Comparing Transfer Learning Techniques for Detection of Traffic Signs using Image Recognition

Hitesh Madhukar Patil
Batch-A, DMML2 Group-F
M.Sc. in Data Analytics
NCI
x19147996@student.ncirl.ie

Vikas Kishanrao Thamke
Batch-A, DMML2 Group-F
M.Sc. in Data Analytics
NCI
x19180080@student.ncirl.ie

Vishal Shakya
Batch-A, DMML2 Group-F
M.Sc. in Data Analytics
NCI
x19182732@student.ncirl.ie

Amandeep Singh
Batch-A, DMML2 Group-F
M.Sc. in Data Analytics
NCI
x19194137@student.ncirl.ie

***Abstract*—A large traffic sign image dataset was imported, explored, analysed, cleaned, pre-processed, transformed and then fed into various models by following the KDD methodology. In total, four models were compared, in which the first was a simple, self-designed CNN model, whereas the rest of the three were pre-trained "Transfer Learning" models, namely, InceptionV3, VGG16 & VGG19. The models were tested comprehensively by applying the Gaussian blurring technique to the image dataset. The classifier layers were intentionally kept the same across all the models to make the comparison easier. The self-designed CNN model managed to perform the best (best accuracy: 97.71%; training time: 361.44s), followed by the InceptionV3 model (best accuracy: 80.59%; training time: 332.37s), then the VGG16 model (best accuracy: 75.02%; training time: 665.41s) and the VGG19 model (best accuracy: 69.66%; training time: 801.50s).**

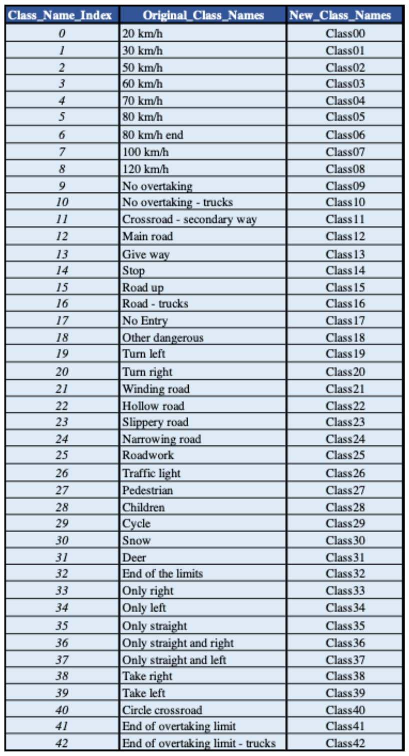
***Index Terms*—Transfer learning, traffic signs, image detection, recognition, CNN, InceptionV3, VGG16, VGG19.**

I. INTRODUCTION

During the mid-1900's, with the advancement in aviation & shipping technologies, the vessels carrying humans and goods around the planet were becoming more and more difficult to operate by human 'drivers'. Although this was partly due to the increasing complexity of the gigantic machines, it was mainly due to the additional variables in the equation - weather, longer more treacherous routes, and the increasing number of vessels operated by other humans. This meant that the job of the 'driver', apart from doing the obvious task of driving the vessel, now also included the responsibility to safely wade through the precarious natural conditions, while also continuously communicating with fellow human 'drivers' to avoid collisions, on much longer demanding routes. This was a recipe for disaster, and disasters did eventuate. But the human in-charge alone cannot be put to fault, there is only so much a puny animal could do against the laws of physics or the wrath of nature - not because of the lack of ability, but because of the natural limitations of the human mind and body. Albeit a few exceptions, it is very strenuous to operate an enormous vehicle autonomously when the lives of other humans are at stake.

This is the reason why the need for automation exists - humans have limited capability and are prone to errors. The concept of autopilot, an automated mechanism to reduce the workload of the human, was born because of these shortcomings, and it was fuelled by the revolution in computer technology. So much so, that in 1969, the lunar module on the Apollo mission that carried the first humans to the Moon, was fully digitally automated. Almost half a century later, and the need for similar autopilot mechanisms is still present, not so much in aviation or shipping, but in cars. After all, millions of people lose their lives in road accidents every year, and the main cause is human error. Constant progress is being made to fully automate the process of driving to eliminate the human from the equation. This means that the vehicle would have to have some inherent understanding for the rules of the road. To achieve this, many Deep Learning techniques and algorithms that analyse the road conditions and traffic rules, on the fly, using camera systems on the vehicle, have been proposed and implemented, with varying degrees of success. But the quest to achieve full automation is going on, and a question has arisen - which is the best Deep Learning algorithm that can be used to achieve full automation of a vehicle? The field of Deep Learning is too broad to sieve through in one project report. So, to make the focus more specific, this project is going to look at the best algorithms that exist in a subset of Deep Learning technique, called Transfer Learning (TL), and compare them to a simple, self-designed "Convolutional Neural Network" (CNN), to attempt to answer the question - "***Which is the best TL algorithm to automatically detect & recognise the traffic signposts on the road?***"

Identifying and classifying traffic signposts is a challenging job due to many complexities, such as words and different symbols, because the self-learning models are normally implemented on pre-defined datasets. Out of the various deep learning networks, CNN has earned noteworthiness for its proved advancements over other networks. CNN has efficient learning skills, with many excellent features such as the translation invariance and local links. Generally, a traffic signpost recognition system consists of two parts: (i) detection of the signs; and, (ii) classification of the detected signs. Compared to a classification problem, the signpost recognition problem needs a more complex network to solve, which is a more difficult task, and research is still being done. Owing to the superior performance produced by CNN, the examination, modelling and evaluation of different aspects of CNN for a large dataset continue to be a compelling area of research.



Class_Name_Index	Original_Class_Names	New_Class_Names
0	20 km/h	Class00
1	30 km/h	Class01
2	50 km/h	Class02
3	60 km/h	Class03
4	70 km/h	Class04
5	80 km/h	Class05
6	80 km/h end	Class06
7	100 km/h	Class07
8	120 km/h	Class08
9	No overtaking	Class09
10	No overtaking - trucks	Class10
11	Crossroad - secondary way	Class11
12	Main road	Class12
13	Give way	Class13
14	Stop	Class14
15	Road up	Class15
16	Road - trucks	Class16
17	No Entry	Class17
18	Other dangerous	Class18
19	Turn left	Class19
20	Turn right	Class20
21	Winding road	Class21
22	Hollow road	Class22
23	Slippery road	Class23
24	Narrowing road	Class24
25	Roadwork	Class25
26	Traffic light	Class26
27	Pedestrian	Class27
28	Children	Class28
29	Cycle	Class29
30	Snow	Class30
31	Deer	Class31
32	End of the limits	Class32
33	Only right	Class33
34	Only left	Class34
35	Only straight	Class35
36	Only straight and right	Class36
37	Only straight and left	Class37
38	Take right	Class38
39	Take left	Class39
40	Circle crossroad	Class40
41	End of overtaking limit	Class41
42	End of overtaking limit - trucks	Class42

Fig. 1. Data Dictionary

A. *Data Sources:*

The dataset used in this project was sourced directly from the Kaggle repository – "*German Traffic Sign Recognition Benchmark*" (GTSRB), created by the user *Mykola* (Kaggle link) [1].

The dataset is licensed under public domain, so its use does not pose any ethical concerns.

B. *Data Overview:*

Most CNN models on traffic signs have trained their models on the GTSRB database as it was used in the "International Joint Conference on Neural Networks" (IJCNN) in 2011 [2]. The GTSRB dataset is a single-image, multi-class dataset with more than 40 classes available for classification. The dataset consists of more than 50,000 images in total. Speed and accuracy of a network will undoubtedly be the two main challenges in a large dataset like GTSRB.

C. *Data Dictionary:*

The data dictionary for the GTSRB dataset used in this project is shown in figure 1.

D. *Organisation of the Project:*

The remainder of this paper is organised as follows – a comprehensive review of the published research concerning deep learning & image recognition in general, and the GTSRB dataset in particular, is presented in section II; while section III lists the methods implemented in the project, section IV details the models used to achieve the intended goal of the project; section V lists and evaluates the results, followed by the final concluding remarks and future research options in section VI.

II. Related Works

Traffic signpost recognition has been an active area of research for the past decade. With many novel techniques, a simple traffic signpost recognition system can be designed using an "*Artificial Neural Network*" (ANN). For recognition of image, ANN shows exceptional performance, both accuracy-wise and speed-wise, as implemented in [3] & [4].

In ANN for image processing, many techniques have been used, among them, "*Convolutional Neural Network*" (CNN) is considered to be the most suitable, and hence is the most popular technique as it has features such as weight sharing, pre-training, feature learning and extraction [5]. The concepts of CNN and AlexNet were thoroughly explained in [6] and [7] respectively. In [6], the author explained a basic CNN model that used pre-defined convolving layers for pattern recognition, whereas [7] described the AlexNet model which did not have any fixed pattern for image recognition. In the "ImageNet Challenge Competition 2012", AlexNet was implemented and got an accuracy of 84.7% in [8], but due to a large number of parameters it became computationally expensive. Since then, many networks are being developed using custom versions of the CNN technique for image processing and recognition problems. For instance, in [9], the performance of the CNN model is increased to 98.21% using a max-pooling technique.

While a back-propagation algorithm, using mini-batch stochastic gradient descent, is shown in [10], Adam optimiser is implemented in [11] for optimisation of low-resolution images. [12], [13], use model-based optimisation techniques for solving problems of low image quality, while the accuracy of the traffic signpost recognition system is improved significantly using a max-pooling function in [9]. Due to many advances in image recognition system, along with the availability of free and open image repositories like ImageNet, OpenCV, etc., CNN has seen numerous improvements [14], primarily because of the "ImageNet Challenge Competition", which laid the foundation for very large image datasets in deep-convolution networks and solidified the foundation of TL networks.

TL method, which is a subset of the Deep Learning stream, has been used in various networks to reduce the computational time on the GTSRB dataset [15]–[17]. Out of the numerous TL algorithms being used, the InceptionV3 network and the VGG networks (VGG16 & VGG19) were selected for comparison in this project because these models were the winner and runner-up (respectively) of "The ImageNet Challenge Competition" held in 2015 [18]. While Hussam et al. [19] successfully introduced the VGG16 network on the ImageNet dataset, Joshua Paolo et al. [20] pioneered the InceptionV3 network through TL on traffic signpost detection & classification. On the other hand, Dickson Neoh et al. [21] expanded upon the findings and presented results for traffic signpost recognition using VGG19, VGG16 and InceptionV3 networks. In [20],

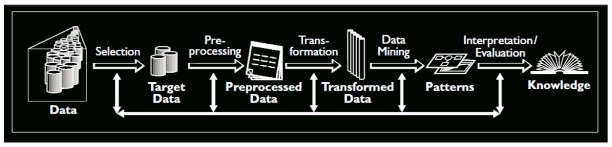


Fig. 2. KDD Methodology

the authors test the feasibility of InceptionV3 model on a traffic signpost dataset and their model achieved an accuracy of 95.70% in compliance mode and 96.20% in degradation mode. Chungkeun Lee Lin et al. [14] also presented a TL-based traffic recognition system on the traffic signpost database. While VGG19 is an advanced version of VGG16, its core does not change, except for the fact that VGG19 has 3 more convolutional layers. As shown in [21], the VGG19 model has an accuracy of 99.47%, which is only improved upon by InceptionV3, another pre-trained network, which also has an accuracy of 99.70% - highest among all the listed pre-trained models.

Total of three pre-trained models namely VGG16, VGG19 and InceptionV3 are used in [21], [20], [22] and [23]. In [19], the VGG16 model got an accuracy of 96.20% which was highest at that time due to its 16 layers deep neural network. In [21], the authors test the feasibility of VGG19 model on traffic signpost dataset, and they received accuracy of 99.47%. In 2015, 48 layers deep, the InceptionV3 model achieved an exceptional accuracy of 96.20% in [20]. In [22], the authors retrained a TL-based model for 5000 epochs, at different learning rates, and got an accuracy of 99.18%. The authors in [12] and [13] show that using a deep CNN denoiser along with Adam optimiser for low-quality image restoration can also have a significant effect on the results.

The significant contributions of [6], [7], [9], [10], [12], [17], [19]–[22] and [23] are given by a complete analysis of three state-of-the-art, pre-trained image recognition models can be followed as an example for selecting a network for the recognition task of German traffic signposts. Each model has its benefits (increased accuracy in some cases) and drawbacks (overfitting, longer training times) depending upon its architecture which can help us to choose the model. In papers [20] and [22], the VGG16 and InceptionV3 achieved better results on GTSRB dataset.

III. METHODOLOGY

The project code was compiled using the free online Python Jupyter notebooks tool available on *Google Colaboratory* (or simply, *Colab*) platform because of the lack of raw graphical processing power in the local machines. There was another merit to using *Colab* - the code and data syncing feature across the *Google Drive* platform. This made the data storage & handling very easy and efficient, while also not compromising with the graphical performance or data security.

The methodology followed in this project was *KDD* or "*Knowledge Discovery in Databases*". Each step in the process flow of this project, as listed in image 2 from [24] & [25], is explained in a step-wise manner below:

A. Data:

The data to be used in the project was selected from a list of different traffic-sign-images datasets on Kaggle. The dataset titled "GTSRB - German Traffic Sign Recognition Benchmark" [1] was finalised because of its large database of $\sim 50,000$ images across 43 classes of different road and traffic signs. The reviews on all the datasets were also checked, to check for any errors/issues, before the final confirmation.

B. Data Pre-processing:

1) Importing the Data: The next step was importing the data for pre-processing in a *Colab* worksheet. Three strategies were explored to accomplish this:

a) Approach-1: The dataset was downloaded from Kaggle to the local machine. The *.zip* file was unzipped on the local machine, and then the $\sim 50,000$ files were uploaded to *Google Drive* to avoid re-uploading the dataset every time the *Colab* notebook is compiled. This approach took $\sim$ 9 hours because of the large number of files in the unzipped dataset. Furthermore, a bug in the *Google Drive* environment forced this whole process to be performed every time the *Colab* notebook session expired. This made this approach unfeasible.

b) Approach-2: The dataset was downloaded from Kaggle to the local machine. This time the *.zip* file was directly uploaded to *Google Drive*. The files were unzipped directly in the cloud using *Colab*. This step, which used to take $\sim$ 8 hours in the previous approach, now took only $\sim$ 45 minutes. After that, the resulting image files were read into the *Colab* notebook. This revealed another bug in the *Google Drive* environment – when this whole process is done for the first time, the file reading process in the *Colab* notebook takes only $\sim$ 10 minutes, but if the notebook is closed and reopened later, the file reading takes more than 4 hours. This again, unfortunately, made this approach unusable.

c) Approach-3: The dataset was loaded directly to *Google Drive* from Kaggle using the *Colab* notebook. This process took $\sim$ 3 minutes to perform, compared to 10-15 minutes in previous approaches. Then the files were unzipped directly in the cloud and then read through the *Colab* notebook. This approach totally bypassed the bug and took $\sim$ 10 minutes to perform. Hence, this approach was finally selected.

2) Using NumPy Arrays: To manipulate the image dataset, and to apply the data mining techniques, the images had to be converted into multi-dimensional NumPy arrays that consisted of numerical values of the RGB (Red-Green-Blue) colour channels for each image. Moreover, these NumPy arrays were saved to the cloud. This step increased the efficiency of the third approach (mentioned above) because once the NumPy arrays were created and saved to *Google Drive* directly, instead of re-converting the images again, the saved NumPy arrays were read and used every time the kernel reconnected.

C. Data Transformation:

1) Resizing the Images: The dimensions of images in the 43 classes were not constant. This would lead to a loss in accuracy and increase in time taken when the algorithms are applied. Furthermore, the TensorFlow-Keras libraries, which were used for implementing the TL algorithms, required a minimum image dimension of 32x32 pixels. Some of the images in the dataset did not satisfy this condition. So, instead of removing all the images less than 32x32 pixels, the dimensions of all the images in the dataset were converted to 75x75 pixels for uniformity. This not only decreased the processing time significantly but would also result in higher accuracies down the line.

2) Blurring: One of the implicit goals of the project, apart from finding the fastest & most accurate TL algorithm, was to emulate the different weather conditions that may affect the input image quality, and then comparing the speed/accuracy of each algorithm. Different weather conditions can affect the sharpness of the image taken by the camera fitted in the vehicle. This effect was reproduced in this project by blurring all the images across the 43 traffic sign classes. This was achieved using the image manipulation libraries PIL ("Python Image Library") and "OpenCV" in Python. The blurring was implemented using the "Gaussian Blur" function inside "OpenCV".

The intensity of the blurring was also varied in three categories: (i) 3x3 blurring; (ii) 5x5 blurring; & (iii) 7x7 blurring. All these blurring categories smoothed the images to various degrees – 3x3 resulted in the least smoothing, similar to cases when light sources are directly in the sight of the vehicle camera, while 7x7 emulated the severe smoothing that may happen during stormy conditions. Of course, real life conditions will vary significantly depending upon the quality of the camera used, the number of cameras used, and the lighting conditions. But Gaussian smoothing can be used to get a ballpark estimate of the accuracy of the different TL algorithms.

Also, this blurring blew up the dataset. Originally, there were $\sim 50,000$ images to work with – 75% training ($\sim 37,500$) and 25% testing ($\sim 12,500$). But after smoothing the test images using three different blurring intensities, the final dataset expanded to: $\sim 37,500$ (training) $+$ $12,500$ (testing)$+ \sim 12,500 \times 3$ (blurred) $= \sim 87,500$. This initially created a storage problem that was rectified by again transforming the blurred images into NumPy arrays and storing them. The data mining techniques were applied on these NumPy arrays to avoid wastage of time and resources as introduced in subsection III-D & detailed in section IV.

D. Data Mining Techniques:

First, a simple, self-designed CNN was used to check all the steps implemented in the process flow, and to set the expectations for the resulting accuracy.

Next, three TL algorithms were applied using the TensorFlow-Keras libraries. The algorithms were: (i) InceptionV3; (ii) VGG16; & (iii) VGG19. Each algorithm used is explained in detail in section IV.

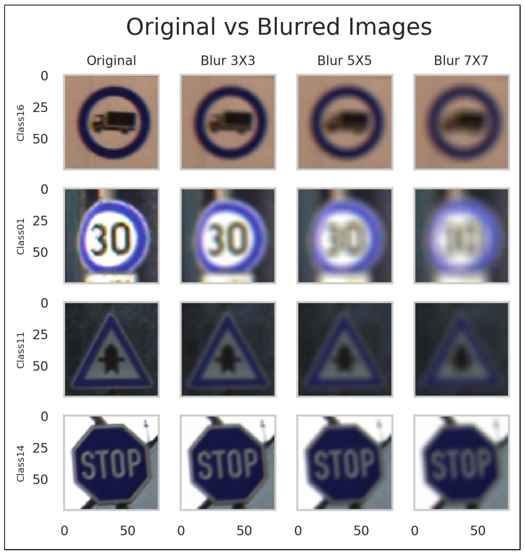


Fig. 3. Original vs Blurred Images

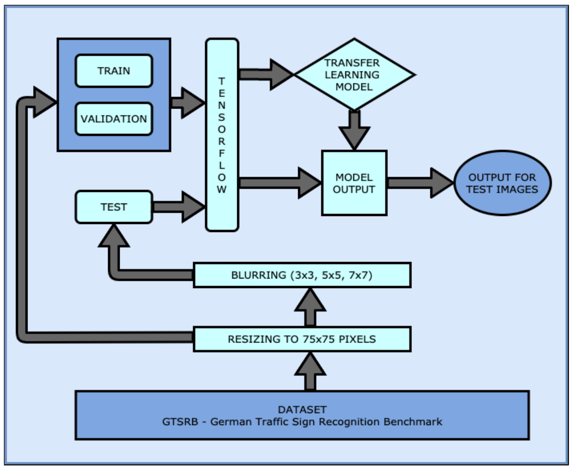


Fig. 4. Project Design Specification Diagram

E. Evaluation of Results:

The training, validation & testing accuracies, and the time taken by each of the algorithms, were noted. The results from the TL algorithms were then compared to the simplified CNN model applied in the beginning. The comparisons were then used to draw conclusions and suggest further work. This step is expanded upon in sections V & VI.

IV. Project Outline

Any neural network takes input and processes it through a number of hidden layers that are made up of a set of neurons. These neurons have a connection with all the neurons from the previous layer and perform functionalities independent of other neurons. The output of the final hidden layer is connected to the classifier layer which represents the score for each category. The class with the maximum score is considered as the output class for the next input.

Following are the hidden layers of CNN with their functionalities:

A. *Convolutional Layer:*

This layer is considered as the fundamental layer which does the intensive processing. A filter is an important parameter of convolution layer, as it convolves through the dimensions. While convolving, it performs the dot product between input positions and entries of the filter. The matrix of the resulting dot product is the output of the convolution layer.

For example, the input image considered in this project has dimensions of 75x75x3. If a filter with 2x2 size and a 2-stride is used, each neuron will have the weight of 2x2x3 region of the input volume, which calculates to total 2*2*3 + 1 filter = 13 weights.

B. *Pooling Layer:*

This layer is added periodically between convolution layers, to reduce the spatial size of representation which leads to the reduction in computation requirement. For example, an input of 224x224x3 requires memory of 224*224*64 = 3.2MB. If 2X2 filter is applied, the output of 112x112x64 is generated which requires 112*112*64 = 800KB memory, i.e. memory requirement is reduced by 75%. This also reduces the computation time for further layers.

C. *Flattening Layer:*

This layer acts as a connection between Feature Layer and Classifier Layer. It converts the multidimensional input to a unidirectional output. For example, an input of (None)x224x3 is converted to (None)x48.

D. *CNN Model:*

A CNN model is created using 3 convolution layers. Images of 75x75x3 dimension are fed as input to the first layer. The first convolution layer has 32 filters of 5x5, which perform operations to give an output of 75x75x32 dimensions. This output is then fed as an input to the second convolution layer which has 64 filters of 3x3 dimensions. The next layer is 'Max Pooling' layer, having 2x2 filters, which reduces the dimensions to 38x38x64. This output is then fed to the 'Dropout' layer, followed by convolution and 'Max Pooling' layer. A 'Flatten' layer is used to add classifier 'Dense' layers to the model. The first 'Dense' layer has 256 features which are then converted to 43 features in the second layer because the total number of required categories are 43. The activation functions used in the first and second 'Dense' layers are 'ReLU' and 'Softmax' respectively.

E. *TL Models:*

TL is a technique where knowledge acquired from one task is used for the prediction of another task. The layers of these models are mainly classified into, 'feature' layer and 'classification' layer. A feature layer, is imported, which was already created by the pre-trained model on a massive dataset. The classifier layer is added after the feature layer to convert its output to the desired number of classes. Following are the models that are used in this project.

1) InceptionV3: InceptionV3 is a convolutional neural network that was started in 2015 by GoogLeNet as a module and used for image recognition and object detection. It is known as the third version of Google's Inception convolutional neural network. It is 48 layers deep. The combination of 1x1 Convolutional layer, 3x3 Convolutional layer, 5x5 Convolutional layer is known as the 'Inception' layer. There are some add-ons to this 'Inception' layer which comprises of 1x1 convolutional layer for dimensionality reduction and a max-pooling layer. It is more computationally efficient than other TL models in terms of the number of parameters generated as well as financial expenses incurred by the network.

2) VGG16: Visual Geometry Group (VGG16) also known as OxfordNet is a convolutional neural network architecture that was developed in Oxford, 2014 to win the ILSVR (ImageNet) competition. It focuses on having convolution layers of a 3x3 filter with stride 1 rather than having a huge quantity of hyper-parameter. The layer of 2x2 filter of stride 2 remains constant always for padding and max-pooling. During the complete architecture, the arrangement of these layers is followed by it consistently. In VGG16, 16 stands for 16 layers with weights including 2 fully connected layers, and Softmax is used for the output, and hence the network is massive with 138 million parameters approximately.

3) VGG19: VGG19 is a trained convolutional neural network. In 2012 for the ImageNet Challenge competition, AlexNet provided the results. After that, the next big invention was provided by the VGG architecture, which had 16 layers & 3 fully-connected layers, i.e., 19 layers in total – hence the name VGG19. As input, the network takes 224x224x3 RGB image. In VGG19, all the convolutional layers use 3x3 filters, and the number of filters increases in powers of two – 64, 128, 256, 512 – and has 5 sets of convolutional layers with a stride length of 1 (pixel) and padding of 1 (pixel) on each side. In between each set of convolutional layers, there are max-pooling layers with 2x2 filters and a stride of 2 (pixels). ReLU is used for activating all the layers. The Softmax layer uses cross-entropy loss and makes the final decision.

V. PROJECT RESULTS & EVALUATION

A total of four models were compared in this project report – the first one was the simple, self-designed CNN, and the rest of the three models were pre-trained TL algorithms, namely InceptionV3, VGG16, & VGG19. Comparing these models produced some interesting results that are summarised in figures 5, 6, 7 & 8.

All the models were trained on the GTSRB dataset and then the different classes were predicted; the resulting test-accuracies were noted and plotted in figure 5.

As can be seen, the self-designed CNN performed better than all the other models across all the original and blurred images. The highest accuracy for the CNN model was 97.71% for the original images, and the lowest was 63.62% for the

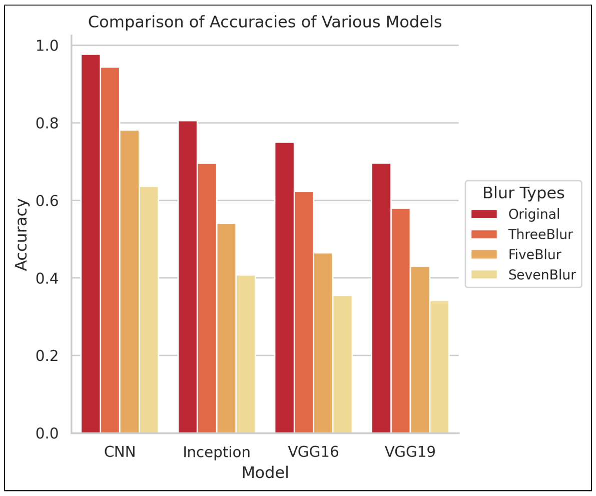


Fig. 5. Accuracy Comparison

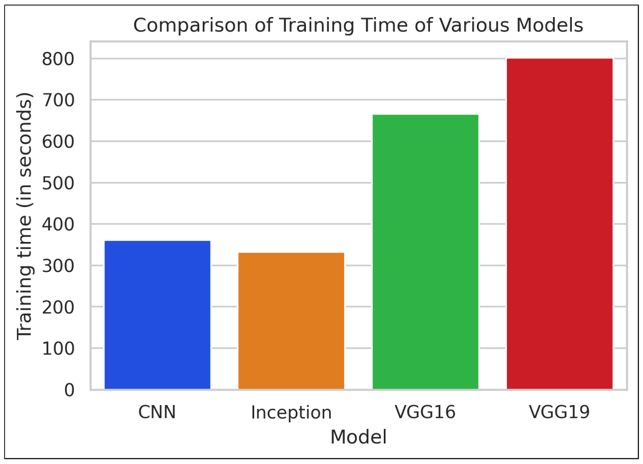


Fig. 6. Training Time Comparison

7x7 blurred images. This is extremely remarkable considering the fact that the 7x7 blurred images are so smoothed that even a human eye can be fooled easily. But even in such extreme cases of smoothing, the CNN model scored a highly respectable 63.62% accuracy.

Next in the accuracy hierarchy is the InceptionV3 model, scoring 80.59% for the original images and 40.79% for the 7x7 blurred images. Although this is a respectable degree of accuracy, it still cannot compete with the CNN model in terms of sheer precision of classification of the predicted classes.

The VGG16 and VGG19 models come next, scoring 75.02% & 69.66% for the original images, and 35.47% & 34.15% for the 7x7 blurred images respectively. These models are comparable in terms of accuracies across all the blurring types. But they are completely blown out of the park by the preceding CNN and InceptionV3 models.

Another deciding criteria for the choice of a model is the time that it takes to train that model. Figure 6 depicts the training times for each of the implemented models. The best performing model, in terms of training time, is the InceptionV3

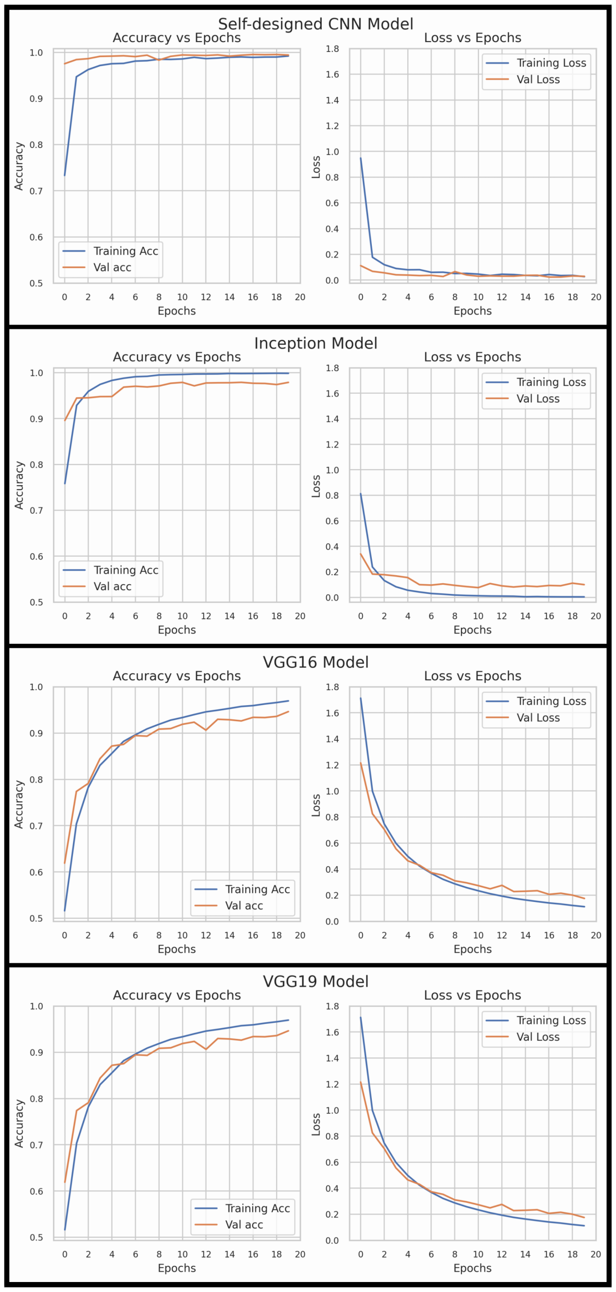


Fig. 7. Accuracy and Loss comparison of Self-designed CNN, InceptionV3, VGG16 & VGG19 models

model with 332.37 seconds, followed very closely by the CNN model with 361.44 seconds. The VGG models perform relatively bad in terms of training times too, with VGG16 managing 665.41 seconds and VGG19 finishing at a crawling speed with 801.50 seconds. This clearly shows that the best performing models not only manage to predict the classes with a respectable accuracy score, but also manage to not overwhelm the system by training for large periods of time.

One interesting comparison of the models is shown in figure 7 where the accuracy and loss for each epoch of model training is plotted. The CNN model manages to achieve the maximum

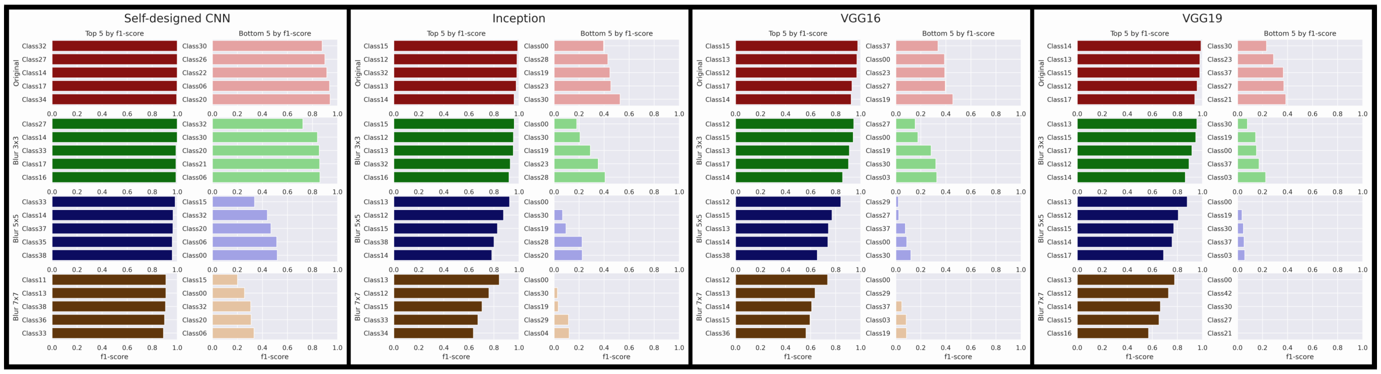


Fig. 8. Top/Bottom five Class-wise f1-score comparison of Self-designed CNN, InceptionV3, VGG16 & VGG19 models

accuracies and minimum loss in about 8 epochs, which means that it can give similar results in even less amount of time. The InceptionV3 model also manages to flatten at maximum accuracies and minimum loss in about 10 epochs. The VGG models, on the other hand, don't flatten out even at 20 epochs, which goes on to show their lack of efficiency in this project.

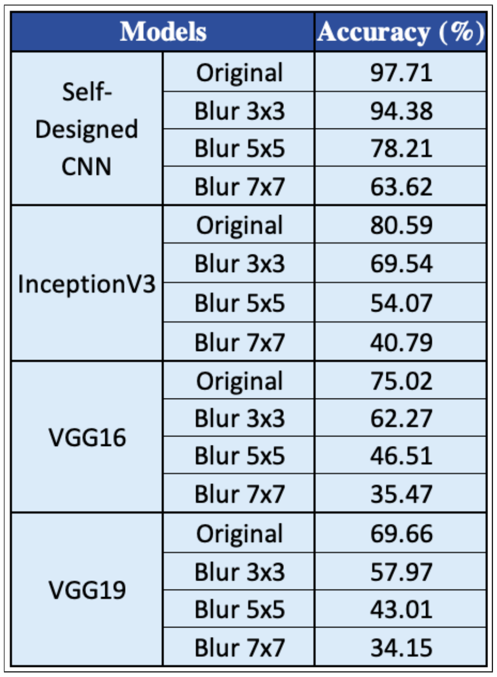
Finally, a class-wise comparison of the f1-scores is shown in figure 8, where the darker colours depict the top-5 classes in each blurring category, and the lighter colours represent the bottom-5 classes in each blurring category. The highest and lowest f1-scores for each class, and for each blurring category, are plotted for all the four models implemented in this project. Unsurprisingly, the CNN model has all its top-5 f1-scores more than 80%, with some even more than 99% for the original image category. The bottom-5, for the CNN model, have the lowest f1-scores, as expected, in the 7x7 blurring category. A similar trend is shown by the TL models as well, with the original image category scoring more than 95%, and the 7x7 blurring category dropping lower and lower with each TL model.

VI. CONCLUSIONS & FUTURE WORK

The GTSRB dataset was downloaded, pre-processed, transformed and then used to train a self-designed CNN model, and then compared to three pre-trained TL models, namely InceptionV3, VGG16 & VGG19. These models were challenged and tested by using blurred images to emulate the bad weather conditions encountered by a vehicle on road. The results of all the models were noted and evaluated, and then comparison was done on the basis of those evaluations.

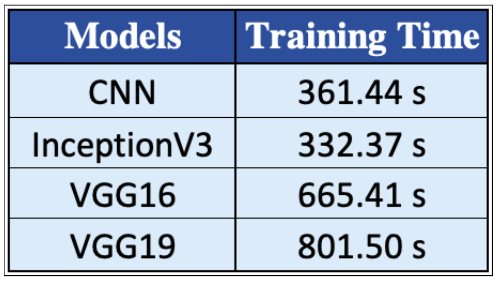
As tabulated in figures 9 & 10, the self-designed CNN model was the best in achieving the intended goal, with 97.71% (best) accuracy and 361.44 seconds of training time. It is followed closely by the InceptionV3 model (best accuracy: 80.59%; training time: 332.37s), then the VGG16 model (best accuracy: 75.02%; training time: 665.41s) and the VGG19 model (best accuracy: 69.66%; training time: 801.50s). Thus, the self-designed CNN model was able to beat the well-established TL models in accurate and timely detection and classification of the traffic signs from the GTSRB dataset.

This whole exercise proved that using a many-layered, ultra-complex neural network does not guarantee a good result, & a simple straightforward CNN model can sometimes produce better results.



Models		Accuracy (%)
Self-Designed CNN	Original	97.71
	Blur 3x3	94.38
	Blur 5x5	78.21
	Blur 7x7	63.62
InceptionV3	Original	80.59
	Blur 3x3	69.54
	Blur 5x5	54.07
	Blur 7x7	40.79
VGG16	Original	75.02
	Blur 3x3	62.27
	Blur 5x5	46.51
	Blur 7x7	35.47
VGG19	Original	69.66
	Blur 3x3	57.97
	Blur 5x5	43.01
	Blur 7x7	34.15

Fig. 9. Accuracies



Models	Training Time
CNN	361.44 s
InceptionV3	332.37 s
VGG16	665.41 s
VGG19	801.50 s

Fig. 10. Training Times

Since deep learning and TL are ever evolving topics, coupled with the constant updates in the self-driving and traffic detection automation fields, much progress can be done in future iterations of this project:

- The GTSRB dataset contained $\sim 50,000$ images, further compounded by the blurring techniques applied and resulting in a massive $\sim 87,500$ image dataset. But by comparison with the massive datasets such as ImageNet, a lot more images can be added in the input data to increase the training materials for the models. One way to achieve this could be augmentation of images, i.e., shifting and/or rotating the images to increase diversity.
- Although the comparison tests done in this project were under controlled circumstances, some other variables could be constricted, in further instalments, to drive the point home. For instance, the code of this project was compiled on *Google Colab* notebooks because of the lack of GPU-enabled machines, which resulted in the addition of a lesser known degree of freedom to the equation – since the GPU was allotted by the *Google Cloud* system automatically, and because there are many varieties of GPUs used by *Google*, the hardware used to train and test the models may have differed at some step of the project. Although this will not affect the results massively, making sure that the same GPU is allotted every time the model is implemented may help train the models faster by a few seconds. But nonetheless, this can be overlooked if the code is compiled on physical hardware that is same for all the models.
- A relatively simple automated system can be designed in future iterations of the project. The best model can be selected and then tested by attaching a low-quality camera that live-feeds the images of traffic signs and symbols into the automated system. This would introduce real-life variables in the stew, and will help shed light on the challenges to fully automated self-driving.

References

[1] Mykola, "Gtsrb - german traffic sign recognition benchmark," kaggle.com, 11 2018. [Online]. Available: https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

[2] F. M. Johner and J. Wassner, "Efficient evolutionary architecture search for cnn optimization on gtsrb," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 56–61.

[3] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[4] S. B. Maind, P. Wankar *et al.*, "Research paper on basic of artificial neural network," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 1, pp. 96–100, 2014.

[5] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[6] A. Abd Almisreb, N. Jamil, and N. M. Din, "Utilizing alexnet deep transfer learning for ear recognition," in *2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP)*. IEEE, 2018, pp. 1–5.

[7] S. I. Rashid, M. A. Islam, and M. A. M. Hasan, "Traffic sign recognition by integrating convolutional neural network and support vector machine," in *2019 International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2)*. IEEE, 2019, pp. 1–6.

[8] C. Alippi, S. Disabato, and M. Roveri, "Moving convolutional neural networks to embedded systems: the alexnet and vgg-16 case," in *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2018, pp. 212–223.

[9] R. Qian, Y. Yue, F. Coenen, and B. Zhang, "Traffic sign recognition with convolutional neural network based on max pooling positions," in *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2016, pp. 578–582.

[10] A. Arcos-Garcia, J. A. Alvarez-Garcia, and L. M. Soria-Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," *Neural Networks*, vol. 99, pp. 158–165, 2018.

[11] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep cnn denoiser prior for image restoration," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3929–3938.

[12] Z. Zuo, K. Yu, Q. Zhou, X. Wang, and T. Li, "Traffic signs detection based on faster r-cnn," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2017, pp. 286–288.

[13] A. Arcos-Garcia, J. A. Alvarez-Garcia, and L. M. Soria-Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," *Neural Networks*, vol. 99, pp. 158–165, 2018.

[14] C. Lee, H. J. Kim, and K. W. Oh, "Comparison of faster r-cnn models for object detection," in *2016 16th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2016, pp. 107–110.

[15] D. Yasmina, R. Karima, and A. Ouahiba, "Traffic signs recognition with deep learning," in *2018 International Conference on Applied Smart Systems (ICASS)*. IEEE, 2018, pp. 1–5.

[16] L. Wei, L. Runge, and L. Xiaolei, "Traffic sign detection and recognition via transfer learning," in *2018 Chinese Control And Decision Conference (CCDC)*. IEEE, 2018, pp. 5884–5887.

[17] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, pp. 211–252, 04 2015. [Online]. Available: https://hci.stanford.edu/publications/2015/scenegraphs/imagenet-challenge.pdf

[19] H. Qassim, A. Verma, and D. Feinzimer, "Compressed residual-vgg16 cnn model for big data places image recognition," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2018, pp. 169–175.

[20] J. P. N. Acilo, A. G. S. D. Cruz, M. K. L. Kaw, M. D. Mabanta, V. G. G. Pineda, and E. A. Roxas, "Traffic sign integrity analysis using deep learning," in *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*. IEEE, 2018, pp. 107–112.

[21] D. N. T. How, K. S. M. Sahari, Y. C. Hou, and O. G. S. Basubeit, "Recognizing malaysia traffic signs with pre-trained deep convolutional neural networks," in *2019 4th International Conference on Control, Robotics and Cybernetics (CRC)*. IEEE, 2019, pp. 109–113.

[22] C. Lin, L. Li, W. Luo, K. C. Wang, and J. Guo, "Transfer learning based traffic sign recognition using inception-v3 model," *Periodica Polytechnica Transportation Engineering*, vol. 47, no. 3, pp. 242–250, 2019.

[23] G. Sapijaszko and W. B. Mikhael, "An overview of recent convolutional neural network algorithms for image recognition," in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2018, pp. 743–746.

[24] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth *et al.*, "Knowledge discovery and data mining: Towards a unifying framework." in *KDD*, vol. 96, 1996, pp. 82–88.

[25] A. I. R. L. Azevedo and M. F. Santos, "Kdd, semma and crisp-dm: a parallel overview," *IADS-DM*, 2008.