Moving Target Classification Through RADAR

Douglas G. Smith, Scott Liu, Xiangyu Gao
Department of Electrical and Computer Engineering
University of Washington
Seattle, WA
smithd57@uw.edu, liuy78@uw.edu, xygao@uw.edu

Abstract

In this work, the authors present classification results for three different classes of moving targets (car, pedestrian and bicycle) using radar data and various neural networks models. Frequency Modulated Continuous Wave (FMCW) RADAR is used to collect the raw data that is then pre-processed by FFT, STFT, and CFAR to generate the MD heatmap. In our experiments, six types of neural networks have been implemented: VGG-like (a simple CNN), AlexNet, VGG16, ResNet50, RNN and RCNN (the combination of an RNN and CNN model). High accuracy is demonstrated in most of the networks, and VGG16 achieves the highest testing accuracy: more than 99%.

1 Introduction

An important topic in the field of machine learning is being able to perform real-time classification. Specifically in regards to the domain of autonomous self driving cars, the system needs to be able to classify objects such as cars, bicycles, and pedestrians in the environment around it at very high accuracy. This is needed to ensure the cars are able to properly navigate the environment with a very low risk of failure, as a failure in this kind of system could have disastrous consequences, including death. To this end a large amount of effort has gone into developing systems that can perform high accuracy classification, with the majority of the effort being focused on using cameras or Light Detection and Ranging (LIDAR) based systems. These have their advantages but also have disadvantages. Classification through cameras is the most common route and provides a lot of information to the system but can be heavily influenced and reduced in effectiveness by environmental factors such rain or glare. Another method is using LIDAR which creates a mapping of the environment using lasers, but these systems can be very expensive, both monetarily and computationally, to implement and use.

Another route for performing classification that has had generally less focus is using RADAR. RADAR is a detection system that uses radio waves to determine the range, angle, and velocity of objects [8]. A radar system consists of transmitters producing electromagnetic waves in the radio or microwaves domain, as well as receivers and processors to determine the listed properties of the objects. Radio waves from the transmitter reflect off the object and return to the receiver, giving information about the object's location and speed.

Recently, FMCW radars are becoming more and more popular in automotive driving since they allow for accurate measurements of distances and relative velocities of obstacles and other vehicles [9]. Therefore, FMCW radars are useful for autonomous vehicular applications (such as parking assistance and lane change assistance) and car safety applications (such as autonomous breaking and collision avoidance), which are shown in Fig 1. An important advantage of radars over camera and LIDAR based systems is that radars are relatively immune to environmental conditions (such as the effects of rain, dust, and smoke). Because FMCW radars transmit a specific signal (called a chirp) and process the reflections, they can work in either complete darkness or bright daylight.

Preprint. Under review.

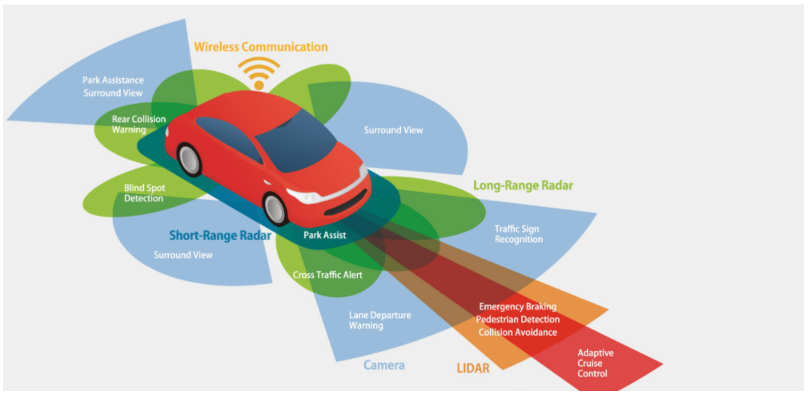

Figure 1: Automotive radar application.

When compared with ultrasound, radars typically have a much longer range and much faster time of transit for their signals

To this end, this project performed testing on a variety of neural networks to determine what accuracies could achieve using RADAR data. The classification challenge was to be able to correctly classify the incoming data as either being a car, a bicycle, or a pedestrian. In the end, the best network was able to achieve over 99% testing accuracy.

2 Methods

2.1 Dataset

The pre-processed input dataset is composed of 5000 labeled images of 256 (height) by 256 (width) by 4 (channels). The width represents the time component of the object and the height represents the velocity of the object (one pixel is 0.781 ms), and the 4 channels represent the data collected from 4 antennas. The output will be compared with the pre-determined labels of the input images: car, pedestrian or bicycle. Three pre-processed MD heatmap images are presented in Fig 2.

2.2 Pre-processing

The micro-Doppler (MD) signature needs to be extracted from the raw radar data, and this procedure has been conducted by a lot of researchers. However, the MD signature can be spread in multiple range bins due to the size and number of objects. Hence it is necessary to detect and track the object prior to performing the Short Time Fourier Transform (STFT), especially in the case of multiple objects crossing their trajectories. To address this problem, the pre-processing procedure shown in Fig3 is implemented on raw radar data to generate the MD heatmap. The processing steps have been summarized as following.

1. Perform Fast Fourier Transform (FFT) on the raw digitized radar data to convert it into the range-time domain and apply an Infinite Impulse Response (IIR) filter to filter out the stationary components.

2. Apply the Constant False Alarm Algorithm (CFAR) on the range domain to perform object detection and store the position of objects for consecutive frames.

3. Use the Kalman Filtering algorithm to estimate the object position given in the previous result and then assign identities to the object detection based on the estimates from Kalman filter. This can help us track the objects and deal with the scenario where new objects enter the radar field of view or some objects leave.

4. Concatenate the object tracking results of several frames and generate the MD heatmap using the STFT. The duration of the overall MD signature can be varied depending on the classification requirement just by concatenating more or less frames together.

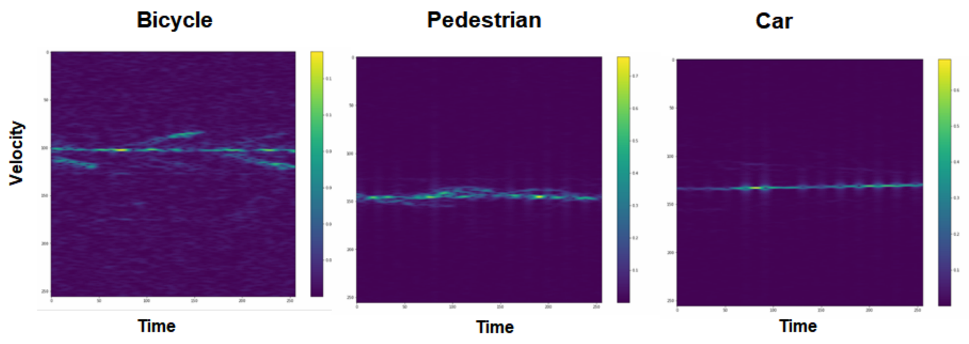

Figure 2: MD heatmap examples for three classes.

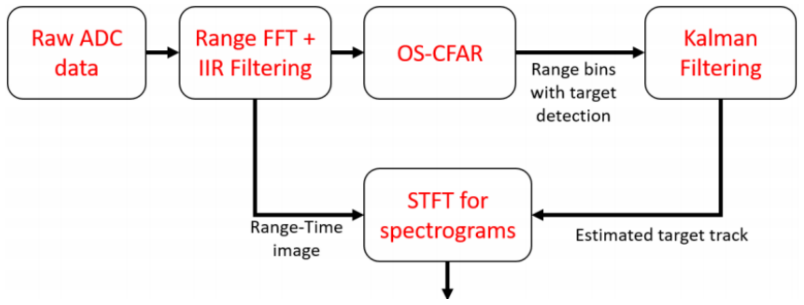

Figure 3: Block diagram of the pre-processing procedure [1].

2.3 Models

For this project we decided to implement two different types of neural network models. These were convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The reason for using CNNs is that these are the classic model to implement when working with image based data. While our raw data doesn't start as an image, the pre-processing performed on the data converts it into an image like structure, so it makes sense to use CNN models on the data. The reason for using RNNs is that the RADAR data has a time component. RNN models are useful for data that has a natural sequence. A common example is language processing as language datasets, such as sentences, consist of sequences of words and learning that sequential nature can be useful or even the goal. For the RADAR data because it has a time component the images can be split into sequentially sets and our theory was that the sequential element of the data could be useful to the learning process and provide valuable data for classification.

2.3.1 CNN Models

With the idea to implement some RNNs and CNNs in place, several models for each type of network were implemented and tested on. For the CNNs a total of four architectures were tested, which are shown in Fig 4. The first labeled VGG-like (though it is closer to LeNet5) was a very simple network consisting of two convolutional/max pooling layers, a dropout layer (which is useful for networks being trained on small datasets), and two full connected layers. This network served as a starting point to see how good of an accuracy could be achieved with a basic CNN model.

From here three more standard models were implemented and tested. These included AlexNet, VGG16, and ResNet50. AlexNet and VGG16 are simply deeper CNNs with AlexNet having 7 layers and VGG16 having 16, but ResNet50 goes up to 50 layers and adds an additional mechanic to avoid vanishing gradients [5]. A common problem with deep neural networks is the effect of vanishing gradients on back propagation which causes the gradient to go towards zero the more layers the gradient goes back, so after a certain point adding more layers to a CNN model provides little benefit. The ResNet50 structure helps avoided this issue by adding shortcuts in the network. What this means

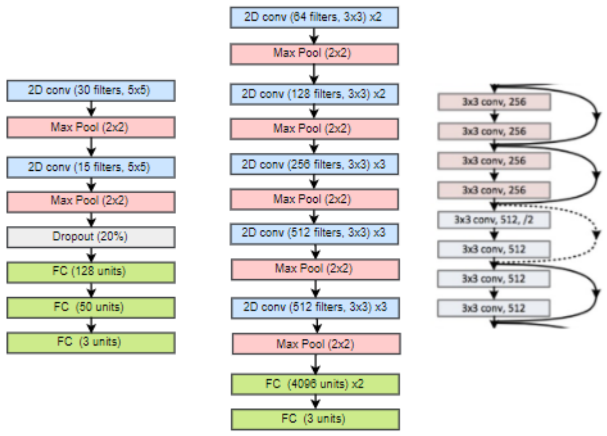

Figure 4: VGG-like network on the left, VGG16 in the center, and a portion of a ResNet model on the right.

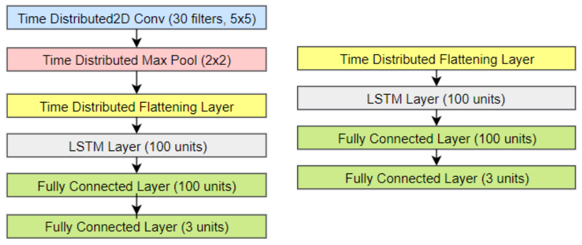
Time Distributed2D Conv (30 filters, 5x5)
Time Distributed Max Pool (2x2)
Time Distributed Flattening Layer
LSTM Layer (100 units)
Fully Connected Layer (100 units)
Fully Connected Layer (3 units)
Time Distributed Flattening Layer
LSTM Layer (100 units)
Fully Connected Layer (100 units)
Fully Connected Layer (3 units)

Figure 5: RNN/CNN combination on the left and the RNN on the right.

is that the output of one layer is added to another layer further down in the network. This provides a path for gradients to more quickly pass through the network, helping diminish the impact of vanishing gradients. All of the layers implement the ReLU activation function with the exception of the output which implements the softmax function. Something worth noting is that for training we used our own implementation of VGG16 and ResNet50 and did not use pre-trained models. This was done because pre-trained models expect input images to have a standard format, i.e. the number of channels per pixel is either 3 or 1. Because our data had 4 channels it was incompatible with a pre-trained model.

2.3.2 RNN Models

For the RNNs two models were implemented. The first is a simple LSTM RNN that flattens the image inputs before feeding them into the LSTM layer. This layer is then connected to some fully connected layers that generate the final classification. The second RNN model tested added a time distributed convolutional layer and max pooling layer before the RNN. This was done to see if combining the effects of both CNNs and RNNs together could produce a better result than either independently. What the time distributed convolutional layer does is allows the network to apply a convolutional layer to each of the images in the input sequence [2]. The final versions of the networks are shown in Fig 5.

2.4 Implementation Details

2.4.1 Splitting Dataset

To test the models, the data was split into $80\%/20\%$ training/testing data. Care was taken to ensure that the testing data was truly independent of the training data since as part of the pre-processing a sliding window is applied to the data, creating a new sample every 50 pixels. This process generates more data to train on, but also creates an inherent similarity between data elements. So if one were to take the entire dataset, randomly shuffle it, and then split the data, a lot of the testing data would look very close to the training data, so care was taken to avoid biasing the testing data in this manner.

2.4.2 Hyper Parameter Tuning

From here we did hyper parameter tuning to achieve the best testing accuracy. We found that training the system with data batches of 5 or 10 (any larger caused memory issues) at a learning rate of 1e-5 generally gave the best performance. For the RNN models there was an additional hyper parameter, the length of the sequence, though all our testing was done for a sequence length of 8. Also several different versions of the RNN models were tested, for example in one test an additional CNN layer was added, but these modifications did little to improve the accuracy. After training, the testing data was feed through the model and the total testing accuracy was calculated using a mean value calculation comparing the ground truth to the prediction produced by the network.

2.5 Challenges

Throughout the project we faced several challenges to our progress. One challenge was that the initial size of the RADAR data was too large. Originally the RADAR images had a size of 512x960x4. However, this caused our hardware to encounter memory issues due to both the size of the data and the resulting increase in the size of the models to process that data. So we had to strike a balance between downsizing the data while still retaining the critical information required to perform the classification. Another challenge we faced was that our dataset was fairly small. We only had 5000 images to work with as this was all the RADAR data that had been collected and processed in time for use in our project. However we made the most of the data that we had.

3 Results and Analysis

3.1 Results

The training and testing accuracy results achieved with the different networks is outlined in table 1. Also included are graphs of the training accuracy, testing accuracy, and loss per epoch for the VGG16 network and the RNN network as these were a good representative of the behaviors we saw in training (see Fig 6). While all of the networks through their training were able to achieve near 100% training accuracy and near zero loss, in general the CNNs achieved higher overall testing accuracies then the RNNs with the CNNs getting up to around 99% accuracy while the RNNs only got to around 94% accuracy. Overall the highest testing accuracy was achieved with VGG16 at 99.1% testing accuracy. This is impressive as the current state of the art for this particular problem using RADAR classification is 93% accuracy. While from an accuracy standpoint VGG16 is the optimal network for this task, an important consideration is the complexity of the network. What is meant by this is that the VGG16/ResNet50 networks are very deep networks and as a result are more computational expensive to train and use than the smaller networks such as AlexNet or the RNN/CNN combination. As a result more hardware would be required to run and maintain the VGG16 network. So if hardware is a concern the second best choice would be AlexNet as it got an accuracy of 98.5% (only slighty worse then VGG16) and is much a much smaller network.

Table 1: Training and Testing Accuracies

Accuracy	VGG-like	AlexNet	VGG16	ResNet50	RNN	RNN/CNN
Training	97.3%	100%	100%	99.6%	100%	99.3%
Testing	87.7%	98.5%	99.1%	98.5%	94.6%	95%

While the RNNs did generally worse than the CNNs in regards to testing accuracy, further testing on these models may be worthwhile. Due to the success of the CNNs, while we were interested in how the RNNs behaved, the RNNs became more of a side focus as the project went on. It could be possible that through more experimentation with the RNN hyper-parameters, better results could be achieved. For example, simply training the network for longer may result in better accuracies.

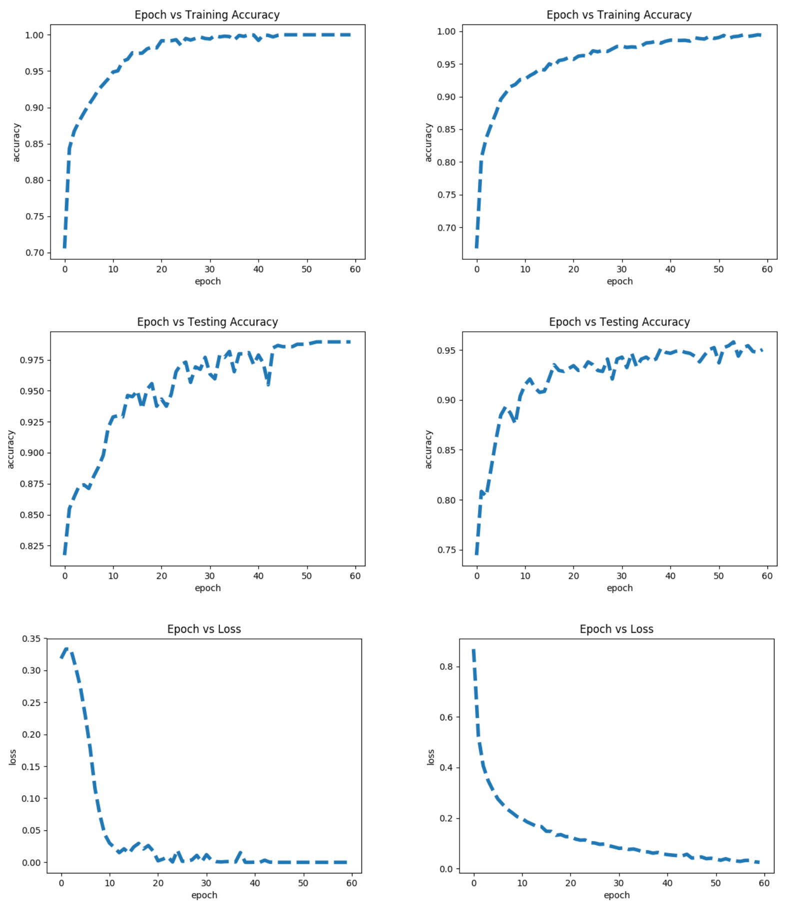

Figure 6: Training accuracy, testing accuracy, and loss of VGG16 (left column) and the RNN/CNN combination model (right column).

Another more interesting parameter to experiment with would be the length of the sequences feed into the model. All of the testing was done at a sequence length of 8, but it is possible that either smaller sequences such as 4 or longer sequences such as 16 may provide overall better results.

3.2 Class Visualization

Also as a side part of the project we used the methods in [6] [7] to make the class visualization. Class visualization is to synthesize an image to maximize the classification score of a particular class. This will provide some intuition on what the network is looking for when it classifies images of that class.

By starting with a random noise image and performing gradient ascent on a target class, it is feasible to generate an image that the network will recognize as the target class. This idea was first presented in [6]; [7] extended this idea by suggesting several regularization techniques that can improve the quality of the generated image.

Concretely, let I be an image and let y be a target class. Let $s_y(I)$ be the score that a convolutional network assigns to the image I for class y; note that these are raw unnormalized scores, not class probabilities. Our objective is to generate an image I^* that achieves a high score for the class y by solving the problem

$$I^* = \arg\max_I (s_y(I) - R(I))$$

where R is a (possibly implicit) regularizer (note the sign of $R(I)$ in the argmax: we want to minimize this regularization term). This optimization problem can be solved by using gradient ascent on the generated image. We will use (explicit) L2 regularization of the form

$$R(I) = \lambda \|I\|_2^2$$

and implicit reguarization as suggested by [7] by periodically burring the generated image.

An example of class visualization from our models are shown in Figure 7. This particular feature map comes from the pre-trained VGG16 model and we used 3000 iterations to generate the feature map.

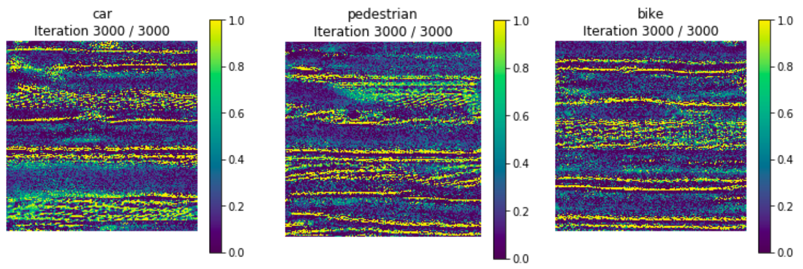

Figure 7: Class visualization for car, pedestrian and bicycle

4 Conclusion and Future Work

Overall the final results of this project were impressive as we were able to beat the state of the art for this particular problem using RADAR classification by several percent. From the testing, purely based on highest accuracy, the VGG16 network is the winners choice at 99.1 % accuracy. While that is a good results in terms of RADAR, an interesting comparison to make is how well the RADAR classification stacks up against camera based classifiers. In doing some research on the accuracies achieved by other groups working on vehicle detection based systems using camera based data, the accuracies they were achieving were in the upper nineties [3][4], similar to the better results we achieved. Also from class we learned about the work Tesla is doing in object classification and they were boosting near 100% accuracies for their camera based systems. So in comparing our RADAR based results to other camera based results, we are achieving similar test accuracies.

For the impact this would have on future work, especially in regards to automatic cars which require high accuracy classifiers, we do not think one system would replace the other. Rather, we would expect both camera and RADAR based will be implemented to work in tandem to provide systems with the most accurate data possible about the environment. In terms of future work for the project there are several next steps. The first would be to acquire more data. As was stated earlier in the project, we only had a limited amount of data to train the system. Future testing would require more data to help further improve the model. The second next step would be to try and expand the number of classes the model can classify. This would improve the versatility of the model and improve its overall usefulness.

References

Our codebase: https://github.com/Xiangyu-Gao/Practical-Introduction-NN-hw/tree/master/Project

[1] Aleksandar Angelov, Andrew Robertson, Roderick Murray-Smith, Francesco Fioranelli, Practical classification of different moving targets using automotive radar and deep neural networks, *University of Glasgow*, 2018

[2] Jason Brownlee, How to Use the TimeDistributed Layer for Long Short-Term Memory Networks in Python, *https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/*, 2017

[3] Hai Wang, Yijie Yu, Yingfeng Cai, Long Chen, Xiaobo Chen, A Vehicle Recognition Algorithm Based on Deep Transfer Learning with a Multiple Feature Subspace Distribution, *Jiangsu University*, 2018

[4] Yaw Okyere Adu-Gyamfi, Sampson Kwasi Asare, Anuj Sharma, Tienaah Titus, Automated Vehicle Recognition with Deep Convolutional Neural Networks, 2017

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, 2015

[6] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

[7] Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML 2015 Deep Learning Workshop

[8] Wikipedia, RADAR, *https://en.wikipedia.org/wiki/Radar*

[9] Automated Parking System Reference Design Using 77- GHz mmWave Sensor