Image Recognition: Using Deep Neural Networks for Traffic Light Management

SYSC5108 – Deep Learning

By:
Jon Menard

Systems and Computer Engineering
Carleton University
1125 Colonel By Drive, Ottawa, ON, CANADA

ABSTRACT

In this paper, we propose using Machine Learning (ML) for computer vision to detect oncoming traffic for traffic light management. We implement three deep neural networks (NN) including a custom deep convolution neural network (CNN) for feature extraction in images. Our CNN is trained to successful identify automobiles present in an image. Our results show that CNN is successful in image recognition and provides better performance than tradition flat NNs. The trained CNN can be deployed worldwide in traffic light camera, to assist in traffic flow management.

TABLE OF ACRONYMS

Binary Cross Entropy	BCE
Convolution Neural Network	CNN
Machine Learning	ML
Neural Network	NN

1 INTRODUCTION

Over the past decade, North American cities have continued expanding while the overall population has become more condensed. Additionally, around five million new cars are added to the road infrastructure each year. Without proper planning, cities quickly become congested, and transportation times significantly increase. New York City has some of the worst traffic seen in the U.S, with 45-minute drives taking over 3 hours to complete [1]. Countless proposals have been made to combat traffic in large cities, such as decreasing the number of cars on the road, substituting traffic signals with roundabouts, adding toll roads, or rerouting and optimizing existing routes. While each proposal offers its own solution, they are not without drawbacks. New construction typically requires reduced road capacity, causing the problem to worsen before it can get better. Furthermore, improvements are not free and must be paid for, usually at the cost of tax-paying citizens.

One method to increase traffic flow is to decrease the number of cars travelling together. When cars travel in a group, the actions of one car affect the entire group. A perfect example of this is seen in Phantom Traffic Jams, in which one car slows down for any reason, causing the cars behind it to slow down as well. Each car behind continues to slow down, propagating the effect backwards to every other car [2]. From a driver's perspective, everyone slowed down and then sped back up for no perceived reason.

Traffic signals are used to control traffic flow when two roads intersect. One road is given a stop signal, while the other road has an advance signal. After a designated amount of time, the traffic signal flips, allowing the other road to advance. As the stopped road waits for the signal to change, commuters begin to cluster together. A magnetic sensor can be used to detect stopped traffic and trigger a cycle period for the traffic signal to change. However, it is not immediate and the average automobile usually waiting at least a quarter of the cycle period (90-120 seconds) [3]. However, sometimes the road receiving the advance signal has no traffic, while the other road is stuck with one or more cars waiting. During this time, cars are unnecessarily grouping together, increasing congestion.

As such, in this paper, we propose using Machine Learning (ML) for image detection to be used in traffic light management. The goal will be to train a deep neural network that can accurately identify vehicles.

Such a policy will be used to signal when a traffic light can be safely changed, reducing unnecessary stopping of the flow of traffic. We will use state-of-the-art techniques to improve training and accuracy, including the use of convolutional neural networks (CNN), pooling, training normalization and randomization, and dropout.

The rest of this paper is organized as follows: Section 2 reviews the background of this work in related literature. Section 3 discusses the system description, including the model architecture, selected dataset, training, and testing process. Section 4 presents our results, and Section 5 states the conclusion.

2 BACKGROUND

2.1 Neural Networks

Neural networks (NN) encapsulate a network of neurons connected to compute data. The neurons are arranged in layers, with each neuron in one layer connected to each neuron in another layer. The connections between neurons are weighted, governing how the state of one neuron affects the state of another. The first layer, called the input layer, is given values by the environment, which propagated forward through each additional layer, hidden layers [4]. The last layer, called the output layer, releases its activation back to the environment representing the computed values of the input. In addition to the weighted connections, each layer or individual neuron may have a bias directing its value. Furthermore, each layer is also assigned an activation function which determines whether a neuron should be active or not [5]. Each weight and bias in the NN can be manipulated to learn the relation between the inputs and desired outputs. The relation does not have to be linear or simple. NNs have shown great success solving a variety of tasks, including speech recognition, computer vision, artificial intelligence, and predictive analysis [6]

Training a NN can be done in a variety of ways. One popular method is supervised learning. Supervised learning is a method to obtain an input-output relationship [7]. The mapping is guided by using a pre-existing labelled dataset where the correct outputs are mapped to the input. The inputs are fed into the NN, and it outputs its predictions. The NN's predictions are compared to the labelled outputs, and the error can be calculated. The goal is to train the NN on the dataset to successfully predict the correct outputs. In doing such, the NN should also develop mapping for inputs not contained in the dataset, allowing unencountered inputs to be predicted. The training dataset can be a subset of the possible data and does not need to be complete. This is particularly useful when the complete dataset is infinitely large, as seen in image recognition.

Tuning the weights and biases of the NN is completed by using the error between the predicted and actual outputs. Calculating the error between the outputs can be completed in various ways. The Binary Cross Entropy (BCE) is one of the most common loss functions used to calculate error with binary classification. To calculate the BCE, equation (1), the difference between the log likelihood of the model's outputs and the actual outputs is averaged with all other errors.

$$BCE = \frac{1}{N}\sum_{i=1}^{N} y_i * \log\big(p(\hat{y}_i)\big) + (1 - y_i) * (\log\,(1 - \mathrm{p}(\hat{y}_i) \tag{1}$$

The NN's goal is to minimize the error, signifying the predicted outputs are the same as the actual outputs. A popular method of error minimization is through Stochastic gradient descent. Gradient

descent is a repetitive process where from a random point of a function, the slope is descended until it the minimum value is found [8]. The slope or gradient of a function is found by taking the derivative. Therefore, by taking the derivative of the BCE, the slope towards the minimum value can be found. Additionally, to update the weights and bias, the gradient can be backpropagated with a learning rate through the NN with respect to each variable.

The complete training cycle for supervised learning, using BCE and Stochastic gradient descent, can be completed using these 3 steps. (1) Using a minibatch of size M from the dataset, predict the outputs for each input. (2) Calculate the BCE for each predicted output. (3) Take the derivative of the BCE multiplied by a learning rate and backpropagation it through the NN. Repeat the three steps until the BCE is at an appropriate level.

2.2 Convolution Neural Networks

Computer vision is an area of ML where a computer program is developed to infer information from pictures like humans. The programs should be able to recognize and identify information from images. Images often contain patterns and features which our brains can use to conclude the subject of the material. Computer vision requires the algorithm to also identify patterns in the image. Patters, however, are not location-dependent and may be located differently for an image of the same subject. Shown in Figure 1 is the pattern for a baby goat.

Figure 1: Two images of a baby goat with a different orientation for each

In Figure 1, humans can easily see this is an image of a baby goat. We identify the ears, nose, eyes, legs, and colour to all match that of a goat. However, we can also easily see the image is almost mirrored, with every feature that was on the left now on the right, and vice versa. While this may be simple for humans, a computer cannot identify this easily.

Traditional NNs inherently include the location of features through the activation of specific nodes. With an image, the location of the feature is not important, only its presence of it. As such, a different style of NN can be used to identify the presence of patterns and features in images. Convolution Neural Networks (CNN) can capture the Spatial and Temporal dependencies in an image by applying numerous filters [9]. Kernels apply a filter to neurons and their neighbors, identifying a pattern in the image. The kernel iterates through the entire image, identifying if the pattern is present at all. Shown in Figure 2 is a 3x3 kernel analyzing an image for the vertical pattern seen in the number 1

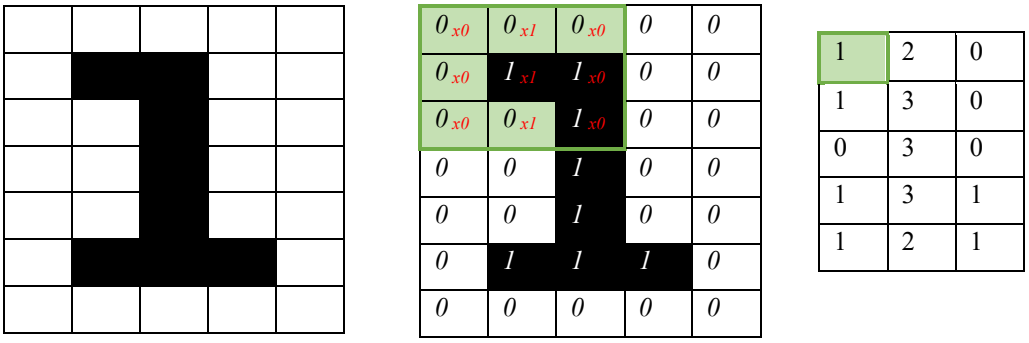

Figure 2: A 3x3 kernel being applied to identify vertical line shapes in images

In Figure 2, the kernel was applied to an image to identify vertical lines. Shown in the last table is a convoluted feature. In the center where the line was present, the table now shows a series of *3s*. No matter where the line would have been, the convoluted feature would still contain *3s* if a vertical line was present. Thus, the kernel successfully identifies the presence of a vertical line. Multiple kernels are applied to an image at once, allowing many patterns to be identified. Additionally, multiple layers of CNN may be combined to continue identifying patterns.

In high-resolution pictures, pixels will often share the same values of all its neighbors. High-resolution makes the image appear clear and crisp to our eye, but the extra information is not always needed. With each extra pixel, the NN must have weights and biases in each layer, which is computationally expensive. Pooling is a method used to condense neighboring features and eliminate excess information. A pooling layer is often placed after a convolution layer [10]. The pooling layer summaries the features identified by a kernel in the convolution layer. This can be done using different methods, such as average pooling or max pooling

2.3 Related Work

An increasing amount of research has concluded that CNN shows promising image detection results and can even outperform other types of NNs. In a paper by [11] 3 major CNNs were tested for their ability in image recognition. Each of the 3 CNN demonstrated significant success in image detection with high testing and validation accuracy. Additionally, in an analysis by [12], the author found CNN's showed better validation accuracy when directly compared to DNNs in image detection. Another paper by [13], used 6 different CNN for image classification, with the results showing all of them to be successful in image classification. With the increasing amount of research showing CNN excel in image detection, we propose creating a custom CNN to detect if an automobile is in the image.

3 SYSTEM DESCRIPTION

The dataset used to train this model was developed by Berkeley deep drive and is a collection of 100,000 images captured from self-driving cars worldwide [14]. The dataset contains annotation with all objects detected in each image. We further refined the dataset into two categories, images with automobiles (i.e. cars, busses, trucks, vans. Etc.) and images without automobiles (i.e. open road).

Using Python 3 and TensorFlow 2.8, the dataset was loaded using the ImageDataGenerator function. The images were augmented with random rotation, vertical and horizontal shifting, horizontal flipping,

and zooming. The images were also resized from 1280x720px, to 400x225px. Of the 100,000 images, 3,000 images were hand selected with 80% used in the testing pool, while the other 20% were added to a validation pool.

We analyzed 3 image detection models and evaluated their performance for image detection. Each of the models in this paper were implemented TensorFlow and Keras model. Each model has an input layer that receives a 400x720px image and a single output. The first model analyzed is the InceptionV3 CNN created and maintained by Google. IncepetionV3 is 48 layers deep with 26 million parameters. The model is included with Keres and initialized with pretrained weights. IncepetionV3 will be used as a target goal for our proposed CNN. The model architecture is included in the supplementary files. The second network analyzed is a traditional flat NN created for a baseline comparison. The model architecture is included in the supplementary files. The third model, and CNN proposed for this paper has 24 layers with 22 hidden layer. Within the hidden layers is four modules of a 2-dimensional convolution layer, connected to a batch normalization layer, connected to a Relu activation layer, and connected to a 2-dimensional pooling layer. Each convolution and Max pooling layer has a stride of (1,1) and (2,2), respectively. Each of the 4 modules feeds into one another, with the final module feeding into a dropout layer, then flatted and fed into 3 dense layers before the output layer. The model architecture is included in the supplementary files. Table 1 lists each model and compares the architecture.

Table 1: Model architecture for each NN

Model	Layers	Trainable Parameters	Activation
InceptionV3	48	24 million	Sigmoid
Flat NN	10	140 million	Sigmoid
Custom CNN	24	6.7 million	Sigmoid

Training the model is completed over 200 epochs, with a batch size of 32. After each epoch, the model is tested against the validation data, and the accuracy is stored. If the validation accuracy is greater than previous epochs, the model is saved before resuming training. An Adam optimizer is used for backpropagation, with a learning rate of 0.00005, and binary cross entropy for the loss metric.

4 RESULTS

After training each of the models for 100 epochs in approximately 15 minutes, the algorithms obtained the results shown in Table 2

Table 2: Model performance after training

Model	Training Accuracy	Validation Accuracy	Epochs
InceptionV3	100%	100%	6
Flat NN	98.75%	99.80%	200
Custom CNN	100%	100%	22

InceptionV3 reached 100% validation almost instantaneously requiring only 6 Epochs as shown in Figure 3.

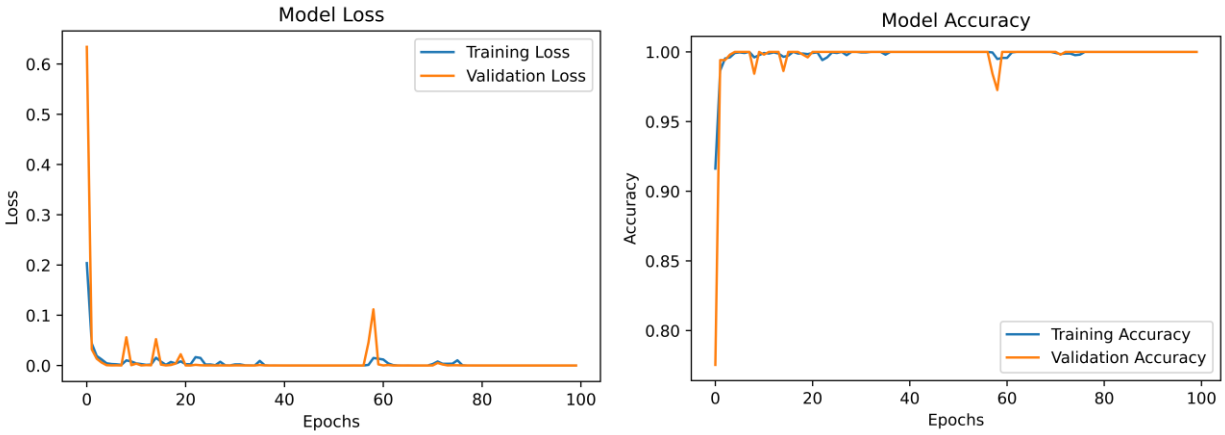

Figure 3: Training/Validation Accuracy and Loss for InceptionV3

However, InceptionV3 is a pre-training model and only needed minor tuning to reach the desired behavior.

The Flat NN slowly improved over the 100 epochs however only reach a final training accuracy of 98.75% and validation accuracy of 99.80% (Figure 4). The decrease in accuracy is likely caused by the location of features impacting the models outputs.

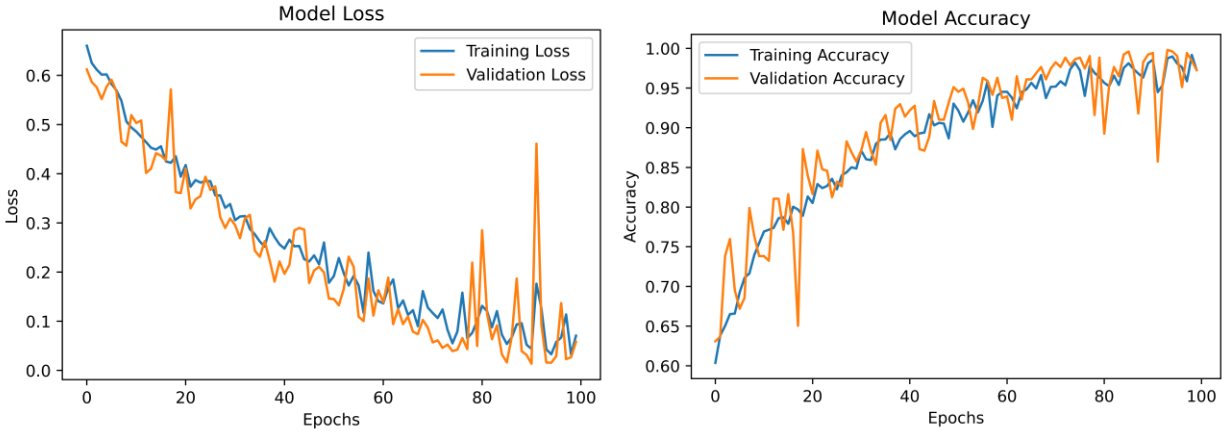

Figure 4: Training/Validation Accuracy and Loss for Flat NN

The proposed CNN was able to quickly learn the specific features within image that signify a car is within it. Shown in Figure 5 in an example features extractions for the red taillight plastic on a car identified by

the first convolution layer in the CNN.

Figure 6: An example feature extraction highlighting the red taillight signals on a car

On the right side of Figure 5, the detection of red turning taillight is highlighted white. After 22 Epochs the model reached 100% validation accuracy compared to the Flatt NN which took 200 epochs to reach 99.8% accuracy. Figure 6 illustrated the CNN model's accuracy during training for testing and validation.

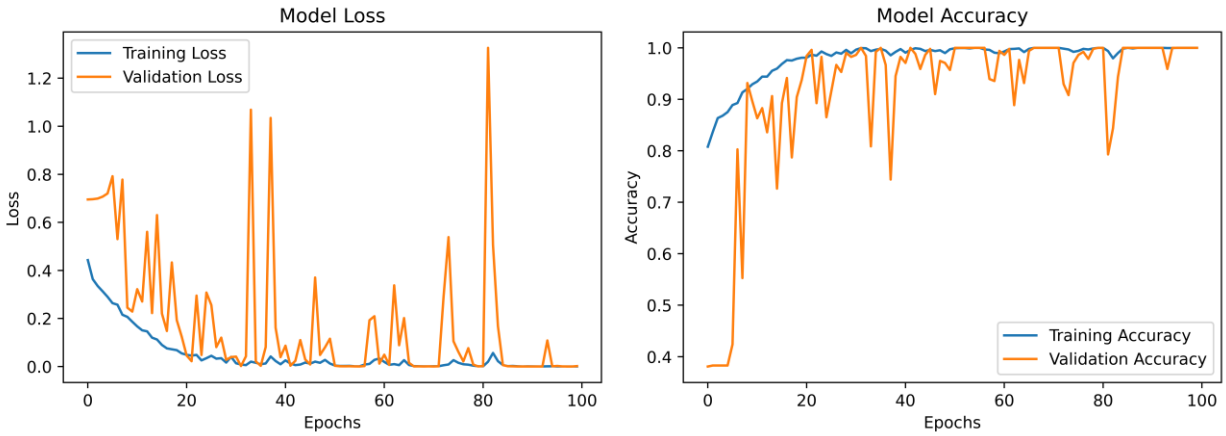

Figure 6: Training/Validation Accuracy and Loss for InceptionV3

Furthermore, after training, an additional evaluation test was done on all images in the dataset. The model was able to properly determine if a car was present in the image 100% of the time. Our results showed in less than 15 minutes a custom ML model can be trained with extremely high accuracy to predict the presence of cars in a picture.

5 CONCLUSION

In this paper we prosed using machine learning to train a CNN for image detection. Using Tensorflow and Keras we were able to implement 3 NNs. A variety of training configurations were attempted with the hyperparameters needing to be tuned multiple times before success was reached. After discovering the right training combination, we were able to produce successful results. Our results demonstrated a CNN could be training with high accuracy to detect cars within an image. The CNN trained can be used to detect the presence of cars, to help with traffic light scheduling.

REFERENCES

[1] NBC New York, "NYC Ranks as the City With Worst Traffic Congestion in the U.S., Study Finds," 12 December 2021. [Online]. Available: https://www.nbcnewyork.com/news/local/nyc-ranks-as-the-city-with-worst-traffic-congestion-in-the-u-s-study-finds/3438472/#:~:text=hours%20in%202019.-,The%20Big%20Apple%20tops%20the%20list%20of%20most%20congested%20cities,and%20Moscow%20(108%20hours)..

[2] L. Wang and X. Liang, "Phantom Traffic Jam Based on MATLAB," 2021.

[3] d. Levinson, "HOW MUCH TIME IS SPENT AT TRAFFIC SIGNALS," 2018.

[4] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," 2014.

[5] P. Baheti, "12 Types of Neural Network Activation Functions: How to Choose?," 2022.

[6] SAS, "Neural Networks: What they are & why they matter".

[7] Q. Liu and Y. Wu, "SUPERVISED LEARNING," 2012.

[8] A. V. Srinivasan, "Stochastic Gradient Descent — Clearly Explained !!," 2019.

[9] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," 2018.

[10] H. Gholamalinezhad and H. Khosravi, "Pooling Methods in Deep Neural Networks, a Review," 2020.

[11] S. Srivastava, A. V. Divekar, C. Anilkumar, Naik.Ishika, V. Kulkarni and V. Pattabiraman, "Comparative analysis of deep learning image detection algorithms," 2021.

[12] M. A. Pratama, "Comparing Image Classification with Dense Neural Network and Convolutional Neural Network," 2020.

[13] F. Sultana, A. Sufian and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network".

[14] Berkley Deep Drive, [Online]. Available: https://bdd-data.berkeley.edu.

[15] F. RINALDI, J. TORSNER, A. IERA and G. ARANITI, "Non-Terrestrial Networks in 5G & Beyond: A Survey," 2020.

[16] R. Dangi, P. Lalwani and G. Pau, "Study and Investigation on 5G Technology: A Systematic Review," 2021.