Jason Ligon
Daniel Kim
CPSC 585

Object Detection with R-CNN Project Report

I. Introduction and Background

Object detection is one of the more classic problems of computer vision, but also very important. The goal of object detection is to find instances of real-world objects or people in images or videos (Amit & Felzenszwalb, 2014). It is used in surveillance systems, automated vehicle systems, image retrieval, and others. The process of detection involves the extraction of features and learning algorithms to determine the presence of the object (Mathworks, n.d.). There are several methods available for performing object detection, but neural networks present a more robust solution to this problem. Some of the challenges in detection are appearance changes, pose variations, occlusion, and background clutter (Wang et al., 2015). Though these challenges are trivial to humans there is active research in the field to answer them.

According to Haykin, a neural network is a machine that is designed to model how the brain works when it performs a particular task. It is also a massively parallel distributed processor made up of simple units called neurons. It is also able to learn and therefore generalize, meaning to say that it produces reasonable outputs for inputs not encountered during its training phase (2009). These properties makes it an ideal tool for solving the object detection problem because it can handle a wide variety of challenges and finds a solution to situations it did not encounter before. The project implemented a Regions with Convolutional Neural Network (R-CNN) to solve the object detection problem. R-CNN is an object detection framework that uses CNNs to classify image regions within the given image (Girshick, 2014). A traditional CNN classifies every region using a sliding window technique. A variant of this is the R-CNN, which only concerns itself with regions that are likely to contain the object to be detected. This makes it computationally easier by reducing the number of areas it has to compute.

Another concept used in this project is transfer learning. Transfer learning involves using a network trained on a large collection of images such as ImageNet or CIFAR datasets as a starting point to solve a new detection task (Mathworks, n.d.). The main advantage of using this method is that the pre-trained network already has learned from a rich set of features that has a wide variety of images. All that is needed is to fine-tune the network so that it can be customized based on the needs of the project. This can be done by making small adjustments to the weights such that feature representation learned from the original tasks are modified to perform the new task. The combination of R-CNN and and transfer learning reduces the number of images needed to train the network, as well as a reduction in training time.

The basis of the project was to perform a single object detection. The project was specifically interested in detecting green traffic lights. The inspiration came from the surging interest in self-driving cars. The program assumes that the cameras are interested in detecting

green traffic lights and the vehicle would move when it has been detected. Despite the simplicity of this goal, the real world environment presents a challenge because we cannot expect our object to be perfectly set up or clear. The traffic light is subject to the aforementioned challenges. One of the primary concern of traffic light detection was intraclass variation, where the traffic light is either red or yellow. The system's goal is to only detect the object if the traffic light is green.

II. Dataset Description

The dataset used for this project is called the CIFAR-10 dataset. It consists of 60,000 32x32 color images in 10 classes. Each class is composed of 6000 images, where 5,000 are training images and 1,000 are test images. The dataset divided into five training batches and one test batch, each batch contains 10,000 images. The test batch contains 1,000 randomly selected images from each class, while the rest are training images in random order (Krizhevsky, 2009). This was chosen for its simplicity and the desired results are achievable using this dataset. Some of the classes in the dataset includes airplane, automobile, bird, truck, and dog. The classes in the dataset are mutually exclusive, meaning to say that there is no overlap between objects, for example items found in automobiles are not found in trucks (Krizhevsky, 2009). This prevents from any confusing within the training data. Though it is not as extensive as VGG, it still has found uses in simple applications.

Another dataset that was used in this project is called Multiple Object Tracking (MOT) dataset. This dataset aims to aid multiple target tracking evaluation. It consists of multiple sets of of captured video in public space. The dataset provides a large collection of datasets, where some are already in use in research projects (Leal-Taixé, 2016). A subset of the MOT dataset, MOT16, was used in the project. The MOT16 provides 7 video sequences of both training and test data. This was filmed in an unconstrained outdoor location with both static and moving cameras. The length of the videos lasts from 30 seconds to 1 minute, therefore it ranges from 600 to 1050 still images. This was chosen because it provided us a good real-world data sample in detecting the traffic lights from the point of view of a moving vehicle. In terms of self-driving cars, this made it a perfect dataset to work with for the purposes of the project.

III. Description of Methods, Algorithms, and System Architecture

The projected was implemented using a MATLAB version R2016b. The frameworks used was the Computer Vision System Toolbox and Neural Network Toolbox. This was also developed in a Windows 10 Operating System. This was chosen due to the desire of implementing a system that is simple and provides promising results accomplished at a reasonable amount of time.

The neural network architecture has the beginning layer which is composed of the input layer. After the input layer we have a middle layer that consists of the convolutional layer, ReLU (Rectified Linear Units) layer, and max pooling layer. This middle layer is repeated three times before heading into the final layers. The final layers consists of two fully connected layer, ReLU

layer, softmax, and classification layer. The network parameters for the project consists of 40 filters with a 5x5 filter size. The learning rate is also set to 0.001.

The first layer, the input layer consists of 32x32x3 images with ‘zerocenter’ normalization. This will be passed onto the convolution layer which consists of 50 5x5x3 convolutions with the stride of [1 1] and padding [2 2]. After the convolution layer, it would pass through the ReLU layer. The max pooling layer, which consists of 3x3 max pooling with stride [2 2] and [0 0] padding. On the third convolution layer we had 100 5x5 convolutions with a [1 1] stride and padding of [2 2]. The final layer consists of 64 fully connected layer before passing onto another ReLU layer. After this, it is then passed to 10 fully connected layer. It would pass into a softmax layer and finally the classification output.

Our method starts with downloading the CIFAR-10 Image data into MATLAB. We then developed a Convolutional Neural Network, where MATLAB’s Neural Network Toolbox provided an easy to design CNN layer-by-layer. The layers are defined at this stage. The input layer is set to the CIFAR-10 data. Next, the middle layers are defined. As aforementioned, the middle layers consists of repeated blocks of convolutional layer, ReLU, and pooling layers. These are the core building blocks of CNN. The convolution layers define a set of filter weights which are updated during the training phase. The ReLU layers adds non-linearity to the network. This allows the network to approximate non-linear functions that map pixels to the semantic content of the image. The ReLU, essentially simplifies the mathematics by removing all the negative numbers found in images. The pooling layers downsample data as it passes through the neural network. This layer was used sparingly to avoid downsampling of data too early in the network, leading to a loss of important features.

A deeper network was created by repeating these three basic layers. The pooling layers, was however reduced to avoid excessive downsampling too early. The input is then passed onto the final layer, which typically consists fully connected layers and softmax loss layer. After the layers are defined, we combined them to one variable.

After defining the network architecture, the network is trained using the CIFAR-10 training data. This was done by using the trainingOptions function provided by MATLAB. The algorithm used to train is Stochastic Gradient Descent with Momentum (SGDM) and the initial learning rate is set to 0.001. As it goes through the training, the learning rate reduces every 8 epochs, where 1 epoch is one pass through the training data set. The training algorithm was set to up 40 epochs.

The trained network must now be validated to ensure that the training was successful. This can be done by performing a quick visualization of the first convolutional layer’s weights to identify any immediate issues with training. A good indicator that the training was a success is that the first final layer weights should appear to have a well defined structure. If the weights visually seems stochastic in nature, then more training is required. The first layer filters should have learned edge-like features from the training dataset.

We continued to validate the training results by using the test data set to measure classification accuracy of the network. If the accuracy score is low, then additional training is required. A 100% accuracy is not necessary, rather, just sufficiently training the network is acceptable for the purposes of object detection. Though further training may improve the overall accuracy, it is not necessary for our purposes.

After a sufficient accuracy score is achieved, we then performed the transfer learning and fine-tuned the network to detect green traffic lights from the MOT training dataset. The training data is created which is a table that contains the image filename and ROI labels for the traffic lights. The ROI label is a bounding box tightly fit around the object of interest within an image. We had 400 training images of green traffic light.

We then trained the R-CNN with the green traffic light using the trainRCNNNObjectDetector function. The input to this function is the ground truth table containing the labeled training data, the pre-trained CIFAR-10 network, and the training options. This function automatically modifies the original CIFAR-10 network into a network that can classify images into green traffic lights and background class. The input network weights are fine-tuned using image patches from the ground truth data. The variables 'PositiveOverlapRange' and 'NegativeOverlapRange' control which image patches are used for training. In order to reduce training time, we used a parallel pool is created automatically by the R-CNN training function automatically creates this for us.

Finally, R-CNN object detector can now be used. For testing, we used the test set from the MOT dataset to determine if the network can detect green traffic lights. The R-CNN object detector returns the test image with the bounding box around the object detected, confidence score, and a class label. The confidence score, which ranges from 0 to 1, indicates how sure the network is that the object it detected is the one we like.

IV. Experimental Results

The first stage of our experimentation involved varying and finding the ideal hyperparameters and variables for our convolutional neural network. One of the first parameters we chose to vary was the filter size. Initially, our experiments began with filter sizes of 3x3 which achieved a mediocre accuracy score of 64.35%. The next filter size that was experimented with was a filter size of 5x5 which was able to achieve 74.25%. Intuitively, the next filter size that was used was 6x6 which gave an accuracy score of 74.06%. The accuracy scores were computed using the CIFAR-10 training data and the attributed test set. With the pre-trained network being the foundation of our R-CNN, we believed it to be most appropriate to test the performance of the hyperparameters on the CIFAR-10 trained network. The other hyperparameters that were varied in our experiments were made constant with the following values: 32 filters, 0.001 learning rate, and 3 convolutional layers. It became clear to see that the ideal filter size for the use of our network was 5x5 and that became the filter size that we used for the rest of our experiments.

The next variable that was varied was the number of filters. Initially, we had started out with 32 when deciding the ideal filter size for our network. After settling on the ideal filter size of 5x5, that was the filter size that was used for our experiments with the number of filters. Starting with the default number we used, 32, we were able to achieve an accuracy of 74.74%. For thoroughness, we then tested the performance of the network using a lower number of filters, 25. Using 25 filters was able to produce an accuracy of 73.76%. The next values that were tried were 40 and 50, which produced 76.24% and 76.54% respectively. One thing to note from the performance of the last two number of filters is that while 50 filters produced the highest

accuracy, the training time increased greatly, roughly twice the amount of time it took on average to perform training with 40 filters. Due to this observation along with the marginal boost in performance, we ended up choosing 40 as the number of filters to use for our network.

The architecture of our network was also varied in order to determine the ideal number of convolutional layers. The starting number of convolutional layers was 3 due to the fact that many people on MATLAB forums and other resource sites such as Quora suggested using 3 as the starting number of layers. Adding one additional convolutional layer was attempted, increasing the total to 4 convolutional layers, and produced very low performance with average accuracy scores hitting around 16%, even after increasing the number of epochs more than triple the normal amount used. Not wanting to reduce the number of convolutional layers below 3 due to its already good performance and possibly losing important features, we decided to stick with 3 convolutional layers. Additionally, the number of epochs chosen was 40 for the CIFAR-10 testing due to its optimal average performance.

Here is a summary of the various parameters and features that were experimentally varied:

Filter Size	Accuracy
3x3	64.35%
5x5	74.25%
6x6	74.06%

Number of Filters	Accuracy
25	73.76%
32	74.74%
40	76.24%
50	76.54%

Convolutional Layers	Accuracy
3	76.24%
4	16.12%

Epochs	Accuracy
30	75.91%
40	76.24%
50	75.61%

Using all of our final hyperparameters and network architecture values, the testing of our traffic light data was able to yield a performance of 89.33% after training the R-CNN with 400 images and a test set of 600 images. During the training process with the traffic light training images, the number of epochs was expanded to 100 in order for deeper learning to occur for traffic light detection and to build invariance.

For a more thorough look and representation of our traffic light test data performance, here is our confusion matrix:

N = 600	**Positive Outcome**	**Negative Outcome**

Real Positive	526 (87.67%) (true positive)	64 (10.67%) (false negative)
Real Negative	0 (0%) (false positive)	10 (1.67%) (true negative)

V. Conclusion

We were able to obtain reasonable results in detecting the traffic light. The test data included several challenges faced in object detection such as scale variation, illumination variation, and intraclass variation. Though the results were able to successfully detect the traffic light, we were unable to attain a good result when occlusion occurs. Occlusion is one of the major challenges faced that must be answered. Overall, the object detection network performed satisfactorily, but improvements are definitely needed.

One of the desired future work for is to be able to obtain object detection under various challenging situations. This is desirable because perfectly clear objects does not often occur in a real world setting. For the purposes of self-driving car, if a truck is block part of the traffic light, we would still want to be able to detect it and determine if it is green or not. This could perhaps be remedied by having a more extensive training data for the traffic light under various conditions.

Another extension of this is to perform multiple object detection. Often times streets may have more than one traffic light at an intersection, detecting all of them might be useful if the system fails to detect a particular traffic light. This is particularly handy if we want to detect people instead. This requires detection of multiple people present within the vicinity.

References

Amit, Y., & Felzenszwalb, P. (2014). Object Detection. *Computer Vision: A Reference Guide*, 537-542.

Haykin, S. S., Haykin, S. S., Haykin, S. S., & Haykin, S. S. (2009). *Neural networks and learning machines* (Vol. 3). Upper Saddle River, NJ, USA:: Pearson.

Krizhevsky, A. (2009). The CIFAR-10 dataset. Retrieved May 18, 2017, from https://www.cs.toronto.edu/~kriz/cifar.html

Leal-Taixé, L., Milan, A., Reid, I., Roth, S., & Schindler, K. (2016, March). MOT16: A Benchmark for Multi-Object Tracking. Retrieved May 18, 2017, from https://motchallenge.net/

Mathworks. (n.d.). Object Detectiong Using Deep Learning. Retrieved May 18, 2017, from https://www.mathworks.com/help/vision/examples/object-detection-using-deep-learning.html

Mathworks. (n.d.). Object Detection. Retrieved May 18, 2017, from https://www.mathworks.com/discovery/object-detection.html

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

Wang, L., Ouyang, W., Wang, X., & Lu, H. (2015). Visual tracking with fully convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 3119-3127).