DS-GA 1008 Deep Learning Final Project
Center for Data Science, New York University

Petri, Guido Sarwar, Nabeel Navarro Garaiz, Esteban

Abstract

For fully autonomous cars, it is essential to know the environment and objects around the vehicle in real time. In this paper, we introduce the Kobe neural network for this task: a single-encoder encoder network, which employs a YOLO-inspired decoder for object recognition and a fractionally strided convolution to generate a road-map.

1. Introduction

In order for autonomous driving to work, it is imperative moving cars can recognize and react to the environments around them. Being able to accurately portray the road on which the car is traveling, the other cars in the road, as well as signals and pedestrians is one of the biggest hurdles to making fully autonomous driving a reality. Given that radar/lidar do not offer a satisfactory solution, the industry has turned to Deep Learning. In this project, we attempt to solve the problem of road and object recognition simultaneously. The data is comprised of 25-second scenes of a car's journey. Each of these has 6-image samples every 0.2 seconds, with the six images coming every $60°$ degrees around the car with a $70°$ view angle.

2. Literature Review

Image processing is interested with the directions of change in pixels, as these happen when there is an object present. We can recognize these changes with a "gradient" on pixels:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f(x+1,y) - f(x-1,y) \\ f(x,y+1) - f(x,y-1) \end{bmatrix}$$

Unfortunately, repeating the gradient computation for every pixel is too slow. One simple solution for this is applying a convolution operator on the entire image; for instance,using the kernel $[-1, 0, 1]$. This is what Deformable Parts models do: they use an equally spaced sliding window with a classifier to detect if an object is there or not. (Weng, 2017)

2.1. Selective Search

Selective search is a common algorithm to provide region proposals that potentially contain objects. It is built on top of image segmentation, which uses a graph-based approach to divide an image into similar regions. It builds an initial proposal based on (Felzenszwalb & Huttenlocher, 2004) and hierarchically binds regions together, similar to hierarchical clustering.

2.2. R-CNN (Girshick, 2015)

Region-based Convolutional Neural Networks uses selective search to identify a number of bounding-box candidates (called regions of interest; approximately 2000 candidates per image). After this, it extracts CNN features for each region to classify them. For the model to work, the CNN needs to first be trained on an image classification task, like ResNet. These features are passed through a binary SVM, trained for every class independently - plus one background / no object class - and thresholded by their intersection over union score. Non-Maximum Suppression gets rid of multiple bounding boxes for a single object by sorting confidence score in descending order and discarding boxes greedily if they have an $IOU > 0.5$ with the highest scoring box. Because the features are generated after the boxes, the network only sees part of the image, and hence fails to take context into account. Furthermore, this decoupling of the process into separate steps leads for the model to take 40 seconds per image at testing.

2.3. You Only Look Once (Redmon et al., 2016)

YOLO does both parts in one pass: it sees the entire image simultaneously and once, by formulating the problem directly as a regression from raw input to bounding boxes and class probabilities in a single convolution network. This allows it to encode contextual information when training and to have real time performance (45-150 FPS) while maintaining the precision of the slower systems. Furthermore, it learns generalizable representations that allow it to outperform other methods on never before seen types of images. YOLO divides images into $S \times S$ grids and assigns an object to a cell if its center is on it. It allows each grid cell to predict B bounding boxes and their respective confi-

dence score, which depend on the $\mathbb{P}(\text{Object in box})$ and how good the box is fit, measured by the IOU of predicted and real box. This way, the box can be characterised by the center of the box (x, y), the height h, the width w and the confidence. Each cell also has a vector of $\mathbb{P}(\text{Class}_i|\text{Object in Box})$ which allows to marginalize the conditional.

Because of the way it runs through the grid, YOLO struggles with small objects compared to its competitors. It also struggles with objects in unusual aspect ratios and configurations that it hasn't seen before.

2.4. Roadmap Detection

The problem of generating a bird's-eye-view map surrounding a moving car is an open and active area of research. (Pan et al., 2019) use a similar six-image set-up to the one in our data to come up with a semantic map at the pixel level, representing the objects in the image. Because of limited real-world annotations, they need to generate synthetic data for training. Their framework requires the network to generate a depth reading, as well as use RGB inputs which then go into a discriminator to decide if the image is real or simulated.

(Mani et al., 2020) takes in a color image, and aims to predict all static (mainly, road and sidewalk) and dynamic (mainly, cars) points on the ground within a certain range in front of the car, whether they are present in the image or not. They do this by producing an amodal BEV layout from an architecture that encodes the context, decodes into the amodal layout, and two discriminators which decide if the static or dynamic points are real or fake.

3. Kobe Architecture

3.1. Pre-Training Task

For our pre-training task, we train a small network to predict from which of the six cameras the image is coming from. The intuition behind this is making the network learn to encode some notion of geographical location from the road context. The network is a two-layer convolutional network with ReLU activation functions and max-pooling. After this encoding is done, the output is fed into two fully connected linear layers with ReLU activation, which are then passed through a log-softmax to generate the final probability output. (Canziani, 2019)

The network predicts with 99.86% accuracy after only two epochs. Figures 1 and 2 have two of the mistakes of the network. In Figure 1, the network is looking at the direction of the parked car - which is right to left - and determining this is a back right photo, when in reality it's a front left and the context car is backwards. In Figure 2, the car that could give the context on the bottom right corner is very dark, as are all other cars in the image, so the network has no way to determine the direction the car is traveling in.

Figure 1. Parked car in opposite direction confuses network

Figure 2. Dark cars can't be used as direction reference

3.2. Main Architecture

Since YOLO uses only bounding boxes that are parallel to the axis, the data set provided has tilted boxes, and we couldn't find a computationally efficient way of calculating IOU on tilted boxes, we decided to surround these by the smallest possible box that contains the tilted box to correct the tilting. This limits the ability of the network to learn tilted boxes and reduces IOU scores for them, with the advantage of simplifying the network architecture (for instance, by not having to calculate the angle of the bounding box) and reduced training time. This takes us from the 8 coordinates of the corners to a vector with $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$.

After fitting the non-tilted box, the encoder converts the (x, y) coordinates of the bounding box to the coordinates that YOLO expects: a tuple referencing to which grid cell they belong, their center relative to the cell, and the width and height of the bounding box normalized to the image size.

The main architecture has two decoders, one specific to each task.

For object detection, the architecture is inspired by YOLO (Redmon et al., 2016) with a grid size of 16 and 2 possible bounding boxes per cell. Given that the images are $80 \times 80 = 64000m^2$, and the typical car is $4.5 \times 2 = 9m^2$, we chose to use a 16×16 grid, which makes every cell have $64,000m^2/16^2 = 25m^2/\text{cell}$. This makes the case where there will be more than one car per cell unlikely, and makes objects per cell be proportionally scaled to those for which the original YOLO network was designed on.

For each cell, the network predicts 2 bounding boxes in a $(x, y, \text{height}, \text{width}, \text{confidence})$; where the last is a score for how confident the network is that a box is actually there. Instead of per box, there is a vector of class proba-

bilities per grid cell. This means the final output has shape $(\text{number of samples}, S, S, (5 * B) + \text{number of classes})$

Much of the implementation is done through PyTorch masks and the architecture for this was defined by us, though we used the code in (Kimura, 2019) as reference for it. The YOLO loss is calculated in this paradigm, and is compromised of multiple parts:

- MSE loss for the confidence, with different parameters for the cases when there is and isn't an object. This makes YOLO flexible to weighting false positives and false negatives according to the application.
- An MSE loss for the square root of the width and height, class probabilities and center coordinates.

We combined the training for both tasks to force the model to learn more general representations in the layers, as opposed attacking each task individually. This is based off the idea, mentioned by Dr. LeCun in class, that teaching your network an additional task simultaneously might make the original task better.

3.3. Alternative Architectures

Upon creating the baseline model, and inspired by the results of (Roddick & Cipolla, 2020), we decided to include attention into our model. This was achieved by adding a two-headed shared decoder before the individual decoders. The shared decoder takes the features from the Pre-Task Encoder and outputs something that, in theory, would benefit both tasks. The attention does not have masks, so every sequence is capable of seeing the others. This is one further step in trying to make the model learn general features.

In detail, it takes the embeddings of the 6 individual images in a sample and turns it into a non-linear sequence of the 6 embeddings with attention. The output matches the dimensions of the input.

We tested the shared decoder with and without the transformer, and with and without batch normalization in the individual decoders. Batch normalization improves the roadmap. Turning the attention on benefited the object detection, but hurt the roadmap's performance. We believe this is because the roadmap task depends on all 6 images equally, as each is responsible for a part of the road; whereas for the object detection task, attention can guide the network to relevant sections of the images that are more likely to feature cars.

Given the above results, we created a variation where the decoder with one-headed attention fed representations to the object detection decoder only; the roadmap decoder received its representations directly from the Pre-Task encoder. Both individual decoders had batch normalization before the non-linear layers (except for the last sigmoid). On early epochs, this had the best performance of all the networks we tried, as it can be seen in Table 2.

Model, training results	Epochs	BB score	RM score
Baseline	4	.010	.514
Shared decoder, attention	4	.016	.4375
Baseline	42	.04	.644
Binary Train/Test Baseline	60	.006	.52
Binary Baseline	113	.059	.777
Baseline113 + Frozen RM	**113+30**	**.083**	**.777**
Binary Baseline	120	.065	.772

Table 1. Main Architecture results

Model, 10 epochs, validation results	BB score	RM score
Shared decoder, no attention	.0024	.3646
Shared decoder, no attention, batch	.0007	.4763
Attention BB, Pre-task RM, batch	**0.02**	**0.75**

Table 2. Alternative Architecture results

4. Results

The results for our main architecture and alternative architectures can be seen in Tables 1 and 2. A significant portion of our time was spent debugging models and their hyperparameters to get them to work. Along the way, we also discarded a few configurations, given their low bounding box score. Once we had a fully-functioning structure we were happy with, it was too late to fully train it. The only version we had trained for a significant number of epochs was our baseline model, one we had coded for the second submission to see if the model was learning at all and to test that our final deliverable format was working. Given that a YOLO style network requires extensive training - per the paper, the authors had to train theirs for "approximately a week" - and we were close to the final submission deadline, we decided the best course of action would be to stay with our baseline model, which was trained on the entire dataset, as opposed to using one of our alternative configurations with a proper train and validation set split.

Our reasoning was that a network that requires that much training will not be in overfitting regions early on and would be unlikely to overfit. Alas, when analysing the results, we found out the network did not generalize well to the out of sample test set, indicating we were wrong to assume this. While our Bounding Box performance on the final test set put us in the 21st position, our roadmap performances lagged behind it and placed us in the bottom half of the competition. The training scores for this architecture were .083 for BB and .777 for RM, as can be seen in Table 1. The object detection had an overall accuracy of 95.3%.

To get a sense of the model's overfitting, we trained a model that matches out final submission model's structure with training and validation sets. After 60 epochs, or less than half the training of our final submission model, it obtains a validation score of .006 for the bounding box task and 0.52 on roadmap.

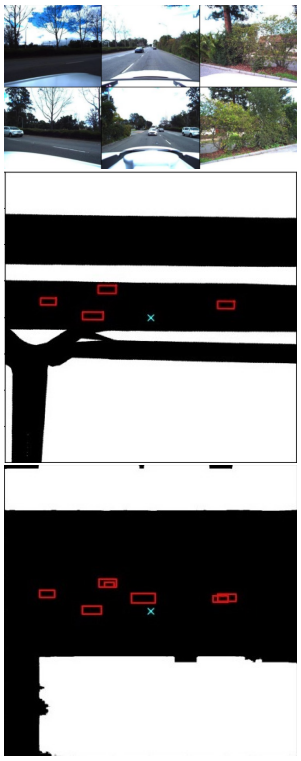

Figure 3. Car images, real and predicted bounding boxes for a successful example. The model picks up all four cars, and only shows some depth uncertainty

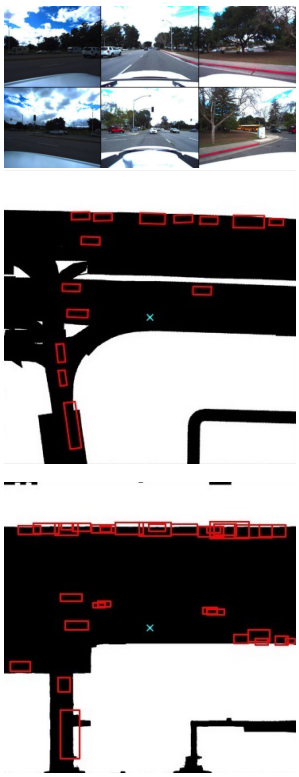

Figure 4. Car images, real and predicted road for a successful example. The model picks up the roads in the horizontal and vertical directions

5. Future Work

- Fixing the training paradigm and including a framework that is reproducible are the obvious next step from our submission model, and indeed, we have already started taking strides in this direction by re-training the delivered model under a train / test split.
- Under this framework, we would next train our best performing variation, as described in Section 3.3 for many epochs, as we believed it would have vastly outperformed our baseline model.
- Based on (Mani et al., 2020), adding an adversarial loss could make the bird's eye view roadmaps look qualitatively better. We wanted to test whether this also helped the quantitative performance of the model. We would have an additional discriminator network, trained to tell apart actual roadmaps from those generated by the decoder, and the decoder would be encouraged to fool the discriminator, yielding more realistic roadmaps. This would have likely added much needed regularization to our framework.
- We would have liked to have extended both the YOLO loss and the roadmap loss with an uncertainty loss. As we learned in class, this can help regularize models and increase performance.
- We believe under the new framework, we could have made the learning rate higher. We would have liked to experiment with different optimizers like SGD; we only used ADAM.

References

Canziani, A. Pytorch-Deep-Learning, 2019. URL https://github.com/Atcold/pytorch-Deep-Learning/blob/master/06-convnet.ipynb.

Felzenszwalb, P. and Huttenlocher, D. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:167–181, 09 2004. doi: 10.1023/B:VISI.0000022288.19776.77.

Girshick, R. B. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, 2015.

Kimura, M. YOLO-v1 Pytorch, 2019. URL https://github.com/motokimura/yolo_v1_pytorch.

Mani, K., Daga, S., Garg, S., Shankar, N., Jatavallabhula, K., and Krishna, M. Monolayout: Amodal scene layout from a single image, 02 2020.

Pan, B., Sun, J., Andonian, A., Oliva, A., and Zhou, B. Cross-view semantic segmentation for sensing surroundings. *CoRR*, abs/1906.03560, 2019. URL http://arxiv.org/abs/1906.03560.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. pp. 779–788, 06 2016. doi: 10.1109/CVPR.2016.91.

Roddick, T. and Cipolla, R. Predicting semantic map representations from images using pyramid occupancy networks. *ArXiv*, abs/2003.13402, 2020.

Weng, L. Object detection for dummies part 1: Gradient vector, hog, and ss. *lilianweng.github.io/lil-log*, 2017. URL http://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html.