KITTI Vision Benchmark Suite
Brigham Young University
2019 Winter CS478

Tomy Huang **Thomas Niedfeldt** **Kolby Nottingham** **Nick Walton**

Abstract

Tens of thousands of people die in the US every year in car accidents. Self driving cars enabled by Lidar (Light Detection and Ranging) have the potential to prevent many of these deaths. But advanced machine learning algorithms are needed to enable autopilot systems to accurately detect and classify objects from camera and Lidar sensor data. Significant work has already been done on reliably detecting objects in images, but not as much research has been done on detecting objects with Lidar. Lidar sensors produce 3D point clouds that are invaluable for understanding the environment around the vehicle. The main challenge of using these point-clouds is that—as opposed to camera images—point clouds are sparse representations of 3D objects and can be computationally expensive to use as features. We plan to reconstruct these point clouds into several 2D planes that can more easily be fed into a deep Convolutional Neural Network (CNN) for object detection.

1 Introduction

The problem of object recognition is fundamental to the development of self-driving vehicles. Outside of the self-driving-car domain, object recognition has been a common task for research and more practical applications. Computer vision is a field of computer science research that aims to improve the way that machines visualize the world. It seeks to give computers a high-level understanding of objects in the visual domain. For instance, recent work on Google Lens provides users with the ability to recognize plant and animal species, products, and other objects via a smartphone camera. This is part of the computer vision sub-field of object detection. Data sets like ImageNet are examples of the object detection problem. ImageNet is composed of millions of hand-labeled images that researchers can use to train image classifiers. Our problem also lies within the object detection task but in the 3D domain rather than 2D. We seek to train a classifier to identify objects with previously-labeled data gathered using 3D sensors, rather than 2D cameras.

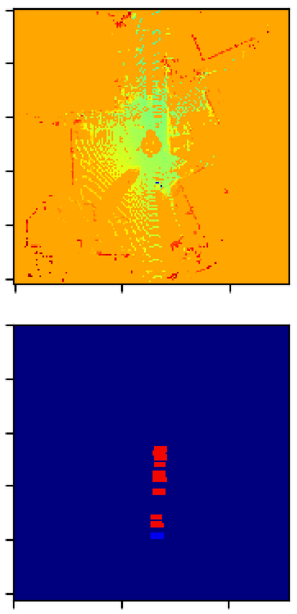


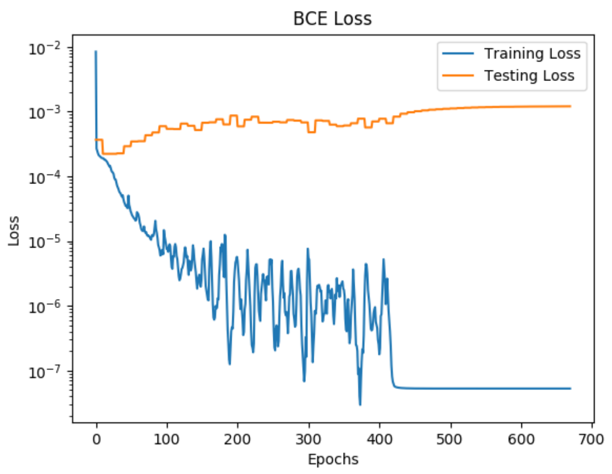
Figure 1: Example Lidar bird's-eye view (top) and object label (bottom) images from the training data set used as inputs and targets, respectively. The input image shows one of three height channels used to train our model. In the target image, the smaller rectangular boxes represent object locations and colors represent object types.

1.1 Motivations

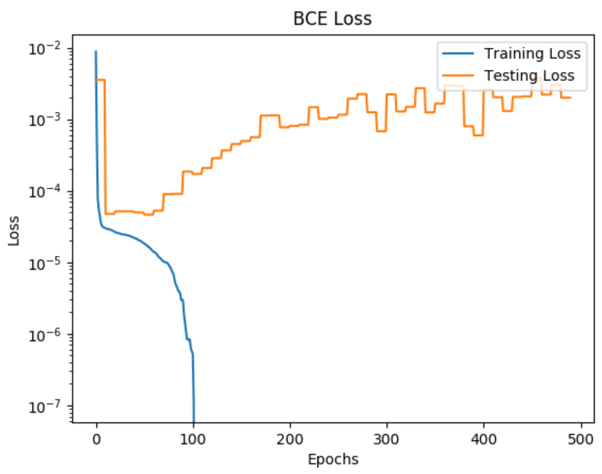
Self driving vehicles are quickly becoming a reality and have the potential to save many lives. Despite recent advancements, reliable computer vision remains one of the key limiting factors to this technology's success. We wanted to create a reliable and simple object detection model that could run quickly on simplified data.

2 Methods and Data

This section discusses how KITTI Vision data and features were chosen and processed in our approach.



(a) Initial Model



(b) Second Iteration Model

Figure 2: Training and Testing Loss over time for our initial and tuned model. The model quickly overfit and testing loss started increasing after 40 epochs. After adjusting hyper parameters, our model achieved lower testing loss before overfitting.

2.1 Data Source

The KITTI vision benchmark suite is a project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago [1]. They have gathered and open sourced a data set of 7481 training point clouds and 7518 test point clouds. Each point cloud contains a set of labeled objects, with a total of 80256 labeled objects. These labels include 3D bounding boxes for cars, pedestrians, and cyclists.

2.2 Data Instances

In order to efficiently run our data through a CNN we decided to compress the 3D point cloud into a 256x256 grid with three channels. The three channels corresponded to average height, average reflectively, and point density that were projected into that grid square. We first removed every point that fell outside of a 100m x 100m area centered on the car. We then projected each point inside that area into the grid box it fell into and calculated the values for each channel. A sample 2D plane might look something like the top image in figure 1.

Similarly the target data for the model is a 256x256 grid with 9 channels where each channel represents a class (car, bicycle etc...) a 1 is placed in the location boxes where the labels indicate an object of that type resides while 0's fill the rest of the grid. Our visualization in the bottom of figure 1 demonstrates the target data structure, with different colors to represent the type of the object in the scene.

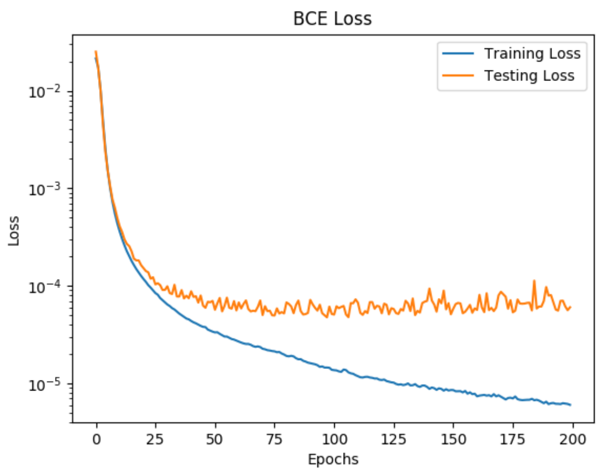
Reprocessing the data into these features allows us to feed a much smaller and denser representation of the point cloud into our network as well as enabling us to use the spatial computation of CNN's. This helps make our algorithm much more efficient then if we had used the entire raw point cloud for each frame. With the average height, reflectively and point density for each grid box we are able to achieve good performance on the test set.

2.3 Models

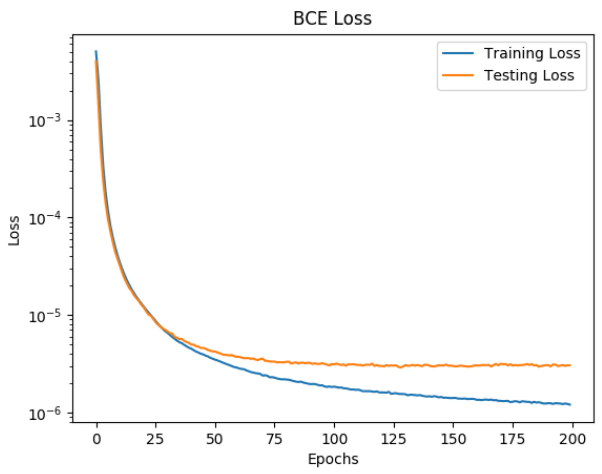
CNN's have shown significant improvements over the state of the art in computer vision. In 2012 AlexNet, a deep CNN achieved state of the art accuracy by a wide margin on the ImageNet data set [2]. This event marked a rise in popularity of deep neural networks and, ever since, the ImageNet competition has continued to be won by similar deep convolutional models. Given this success we decided to use a deep CNN in our task.

Neural networks are computing systems inspired by the biological brain that rely on neurons for their structure—such as the multi-layer perceptron. CNNs implement a calculation called a convolution between their layers that captures spatial data. For this reason, they are ideal for image data. Deep networks refer to networks with many layers. The deeper layers use the output of previous layers to learn higher level features about the data. In the case of convolutions, deeper layers are able to learn to recognize shapes and patterns in images.

Our classification model relies on a pixel to pixel like structure CNN structure. Several convolutional layers downsize the input image into a 128-dimensional vector. Then, transposed convolutional layers upscale the image back into its width and height, but with 9 channels, one for each class. This process encourages the first encoder layers to learn simplified representations of the input data, which can then be used to compute the position and type of detected objects. In this case, our output was a mask labeling the individual objects in our input scene.



(a) Final Model



(b) Large Dataset Model

Figure 3: Training and Testing Loss over time for our final model. Using dropout, we solved our model's overfitting problem and achieved better testing loss. We then experimented using a data set seven times larger than the one used in the other experiments. This larger data set decreased testing loss even further and significantly smoothed our model's loss.

3 Initial Results

Our initial experiments used a training set of 1000 instances and a test set of 100 instances sampled from the KITTI data set. The data was processed using the methods described in section 2.3. Input data was passed forward through our network and the network's output was compared to the target output using Binary Cross Entropy (BCE) Loss. We plotted the loss of the training and test sets throughout model training.

Our initial model performed well on our training set, but overfit the training data and failed to perform well on our test set. The results of our initial model are illustrated in figure 2a. Both training and test sets performed well at first but then the test set loss would start to increase as the training set loss continued to decrease.

4 Data and Feature Improvements

Models overfit due to a failure to generalize. Rather than learning a general mapping from LIDAR images to object labels, our model overfit to the training data set and only learned how to map those instances to their respective outputs. Several methods exist to prevent overfitting. These include tuning, improved or simplified architectures, larger data sets, and regularization techniques.

Our first approach involved hyper parameter turning. We tried a range of learning rates and mini-batch sizes. As shown in figure 2b, this improved testing loss and the model took longer to overfit, but the testing results were still not satisfactory. Since the model always ended up overfitting we decided to adjust our architecture.

Eventually, we decided to add batch normalization and a dropout layer to our convolutional auto-encoder network. Batch normalization normalizes the data at each activation layer of the network during training, helping the network to learn a general mapping of the data sooner [3]. Dropout randomly zeros a portion of the nodes during training each forward pass [4]. This helps the model to make full use of its weights since it cannot rely on any particular set of weights. During testing all weights are used. Dropout essentially creates an ensemble of different weight combinations during testing. After some experimenting, we decided to add the dropout layer in-between the encoder and the decoder of our network. This improved our testing loss dramatically.

5 Final Results

By adding dropout between the auto encoder's encoder and decoder module, adding batch normalization at each convolutional layer, and tuning our hyper parameters, our model was able to correctly identify objects in both the training and testing data sets. Figure 3a shows the results of our model on the same data set used for our initial tests. Our final experiment used the same model on a 6733 instance training set and a 748 instance test set. The larger data set improved both the training and testing loss even more and smoothed out the loss curves.

Figure 4 shows example output from the test set at 0, 10, 100, and 200 epochs. This scene contains a car, a truck, and a cyclist. After 10 epochs the model already learned the location of several objects, but was unable to identify what object they were. After 100 epochs the model performed with few errors, and after 200 epochs the model was near perfect on this testing instance.

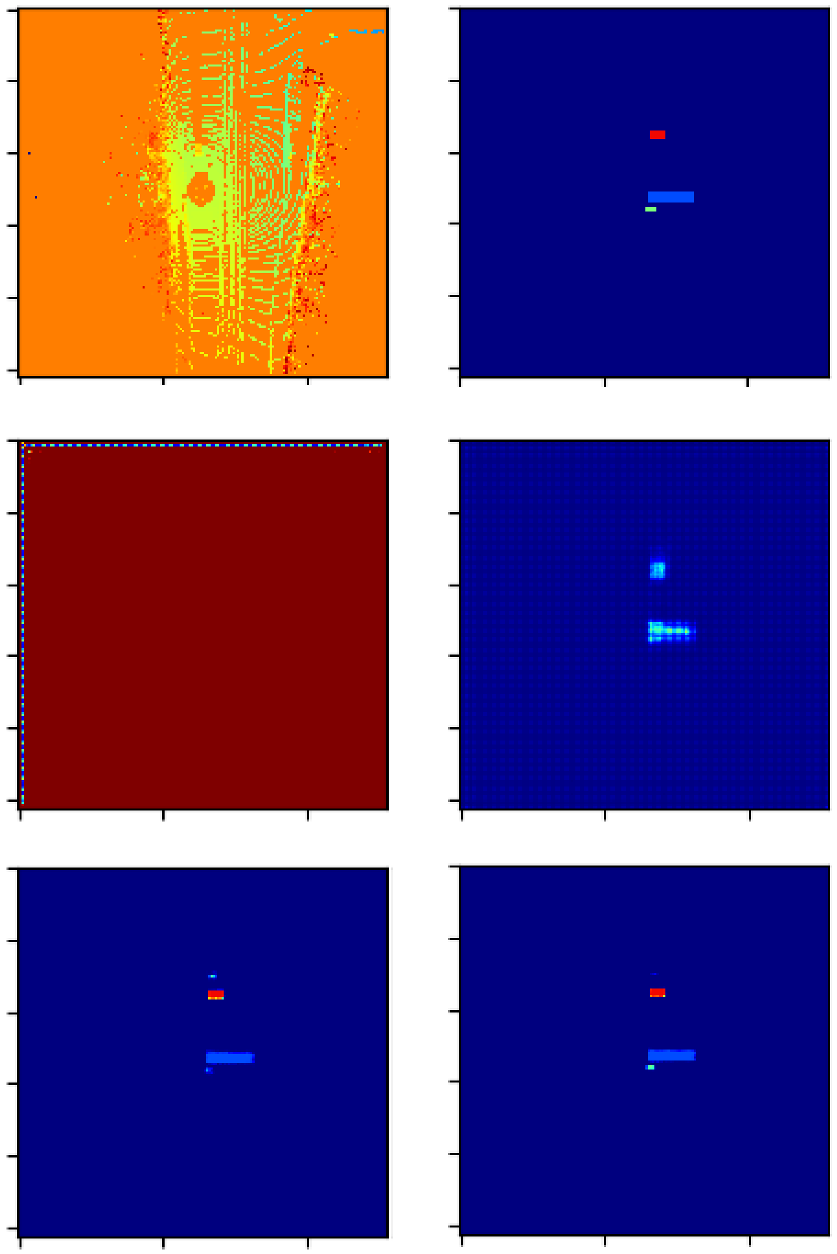


Figure 4: Visualizations of input, target, and output data sampled from our testing data set. This scene contains a car, a truck, and a cyclist. The top row of images depicts the input channel (left) and the target image (right). The bottom four images show the model's output after 0 (middle left), 10 (middle right), 100 (bottom left), and 200 (bottom right) epochs.

6 Conclusion

We found that we were able to learn an object detection model of 3D data transformed into 2D input. Our outputs, a bird's-eye semantic image of objects in the scene, correctly matched the target images for most testing instances. We consider our work successful.

7 Future Work

Although we were able to correctly produce 2D output images that matched our target semantic images, some of the more complicated scenes in our test set were difficult for our model. Future work may focus on making our method more reliable or expanding it to richer domains. Our bird's-eye semantic images were accurate but may not be detailed enough for the purposes of self-driving cars. Our methods should be expandable to more complex representations that incorporate the shape of the objects rather than simple rectangles, as well as height information.

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.