Traffic Sign Classification

Ajay Dahiya

C0783939@mylambton.ca

Angelica Serrano C0785370@mylambton.ca

Jonatas Aguiar C0790419@mylambton.ca

Akshita Khatri C0785493@mylambton.ca

Kuldeep Singh C0790419@mylambton.ca

AML-3104 Neural Networks and Deep Learning

Instructor: Mohammad Islam

Lambton College – Toronto

Abstract— **Image classification is a Machine Learning method that recognizes an input image and predicts its category or class. Several factors and challenges are associated with building classification models, such as image variations, imbalanced datasets, and overfitting. The most common network model for building an image classification problem is Convolutional Neural Network (CNN). Tensorflow Keras and Pytorch are the most widely used libraries to train and compile a CNN model. Training a CNN model requires just enough features of an image to produce a good result, unlike other types of Neural Networks. In this project, CNN was leveraged to solve a Traffic Sign Classification problem.**

# INTRODUCTION

Road safety is one of the priorities in the Canadian transport system. For several years now, applications dealing with driver assistance, traffic detection, traffic sign classification and smart cars have become an important topic that Machine Learning researchers have been putting immense work on. Their research has led to the creation of brilliant systems using deep learning.

Deep learning is a subset of machine learning that deals with the construction of multiple layers of Neural Networks. Essentially, deep learning works by training a model to imitate human actions and thinking. One of the applications of deep learning is to solve an image classification problem using Convolutional Neural Network.

Traffic sign classification is one example of an image classification use case. With deep learning, an application that can accurately recognize and identify common traffic signs along the road is now possible. Devices that may act as driver alert systems can generally assist drivers with poor eyesight, such as those with Myopia or nearsightedness. More often, this condition leads to driving mistakes, road violations and accidents. Aside from having proper road education and good vehicle condition, it would also be valuable to have a proactive system in our cars. This may serve as additional security while drivers are focusing on the wheels.

The main goal of this project is to build a Traffic Sign classification system using a Convolutional Neural Network model. Tensorflow, Keras and Pytorch were utilized for training a model using a publicly available dataset. A web application was created on top of the CNN model to execute it on the back end and return the interpreted output in high level format. Additionally, like any driver alert system - a speech generator was also embedded in the system to utter the predicted traffic sign.

# LITERATURE REVIEW

Convolutional Neural Networks (CNN) are becoming one of the top choices for image classification since the launch of AlexNet [1]. Apart from AlexNet, ResNet [2] and DenseNet [3], other examples of new architectures are used for CNN. Various model improvements have also been seen since then. For instance, validation accuracy on ImageNet was increased to 82% from 62% [4]. These improvements are attributed to enhancements in model infrastructure, loss functions, optimization methods, and data processing [5].

A prevalent problem in improving the model accuracy is the lack of quality data. The noises can be easily added to images by misclassification of these images [6]. One of the essential aspects of improving image classification, specifically the CNN model, is data augmentation. Various data augmentation methods have been analyzed and compared to shed more light on enhancing the capabilities of augmenting the data and improving CNN [7]. A similar type of work involving comparing various data augmentation methods was also performed by many authors [8], thereby highlighting the importance of data augmentation in image classification.

Nowadays, CNN has been used in many applications, such as disease classification using medical image techniques [9], abnormal brain image classification [10], environmental classification such as coconut tree classification [11], remote sensing hyperspectral image classification [12], and traffic sign classification [13, 14]. However, no research is done to convert the traffic sign into a textual-based speech for drivers' assistance while driving. Our research is focused on using CNN to classify traffic signs and convert the text to speech.

# METHOD

## Data description

The data used in this project is the GTSRB (German Traffic Sign Recognition Benchmark) dataset which originated from INI Benchmark website. It was a collection of real-life images of German traffic road signs and was formerly used for a competition at IJCNN in 2011.

**A.1 Train and Test data**

A total of 3,3799 images were assigned for training data, labelled with 43 classes. Test data consist of 12,630 non-labelled images, and 4,410 images were used for validation.

A screenshot of a computer screen

Description automatically generated with medium confidence

***Figure 1 –*** *Preview of GTSRB dataset*

The training dataset is imbalanced with a higher number of images of a specific class. For instance, Class 3, Speed limit (50 km/h) class, has the highest image count while Class 0, the Speed limit 20km/h class, has the lowest value count. Thus, it is more likely that majority of predictions will return Class 3, among others.

Chart

Description automatically generated

***Figure 2 –*** *Graph showing train dataset distribution*

Nevertheless, the original dataset was used in the same way as it is. Further in the experiment phase, various approaches were conducted such as inclusion of non-German signs. These will be discussed in the subsequent sections of this paper.

## Data Preprocessing

On this section, different preprocessing techniques were applied on the dataset. The goal of this step is to make the train images ready for ML model training and analysis.

**B.1 Image augmentation using Augly**

Augly, a newly released library by Facebook, was primarily created for data augmentation. It supports four modalities, including images, and has been used in several applications to identify fake images.

For the initial approach done in this project, Augly was used to perform arbitrary transformations on the train images. These new images were then added to the original set to create input data for the CNN model. The following options were used for augmentation:

1. Sharpen – to increase the sharpness of image
2. Saturate – to increase the intensity of image colors
3. Shuffle – to shuffle the pixels
4. Rotate – to randomly rotate the image to a certain degree
5. Pixelization – to pixelate the image
6. Blur – to decrease the sharpness of image
7. Perspective transform – to change the perspective
8. Change Aspect Ratio - to randomly change the dimension

**B.2 Image augmentation using Keras Image Data Generator**

Image Data Generator from Keras is another augmentation library utilized in the second approach for CNN modelling. Although the transformation options are somehow similar with Augly, this preprocessing library works a little bit different.

The generated batches of image and labels were transformed into the .flow data, which was then used as input images for the Keras model.

Text

Description automatically generated with medium confidence

***Figure 3 –*** *Snippet of .flow data generation. The defined batch size means that the train images are divided into groups of 15 images to be transformed.*

The following geometric transforms were applied on the train set:

1. zoom\_range – to randomly zoom images
2. shear\_range - to randomly apply shearing transformations
3. rotation\_range - to randomly rotate pictures.
4. width\_shift\_range – to randomly transform image horizontally.
5. height\_shift\_range – to randomly transform image vertically.

**B.3 Image conversion to Torch format**

To train a Pytorch model, it is required to have input data with torch format. Hence, the train and test dataset have been converted to appropriate format using torch.from\_numpy(). This module allowed transformation of original data from ndarray to tensor, in which both shared the same memory.

Subsequently, using torch utils library, the newly converted data were transformed into data loader set. This set was used as input during Pytorch modelling phase.

**B.4 OpenCV Image transformation**

As additional image transformation approach, these pre-processing techniques were utilized as well:

1. Conversion to grayscale
2. Histogram equalization for contrast adjustment.

**B.5 Image Normalization**

Images, same with text and numerical data, needs to be scaled or normalized as well. This process works by converting the images into numpy array, diving the values by 255 and producing an output ranging from 0 to 1.

## Convolutional Neural Networks

**C.1 CNN Model - Experiment**

To generate the best CNN model, three methods were performed during experimentation stage.

**C.1.1 Method 1 - CNN model using Keras**

Keras is a deep learning API running on top of TensorFlow. Sequential function, particularly, was leveraged in creating seven models with individual hyperparameters. A conventional CNN architecture, applied to each model, was composed of convolutional layers, ReLU activation function, pooling layers, a fully connected layer and a SoftMax layer.

A picture containing timeline

Description automatically generated

***Figure 4 –*** *CNN model architecture*

After training stage, the models were compiled using Categorical cross entropy loss function to calculate the loss and the Adam optimizer to optimize the loss function.

Figure 5 shows plots of loss and accuracy curves for both training and validation. It was evident that the first model has the highest validation loss, which was caused by overfitting. Overfitting happens when the train data have high variance. Therefore, the following overfitting reduction techniques were applied:

1. Regularization – to enforce simple neural network.
2. Weight initialization – using “he initialization”
3. Weight constraints – to rescale the network weights if it exceeds the pre-defined limit.
4. Dropout regularization – to ignore random subset of units during training.

Diagram

Description automatically generated

Diagram

Description automatically generated

***Figure 5 –*** *CNN model loss and accuracy graph*

Applying overfitting reduction steps led to lower loss level and higher accuracy of 88% for the final model.

Graphical user interface

Description automatically generated



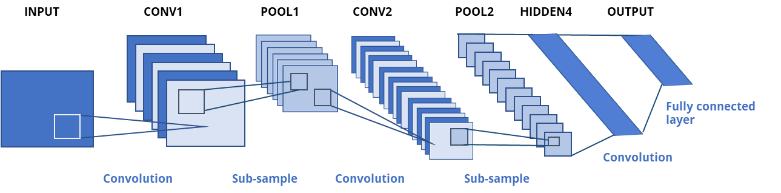
***Figure 6 –*** *Preview of the final model output, loss/accuracy graph and accuracy score*

Additionally, since this model used 43,000 input images with 256x256 dimension, it produced a 3GB size of H5 (model) file. Thus, pruning technique was implemented. Pruning is a method of compressing a machine learning model by eliminating non-critical sections or parameters. The process has successfully brought down the model file size to 1GB.

**C.1.2 Method 2 – CNN model using Keras (LeNet architecture)**

Similar libraries and API were used in this approach. But pre-processing and model training were handled differently. Data augmentation on 32x32 input images were achieved using Keras Image Data Generator and the generated flow data were fed into the model directly during training phase. The main benefit of this approach is that its more straightforward and faster.

Primarily, this model adopted LeNet 5 architecture, a classic model proposed by Yann LeCun et al. in 1989. The model involves two sets of convolutional, activation (ReLU) and pooling layers; followed by a fully connected layer, activation, another fully connected layer, and lastly – a SoftMax layer. This model is simple, easy to understand and enough to provide good result. To reduce overfitting, a dropout layer with rate of 0.5 was added to the sequential model.



***Figure 7 -*** *Classic LeNet 5 architecture*

A picture containing chart

Description automatically generated



***Figure 8 –*** *Preview of model output, loss/accuracy graph and accuracy score*

**C.1.3 Method 3 – CNN model using Pytorch**

Pytorch based CNN model was developed from scratch using both ‘cpu’ device and cuda to leverage ‘gpu’ power. The generated model has four total layers and input of 32x32x3. This approach was the most complex among the methods applied for this project. Several hyper-parameter tunings like increasing and decreasing input/ouput channels, reducing the batch\_size and decreasing memory size requirements were performed to get the most logical result. Moreover, running a torch model required higher RAM capacity which is the most challenging part.

Nonetheless, final model was successfully generated with an accuracy score of 23%.

A picture containing text, person, screenshot

Description automatically generated

***Figure 9 –*** *torch model details*

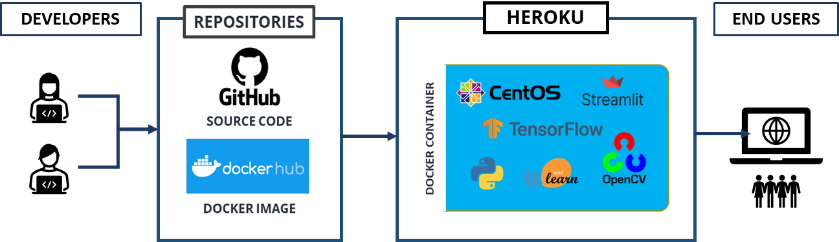
**C.2 CNN Result and Analysis**

Out of the three approaches conducted, Method 2 provided the highest test and validation score with 94%. Although overfitting still exist, the testing done on new images (all German signs) yield 60-70%.

During User Acceptance Test (UAT), the model was tested against non-German traffic signs and the result was poor with accuracy score ranging from 20-30%. Adding 70 non-German signs on the train dataset improved the accuracy, but percentage was not significant.

## User Interface

A web application was built to run the final CNN model with a graphical user interface.Various tools were used to create a simple but well-structured architecture.



***Figure 10 –*** *TS Classification application architecture*

**D.1 Web Framework using Streamlit**

Streamlit, a python library that was created mainly for displaying any machine learning model output in a browser, was utilized for this project. Streamlit is easy to use and comes with pre-built web components like the image upload function and selection buttons. By default, Streamlit service runs in port 8501 and it was retained for this project, so that port 8080 can be used in other purpose like debugging process. The code workflow starts with loading the model and the datasource csv files respectively.

Then the uploaded image will be pre-processed by converting to grayscale, applying histogram equalizer (equalizeHist), normalizing and reshaping.



***Figure 11*** *– Image pre-processing (*[*image source*](https://www.freepik.com/premium-photo/traffic-sign-overtaking-is-forbidden-moroccan-highway-no-passing-road-sign_3616882.htm)*)*

Model.predict\_classes() function returns the predicted class number and its corresponding score. The resulting class will then be mapped against the data source file which contains the road sign names. Finally, result will be generated as the corresponding road sign name of the resulted class.

Graphical user interface, application

Description automatically generated

***Figure 12 –*** *Sample prediction output*

As an additional function, speech synthesis API was applied to generate speech or utter the predicted road sign. In the example shown at Figure 11, the speech API will speak “Overtaking is prohibited”. Although, this function is manually triggered through a button click, it can significantly contribute to the creation of more robust version of this application in the future.

**D.2 Containerization using Docker**

Docker was leveraged to compile the whole application into one image file. The main benefit of containerizing any application is the elimination of manual configuration. Since the application uses various tools like Streamlit service, proper port setup and dependency installations are required. Hence, it would bring massive reduction of man hours and effort if these set-up procedures will be automated through Dockerfile and requirements.txt file.

**D.3 Heroku: Cloud Application Platform**

To expose the application in a public DNS, the image was deployed to Heroku, a Platform-as-a-service (PaaS) that offers free tier hosting of simple applications with slug size limit of 500MB. Heroku has its own code repository and allows deployment of an image file not exceeding the size limit.

**D.4 Code Versioning**

GitHub was used primarily to store and manage the versioning of the application code. To efficiently manage all the project artifacts, proper branches were created for smooth SCM (Software Configuration Management) process. Furthermore, the updated version of application was also compiled regularly and pushed to the Docker Hub with image name madserrano/ts-classify.

## Summary and Conclusion

The project is mainly focused on creating a multi-class classification model using Convolutional Neural Network. The application involved traffic sign detection that can recognize common traffic signs.

The data used for training was GTSRB (German Traffic Sign Recognition Benchmark) dataset, originally from INI Benchmark website. GTSRB is a popular multi-class classification dataset used for a competition at IJCNN in 2011.

Three methods were conducted to generate a model that could provide the most logical result. Two methods used Keras sequential API and the other one used Pytorch. As a result of this experiment, Method 2, which leveraged LeNet architecture, produced the highest validation score of 94%. Evaluation using new image sets of German traffic signs yield 60-70%, while non-German traffic signs produced poor accuracy.

It was concluded that a good quality model can be achieved with adequate and balanced dataset. The accuracy of Model 2 can be sufficient for experimental level, but for a more robust model, one must invest a great amount of time and sufficient machine processor to train it.

The final version of this project runs in a public DNS with URL: <http://ts-classification.herokuapp.com/>. Streamlit web framework was used to build the user interface and the whole set-up was compiled in a docker image, deployed to Heroku cloud service.

# REFERENCES

1. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, 1097-1105. <https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>

2. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778). <https://arxiv.org/abs/1512.03385>

3. Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708). <https://arxiv.org/abs/1608.06993>

4. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211-252. <https://arxiv.org/abs/1409.0575>

5. He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., & Li, M. (2019). Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 558-567). <https://arxiv.org/abs/1812.01187>

6. Jin, J., Dundar, A., & Culurciello, E. (2015). Robust convolutional neural networks under adversarial noise. *arXiv preprint arXiv:1511.06306*. <https://arxiv.org/abs/1511.06306>

7. Mikołajczyk, A., & Grochowski, M. (2018, May). Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)* (pp. 117-122). IEEE. <https://ieeexplore.ieee.org/document/8388338>

8. Wang, J., & Perez, L. (2017). The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit*, *11*, 1-8. <https://arxiv.org/abs/1712.04621>.

9. Yadav, S. S., & Jadhav, S. M. (2019). Deep convolutional neural network based medical image classification for disease diagnosis. *Journal of Big Data*, *6*(1), 1-18. <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0276-2>

10. Hemanth, D. J., Anitha, J., Naaji, A., Geman, O., & Popescu, D. E. (2018). A modified deep convolutional neural network for abnormal brain image classification. *IEEE Access*, *7*, 4275-4283. <https://ieeexplore.ieee.org/document/8571163>

11. de Souza, I. E., & Falcão, A. X. (2020). Learning cnn filters from user-drawn image markers for coconut-tree image classification. *IEEE Geoscience and Remote Sensing Letters*. <https://arxiv.org/abs/2008.03549>

12. Wu, H., & Prasad, S. (2017). Semi-supervised deep learning using pseudo labels for hyperspectral image classification. *IEEE Transactions on Image Processing*, *27*(3), 1259-1270. <https://ieeexplore.ieee.org/document/8105856>

13. Arcos-Garcia, A., Alvarez-Garcia, J. A., & Soria-Morillo, L. M. (2018). Evaluation of deep neural networks for traffic sign detection systems. *Neurocomputing*, *316*, 332-344. <https://www.sciencedirect.com/science/article/abs/pii/S092523121830924X>

14. Wang, C. (2018, May). Research and application of traffic sign detection and recognition based on deep learning. In *2018 International Conference on Robots & Intelligent System (ICRIS)* (pp. 150-152). IEEE. <https://ieeexplore.ieee.org/document/8410256>