**Traffic Sign Detection and Recognition**

Saroj Dayal, S.D., and Dayal

School of Computer Science, University of Windsor, dayals@uwindsor.ca

1. **Abstract:**

Because of safety concerns, we have yet to have practical access to autonomous cars in the fast-paced era of automation. The detection and recognition of traffic signs is critical for such vehicles' safety. In this research, we offer a CNN-based technique for detecting and recognising traffic signs. The experimental findings revealed that the proposed system detects and recognises traffic signals with more precision, allowing it to be efficiently integrated into autonomous cars.

1. **Introduction:**

In current days, we highly rely on cars as our mode of commute, and statistics also state the same. The car is the dominant mode of transport in the EU, with slightly less than 2 persons on average per car [1]. Over 85 percent of Americans drive cars to work every day [2]. This includes lone drivers and carpools as well. In 2016, as per census reports, 12.6 million Canadians used cars as mode of their commute to work [3]. These numbers show us our dependency on cars and road transport in our daily life. There are 1.3 million deaths around the globe yearly, just because of road accidents. Human error, driving under the influence of alcohol or drugs, distracted driving, failure to follow traffic signs properly, and other factors all contribute to this [4].

Autonomous vehicles, on the other hand, are the future and are slowly gaining market share. The number of autonomous vehicles around the road is expected to rise to 54 million by 2024 [5] So naturally, self-driving cars also must adhere to road legislation and follow the traffic signs and therefore recognize and understand the traffic signs. Traffic signs detection and recognition plays an important role in advanced driver assistance systems (ADAS) to increase driver safety by warning or notifying the driver about the various traffic signs. This field of study has been extensively researched by researchers for a long time and has received considerable attention from the community.

To identify and classify traffic signs, we used a convolution neural network (CNN) in this article. We have used German Traffic Sign Recognition Dataset (GTSRB), which is an image classification dataset. Dataset consists of traffic signs of 43 different classes. Dataset contains more than 50000 images out of which 39209 are labelled images as training sets and 12630 are unlabelled images for testing.



Figure 1. Traffic signs categories contained in the dataset.

We use a three-layer CNN architecture to address the problem of traffic sign recognition and classification in this paper. The flow goes from the input from the car's desk camera through feature learning, which leads to the final classification.

The rest of the paper is organised as follows: in section 2, we follow the problem definition and motivation behind the research; in section 3, we review existing related publications in the field; in section 4, we describe our technique and provide a full description of flow and implementation. In part 5, we do an experimental evaluation, and in section 6, we discuss the result and future scopes.

1. **Problem Statement:**

**3.1. Problem Definition**

As we are highly dependent on road transport for our daily life activities, it is necessary to pay attention to safety while we commute. It is important to note that road safety signs play a vital role in this. Depending on the type of safety sign, it delivers various critical information to the passing motorists. The issue, however, is that the accidents are caused by the driver’s irresponsibility and unawareness.

**3.2. Motivation**

Designing an application that can aid drivers in identifying and recognising traffic signs, as well as advising the driver depending on the recognised sign is the need of an hour. In this paper, our proposed solution helps detecting and classifying the traffic signals available in RGB colour format.

**References:**

*[1]: Passenger mobility statistics*. (2021, February 11). Retrieved from eurostats: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Passenger\_mobility\_statistics

[2]: Adie Tomer. (2017, October 3). *America’s commuting choices*. Retrieved from brookings.edu: https://www.brookings.edu/blog/the-avenue/2017/10/03/americans-commuting-choices-5-major-takeaways-from-2016-census-data/

*[3]: Study: Long commutes to work by car*. (2019, February 25). Retrieved from Statistics Canada: https://www150.statcan.gc.ca/n1/daily-quotidien/190225/dq190225a-eng.htm

*[4]: Road traffic injuries*. (2021, June 21). Retrieved from World health Organization: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[5]: Statista Research Department. (2021, May 28). *Autonomous vehicles worldwide - statistics & facts*. Retrieved from Statista: https://www.statista.com/topics/3573/autonomous-vehicle-technology/

1. **Literature Review**

A variety of efforts have previously been made to detect and recognize traffic signs.

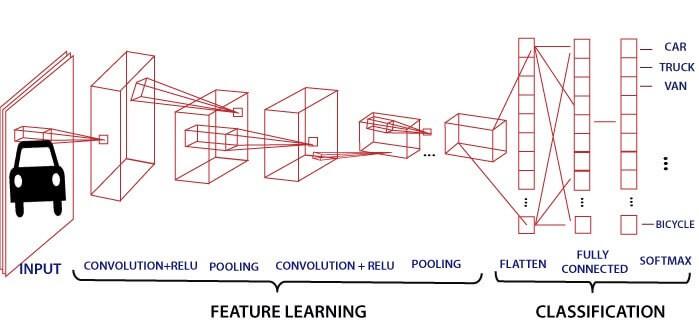
One scenario that was considered in previous works is for traffic sign recognition, a revolutionary energy-efficient Thin but Deep convolutional neural network design [1] suggested. Each convolutional layer in the proposed design has fewer than 50 features, allowing the convolutional neural network to be trained quickly even without the use of a graphics processing unit. First, the enormous German Traffic Sign Recognition Benchmark dataset used to train and test the suggested architecture.Then, to evaluate test performance, retrain the network models using transfer learning on the more difficult Belgian Traffic Sign Classification dataset. The suggested design is based on the concept that adopting thin yet deep networks, significant features per layer may be captured and learned with minimal resources. With minor modifications, this proposed architecture can readily be extended to other comparable applications.

In another work a method for recognising traffic signs uses small-scale deep convolutional neural networks (CNN) and can be used in a variety of applications [2]. For such real-time goals, sophisticated transformations or highly computational image processing algorithms are out of the question. The German Traffic Sign Recognition Benchmark (GTSRB) dataset is used to build the described approach. This dataset is dependable and vivid, and it has been used to train a variety of algorithms.

CNN is primarily used for deep learning in another work [3], and proposed to be used in traffic sign detection and identification from any source. The CNN-based classifier "WAF-LeNet" is proposed in this work. This could aid in the development of autonomous vehicle driving that is safer. With a significantly smaller and faster network than its competitors, the proposed technique was successful in accurately identifying 96.5 percent of the testing information set and 100 percent of the strength information set. WAF-LeNet has demonstrated a remarkable ability to recognise 43 different traffic sign classes.

1. **Architecture of The Models Implemented**

A CNN is a type of neural network that is quite similar to traditional neural networks. They're made up of neurons with learnable weights and biases. CNN differs from a standard neural network only in that it anticipates image input. This drastically minimises the amount of parameters that must be tuned in a network, and it also allows the model to be made more computationally efficient and faster by using this information.



The CNN architecture is made up of three types of layers: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. In a simple CNN design, the architecture[INPUT-CONV-RELU-POOL-FC] could be used. To be more specific :

1. The input image would be stored as a three-dimensional array of pixel values in the INPUT layer.
2. The dot product of a kernel and a sub-array of an input picture of the same size will be computed by the CONV layer. The single pixel value of an output image will be calculated by adding all of the values obtained via the dot product. This procedure is continued for each kernel until the entire input image has been covered.
3. The activation function max(0,x) will be applied to all of the pixel values in an output image by the RELU layer.
4. The POOL layer will do downsampling along the image's width and height, resulting in the image's dimension being reduced.
5. The class score for each classification category will be calculated by the FC (Fully-Connected) layer.

The original image is layered from the first pixel values to the final class scores by ConvNets. The RELU and POOL layers implement the constant function, and no variables should be trained at this layer. The parameters at the FC and CONV layers will be trained using the gradient descent optimizer.

**2.1 Convolutional Layer**

ConvNet is built around this layer. It takes care of the majority of the heavy work in terms of computation. This layer's hyperparameters are as follows:

The number of layers present in an output volume is represented by its depth.

The height and width of the filters utilised are represented by the filter size (F).

The step size to take when travelling horizontally and vertically along the height and weight of an input image is represented by stride (S).

Padding (P) helps in the maintenance of an input image's height and width. For each layer, hyperparameters are set. The values of multiple convolutional layers may differ. The filter matrix is also a weight matrix for which backpropagation must be used to train values. All of the filters in this layer must be trained, and they are all initialised with small random numbers.

An output volume's height and weight are determined by:

height, weight = floor( ( W+2\*P-F )/S +1 )

depth = K (number of filters used)

**2.2 Pooling Layer**

This layer is used to reduce the size of a volume input. The depth of an input is not reduced by this layer. This layer can be used to reduce the spatial size of the image, lowering the computational power required to process it. The important property of an image is not lost while using a pooling layer.

There are two types of pooling: maximum pooling and average pooling.

The maximum value present in a specified kernel is maintained in max pooling, while all other values are eliminated. The average of all the values present in a kernel picked is stored when using average pooling. Pooling also serves as a noise dampener. However, because max pooling outperforms average pooling, it is more commonly utilised.

This layer's hyperparameters include:

Filter size (F), which represents the size of the kernel to be used.

The number of steps to take while sliding the kernel window is represented by stride (S).

The padding (P) parameter specifies how much padding should be applied to an input image. Padding is usually not used at this layer.

There is no need to train the filters. Backpropagation has no effect on this layer as a result. And once the hyperparameters are set, they're fixed.

Now, the dimension of an output volume can be determined in the same way as the convolutional layer's dimension can be calculated. The depth of an output volume is comparable to the depth of an input volume in this case.

**2.3 Fully-Connected Layer (FC-Layer)**

This layer is used to classify the complicated features that were taken from the preceding layers. This layer is similar to neural networks, in which each neuron is connected to every other neuron on the following layer. Softmax is used to produce the final output, which provides the probability of each class for the given features.

FC-layer functions in a similar way as deep neural networks, which are utilised for classification.

To transfer an image to the FC-layer, it must be flattened out such that all of the pixel values are placed in a single column. The FC-layer now receives this flattened feature. All of the weights in the FC-layer are trained using a back-propagation technique and are initialised with tiny random integers.

1. **Methodology**
   1. **Preprocessing the Dataset:**
   2. **Training:**
   3. **Evaluation Criteria and Results:**
   4. **Challenges:**

**Material and Data**

GTSRB(German Traffic Sign Recognition Benchmark) dataset is used for the experiment. The dataset was created from approx 10hr of video while driving on different road types of Germany in the daytime[1]. It is a multi-class classification problem having 43 different traffic sign classes. The dataset contains total 51839 images. Among them, the train set and test set contains 39209 and 12630 images respectively. The GTSRB dataset [2] is structured into two separate directories, training and testing. The training directory has 43 directories, one directory for each class. Images in the training directory are grouped by tracks and there are 30 images of one traffic sign in each track. Every image has only one traffic sign and its size is between 15\*15 to 250\*250 pixels. It is stored in the PPM format and every image is not necessarily squared and the traffic sign within the image is not always centered. Two separate CSV files for train and test set are given for reading the images and it contains the following fields. The width and height field has stored image width and height. ClassId contains the assigned label of the image and the Path field has the image path.

**5) Computational Experiments**

**5.1 Experiments Performed:**

GTSRB dataset is a widely used dataset for traffic sign classification problems. The dataset has two directories containing train and test images. We divided the images which are in the train directory into two parts - train set and validation set. The final distribution of the data contains 60% train images, 20% validation images and 20% test images. Train images are used for training the model, while validation images are used for improving the training accuracy. Test images are used for assessing our model accuracy.

Chart, histogram

Description automatically generated

Figure 1: Frequency distribution of training data set

From the histogram, it is clearly seen that the dataset is not evenly distributed. Some image classes contain more images than others. It depicts that our model will be biased to some classes. To handle this, we are applying the class weight concept. In which, every class has assigned a weight based on the number of samples it contains in the train set. It penalizes more for samples that have a higher weight. The compute\_class\_method of scikit learn is used for assigning the weight to classes. The balanced heuristic is used for calculating the class weight.

In order to generalize our model better and improve the performance, we have applied some image data augmentation techniques to images. It will create a modified version of the image and from that our model will be able to generalize better and model accuracy will also be improved. Different techniques that were used for the image data augmentation[3] in our experiment are as follows. Horizontal flip - it will randomly flip the image into a horizontal direction and it is a much more common technique. Shear Range -  it will twist the image along with the axis to create a different perception angle of the image and for that, we choose 0.2 shear intensity. Zoom range - It randomly zoom the image in or out for creating an augmented image. We have specified zoom range value to 0.2 so, it will randomly zoom the image between 0.8 to 1.2 range. It means that the range will be between 80%(Zoom In) and 120%(Zoom out). Rescale - we are using the RGB image and its pixel value is between 0 to 255 so to apply the rescaling we are specifying rescale parameter to 1./255 to rescale the image between 0 to 1 range. For applying all these image data augmentation ImageDataGenerator class of the Keras library is used. It is very simple to apply these image transformations using this class.

Once the splitting of the data and image augmentation methods applied to the images, we will train our model using two different architectures.

**Architecture-1**

This architecture is based on the LeNet-5 Architecture[4]. LeNet-5 consists of 7 layers without counting the input layer. All the layers consist of trainable parameters. The input is of 32x32 image. The layers are in the following order in the LeNet architecture. Convolution layer-1, Max pooling layer-1, Convolution layer-2, Max pooling layer-2 then two fully connected dense layers followed by the output layer. The activation function used in this architecture is tanh[5] (hyperbolic tangent function) and for the output layer, sigmoid[5] activation function is used.

| **Layer (type)** | **Output Shape** |
| --- | --- |
| conv2d (Conv2D) | (None, 28, 28, 6) |
| max\_pooling2d (MaxPooling2D) | (None, 14, 14, 6) |
| conv2d (Conv2D) | (None, 10, 10, 16) |
| max\_pooling2d (MaxPooling2D) | (None, 5, 5, 16) |
| flatten (Flatten) | (None, 400) |
| dropout (Dropout) | (None, 400) |
| dense (Dense) | (None, 120) |
| dropout (Dropout) | (None, 120) |
| dense (Dense) | (None, 84) |
| dropout (Dropout) | (None, 84) |
| dense (Dense) | (None, 43) |

Table 1: Output data shape and layers of Architecture-1

We have modified the LeNet architecture and proposed a new architecture in which we have made few changes for better accuracy and performance. The input image has been transformed to 32x32 height and width and the image is RGB so the dimension is set to 3. Instead of using the tanh activation function for the convolution and dense layers, we used ReLU[5] activation function because ReLU will quicken the convergence compared to tanh. The kernel size for the first convolution layer and second convolution layer is 5x5 but the first convolution layer has 6 filters while the second convolution layer has 16 filters. A single convolution layer will extract the low-level features from the images. The first and second max-pooling layers have a 2x2 pool size and stride of 2. It is used for downsampling. Then, there is a flatten layer that converts the dimension to one and flattens the output. This output is then fed to two fully connected layers with units 120 and 84 followed by the output layer, which uses the softmax as the activation function[5]. It takes the vector of real numbers as input and converts it to the probability vector that sums up to 1. To minimize the loss and improve the accuracy we have used the Adam optimizer.

Firstly, the dropout layer was not used in the architecture and the model was overfitting to the training data because the training accuracy was much higher than the testing accuracy. Then we added the Dropout layers to solve the overfitting problem. It randomly sets the output of the hidden neuron to 0 at each update of the training phase. After adding 3 Dropout layers with a dropout rate of 0.25, 0.25 and 0.5 respectively, we got the best fit to our model. Still for better accuracy and to decrease the loss we have added additional convolutional layers to extract high-level features and that we will discuss in Architecture 2.

**Architecture 2**

| **Layer (type)** | **Output Shape** |
| --- | --- |
| conv2d (Conv2D) | (None, 28, 28, 32) |
| conv2d (Conv2D) | (None, 24, 24, 32) |
| max\_pooling2d (MaxPooling2D) | (None, 12, 12, 32) |
| dropout (Dropout) | (None, 12, 12, 32) |
| conv2d (Conv2D) | (None, 10, 10, 64) |
| conv2d (Conv2D) | (None, 8, 8, 64) |
| max\_pooling2d (MaxPooling2D) | (None, 4, 4, 64) |
| dropout (Dropout) | (None, 4, 4, 64) |
| flatten (Flatten) | (None, 1024) |
| dense (Dense) | (None, 256) |
| dropout (Dropout) | (None, 256) |
| dense (Dense) | (None, 43) |

Table 2: Output data shape and layers of Architecture-2

This architecture is a modified version of Architecture-1. The input shape of the image is the same as the Architecture-1. But here we have added an additional convolutional layer. By doing this we stack the convolutional layer and the first convolutional layer will apply directly to the image and extract raw pixel values with low-level features. An added convolutional layer will help to identify high-level features by extracting the shapes or arcs. The first set of convolutional layers uses 32 filters and kernel size is 5x5, while the second set of convolutional layers has 64 filters and kernel size is 3x3. Still, ReLU activation function is used for increasing the non-linearity in images. We have applied the Maxpolling for subsampling with the pool size of 2x2 and stride 2. Then Flatten layer for converting the output to 1 dimension and then feed it to the fully connected layer with 256 units. The output layer has 43 units for classifying 43 image classes of the GTSRB dataset. It uses the softmax function. The 3 dropout layers have set dropout ratios of 0.25, 0.25 and 0.5 respectively. This architecture has given better accuracy compared to Architecture-1.

**5.2. Evaluation Metrics:**

For both architectures, we have used adam as an optimizer function. This optimizer is based on the Adam algorithm which performs adaptive estimations of the first order and second-order moments. Adam’s optimization is a stochastic gradient descent method that is computationally efficient and is well suited for larger datasets.[6]

For loss, We have used the categorical crossentropy loss function because there is presence of more than two label classes. This loss function is used for calculating the crossentropy loss in between the labels and predictions.

**5.3 Implementation Details:**

For implementation we have used the Python programming language. Keras and tensorflow libraries are used for model building. For data structuring and manipulation, pandas and numpy libraries are used. Scikit learn is used for handling the imbalance dataset. Matplotlib and Seaborn are used for data visualization. We have train our model using GPU on google colaboratory.

**5.4 Results:**

| **Performances** | **Architecture-1** | **Architecture-2** |
| --- | --- | --- |
| Epoch 1 [train acc/val acc]; [train loss/val loss] | [0.1776/0.4875]; [2.9318/1.6889] | [0.3028/0.6619]; [2.4585/1.0886] |
| Epoch 5 [train acc/val acc]; [train loss/val loss] | [0.6260/0.7961]; [1.0413/0.6394] | [0.8259/0.9250]; [0.4498/0.2274] |
| Epoch 10 [train acc/val acc]; [train loss/val loss] | [0.7217/0.8397]; [0.7517/0.4658] | [0.8884/0.9674]; [0.2818/0.1105] |
| Epoch 15 [train acc/val acc]; [train loss/val loss] | [0.7621/0.8783]; [0.6185/0.3824] | [0.9184/0.9723]; [0.2174/0.0824] |
| Epoch 20 [train acc/val acc]; [train loss/val loss] | [0.7897/0.9045]; [0.5491/0.2989] | [0.9261/0.9736]; [0.1934/0.0804] |
| Epoch 25 [train acc/val acc]; [train loss/val loss] | [0.8122/0.9245]; [0.5101/0.2615] | [0.9291/0.9782]; [0.1815/0.0627] |
| Epoch 30 [train acc/val acc]; [train loss/val loss] | [0.8245/0.9171]; [0.4676/0.2652] | [0.9386/0.9809]; [0.1579/0.0565] |
| Epoch 35 [train acc/val acc]; [train loss/val loss] | [0.8321/0.9296]; [0.4401/0.2237] | [0.9425/0.9852]; [0.1485/0.0463] |
| Epoch 40 [train acc/val acc]; [train loss/val loss] | [0.8421/0.9283]; [0.4319/0.2126] | [0.9460/0.9844]; [0.1416/0.0460] |
| Training Time | 20.15 min | 22.58 min |
| Test Accuracy | 0.8695 | 0.9505 |

Table 3: Performance comparison of our two models for 40 epochs

Chart

Description automatically generatedA picture containing graphical user interface

Description automatically generated

1. Accuracy and loss for model 1

Graphical user interface

Description automatically generated with low confidenceA picture containing graphical user interface

Description automatically generated

1. Accuracy and loss for model 2

Figure 2: Accuracy and loss graphs for our two models

From the accuracy and loss graph of architecture-1, It can be noted that with the increase in the number of epochs, both the training and validation accuracy increases significantly. Also, it can be clearly seen that the gap between the training accuracy and validation accuracy is significantly lesser, indicating lesser overfitting. Training accuracy is 84.21% and test accuracy is 86.95% for architecture 1. It can be noted from the loss graph that, with the increase in the number of epochs, there is a notable decrease in training and validation loss. Nearly after 15 epochs, validation loss and training loss were almost less than 0.75.

Accuracy for architecture 2 came out to be better than the accuracy of architecture 1 and with an increase in the number of epochs, there was a slow and steady increase in both accuracies. Also, the gap between the training and validation accuracy was significantly lesser, which indicates lesser overfitting. Training accuracy is 94.60% and test accuracy is 95.05% for architecture 2. It can be noted that there is a decrease in loss with the increase in the number of epochs. Nearly after 10 epochs, validation loss and training loss were less than 0.5.

**6) Conclusion:**

**6.1 Summary:**

Main goal of traffic sign recognition is to make a system which identifies traffic signs using the Convolutional Neural Network in deep learning and it can be useful in autonomous vehicles. For identifying traffic signs, we have used different image data augmentation techniques to ensure lesser overfitting and trained our model using modified LeNet 5 architecture and obtained an accuracy of 95.05% for our model.

**6.2 Further Research:**

Following are the essential challenges that need to be addressed and they also serve as a basis for future work. Difficulty in classifying traffic signs during adverse weather conditions such as heavy rain, fog and snow. At nighttime, due to darkness it can be complex to capture and classify traffic signs, which can lead to traffic accidents. Traffic sign images that are distorted or sometimes covered by shadows of other nearby objects can lead to improper traffic sign recognition by the automobile system.

OR

We have some limits, as described in the challenges section above, such as unfavourable weather conditions such as rain, fog, and snow; it is difficult to identify and classify such photos. In the same way, intense sunlight and darkness at night make it difficult to see traffic signals. We have yet to uncover a strategy for classifying photos that are affected or obscured by tree or other object shadows. This gives us a substantial amount of work to undertake in the future in this approach.

**References**

[1] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural Networks*, vol. 32, pp. 323–332, 2012.

[2] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “The German Traffic Sign Recognition Benchmark: A multi-class classification competition,” *The 2011 International Joint Conference on Neural Networks*, 2011.

[3] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *Journal of Big Data*, vol. 6, no. 1, 2019.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] T. Szandała, “Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks,” *Bio-inspired Neurocomputing*, pp. 203–224, 2020.

[6] Z. Zhang, “Improved Adam Optimizer for Deep Neural Networks,” *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018.

**REFERENCES**

[1] Haque, W. A., Arefin, S., Shihabuddin, A. S. M., & Hasan, M. A. (2021). DeepThin: A novel lightweight CNN architecture for traffic sign recognition without GPU requirements. Expert Systems with Applications, 168, 114481.

[2] Chaudhari, T., Wale, A., Joshi, P., & Sawant, P. (2020). Traffic Sign Recognition Using Small-Scale Convolutional Neural Network. Available at SSRN 3645805.

[3] Jabbar, S., Khan, M., Khan, H., & Naseer, Z. Recognition of traffic signs by convolutional neural nets for Autonomous Vehicles.