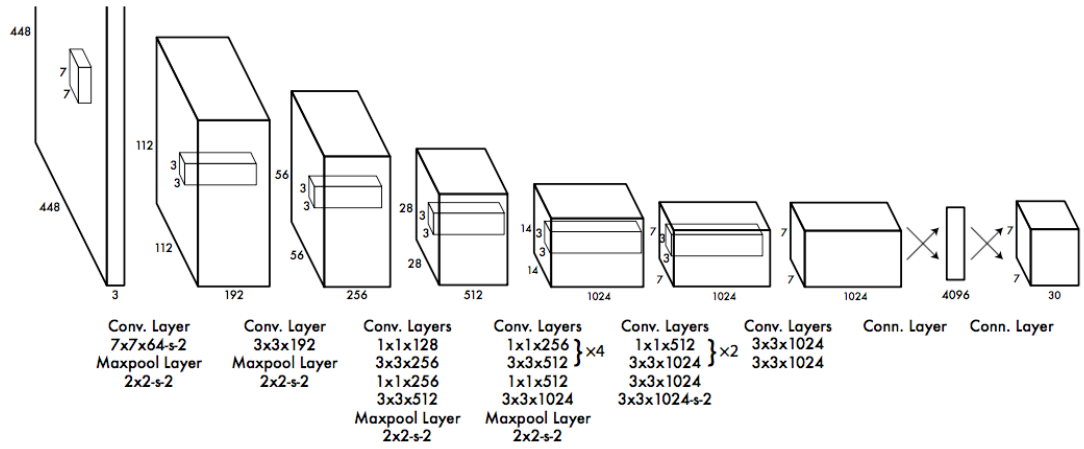

Figure 1: Original YOLO Architecture.

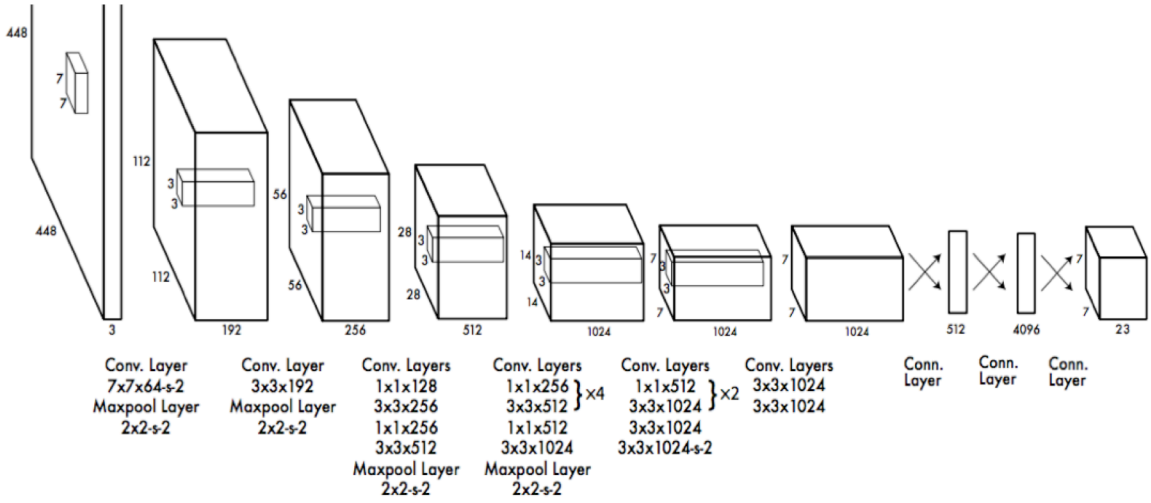

Figure 2: Traffic-Small Sign Detection Architecture.

3. Dataset and Features

We used the Belgium Traffic Sign Dataset (BTSD) to train our models. In total, there are 13 classes of varying shapes and colors, including an "unlabeled" class for any sign outside of the 12 original classes. These classes include triangle signs, red circle signs, blue circle signs, red and blue circle signs, diamond signs, reverse triangle signs, stop signs, forbidden signs, square signs, vertical rectangle signs, horizontal rectangle signs, other traffic signs, and an undefined class as a catch all.

The images were generated from different cameras on vehicles in urban streets. There are 5,905 training images, 1,301 validation images, and 1,750 test images. Within these images, there are zero, one, or multiple traffic signs. Usually, the traffic signs take up a small portion of the image and may suffer from variations in lighting, orientation, or occlusions.

Figure 3: Example Images from BTSD.

As part of the infrastructure, we are also able to easily augment our training data by conducting horizontal flips and random crops on the training images. Since our classes are based on geometries that are vertically symmetrical, horizontal flips can effectively boost our training size by a factor of two. Random crops are controlled by a jitter hyper-parameter that randomly crops an image by the specified percentage. By default, we set the jitter hyper-parameter to 0. To reduce the memory footprint and increase computational efficiency, we also

resized all images to 448 x 448 pixels at both training and testing times.

For each image, BTSD provides labels for all classes and associated bounding boxes. The format for each bounding box label is: class of the traffic sign, x coordinate corresponding to the center of the bounding box, y coordinate corresponding to the center of the bounding box, width of the bounding box, and height of the bounding box.

4. Technical Approach

This project is built on top of the YOLO Architecture, focusing on fine-tuning the existing model based on the traffic signs. We have adopted the small YOLO model due to computational constraint.

4.1. YOLO Architecture

YOLO, unlike R-CNN, directly regresses on the bounding box locations and class scores. This simpler approach in YOLO resulted in a mAP of 57.9% while Faster R-CNN attains mAP of 70.4% on the VOC 2012 test set. However, the base YOLO model runs at 45 fps, achieving more than twice the mAP of other similar real-time detection systems [1].

The key idea of YOLO is in dividing each image into $S \times S$ regions and within each region it directly regresses to find B bounding boxes and a score for each of the C classes. For each of the B bounding boxes there are 5 numbers the neural network regresses on; these are center x, center y, width, height and confidence of the bounding box. For each region, there will only be one set of class scores C for all bounding boxes in that region. Therefore, the output of the YOLO network will be a vector of $S \times S \times (5B + C)$ numbers for each image. YOLO was pre-trained on ImageNet with $S = 7$, $B = 2$, and $C = 20$. Overall, the existing YOLO architecture consists of 24 convolutional layers followed by 2 connected layers and a final output layer as shown in Figure 1.

4.2. Our Architecture

The key architecture modifications arise from the last layers after the convolutional layers. Since there are only 13 classes of traffic signs, our last layer requires $C = 13$. In the original YOLO architecture, there were only 2 connected layers at the end, but due to memory constraints, we inserted a down-sampling connected layer before the original two connected layers, effectively reducing the number of parameters by 84%. Our final architecture, Traffic-Small, is shown in Figure 2.

4.3. YOLO Loss Function

YOLO uses ones single loss function for both bounding box and the classification of the object. The loss function is:

$$Loss = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \quad (1)$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \quad (2)$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 \quad (3)$$

$$+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2 \quad (4)$$

$$+ \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2 \quad (5)$$

The loss function can be parsed into 5 parts, where parts (1) and (2) are focusing on the loss of the bounding box coordinates, parts (3) and (4) are penalizing the differences in confidence of having an object in the grid and part (5) is penalizing for the difference in class probability. It is interesting to note that the loss function for the bounding box size is based on the square root of the dimensions. This is used to address that the small deviations in larger bounding boxes should incur less of a penalty than in smaller bounding boxes. We also note that there are λ_{coord} and λ_{noobj}, which are set to be 5 and 0.5 by the original authors. The λ_{coord} hyper-parameter is set to ensure "fair" contribution of the bounding box location penalty and the classification penalty to the overall loss function, and the λ_{noobj} is set to penalize less for the confidence of identifying an object when there is not one.

4.4. Data Preprocessing / Data Augmentation

To reduce the memory footprint and increase computational efficiency, all images are first resized to 448 x 448 pixels at both training and testing times.

As part of the infrastructure, we are able to easily augment our training data by conducting random crop and horizontal flip of the training image. Since our classes are based on geometries that are vertically symmetrical, the horizontal flip can effectively boost our training size to be twice as large. Random crop does provide a means of increasing our training data set. However since many of our traffic signs are near the borders of images, random corps often result in having 0 traffic signs per batch of training, leading to 0 updates in these batches. Therefore, the actual benefit of random crops is not definitive without further supervision on the random crop.

4.5. Configuration

Current architecture runs 350x faster on GPU mode than on CPU mode. As a result, we are running the optimized GPU code for training on the Stanford Rye

cluster, which has 6 parallel NVidia Tesla C2070 GPUs, each with 6GB memory. However, due to the limitation on the shared GPU resources, we are only capable of running at most 64 training images per batch during times when the GPU load is low and 4 training images per batch during times when the GPU load is high. This constraint is based on the GPU load at the time when we start training. As such, most of our training is done with either 32 or 64 training images per batch.

4.6. Training Variations

We trained several models, each with slightly different training hyper-parameters or methodology, with the exception of Traffic-Tiny, which uses only around 600 MB of GPU memory. Traffic-Tiny architecture is very similar to Traffic-Small, with the exception of only having 15 convolutional layers instead of 28.

When we trained Traffic-Small, we attempted numerous mechanisms that are provided by the framework to combat the overfitting issue that we see from our training. We introduced higher drop out rates, larger training data size by random crops and freezing the convolutional layers. The results of these experiments are discussed in the next section.

4.7. Saliency Map Generation

As part of our analysis, we have incorporated the saliency map generation into the YOLO framework in C. It is very interesting to note that the saliency map generation for detection is very similar to that for classification. Even though the true label for bounding boxes are real values between 0 and 1, feeding these true label values directly back into the network from the last layer seemed to give reasonable and interpretable results. The results of these saliency maps are discussed in the next section.

5. Results

We first ensured that our model can over-fit a small training set of 1 image and 20 images as our preliminary model checking. We then picked a mix of hyper-parameters based on initial experimental results and computational power constraints and trained multiple models and compared the results.

5.1. Overfitting Model

As a sanity check to ensure that we set up YOLO correctly, we attempted to over-fit our model on a single training example. We found we were able to fully over-fit the model, reaching around 0.99 IOU training accuracy, after 400 iterations with a batch size of 4 as shown in Figure 4.

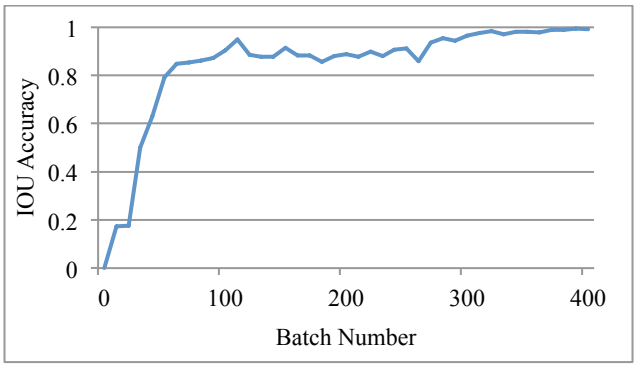

Figure 4: IOU Accuracy on 1 training image.

As another sanity check, we expanded the model to train on just 20 training examples and observed a similar convergence pattern. When tested on one of the training images, the trained model was able to retrieve the true bounding box and class correctly as shown in Figure 5.

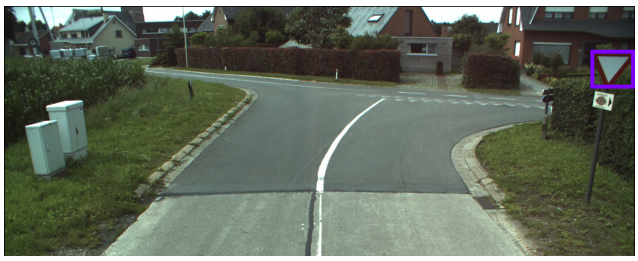

Figure 5: Overfitting based on 20 training images.

5.2. Identifying the Hyper-parameters

Detection problems often take a while to train before starting we start seeing if the learning rate is truly too slow or is truly unstable. Because of that and due to the computational resources available, we picked our learning rate for all models based on the Traffic-Small model with drop-out of 0.5 and batch size of 64. We tried a few learning rates and immediately stop the ones that are unstable. Figure 5 is the trend on the training loss function of the final two learning rates of 1e-3 and 1e-5 we considered. From the figure, we found that the learning rate of 1e-3 is more efficient than 1e-5, while attaining numerical stability.

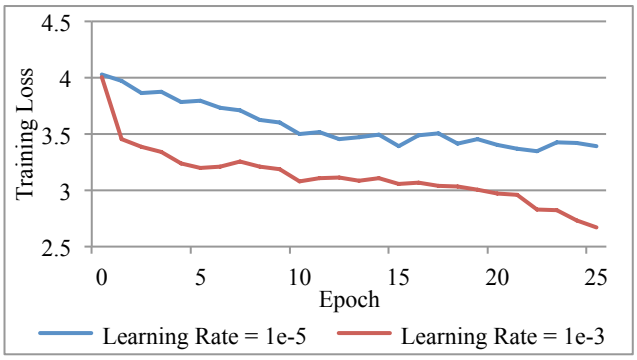

Figure 6: Training Loss on different learning rates.

Due to the computational resource constraint we have, we were not able to run our model with more than a batch size of 64 and hence that is what we used.

Although many update rules are not present in the YOLO framework, momentum update and the vanilla SGD is. Since we learnt that momentum update usually outperforms vanilla SGD, we chose to use momentum update for training.

5.3. Validation

We conducted a held-out validation using 1,301 images from the BTSD dataset. Although cross-validation may be a more effective use of the dataset, it would mean training multiple models with the same set of hyper-parameters, which is something we cannot afford due to computational resources at this moment.

5.4. Metrics

In order to evaluate our results, we used interpolated mean average precision from VOC 2007, which has become a standard metric in VOC. Specifically, the average precision approximates the shape of the precision/recall curve by looking at the curve via a step function. For every recall level between 0 and 1 (i.e. [0, 0.1, …, 1]), we interpolate the precision by finding the maximum precision across all recall levels larger than the current recall level. We then take the average of all of these precisions to find the AP.

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r}\geq r} p(\tilde{r})$$

$$AP = \frac{1}{11} \sum_{r\in\{0,0.1,\ldots,1\}} p_{interp}(r)$$

Note that this is different from mAP in most other object detection papers, where we do not interpolate the precision at each recall level. Because we are comparing our mAP results with models trained and tested on VOC, we used the way VOC defined mAP.

We determine true and false positive bounding boxes by their respective intersection over union (IOU) with the ground truth bounding boxes. IOU is defined as the intersection divided by union between the predicted bounding box and the ground truth bounding box. In order for a bounding box to be a correct prediction, the bounding box must have an IOU > 0.5.

$$a_0 = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

5.5. Qualitative Results

Our model is successful in detecting many of the traffic signs. However, we noticed that there are a few areas that our model has not fully learned to pick up yet. These include traffic signs that are further away and hence smaller and with lower contrast in the image as shown in Figure 7, and traffic signs that are partially occluded as showing in Figure 8. These are some of the canonical challenges we face in computer vision, but possibly with longer training and larger data set, these issues can be resolved.

Figure 7: Detection of close up traffic signs.

Figure 8: Detection of partially occluded traffic sign.

5.6. Quantitative Results

After we trained our initial Traffic-Small architecture on the 5901 training examples, we realized that we were overfitting to the dataset. Specifically, after 54 epochs, our training accuracy was 66.8% mAP and our validation accuracy was 27.8% mAP, causing about a 39% mAP gap between training and validation. This is indicated in Figure 10.

In order to remedy the overfitting issue, we looked into several experiments:

- Increasing the dropout threshold to 0.75 (Traffic-Small-Dropout-0.75)
- Setting jitter to be 0.2 as a means of data augmentation (Traffic-Small-Jitter-0.2)
- Increasing the dropout threshold to 0.75 and setting jitter to be 0.2 (Traffic-Small-Dropout-0.75-Jitter-0.2)

In addition to these methods, we also looked into reducing the model complexity by using the Traffic-Tiny architecture instead of the Traffic-Small architecture. In this way, because there are fewer parameters, Traffic-Tiny is less likely to over-fit to the dataset. Changing this

architecture also helps in overall computation time and memory.

As an alternative to the current Traffic-Small model, we also looked into freezing the preexisting convolutional layers trained on ImageNet and only back-propagate to the fully connected layers (Traffic-Small-Freeze-Conv).

We ran these five models for 54 epochs and achieved the mAP results in Figure 9.

Looking at the results, we find that Traffic-Tiny has the lowest validation accuracy, with 1.4% mAP. This is most likely due to the limited number of parameters within the Traffic-Tiny architecture. As a result, our hypothesis about reducing the model complexity was incorrect, causing Traffic-Tiny to underfit.

Traffic-Small-Jitter-0.2 and Traffic-Small-Dropout-0.75-Jitter-0.2 both seemed to also have trouble in producing reasonable results. This is most likely due to the fact that the jitter parameter was too high. Because traffic signs are usually around the peripheries of images, setting the jitter parameter too high would crop the traffic signs out.

The Traffic-Small-Freeze-Conv model seemed to also struggle in accurately predicting bounding boxes and classes. One possible reason is that the convolutional extraction weights were trained on ImageNet and most images in ImageNet have large objects in the center of the image. This property is undesirable for traffic signs, as most traffic signs are relatively small and are towards the edges of the image.

Of the five models, only Traffic-Small-Dropout-0.75 produced results comparable to our original model. It was also able to alleviate some of the overfitting issues in Traffic-Small as indicated by Figure 11. Specifically, we find that the final training accuracy for Traffic-Small-Dropout-0.75 was 56.3% mAP and the final validation accuracy was 27.4% mAP, resulting in a smaller gap between training and validation. Unfortunately, the Traffic-Small-Dropout-0.75 model validation accuracy did not improve over the original model.

In the end, as a means of improving overall accuracy, we combined Traffic-Small and Traffic-Small-Dropout-0.75 into an ensemble. While evaluating mAP for the ensemble, we removed duplicate bounding box predictions. Using this ensemble, we achieve a final result of 32% mAP on the validation set. Running the ensemble on the test set achieves 33.8% mAP.

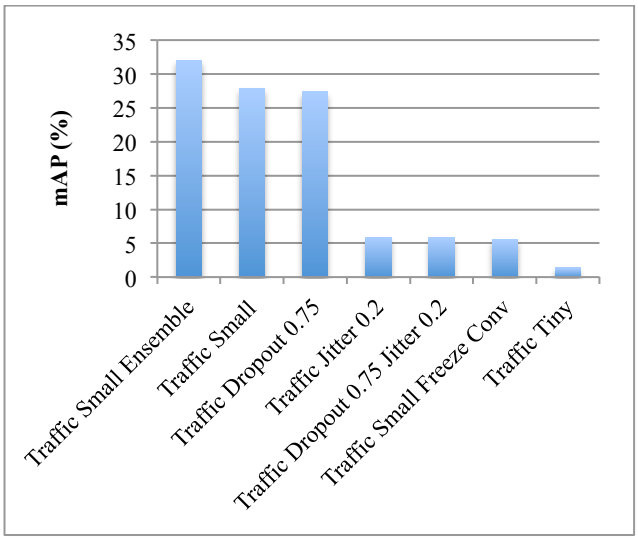

Figure 9: Model Comparisons.

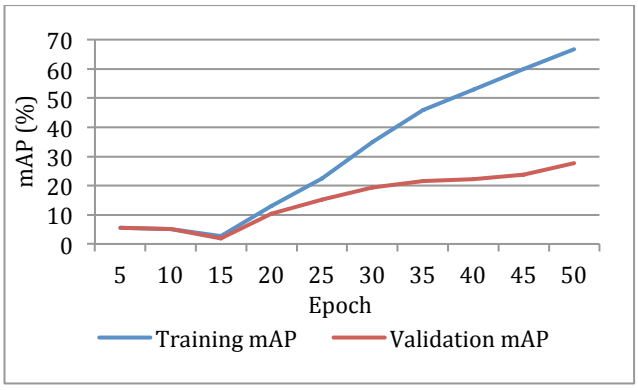

Figure 10: Traffic-Small Training Progression.

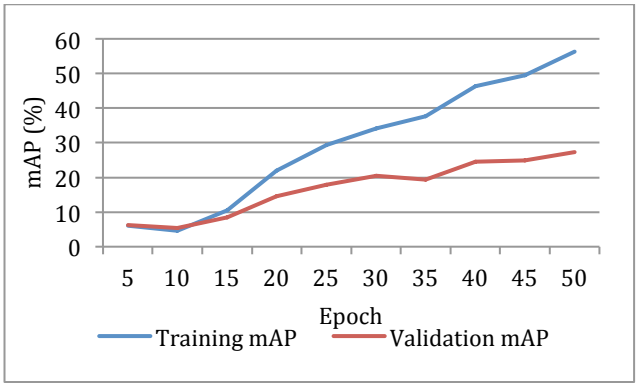

Figure 11: Traffic-Small-Dropout-0.75 Training Progression.

6. Error Analysis

Using our current architecture, we have achieved 28.3% mAP on the test set. In this section, we look into the distribution of the errors and the saliency maps to analyze the YOLO architecture.

6.1. Distribution of Error Types

Each detection is either correct or within the five main error types are localization, similar classification, other classification, background detection and no detection errors:

- Correct: correct class and IOU > 0.5
- Localization: correct class but 0.1 < IOU < 0.5
- Similar: similar class and IOU > 0.1
- Other: wrong class and IOU > 0.1
- Background: Any class and IOU < 0.1
- No Detection: No detection with confidence > 0.1

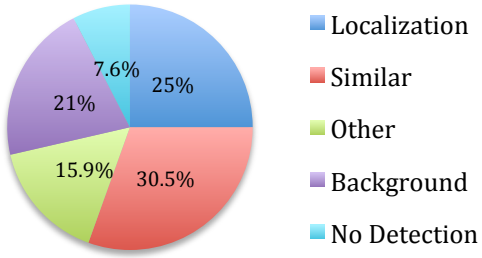

Figure 12: Error Distribution

From Figure 12, we see that the most prominent errors fall into either similar class error or localization error, which is similar to what was found in the original paper [1]. However, our top error is attributed to the similar class error instead of the localization error, reversed from the original paper. One possible reason is that YOLO architecture seems to weigh the global features from the entire image more for classification of an object than for the localization of bounding boxes. However, as most traffic signs can reasonably appear in any background scene, this global feature approach seems to impact the classification more negatively than it would have been for other types of classifications found in PASCAL VOC dataset.

6.2. Saliency Maps

We have found results that corroborated our understanding of the YOLO architecture. Figure 13 shows the saliency map that we get on a typical test image. It is interesting to note that in this image, there are the "blue circle" sign and the "stop" sign. Although only the blue circle sign is correctly detected based on the confidence threshold, it is incredible to see that the partially occluded stop sign also created positive response to the ground truth.

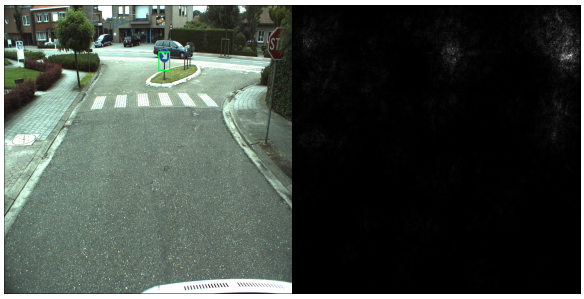

Figure 13: Saliency Map Generation

Aside from just applying saliency map on images with correct detections, we also use it to analyze the images where we have localization and similar classification errors.

Figure 14 and 15 is the result of applying saliency map on an image with localization error. We generated Figure 14 by using the coordinates the ground truth bounding box and Figure 15 by using the coordinates of the predicted bounding box. We see that the two saliency maps are almost identical, suggesting that the similar pixels may fire for the same class but for different bounding boxes. The main difference is the strength of these pixel level responses. The highest pixel level response in Figure 14 is 5% more than the highest pixel level response in Figure 15.

Figure 14: Saliency Map Generation on Ground Truth – Localization

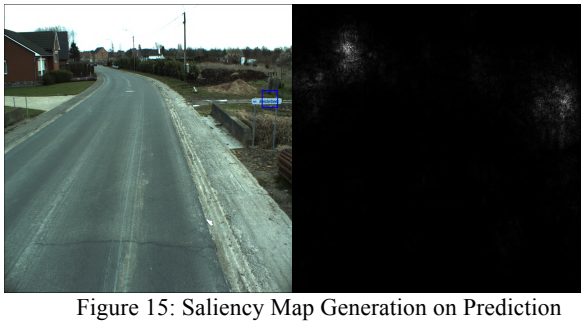

Figure 15: Saliency Map Generation on Prediction – Localization

Figure 16 and 17 are the saliency maps for an image with similar class error in predicted detection. We generated Figure 16 using the ground truth "blue-circle" class label and Figure 17 using the predicted "other" class label. What we noticed is that the pixels that have peak responses are within the bounding box, showing good

localization. However, pixels outside of the bounding box for the predicted class generated noticeable responses, confirming that YOLO detection pipeline does indeed look at the global image for detection. This is very different from the traditional approach where we conduct classification on proposed bounding boxes. As these traditional approaches tend to ignore the information outside of the bounding box when making a prediction. However, this global approach can be either advantageous or disadvantageous depending on the application. Since most traffic signs can occur anywhere regardless of the background context, this additional information seems to confuse the prediction more for our problem.

Figure 16: Saliency Map Generation on Ground Truth – Similar Class

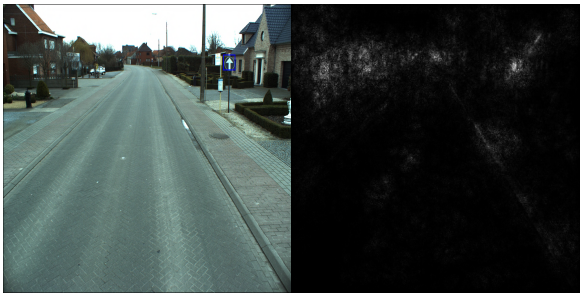

Figure 17: Saliency Map Generation on Prediction – Similar Class

7. Conclusion & Future Work

YOLO model has delivered what it promises, fast speed, but somewhat suboptimal accuracy. Our best model is based on the architecture presented in Figure 2, using 28 convolutional layers and 3 connected layers on top of the convolutional layers. This model has yielded around 28% mAP on the test set and at a blazingly fast speed of about 10 ms/frame, achieving the real-time requirement.

Although the dataset isn't particularly large even after data augmentation, we found only updating the last two layers attained a worse result than updating through all the layers. One probable reason is that the original convolutional layers were trained on ImageNet, where the object of interest is usually centered and large relative to the surrounding. However, this is not the case for the traffic signs within an image.

Furthermore, it seems like YOLO architecture is in general not very good at detecting skinnier objects like traffic signs, bottles and plants. This suboptimal performance was also found in the original YOLO model trained on PASCAL VOC 2012, where it achieved mAPs of mere 22.7% on bottles and 28.9% on plants.

Despite the low test mAP, we attained a relatively high training mAP, suggesting that we have overfitted the data. Hence we started training multiple other models trying to remediate this effect. In particular, we saw that using a higher dropout rate of 0.75 instead of 0.5 closes the gap between the training and validation mAP, and we strongly believe that with a few more epochs, the model trained with higher dropout rate will attain better performance of the model trained with lower dropout rate.

YOLO detection architecture is very different from that of traditional approaches with bounding box proposal before classification. YOLO is able to conduct classification on each of the proposed object based on the entire image, instead of only on the information within the bounding box. This may be very informative for some general detection applications, like the detections based on PASCAL VOC dataset, but not so much for traffic signs as many traffic signs can appear in various places.

YOLO is a new and developing framework, providing an exciting new way for object detection. Currently, there are very few supported modules and therefore we developed our own post-processing validation pipeline and saliency image visualization tools in C. We would strongly encourage people to use higher-level language frameworks to have faster development cycle.

As part of our future work, we would attempt to migrate the architecture to some more developed framework like Caffe or Torch to facilitate faster development cycle. In these more developed frameworks, we would like to try different regularization mechanisms like adding in batch-normalization layers and possibly adding in the regularization terms in the loss function. There are multiple hyper-parameters that are also built into the model, like the number of grids per image and the number of bounding boxes per grid that might seem reasonable to tune down as most images only contain at most 3 traffic signs. In addition to just changing different hyper-parameters of the YOLO architecture, it would also be interesting to see if the ensemble of the variants of R-CNN and YOLO will achieve better results on traffic signs.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv: 1506.02640, 2015.

[2] Dalal, N., Triggs, B., 2005. Histograms of oriented gradients for human detection. In: Proc. CVPR 2005, vol. 1, pp. 886-893. <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1%467360>.

[3] P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan Object Detection with Discriminatively Trained Part Based Models. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, No. 9, September 2010.

[4] Girshick, R. B. and Felzenszwalb, P. F. and McAllester, D.: Discriminatively Trained Deformable Part Models, Release 5.
[5] Sermanet, Pierre, et al. ”Overfeat: Integrated recognition, localization and detection using convolutional networks.” arXiv preprint arXiv:1312.6229 (2013).
[6] Segmentation as Selective Search for Object Recognition. Koen E. A. van de Sande. Jasper R. R. Uijlings. Theo Gevers. Arnold W. M. Smeulders.
[7] R. Girshick, “Fast R-CNN,” in IEEE International Conference on Computer Vision (ICCV), 2015.
[8] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. arXiv preprint arXiv:1506.01497, 2015.
[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results,” 2007.
[10] Radu Timofte, Markus Mathias, Rodrigo Benenson, and Luc Van Gool, Traffic Sign Recognition - How far are we from the solution?, International Joint Conference on Neural Networks (IJCNN 2013), August 2013, Dallas, USA.
[11] J. Greenhalgh and M. Mirmehdi, “Real-time detection and recognition of road traffic signs,” IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 4, pp. 1498–1506, 2012.
[12] The PASCAL Visual Object Classes (VOC) Challenge Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A. International Journal of Computer Vision, 88(2), 303-338, 2010.