Multimodal Fusion Object Detection System

Michael Person, Mathew Jensen, Hector Gutierrez, Anthony O. Smith

IGVC Spec 2 Team
College of Engineering and Computing
Florida Institute of Technology, Melbourne, Florida 32901
mperson2016@my.fit.edu, mjensen@fit.edu, hgutier@fit.edu, anthonysmith@fit.edu

ABSTRACT

In order for autonomous vehicles to safely navigate the road ways, accurate object detection must take place before safe path planning can occur. Currently, general purpose object detection CNN models have the highest detection accuracies of any method. However, there is a gap in the proposed detection frameworks. Specifically, those that provide high detection accuracy necessary for deployment but do not perform inference in realtime, and those that perform inference in realtime but detection accuracy is low. We propose Multimodal Fusion Detection System MDFS), a sensor fusion system that combines the speed of a fast image detection CNN model along with the accuracy of light detection and range (LiDAR) point cloud data through a decision tree approach . The primary objective is to bridge the trade-off between performance and accuracy. The motivation for MDFS is to reduce the computational complexity associated with using a CNN model to extract features from an image. To improve efficiency, MDFS extracts complimentary features from the LIDAR point cloud in order to obtain comparable detection accuracy. MFDS achieves 3.7% higher accuracy than the base CNN detection model and is able to operate at 10 Hz. Additionally, the memory requirement for MFDS is small enough to fit on the Nvidia Tx1 when deployed on an embedded device.

Keywords: Autonomous Vehicle Perception, Obstacle Detection, Embedded GPU Computing, Convolutional Neural Networks, Sensor Fusion

1. INTRODUCTION

Vehicles are an integral part of the world, interweaved in our everyday tasks with the primary objective to provide point-to-point transportation. They transport goods between factories, shipping ports, stores, and also are a primary mode of transportation for many populations of the world. Vehicles may be very beneficial; however, their usefulness does not come without a cost. Every year roughly 30,000 are killed and over 2 million are injured in the US in automobile accidents [1]. Therefore, safety is a large concern not only for government regulators but also for automotive manufacturers. It is desired to have vehicles that are capable of driving themselves, since they would never be distracted leading to a significant improvement in vehicle safety. In 2005, Stanley the robotic vehicle from Stanford University (Figure 1), under the guidance of Sebastian Thrun won the DARPA Grand Challenge [2]. This success sparked a surge in research and commercial work towards the development of autonomous vehicles. This

Figure 1. Stanely, winner of the DARPA Grand Challenge in 2005 [2]

rush for autonomy was increased when Boss, from Carnegie Mellon University won the DARPA Grand Urban Challenge in 2007 [3]. Most vehicles are now equipped with Advanced Driver Assistance Systems (ADAS), such as Adaptive Cruise Control, that are classified as National Highway Traffic Safety Administration (NHTSA) level 1 or 2 autonomy (partial autonomy). However, one of the challenges in designing level 4 or 5 autonomous vehicles (highly or completely autonomous) is accurate perception in all environments of the world around the vehicle [4].

At the heart of perception lies computer vision. The vehicle must take in raw data from sensors such as cameras, LiDAR, and RADAR and then process it to form a meaningful representation of the world around it including object classification, detection, and localization. These different modalities of data provide unique benefits, but also have shortcomings and failure points. These different streams of data must be analyzed in order to gather important information from the environment around the vehicle in order to be passed to later stages of the autonomy system such as path planning.

A modern approach for object detection and classification is to use a convolutional neural network (CNN). The CNN is passed an image and will output a set of both bounding boxes and object labels for each bounding box. The CNN learns which features to extract by optimizing its convolutional kernels in order to form rich representational power to detect and classify objects. This class of algorithms yields high accuracy and generalizable detections. However LiDAR data is able to directly reason with the 3D world instead of a 2D projection of it. Directly manipulating 3D data allows detections to be used for path planning in the next stage of an autonomous vehicle, which would not be possible with the 2D detections of an image algorithm. Although LiDAR analysis would be useful because of its 3D reasoning, detection accuracy is traditionally worse for LiDAR based algorithms than camera based ones, making LiDAR only algorithms unsuitable for autonomous vehicles [5].

In order to overcome the weaknesses of these independent modes of data, it is possible to fuse the data. The two defining types of data fusion between image and LiDAR data are decision and feature level fusion. Both of these fusion techniques have their own weaknesses, feature level fusion is difficult to find equivalent representations for each type of data and decision fusion can be less accurate as the system relies on both sets of features independently. The benefits though are increased detection quality over LiDAR only algorithms and 3D detections which can not be obtained by monocular camera algorithms, thus making sensor fusion algorithms the final output of an autonomous vehicle's perception system.

The purpose of this research was to develop MFDS to fuse camera and LiDAR data in order to provide accurate, fast object detection and classification. MFDS attempts to avoid large memory consumption while still being accurate, and operate at close to real time by using information from different types of sensors to efficiently augment one another, rather than greatly increase the computational cost of a single sensor for marginal benefit. The target deployment platform for MFDS is the Nvidia Tegra series, specifically the Tx1.

2. LITERATURE REVIEW

Convolutional Neural Networks (CNNs) have seen great success in image classification [6][7][8][9][10]. CNNs have also proven successful in a variety of other computer vision tasks including, but not limited to, detection and segmentation [11][12][13][14][15][16]. Although classification, detection, and segmentation tasks are all different, neural networks are called universal approximators and are able to learn how to perform each of them with high levels of accuracy [17]. Each task requires specific network architectures for optimality, but due to their generality, many improvements to CNNs in one task also prove to be improvements in the others.

By 2017, image classification had reached the necessary accuracy levels for real world performance; however, to be useful for most applications, models would need to take up less memory and perform inference faster. The first major paper to address this problem of model deployment was MobileNets, which developed an accurate, small, fast classification network and was shown to perform well at transfer learning to more complicated tasks [18]. MobileNets was based upon the idea of the depthwise seperable convolution, which was shown to be nearly equivalent to normal convolution. Depthwise seperable convolution is faster than standard convolution because they are optimized for the General Matrix to Matrix (GEMM) function call from within the cuBLAS library of CUDA, which is utilized by the cuDNN library [19]. cuDNN is a library specifically dedicated to GPU implementations of CNNs due to their high computational complexity, making them intractable on standard CPUs.

Once classification networks had reached a high enough level of accuracy, CNNs were applied to more complicated tasks such as detection tasks. One such successful detection network is Single Shot MultiBox Detector (SSD) [20]. Instead of only outputting a class label, SSD outputs a list of detections where each detection consists of a class label, a confidence, and the four coordinates of a bounding box. SSD is a unified detection network meaning that it performs a single pass through a network without using a Region Proposal Network popularized by Fast and Faster RCNN [12][13]. Detections are outputted at different stages of the network because the feature map size decreases as the layer depth increases, meaning that larger objects will be outputted earlier on in the network since their feature maps are larger.

In addition to camera, or image, based methods are LiDAR, or point cloud, based methods for object detection. As range sensors such as LiDAR have become more affordable and more readily available, point cloud detection methods have increased in popularity but are still not as accurate as image based methods [21][22][23][24][25][26]. Unlike image methods which are almost entirely handled by deep learning methods, LiDAR methods utilize both modern deep learning techniques and traditional hand crafted feature extraction methods. This is largely due to the computational complexity of 3D convolution, making it difficult to form CNNs for a LiDAR point cloud. Therefore CNN algorithms that operate on LiDAR data typically perform some sort of projection or form a 2D representation of the point cloud which 2D convolution can be applied to.

There are also fusion based methods which take multiple modes of data as input and output detections similar to the camera and LiDAR methods previously mentioned. These fusion methods can generally be grouped into two classes, feature fusion and decision fusion. Feature fusion methods generally involve projecting the LiDAR point cloud into a 2D space and then processing both the image and the projection with some sort of CNN in order for the learned features that are extracted from both mediums to compliment one another [27][28][29]. The other group of fusion algorithms are decision level fusions, which perform independent detections in both mediums and then combine both sets of detections together in order to output a single set of superior detections [30]. The idea behind fusion methods is to utilize the advantages of each type of data to augment one another in order to provide superior detection quality over each independent medium alone.

3. METHODOLOGY

The proposed Multimodel Fusion Detection System (MFDS) is a decision-feature fusion, seen in Figure 2. At the start of the algorithm, an image based CNN performs detection to output a set of possible objects represented as bounding boxes with classes and probabilities. After which, the synchronized point cloud is masking to the same viewing angle's as the camera, the ground plane is removed, the cloud is transformed into the camera coordinates, and the remaining points are separated into clusters based upon Euclidean distance. Then, image detections and clusters are associated and paired together. Next a vector of hand selected features is extracted from the cluster of each pair, if there are any, and run through a Multi-layer Perceptron (MLP) to regress class probability, object length, distance to the object's center, and orientation of the object. The pairs confidence's are adjusted, or could potentially be removed, based upon the output of the MLP. The output of the fusion algorithm are confident 3D localized detections.

3.1 Creation of Fusion Models

The first step was acquiring a pretrained image detection CNN model, specifically SSD, and then performing transfer learning on the SSD model to be fine tuned on the KITTI dataset [5]. The KITTI dataset was split to create training and validation sets, out of the 7481 images and point clouds, 5611 were used for training and 1870 were used for validation. Next, the LiDAR processing pipeline was built up to the feature extraction of identified clusters

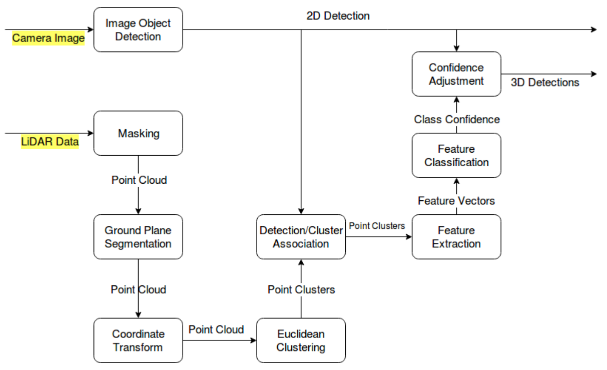

Figure 2. System Diagram for the operation of MFDS

stage. A converted LiDAR dataset was then created by running every LiDAR point cloud in the KITTI dataset through the processing pipeline to extract features of every cluster, which were then compared to KITTI's labels and saved to disk if they matched. After the converted dataset was formed, the classifying MLP was built and trained.

3.2 Feature Extraction

After point cloud clusters had been formed, the last step before the MLP was to perform feature extraction. The features that were decided upon was the sample mean and standard deviation of the clusters x, y, and z coordinates, ranges for all 3 dimensions, as well as ratios of x to y, x to z, y to x, y to z, z to x, and z to y. Although these ratios were redundant and included inverses, they proved to be beneficial as the MLP did not learn as well if some were removed. It is believed that this is because this redundancy slightly reduced the complexity of the nonlinear decision boundary that the network needed to learn.

These features were chosen because they represent important geometric information about the point cloud cluster while being computationally cheap to compute. The geometric data of the point cloud is used to augment the color intensity data stored in the camera image.

3.3 Dataset Creation

After the LiDAR point cloud pipeline had been created, the dataset needed to be created. To form the dataset, each point cloud in the separated training portion of KITTI was processed with the same pipeline that was used during inference, which included masking, ground plane extraction, transformation, and clustering. Each cluster was checked against each labeled cuboid to determine if that cluster represented one of the labeled objects. Since the labels were not perfect, a cluster was considered a labeled object if no more than 5% of the points existed outside the label cuboid. In addition to the KITTI labels of Vehicle, Pedestrian, or Cyclist, clusters with no class label were included in the final dataset in order for the network to learn to be able to reject objects that were not relevant, which meant that the MLP would output a score for four classes instead of three. Therefore, the final classes used were Don't Care, Vehicles, Pedestrians, and Cyclists. In total there were 17,382 point clouds, 17% was other, 69% was vehicles, 10% was pedestrians, and 3% was cyclists, seen in Figure 3. Therefore, cyclists were the most difficult class to classify since they had the least number of training examples. After the datasets clusters were identified, each cluster had their features extracted and their class, distance to

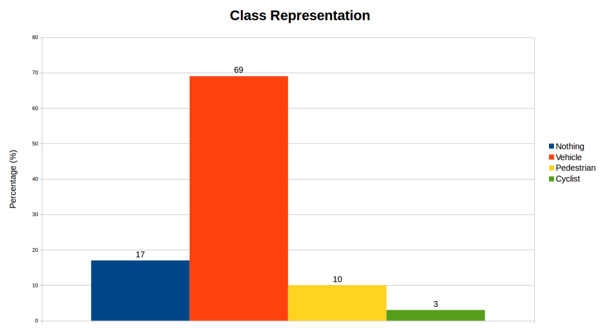

Figure 3. Class Representation in the created MLP Cluster Dataset

center, length, orientation, and features saved to disk. 75% of the point clouds were used for training and the remaining 25% were used for validation.

3.4 Mutli Layer Perceptron Architecture

The MLP to classify point cloud clusters was built in Python with the Tensorflow library. The MLP's jobs were to predict the class of the object's cluster, the distance to the center of the object from the LiDAR sensor's origin, the size of the object along the z axis, and the rotation of the object given the cluster's features and the trained model. The class probability distribution is an unknown nonparametric distribution. The MLP is used to form an estimate of the posterior distribution given the trained model and input feature vector. The class probability is mathematical stated in Equation 1.

$$\hat{P}(C_i|\mathbf{f}_i,\Theta) \tag{1}$$

C_i represents the class probability for the ith feature vector and Θ represents the trained MLP model parameters. The distance to the center of the object, z_i, the size of the object, l_i, and the rotation of the object, α_i, are all estimated values.

The MLP needed one output layer for every value that needed to be regressed, therefore the MLP had four different output layers. The MLP architecture is seen in Figure 4 but should be noted that the nodes are not draw to scale according to their size. The input layer for the feature vector had a size of 15 since that was the length of the feature vector, the class output layer had a size of 4 to be able to regress each class probability, and distance, size, and rotation layers each had a size of 1.

The number and shape of the MLP's hidden layers needed to be determined. The number of layers tested varied from 1-7 and the number of neurons in each layer was varied between 10-200. The hidden layer configuration that converged to the best accuracy had a single hidden layer with 150 neurons.

A truncated normal distribution with a standard deviation of .01 was sampled from to initialize all parameters. As is standard for detection networks, the distance, length, and rotation was not predicted outright, an encoding was used instead. The entire dataset was analyzed and the maximum distance was 50m as was the maximum length. The maximum length was this large because of random clusters that were included in the dataset. Therefore, the distance and length did not need to be predicted but instead an encoding was predicted, which was easier for the network to learn. In addition, the rotation value can only assume values between $-\pi$ and π so the natural scaling factor used was π. The encodings are seen below:

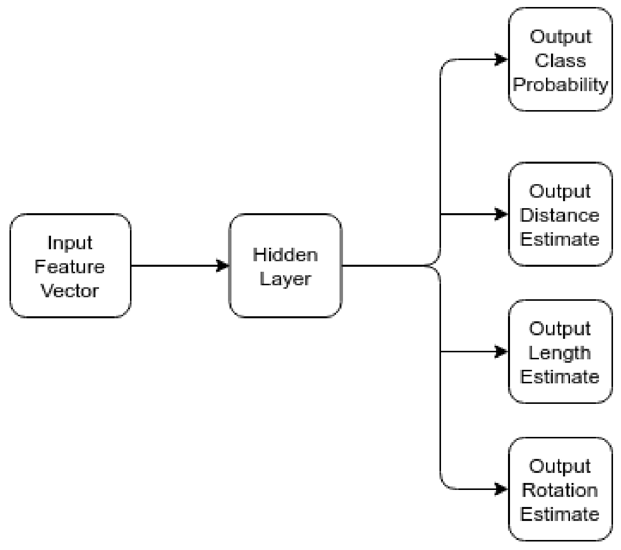

Figure 4. MLP Network Architecture

$$\phi(z_{pred}, l_{pred}, \alpha_{pred}) = \left[\frac{z_{pred}}{50}, \frac{l_{pred}}{50}, \frac{\alpha_{pred}}{\pi}\right] \tag{2}$$

The activation function that was used for the hidden layer was a rectified linear unit (RELU), the class output layer was a softmax, the distance and length output layers were both sigmoids, and the rotation layer was a hyperbolic tangent [7]. The class layer used the softmax to squash all class probabilities between 0 and 1 and to make the sum of the probabilities equal 1. The distance and length layers used the sigmoid because the predictions fell between 0 and 1 since neither of the values could be negative and also the scaled value could never be over 1. The rotation layer used a hyperbolic tangent since the values fell between -1 and 1.

3.5 Training

In order to train the MLP to regress all four values simultaneously, a multi-part loss function was created. The total loss would be the summation of a weight regularization penalty term, L_{reg}, the weighted class error, L_{class}, and the weighted distance, length, and rotation errors, L_{dist}, L_{size}, L_{rot}, respectively. The final multi-part loss equation can be seen below:

$$L = L_{reg} + \frac{1}{N_{total}} \sum_{i=0}^{N_{total}} (\lambda_{class} L_{class}) + \tag{3}$$

$$\frac{1}{N_{care}} \sum_{j=0}^{N_{care}} ((\lambda_{dist} L_{dist}) + (\lambda_{size} L_{size}) + (\lambda_{rot} L_{rot})) \tag{4}$$

L_{class} was Cross Entropy Loss and L_{dist}, L_{size}, and L_{rot} were Smooth L1 Loss or Huber Loss [**?**]. L_{reg} was the summation of the l2 norm of all trainable parameters with decay value of 0.001. Finally, $\lambda_{class} = .8$, and $\lambda_{dist} = \lambda_{size} = \lambda_{rot} = \frac{1-.8}{3}$. The class loss weight, λ_{class}, was set higher than the other three weight terms as that output was the most important and needed to be emphasized to learn to the highest degree of accuracy possible. N_{total} is the total number of training examples per batch and N_{care} is the number of training examples that do not have a Don't Care class label. The reason why the summations were over N_{total} and N_{care} and not the same is that there were no labels for length, size, or rotation of the clusters that were labeled Don't Care. Therefore, the Don't Care terms in the summation needed to be avoided when computing loss, which would impact the gradients and degrade the performance of the MLP.

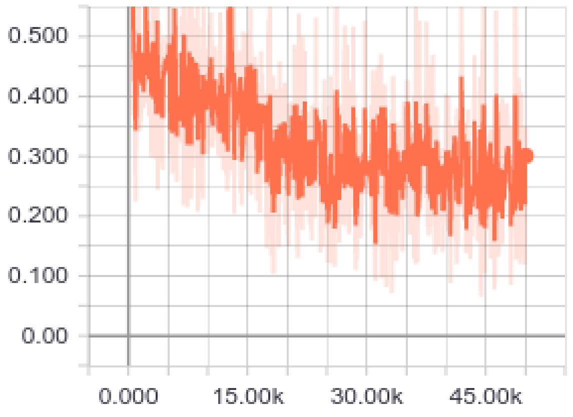

Figure 5. MLP Total Loss

The batch size was set to 12 on each GPU so there was an effective batch size of 36, since three GPUs were used. The network was trained for 50,000 iterations. Both Nesterov Momentum, with learning rate of 0.1 and momentum of 0.9, and Adam were tested; however, Adam converged to a higher class accuracy and was therefore used. The Total Loss during training can be seen in Figure 5.

Due to the unknown shape of the error surface, the MLP accuracy was extremely sensitive to initial conditions used for the learnable parameters. Since the MLP was so sensitive, the network was trained 75 times and the network that scored the highest classification accuracy on the validation set was selected to be the final network.

3.6 Association Problem

In order to perform the fusion algorithm after trained models were created for the image detection CNN and LiDAR classifying MLP, image detections and clusters needed to be associated together due to potentially noisy sensor readings detecting the same object. The projection matrix from 3D LiDAR space to 2D camera image space is provided for every timestep in the KITTI dataset. The projection matrix allows to compute the pixel location for each 3D point within the cloud, seen in Figure 6. The projection can be computed in the following way:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5}$$

$$\mathbf{P} \in \mathbb{R}^{3,4} \tag{6}$$

After the matrix multiplication had been computed, the output vector needed to be normalized by the last element and the projection from 3D LiDAR to 2D camera image space. After the projection had been developed, clusters mean x, y, and z coordinates were projected into image space, along with forming the centroids of each of the CNN detection's bounding box. With the two sets of projected cluster centroids and bounding box

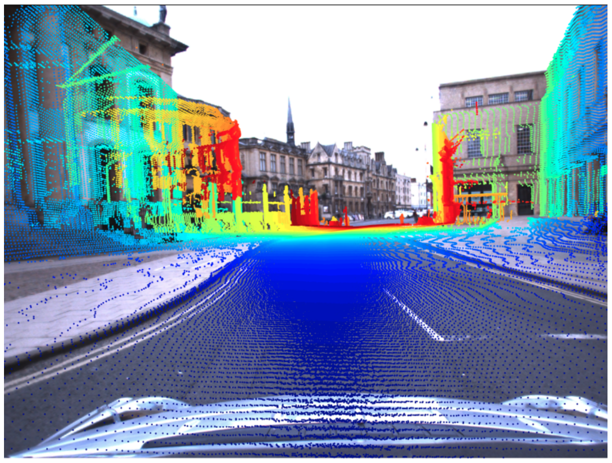

Figure 6. Example LiDAR Projection to Image Pixel Space, [31]

centroids, a simple Euclidean distance comparison in pixel space was performed. Any pair of centroids that were under a certain threshold were considered to to be a pair of the same object. The threshold that was used was 75 pixels due to the large variance of the cluster centroid's means.

3.7 Confidence Adjustment

After each pair's cluster had been classified, a check was performed between the pair's image detection class and cluster classification class. If the classes were not the same then the detection was eliminated, however if the classes were the same then the detections confidence was increased by 50%. The class probability was normalized afterwards. This confidence adjustment allowed for uncertain image detections, which would be eliminated, to gain the required confidence to be considered true detections.

3.8 MFDS Deployment Details

The fusion algorithm was jointly implemented in Python and C++ using ROS as the communication and build platform. ROS serves multiple purposes in this code; first to allow for messages to be passed with between the two languages, second for the use of powerful debugging tools such as topic monitoring and it's visualization package RVIZ, and third because MFDS needs to be easily deployable on robotic systems [32].

MFDS requires two ROS nodes to operate concurrently, seen in Figure 7; the first of which is a Python node to perform image detection and the second is a C++ node to perform point cloud manipulation, classification, and fusion. The Python node is subscribed to a Compound Data Message (CDM) containing both an image and a synchronized point cloud. Once the CNN finishes computing detections, another compound message is formed containing the detections and the CDM, which is published to the C++ node. The C++ node performs all point cloud preprocessing, cluster formation, cluster detection pairing, feature extraction, feature classification, and fusion. The output of the C++ node is the final set of fused 3D localized detections. Python was chosen for the image detection CNN because of Python's easy access to the Tensorflow library, eliminating the need to write the network inference code. C++ was chosen for the fusion node because PCL only operates in C based languages.

Since the LiDAR classifing MLP was trained with Tensorflow in Python, but inference was performed in C++, the MLP needed

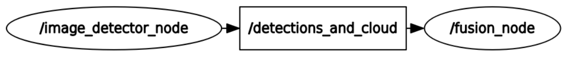

Figure 7. Fusion Algorithm's Compute Graph

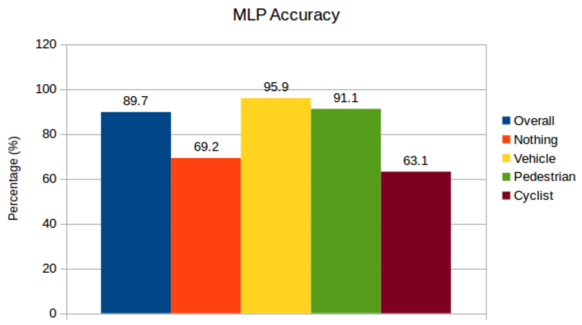

Figure 8. MLP Accuracy

to be ported over to C++ [33]. Since the network involved no convolutions and only a simple MLP, the cuDNN library did not need to be called. A MLP is a series of matrix multiplications followed by nonlinear activation functions so naturally an efficient, large scale matrix multiplication library was required. cuBLAS was decided upon over Eigen due to the size of the matrices that needed to be multiplied. cuBLAS is a library of CUDA that operates similarly to the Basic Linear Algebra Subprograms (BLAS) library in C++; however, cuBLAS performs the same operations on a GPU instead of a CPU. The cuBLAS function call Sgemm that is used to implement the MLP is the same function call that MobileNets optimized their depthwise seperable convolutions around [18].

4. RESULTS

4.1 LiDAR Cluster MLP

The MLP was able to learn to classify each cluster with 90% accuracy, on average, without the need for a complicated feature extraction method. The MLP struggled with classifing the nothing, or Don't Care class, as well as cyclists. It is believed that classifying the nothing class was difficult due to the large variance in shapes and sizes, making it difficult for the network to form a relationship between the high variance features and the class label. Cyclists were difficult to classify due to the relatively low number of training examples to learn from.

All distance, length, and rotation errors presented are after rescaling the MLP's outputs and labels back to dimensional values from their nondimensional outputs. The MLP output of predicted distance to the object had a Mean Square Error (MSE) of 1-2 square meters, which is a much better predictor than the centroid of the cluster (MSE of 39.52 square meters), but not an optimal value. The MLP distance prediction is an order of magnitude more accurate than the naive analysis of the point cloud. The length of the object's prediction had an MSE of approximately a quarter of a meter squared when the average length of vehicles, pedestrians, and cyclists were 18.3 meters. Unlike distance, there is no easy way to predict object length or rotation without making strong assumptions about each class's shape. Therefore, the MLP provides access to reasonable predictions for these values very quickly at the expense of some accuracy.

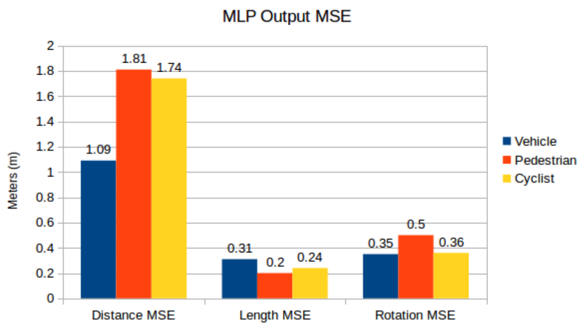

Figure 9. MLP Output MSEs

As the MLP was progressing through its 75 separate training initializations, the network appeared to converge to only three different levels of accuracy. It became apparent that these three levels of accuracy were proportional to each classes representation in the dataset. The network would converge to roughly 79%, 87%, or 90%. The networks that converged to 79% had not learned how to predict pedestrians or cyclists and instead only predicted class labels of vehicles and nothing since they made up the majority of the training examples. 87% convergence represented not learning cyclists and 90% represented learning how to predict all classes. This theory was tested by tallying each of the outputs of the test set and it was confirmed that the network never outputted the corresponding class labels.

4.2 MFDS and Image Detection CNN

Testing was performed on the remaining 1870 images and point clouds that were set aside in the test set. The image only detection method was analyzed in order to see how well the primary SSD model, as well as the RFCN and FRCNN reference models, performed on the KITTI dataset, as well as to act as a benchmark to see how much improvement MFDS provided. The KITTI dataset uses the mean Average Precision (mAP) metric for reporting results and is computed by finding the area under the precision recall curve. KITTI defines a true positive as a detection that scores over 70% IOU for cars and 50% for pedestrians and cyclists. However, mAP is a poor indicator of MFDS's detection quality since it is dependent on how many detections an algorithm can output.

Recall is a measure of how completely the detector's output detection set covers all labeled objects, while precision is a measure of how few incorrect detections are in the output set. Recall is largely dependent on how many detections a detector can output, generally in the range of 10 to 300 [34]. This variable number of detections, at varying confidence levels, allows for increased recall. As more detections are outputted, the likelihood of covering all labeled objects increases. MFDS operates in direct opposition to the idea of a variable number of detections with different confidence levels and works to only output as many detections as necessary, each with high confidence. This difference in output ideology means that recall is not a good evaluation metric for MFDS and as a result, neither is mAP. As the few high confidence outputs of MFDS are used to compute the systems recall, the precision recall curve drops to zero when there are no more available detections to cover the labels. Therefore a more applicable metric to evaluate the improvement of MFDS over the base SSD image detection CNN was used.

In order to determine the viability for deployment of the image detection CNN model, three main metrics were considered; the

	RFCN	FRCNN	SSD	MFDS
CPU Memory (GB)	2.592	2.816	1.824	2.176
GPU Memory (GB)	1.962	7.876	0.554	0.703
Inference Time (s)	0.073	0.54	0.0167	0.1083
Adjusted Accuracy (%)	58.11	57.84	37.18	40.89

Table 1. SSD Results in MFDS

memory footprint on the CPU and GPU, the inference time, and our own accuracy metric called adjusted accuracy. We define Adjusted accuracy, Equation 7, as the percentage of accurate detections with a penalty for incorrect detections, divided by the total numbed of labeled objects. In addition to these three metrics, it was beneficial to view the distribution of different types of detections in order to see the room for improvement that MFDS could add. The different types of detections were based upon varying combinations of confidence and correctness. Correctness was defined as the detection predicting the correct class with appropriate IOU. Confidence was defined as the confidence MFDS had in the detection's class, and a miss was defined as the detection CNN placing a detection around no labeled object. MFDS's main objective was to reduce the number of unconfident but correct detections and also reduce the number of confident misses. Therefore, MFDS is best suited to be used in conjunction with a CNN detection model that has a high number of unconfidently correct detections, a high number of confident misses, and performs inference at a high rate of speed.

$$\text{Adjusted Accuracy} = \frac{\text{True Positives} - \text{False Positives}}{\text{True Positives} + \text{False Negatives}} \tag{7}$$

The command line tool nvidia-smi was used to find each model's GPU memory consumption and the command line top was used to find each model's CPU memory consumption, which included all memory required by the process. The memory consumption is much larger than the size of the trained models because the memory consumption includes the amount of memory for storing temporary values and all additional required libraries for performing inference.

The SSD model was chosen for its small memory footprint, fast inference time, and ideal detection type distribution. Roughly 11% of SSD detection's fell within the categories of unconfidently correct and confidently incorrect, seen in Figure 13, as the targeted types of detections for MFDS to eliminate. MFDS takes the small, lightweight SSD model and increases the confidence of it's detections in order to output high confidence, correct detections with a minimal increase in memory demands, while still being able to operate at 10 Hz, seen in Table 1. Figures 10, 11, and 12 show that MFDS increases the confidence of detections that are able to become confident enough to become final detections. MFDS' output is 96% high confidence, correct detections, which is roughly a 50% improvement over the base SSD model, seen in Figure 13. MFDS suffers from a slight, roughly 0.8%, increase in confidently incorrect detections and misses. With the increase in the number of confident detections, MFDS was able to increase the adjusted accuracy by 3.7%. A comparrison to the popular RFCN and Faster RCNN models are supplied in Table 1.

In order to evaluate the inference speed of MFDS, each function in MFDS was timed, shown in Table 2. Cluster formation took up roughly half of the entire MFDS inference time with the other half evenly spread out amongst the other functions. Although the MFDS processing is roughly as fast as the RFCN and faster than the FRCNN inference time, they are not directly comparable because

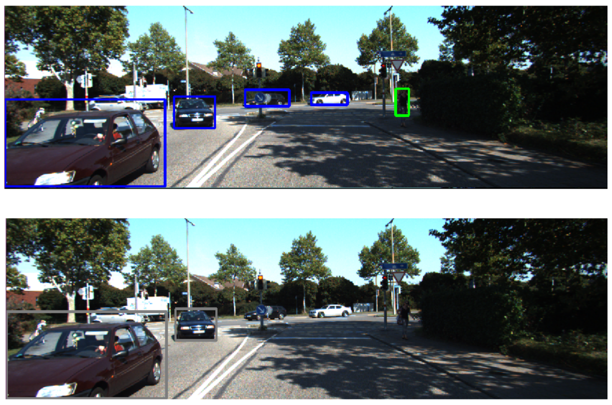

Figure 10. SSD to MFDS Comparison; MFDS above, SSD below

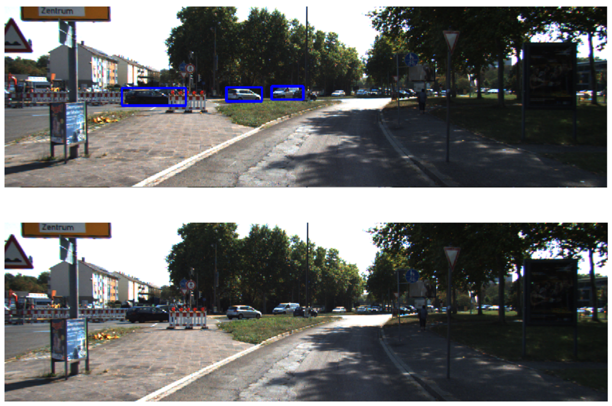

Figure 11. SSD to MFDS Comparison; MFDS above, SSD below

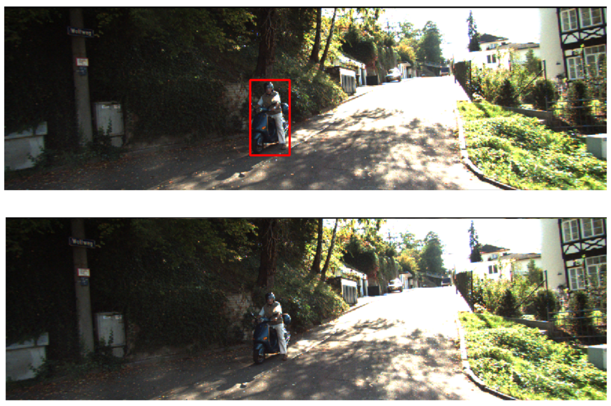

Figure 12. SSD to MFDS Comparison; MFDS above, SSD below

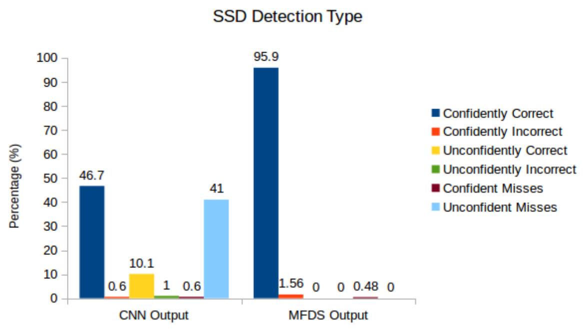

Figure 13. SSD Model Detection Type

Task	Time (s)
CNN	.0167
Masking	0.014
Segmenting Ground Plane	0.009
Coordinate Transform	0.015
Cluster Formation	0.039
Detection/Cluster Association	0.001
Feature Extraction	0.0004
Classification	0.008
Confidence Adjustment	0.0001
Total	0.1083

Table 2. Time Analysis of MFDS Inference

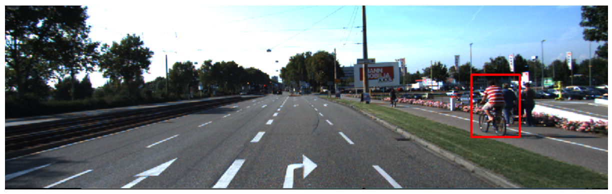

Figure 14. Occluded Objects from the Image Viewpoint

the CNN detection models operate on the GPU and MFDS operates on the CPU and GPU with non optimized functions.

One of the main sources of error in MFDS proved to be the detection-cluster association. A known problem is when objects appear very close to one another, their bounding boxes will be nearly on top of one another and will be erased by NMS [35]. MFDS is very susceptible to this problem due to the way it associates detections and clusters together. An example of this can be seen in Figures 14 and 15 where a cyclist partially occludes a pair of pedestrians. There is only a single bounding box after NMS, due to their high IOU, however, there are two different clusters that are paired with it, visualized as the pink and blue points in Figure 15. There will be two final detections with the same bounding box, but with different 3D localized values which, by KITTIs definition, is one true positive and one false positive since labels can only have one detection after which all detections are considered errors.

Another source of error for MFDS was the false positive detection rate due to confident misses. Since many clusters are generated from a point cloud and are then paired with image detections with MFDS's association method, non object clusters make it to the cluster classification stage. The classifying MLP has a classification accuracy of 90%, which means that for every

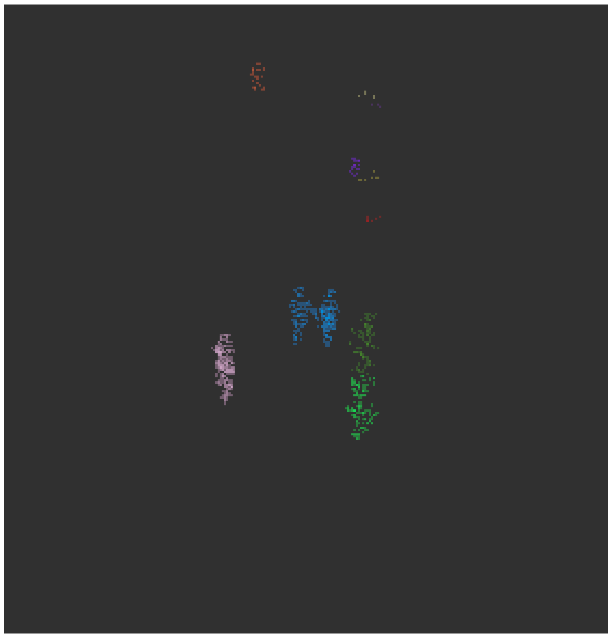

Figure 15. Occluded Objects from the Point Cloud Cluster Viewpoint

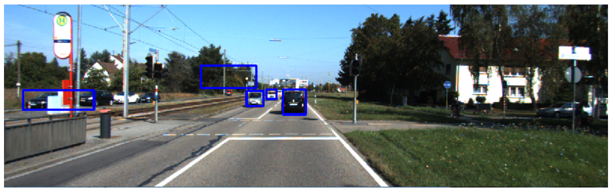

Figure 16. False Detection Much Higher than Ground Plane

non object to reach the MLP, 10% will be viewed as confident detections due to false cluster classification and a poor confidence image detection. An example of this can be seen in Figure 16 where the treetop canopy's cluster is falsely detected as a car.

5. CONCLUSSIONS

An object detection system for autonomous vehicles was discussed in this article. MFDS was able to increase the adjusted accuracy by 3.7% over SSD while providing detections at 10 Hz. This increase in adjusted accuracy was achieved by changing unconfident into confident detections by performing an analysis on the corresponding point cloud cluster. MFDS performed inference comparably or faster than the reference CNN image detectors, takes up significantly less memory, and provided 3D localized detections. MFDS was able to take unconfident detection proposals from the image CNN and use LiDAR data to add enough confidence for the detection proposals to be considered true detections. MFDS was a step towards a deployable object detection system for autonomous vehicles. It fused information from multiple sensors to produce outputs directly usable by the path planning module of an autonomous vehicle. Although there are limitations to MFDS; the benefits and information that MFDS produces outweigh the problems it faces.

6. ACKNOWLEDGMENTS

We are thankful for personal funding from the SMART Scholarship, which is funded by: USD/R&E, National Defense Education Program / BA-1, Basic Research. We would also like to thank both Magna and Polaris for funding Florida Tech's IGVC team.

7. REFERENCES

References

[1] Nation Highway Traffic Safety Administration. Traffic safety facts 2015, a compilation of motor vehicle crash data from the fatality analysis reporting system and the general estimates system. 2015.

[2] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, 23(9):661–692, September 2006.

[3] Christopher Urmson, Joshua Anhalt, Hong Bae, J. Andrew (Drew) Bagnell, Christopher R. Baker, Robert E. Bittner, Thomas Brown, M. N. Clark, Michael Darms, Daniel Demitrish, John M. Dolan, David Duggins, David Ferguson, Tugrul Galatali, Christopher M. Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Sascha Kolski, Maxim Likhachev, Bakhtiar Litkouhi, Alonzo Kelly, Matthew McNaughton, Nick Miller, Jim Nickolaou, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Varsha Sadekar, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod M. Snider, Joshua C. Struble, Anthony (Tony) Stentz, Michael Taylor, William (Red) L. Whittaker, Ziv Wolkowicki, Wende Zhang, and Jason Ziglar. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(8):425–466, June 2008.

[4] C. Berger and B. Rumpe. Autonomous Driving - 5 Years after the Urban Challenge: The Anticipatory Vehicle as a Cyber-Physical System. *ArXiv e-prints*, September 2014.

[5] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *Int. J. Rob. Res.*, 32(11):1231–1237, September 2013.

[6] Yann LeCun, LÃl'on Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. *ArXiv e-prints*, September 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, December 2015.

[11] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[12] R. Girshick. Fast R-CNN. *ArXiv e-prints*, April 2015.

[13] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *ArXiv e-prints*, June 2015.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *ArXiv e-prints*, June 2015.

[15] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger. *ArXiv e-prints*, December 2016.

[16] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016.

[17] Gerald H. L. Cheang. Approximation with neural networks activated by ramp sigmoids. *J. Approx. Theory*, 162(8):1450–1465, August 2010.

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[19] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *ArXiv e-prints*, October 2014.

[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[21] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[22] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017.

[23] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[24] M. Engelcke, D. Rao, D. Zeng Wang, C. Hay Tong, and I. Posner. Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks. *ArXiv e-prints*, September 2016.

[25] Shuran Song and Jianxiong Xiao. *Sliding Shapes for 3D Object Detection in Depth Images*, pages 634–651. Springer International Publishing, Cham, 2014.

[26] Shuran Song and Jianxiong Xiao. Deep Sliding Shapes for amodal 3D object detection in RGB-D images. 2016.

[27] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-View 3D Object Detection Network for Autonomous Driving. *ArXiv e-prints*, November 2016.

[28] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. *CoRR*, abs/1711.10871, 2017.

[29] V. Hegde and R. Zadeh. FusionNet: 3D Object Classification Using Multiple Data Representations. *ArXiv e-prints*, July 2016.

[30] Sang-Il Oh and Hang-Bong Kang. Object detection and classification by decision-level fusion for intelligent vehicle systems. *Sensors*, 17(1), 2017.

[31] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017.

[32] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[33] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[34] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *ArXiv e-prints*, November 2016.

[35] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Improving object detection with one line of code. *CoRR*, abs/1704.04503, 2017.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.