ROB 535 Perception Project

Fall 2021, Team 21
University of Michigan, Ann Arbor
GitHub Repository: https://github.com/ajeffries0492/vehicle-recognition-and-localization

***Abstract*—We present a method of using a deep neural network to classify vehicles in still images taken from the video game Grand Theft Auto. First, we used a Single Shot Detection (SSD) algorithm from the TensorFlow Object Detection API to detect the vehicles within the overall image. Bounding boxes were then drawn around those detected objects and the image background was removed, resulting in a smaller image of only the pixels within that bounding box. Finally, after training a ResNet-50 neural network using ground truth bounding boxes from the training data, the network was able to classify those detected objects into one of three vehicle classes when given new test data. The overall performance of the algorithm resulted in an accuracy of over 73 % in the competition.**

***Index Terms*—perception, detection, localization, TensorFlow, SSD**

I. Introduction

In recent years, the automotive industry has preferred utilizing cameras over other sensors due to their low cost of implementation, especially when compared to RADAR and LiDAR. They allow the vehicle to "see" its surroundings by providing data to process and identify objects around the vehicle. However, utilizing this camera information is often difficult. To identify objects such as vehicles and even classify the differences between surrounding object types, the system must be able to parse complex data and process it through the model to generate useful detections. For instance, it is important to differentiate between a biker and a bus for predicting behavior. Furthermore, localizing these objects with respect to the vehicle enables motion controllers to plan a path in automated driving systems.

This project looked at a simulated Grand Theft Auto camera and LiDAR dataset and sought to detect, classify, and localize vehicles (as shown in Fig. 1) when presented with a new similar test dataset.

II. Methods

A. Task 1

The first task was to recognize a vehicle and classify it into one of three classes from a novel test image. The classes often included overlapping vehicle types. For instance, a compact, sedan, and SUV all shared the same class of 1.

Training an SSD Object Detector [1] for Three Classes: Upon achieving an acceptable accuracy from training on full sized images, the team hoped to improve results by training and testing on only regions of interest in an image. After doing a literature survey, many methods were discovered including Faster R-CNN [2]. Faster R-CNN seeks to find region proposals at the same time as it classifies images. This approach seemed promising for finding regions in our full-sized images. Furthermore, TensorFlow has an Object Detection API [3] that leverages object detection models like Faster-RCNN and SSD [1]. Using these pre-built models which were trained on the COCO dataset, we trained an object detection model that took in a full-sized image and output both bounding boxes and class labels. This method worked well for finding region proposals of vehicles in test images, but the classifications were poor.

Utilized ResNet-50 on Detection Boxes: To further improve improve our classifications, we trained a ResNet50 classifier on the ground truth bounding boxes. Keras includes several pre-trained as well as well-defined convolutional neural networks for use. These pre-trained neural networks are generally trained on ImageNet, COCO, or KITTI datasets. The layers as defined can be re-trained using the image data for this project. In addition to these layers, we added four layers after the provided layers to work with a three-class classification. These included a flattening layer, a dense layer with ReLU activation function, a dropout layer, and finally the Softmax prediction layer. The overall neural network parameters used are listed in Table I. In this implementation, we froze the first two layers of the off-the-shelf ResNet-50 neural network, which generally capture the more common features. The rest of the provided layers and our added layers were trained on our training data. In order to improve the performance of the network, we used the Keras Data Generator - random rotations and horizontal flipping - to help balance and vary the provided training data.

B. Task 2

The method used to localize the vehicle relative to the camera involved making the initial assumption that the center of the bounding box, c, from the classification task is centered on the vehicle. The team then used this center point, c, to find the nearest LiDAR data point, l, by finding the index with the smallest linear distance between c and all the points in the point cloud. The LiDAR point cloud distance at this point l was used as the localization value for the vehicle.

III. Results

A. Task 1

The parameters of our neural network provided good loss that decreased epoch-by-epoch and showed that our model is indeed learning. On of our other interested parameters that defined neural network performance was precision-recall. This metric, along with other metrics, are shown in Fig. 2. In the figure, we show how these parameters vary for both the training data and the hold out validation data. These metrics yielded a decent confusion matrix (shown in Fig. 3) for our hold out test data.

Ultimately, our neural network provided over 73% classification accuracy on the Kaggle test dataset.

B. Task 2

An example of the calculated center of the predicted bounding box is the white star in Step 3 of Fig. 1. Utilizing the predicted bounding boxes in the test data produced localization results with a root mean squared error (RMSE) of 10.35494, beating the baseline of 18.78165.

IV. Conclusions & Future Work

Currently, our classification network is trained to only use the image pixel data within the 2D predicted bounding box on the image for classification. While we showed that this gives adequate classification accuracy, it does, however, have a significant error rate. One potential opportunity for improvement would be to also include the LiDAR point cloud data in the object classification step. Given the 2D predicted bounding box from the detection step, a clustering algorithm could be used to determine a 3D bounding box of LiDAR point cloud data at the location of the detected object for improved classification.

This method would require that the LiDAR point cloud data be included when training the classification network. The provided training data already includes 3D bounding boxes of the object and the point cloud data within it. These LiDAR points could then be included with the image data so that the classification network is trained on the combination of image and LiDAR data. Instead, for potentially more improvement, a separate LiDAR-only classification network that uses these 3D bounding boxes could be trained. Once the LiDAR points given from the clustering algorithm are utilized with the proposed LiDAR-only network, the results of both networks could be fused to further increase the accuracy of the prediction.

Team Member Contributions

For Task 1, Austin Jeffries wrote the entire codebase in Python while utilizing Jupyter Notebooks and Google Colaboratory as needed. He researched different off-the-shelf neural networks. He compared and contrasted them to see which one could run on his computer and on the cloud. He worked on the methods listed above. He trained the neural network and produced CSVs for Kaggle. He attended office hours to find improvements to the methods. He also attended the team meetings to share his findings and results throughout the project period. Arnav Sharma used Austin's code setup to train more iterations (with different neural network parameters) and produced CSVs for Kaggle. Alexander Jaeckel attended the project meetings and also trained iterations of the final neural network setup. Saurabh Sinha attended the project meetings and also trained iterations of the final neural network setup.

For Task 2, the team collectively helped Austin write the codebase in Jupyter Notebook. We brainstormed the idea together, wrote and submitted the CSV to Kaggle.

The team completed the project paper together.

Acknowledgment

We would like to acknowledge Professor Matthew Johnson-Roberson at the University of Michigan, Ann Arbor, for teaching us different neural networks and how they relate to current-day problems. We thank the GSIs Ming-Yuan Yu and Tianyi Zhang for their help during office hours to help us improve our methods and results for Task 1.

References

[1] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, A. Berg, "SSD; Single Shot MultiBox Detector," CoRR, 1512.02325, 2015.

[2] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," CVPR 2017, 2017.

[3] S. Ren, K. He, R. Girshick, J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," CoRR, 1506.01497, 2015.

FIGURES

Fig. 1. General Approach

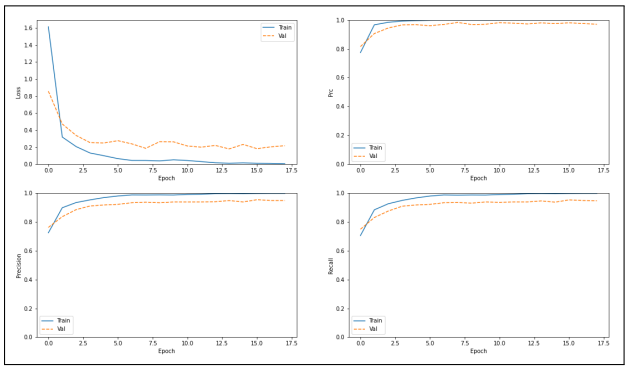

Fig. 2. Metrics of Neural Network Across Epochs. Top-left: Loss; Bottom-Left: Precision; Top-Right: Percent Recall; Bottom-Right: Recall

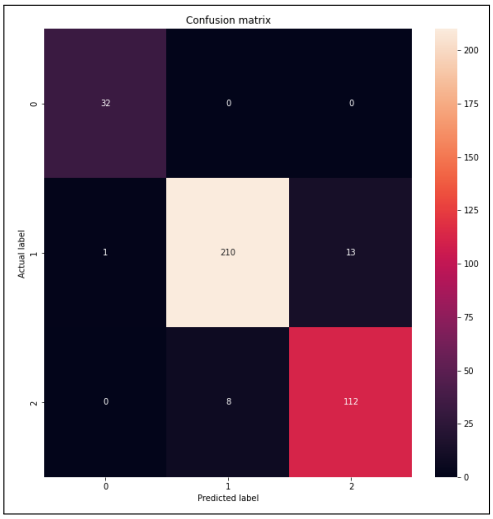

Fig. 3. Confusion Matrix Comparing Actual vs Predicted Class after Training the Neural Network

TABLES

TABLE I
RESNET-50 NEURAL NETWORK PARAMETERS

Dropout Percentage	20%
Early Stopping Patience	10 epochs
Validation/Test Data Split	15%
Test Data Split	20%
Batch Size	325
Rotation Range	$\pm$25 deg
Horizontal Flipping	YES

APPENDIX A

Previous Iterations:

The team experimented with multiple different methods, often combining them to find the best outcome. These included methods such as SIFT features extraction, scaling images, using different models within the TensorFlow Object Detection API [2], and different Keras models to classify the vehicles. The details of each method are detailed below.

Full Image Classification: One of our first set of approaches sought to classify the full sized image into one of the three classes. With the full sized images, we first tried a more traditional machine learning approach which first extracted SIFT features from the image prior to running it through a classifier. The reason that SIFT features were chosen was because they are scale-invariant features and are often useful when needing to describe objects (such as cars) at multiple scales. After extracting these features from each image, we split the data into validation and training data. With the data split we ran the features through the Support Vector Machine to see if the training data could differentiate between our three classes. This method seemed to be acceptable but did not achieve state-of-the-art results.

Expanding upon this method, where we use the full sized images, our next approach used a popular convolutional neural network to classify the images. As opposed to extracting features beforehand, this method took in the full sized images to learn both the features and the classes. The first network tried was VGG16. One nice things about the TensorFlow library is that it has pre-built and pre-trained versions of popular convolutional models like VGG16 and ResNet-50. Taking the pre-built VGG16 network, we added a few custom layers - flatten and dense - prior to a Softmax output into the three classes. Overall, this method - including the transfer learning - ended up working quite well and was able to achieve a 58 percent accuracy.