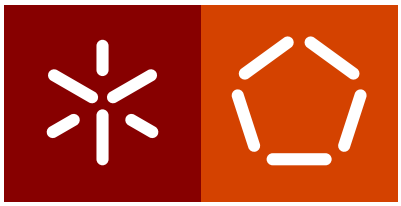
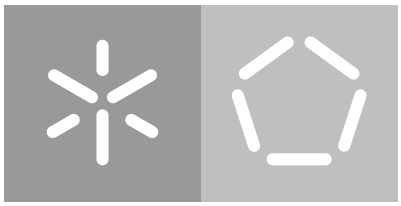


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Diogo Lopes da Silva

Synthetic Data Approach for Traffic Sign Recognition

December 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Diogo Lopes da Silva

Synthetic Data Approach for Traffic Sign Recognition

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
António Ramires Fernandes

December 2019

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

To my beloved mother, *in memoriam*.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to my supervisor, Professor António Ramires Fernandes, for his guidance, availability, dedication, and continuous support. I would not have accomplished this work without his valuable suggestions and truly remarkable help.

I owe my profound gratitude to my mother, Avelina, for her affection, unconditional support, and invaluable lessons that allowed me to reach this far, making me the person I am today; and for which I am forever indebted.

I am very grateful to my friends for their support and fellowship throughout my life and education, providing me moments that allowed me to stay focused in order to complete this dissertation.

I would like to thank my girlfriend, Flávia, for her understanding and advice in the most difficult times.

Last but not least, I am thankful for my family, especially my sister, Mariana, for her ceaseless encouragement.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Currently, *Advanced Driver Assistance Systems (ADAS)* have been gradually increasing their presence in everyday life, thanks in part to its ability to recognize several distinct types of objects in the road, namely, traffic signs. These systems employ *Convolutional Neural Networks (CNNs)*, a type of classification algorithms that relies on an enormous amount of data in order to be effective. Current traffic sign datasets suffer from a scarcity of samples due to the necessity of compiling and labeling them manually. Such task is highly resource and time consuming. Thus, researches resort to other mechanisms to deal with this problem, such as increasing the architectural complexity of the neural networks or performing data augmentation.

This work addresses the data shortage issue by exploring the feasibility of developing a synthetic dataset. Such set would not require gathering and labelling manually thousands of real word traffic sign images, requiring only easily collectable information and no human intervention.

The only data required is a set of templates for each sign given that a particular sign may have more than one template. This is required to cope with outdated pictograms that are still present in streets and roads.

We apply several colour and geometric processing methods to the templates aiming to achieve a look similar to real signs, from the CNN point of view. One of such methods is the usage of Perlin noise to both simulate shadows and avoid the clean and homogeneous look that templates have.

Two use cases for synthetic data usage are presented: considering the synthetic dataset as a standalone training set, and merging synthetic data with real samples when real data is available. The first option provided results that not only clearly surpass any previous attempt on using synthetic data for traffic sign recognition, but are also encouragingly placing the accuracies obtained close to state-of-the-art results, with much simpler networks. The second approach provided results on three distinct test datasets that consistently beat state-of-the-art results, either in accuracy or in simplicity of the network.

Keywords: Synthetic data, Traffic sign recognition, Convolutional neural networks, European traffic signs

RESUMO

Atualmente, Sistemas Avançados de Assistência ao Condutor têm vindo a aumentar gradualmente a sua presença no quotidiano graças, em parte, à sua capacidade de reconhecer vários objetos distintos na estrada, nomeadamente, sinais de trânsito. Estes sistemas empregam Redes Neuronais Convolucionais (CNNs), um tipo de algoritmos de classificação que dependem de uma enorme quantidade de dados de forma a serem eficientes.

Os conjuntos de dados de sinais de trânsito atuais sofrem de escassez de amostras devido à necessidade de as compilar e rotular manualmente. Tal tarefa consome imenso tempo e recursos. Por conseguinte, investigadores recorrem a outros mecanismos para serem capazes de lidar com esse problema, tais como, aumentar a complexidade arquitetural das redes neuronais ou efetuar *data augmentation*.

Desta forma, este trabalho aborda a questão da escassez de dados, explorando a viabilidade do desenvolvimento de um conjunto de dados sintéticos. Tal conjunto não exigiria recolher e rotular manualmente milhares de imagens de sinais de trânsito, necessitando apenas de informação facilmente recolhida sem intervenção humana.

Os únicos dados necessários são um conjunto de modelos para cada sinal uma vez que um sinal particular pode apresentar mais que um modelo. Tal é necessário para lidar com pictogramas desatualizados que ainda se encontram nas ruas e estradas.

Aplicamos vários métodos de processamento de cor e geometria aos templates visando obter uma aparência semelhante a sinais reais, do ponto de vista da CNN. Um desses métodos é a utilização do ruído de Perlin para simular sombras e evitar a aparência limpa e homogênea que os modelos apresentam.

Dois casos de uso com dados sintéticos são apresentados: considerar o conjunto de dados sintético como um conjunto de treino independente, e unir dados sintéticos com amostras reais sempre que estas estiverem disponíveis. A primeira opção forneceu resultados que, não apenas superam claramente qualquer tentativa anterior de usar dados sintéticos para reconhecimento de sinais de trânsito, como também colocam as precisões obtidas próximas dos resultados do estado da arte, com redes muito mais simples. A segunda abordagem forneceu resultados em três conjuntos de dados de teste distintos que superam consistentemente os resultados do estado da arte, tanto na precisão quanto na simplicidade da rede.

Palavras-Chave: Dados sintéticos, Reconhecimento de sinais de trânsito, Redes neuronais convolucionais, Sinais de trânsito europeus

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

A

ADAS Advanced Driver Assistance Systems.

B

BTSC Belgium Traffic Sign Classification.

C

CNN Convolutional Neural Network.

D

DITS Data set of Italian Traffic Signs.

DNN Deep Neural Network.

E

ELU Exponential Linear Unit.

G

GPU Graphical Processing Units.

GTSRB German Traffic Sign Recognition Benchmark.

H

HOG Histogram of Oriented Gradients.

M

MCDNN Multi-column Deep Neural Network.

P

PCB Printed Circuit Board.

PRELU Parametric Rectied Linear Unit.

R

RELU Rectified Linear Unit.

RMASTIF revised Mapping and Assessing the State of Traffic InFrastructure.

ROI Region of Interest.

S

SGD Stochastic Gradient Descent.

SOFTMAX Smooth approximation of maximum.

STN Spatial Transformer Network.

STS Swedish Traffic Signs.

SVG Scalable Vector Graphics.

SVM Support Vector Machine.

Y

YOLO You Only Look Once.

1

INTRODUCTION

It is of crucial importance for autonomous vehicles to be able to recognize and classify vehicles, pedestrians, road marks, and traffic signs among other objects in the public highway accurately, as it is imperative to ensure the safety of all road traffic intervenients.

In this work we address the recognition of traffic signs. This chapter aims to introduce the domain of traffic sign recognition and its motivation, with particular focus on the training sets.

1.1 CONTEXTUALIZATION

Due to *Graphical Processing Units (GPU)* advancements in recent years, previous techniques used to classify traffic signs such as traditional computer vision methods have been replaced by deep learning classifiers. In particular, the domain of traffic sign recognition resorts mainly to CNN based architectures.

Typically, to be able to train CNNs, in the context of traffic sign recognition, a high volume of samples is necessary. However, the number of samples is not the only criterion to define a training set. The environmental diversity, the time of day, atmospheric conditions, and camera viewpoints, are all factors that influence the ability of a neural network to properly recognize traffic signs.

In this context, collecting a truly representative dataset becomes a time and resource consuming task. For instance, traffic signs that exist only in highways require long journeys to get collected. Other traffic signs only appear in rural areas. And, this is only considering the location influence. When all factors are taken into account, the process of building a truly representative dataset of real images is not to be taken lightly.

Furthermore, all tests reported on the existing datasets assume perfect scenarios, i.e. that the network will only be asked to classify an image containing a traffic sign that belongs to one of the classes that are included in the training set. This is far from a real scenario where a neural network may be asked to classify images that are not even traffic signs.

Considering only traffic signs, unless the training set contains a large set of classes, the CNN is most likely to perform poorly on a real scenario where a multitude of different traffic signs are present.

To aggravate things further, it must be taken into account that a training set build with collected signs from a country will not be optimal when used to train a network to classify traffic signs from a different country. Small differences in letter fonts and pictograms are sufficient to decrease significantly the accuracy of a CNN.

1.2 Motivation

As mentioned before, collecting real traffic sign images can be an expensive and time consuming task. Furthermore, the datasets should not be reused across different countries at the risk of a poor accuracy.

The motivation of this work derives from these issues. The solutions found so far address the accuracy problem from the network perspective. Complex architectures, including STNs and Inception modules, have been able to achieve considerable accuracies when asked to classify a sign from a class that is included in the training set.

What is proposed in this work is to address the issues pertaining to the datasets. The approach is to explore the construction of a dataset based only on synthetic data, thereby eliminating the above mentioned issues. The number of classes is no longer a limitation, and neither is it necessary to collect real images of traffic signs, some of which may require long journeys to be able to gather a significant number of samples. Furthermore, conceptually, it makes sense to consider atmospheric conditions, time of day, and craft multiple scenarios that are hard to encounter in real life in meaningful numbers, sufficient to train CNNs.

When real data is already available, synthetic data can be used to complement the training set, providing a hard to obtain diversity when considering only real samples, thereby potentially increasing the overall classification accuracy.

1.3 Aim of the work

The goal of this work is to explore the assembly of a synthetic traffic sign image dataset suitable for training a CNN for real traffic sign recognition.

The only input required must be the set of templates that define the classes of the traffic signs included in such a training set. Most of these can be collected from a number of sources in the internet. Note that for a particular class, there may be more than one template. This is because the same class may have different pictograms displayed in the streets. This occurs with the introduction of new pictograms, or text fonts for particular classes. Old

signs are not removed so several versions may coexist. In order to maximise the probability of having high accuracy for both new and old versions, all versions should be part of the template set.

For each template, a background is applied. Once again this can be a synthetic or a real scenario. Afterwards, several color and geometric modifications are applied to the resulting image. Perlin noise is also used with a dual purpose: on the one hand it allows the simulation of shadows from tree leaves, and, in the other, it simulates the wear of materials exposed to an open environment.

To compare the potential of such a dataset, tests are performed against state-of-the-art reports on existing European traffic sign repositories, considering two usage scenarios: the synthetic dataset as a standalone training set, or merged with a real-world training set.

1.4 DOCUMENT STRUCTURE

The remaining of the dissertation is divided into the following chapters:

- Chapter 2 reviews previous studies regarding existing traffic sign datasets, widely used data augmentation techniques, and some of the employed synthetization approaches for data generation, not directed exclusively at traffic signs;
- Chapter 3 explores the different included approaches in the generation of synthetic traffic signs;
- Chapter 4 presents the contribution of each technique in the generating process assessing each individual technique as the final synthetic data generation pipeline in the above mentioned usage scenarios;
- Chapter 5 presents the conclusions regarding the usage of synthetic data in training sets, providing prospects for future work.

2

STATE OF THE ART

Over the past few years, several studies have been conducted on the subject of traffic sign classification involving datasets from various countries. As stated before, the adopted approach consists in the use of convolutional neural networks, a machine-learning based set of algorithms that has achieved fruitful results and have been widely adopted for both object detection and recognition, namely for traffic signs.

This chapter focuses on existing European traffic sign repositories and the CNN architectures reported in the research literature trained with these datasets.

Ultimately, it also reports on research relating to synthetic datasets.

2.1 TRAFFIC SIGN DATASETS

Only for a few countries samples of traffic signs have been collected and labelled, and released as a public dataset. The volume and quality of samples varies greatly across countries. This section will provide a brief overview of some of the largest European traffic sign classification datasets. For the purpose of this work, only classification data sets above 5000 samples are considered. However, smaller datasets do exist, such as the *Swedish Traffic Signs (STS)* dataset which has just a total of 3488 traffic signs obtained from a 20000-frame video (Larsson and Felsberg, 2011).

2.1.1 *Germany*

The GTSRB dataset is Europe's largest publicly available set of traffic signs, consisting of 51839 samples divided by 43 classes. An example of each class can be seen in Figure 1.

The traffic signs were extracted from a 1-second video sequences by Stallkamp et al. (2012). The entire dataset is divided between the training set, consisting of 39209 images, and the test set, consisting of 12630 images. The dataset samples are not necessarily square with resolutions ranging from 25×25 to 232×266 pixels with each sample containing approximately 10% of margin around the traffic sign. The distribution of the dataset is shown in Figure 2 with the red dashed line representing the mean sample count per class.

Figure 1.: Examples for each of the 43 classes of the German dataset. The classes are labeled from left-to-right and top-to-bottom.

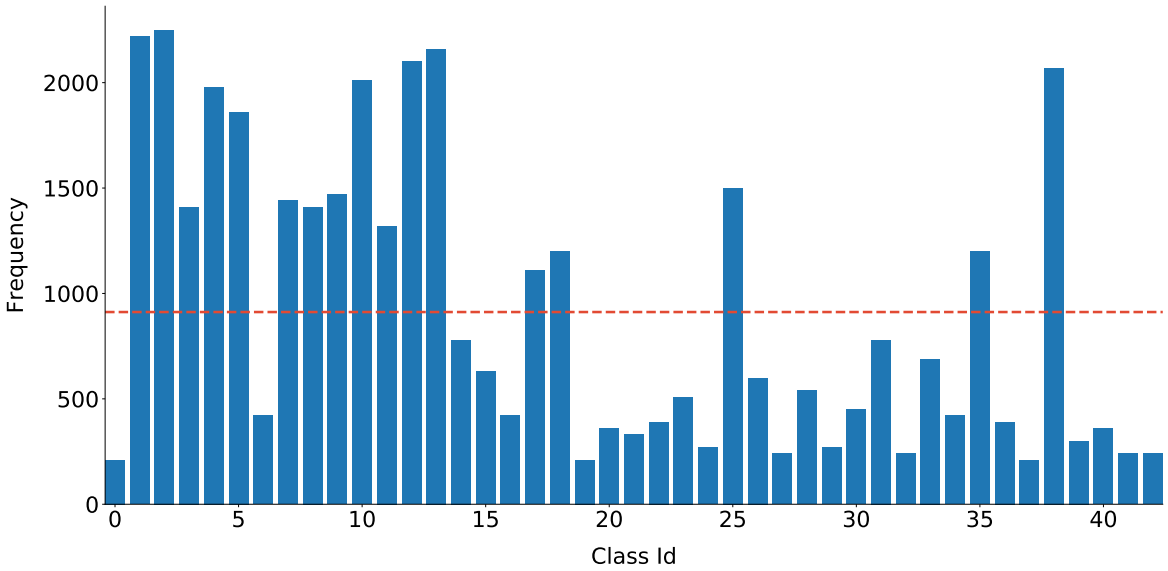


Figure 2.: Traffic sign samples frequency by class of the GTSRB dataset.

The dataset is clearly unbalanced, i.e. there are classes with a much higher sample count than others. This issue is recurrent in the other datasets covered in this section as well.

2.1.2 *Italy*

The DITS data set possesses a total of 9254 traffic signs distributed by 59 classes captured by Youssef et al. (2016) in challenging conditions such as night-time, fog conditions, and complex urban environments. An example of each class from the Italian data set is shown in Figure 3.

The data set is split in training set, consisting in 8048 samples, and test set, consisting in 1206 samples. From the 59 classes, 11 do not contain any sample in the test set. The samples resolution ranges from 19×20 to 321×348 pixels. The distribution of the data set is shown at Figure 4, also unbalanced like other datasets.

Figure 3.: Examples for each of the 59 classes of the Italian dataset. The classes are labeled from left-to-right and top-to-bottom.

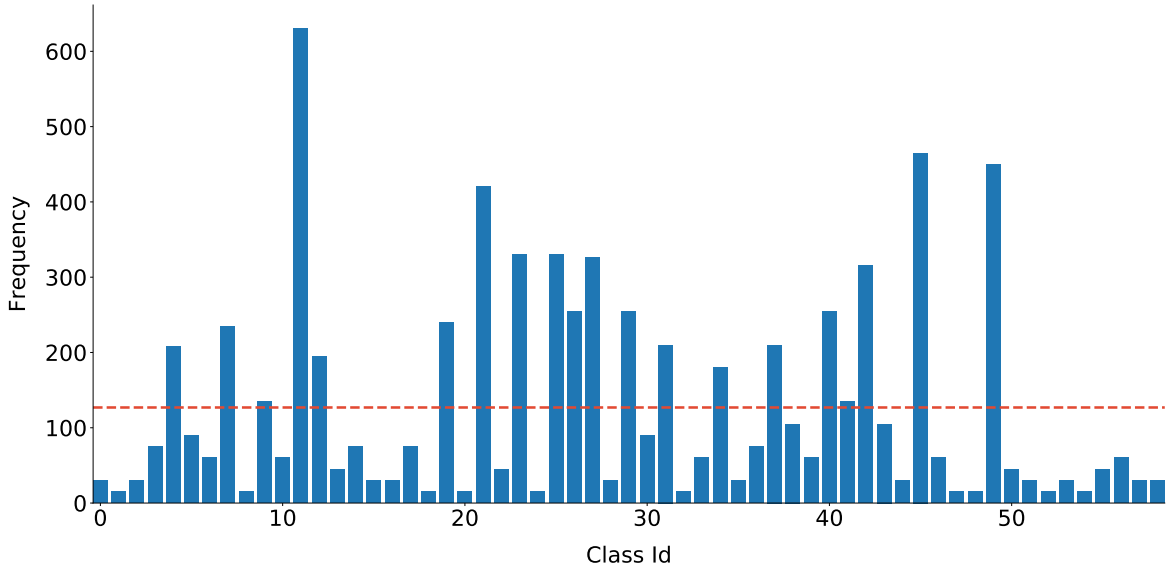


Figure 4.: Traffic signs samples frequency by class of the DITS dataset.

Figure 5.: Examples for each of the 62 classes of the Belgian data set. The classes are labeled from left-to-right and top-to-bottom.

2.1.3 *Belgium*

The BTSC dataset contains a total of 7095 traffic sign samples divided by 62 distinct classes that were collected and processed by Timofte et al. (2009). An example of each class from the Belgian dataset is shown in Figure 5.

The entire data set is split in training set, consisting in 4575 samples, and testing set, consisting in 2520 samples. The resolution of the samples varies, ranging from 22×21 to 674×527 pixels. In a similar fashion to the GTSRB, the Belgian data set samples contains a border of 10% around the traffic sign of, at least, 5 pixels. On average, the BTSC data set contains 3 samples per each traffic sign as seen by different viewpoints at 3 different time moments (Mathias et al., 2013). In order to improve diversity, the samples have been extracted from an average of 15 video tracks per class. However, some classes have more examples in the test set than in the training set, and 9 classes do not contain a single example in the test set. The distribution of the entire dataset is shown at Figure 5 where it is possible to observe that it is also unbalanced, similarly to all other data sets.

2.1.4 *Croatia*

The rMASTIF dataset has a total of 5828 traffic sign samples divided by 31 classes (Šegvic et al., 2010). An example of the dataset can be seen in Figure 7. Each class has at least 12 distinct tracks and each track has 4 examples. The data set is split in training and test sets, consisting in 4044 and 1784 samples, respectively. The samples resolution ranges from 17×16 to 185×159 pixels. Each sample has an additional margin of around 10% similarly to other datasets.

Once again, the data set distribution is not uniform as shown in Figure 8.

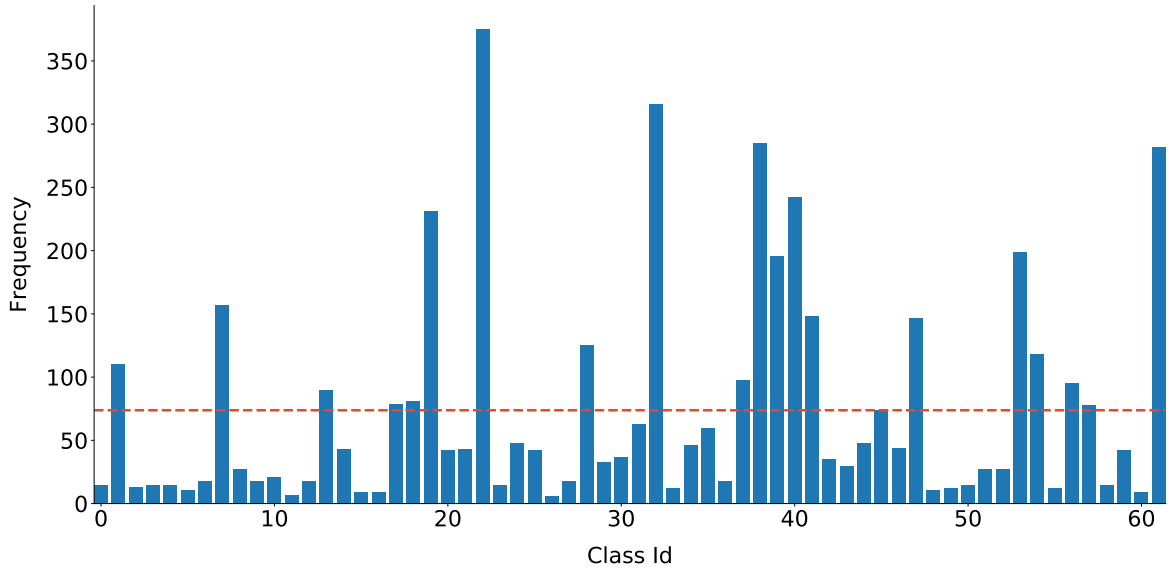


Figure 6.: Traffic signs samples frequency by class of the BTSC dataset.

Figure 7.: Examples for each of the 31 classes of the Croatian dataset. The classes are labeled from left-to-right and top-to-bottom.

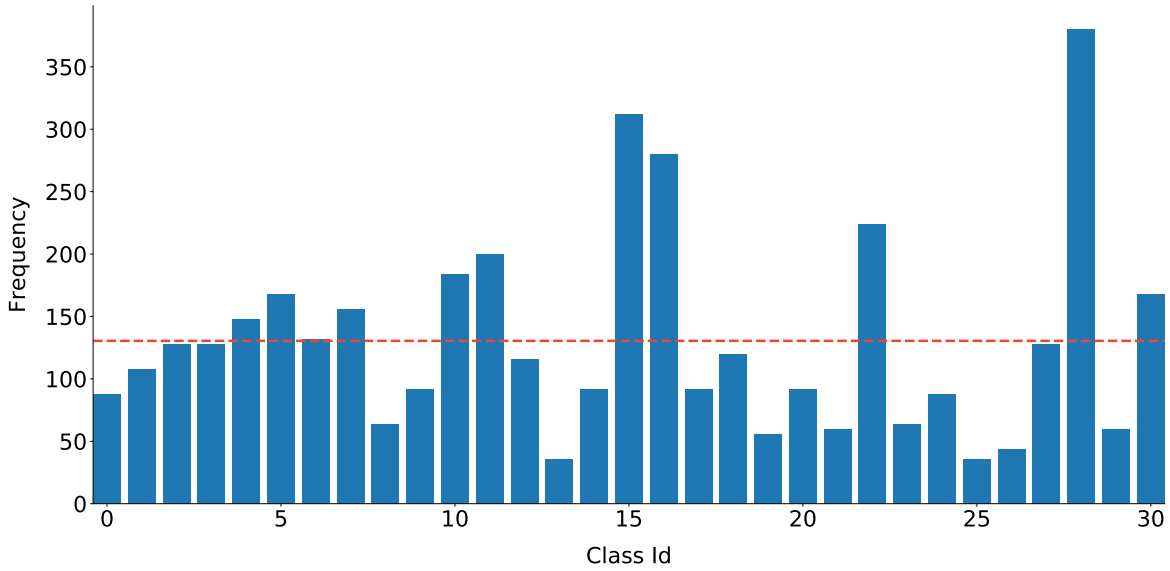


Figure 8.: Traffic signs samples frequency by class of the rMASTIF dataset.

Table 1.: Individual CNN architecture used for classification of the GTSRB dataset by Cireşan et al. (2012) with a total of 1.1 million parameters. Table reproduced from Stallkamp et al. (2012).

Layer	Type	Kernel	Output Shape	Parameters
0	Input		$48 \times 48 \times 3$	
1	Convolution	7×7	$42 \times 42 \times 100$	14800
2	Max Pooling	2×2	$21 \times 21 \times 100$	
3	Convolution	4×4	$18 \times 18 \times 150$	240150
4	Max Pooling	2×2	$9 \times 9 \times 150$	
5	Convolution	4×4	$6 \times 6 \times 250$	600250
6	Max Pooling	2×2	$3 \times 3 \times 250$	
7	Fully Connected		128	288128
8	Fully Connected		43	5547

2.2 TRAFFIC SIGN CLASSIFICATION

In 2011, Cireşan et al. (2012) won the the International Joint Conference on Neural Networks (IJCNN) competition held by Stallkamp et al. (2012), achieving, at the time, a state-of-the-art accuracy of 99.46% for the GTSRB dataset. They accomplished this result employing a *Deep Neural Network (DNN)* in a form of a committee of 25 CNNs, which they referred to as a *Multi-column Deep Neural Network (MCDNN)*. Each individual CNN consists in a succession of convolutional and max-pooling layers as shown in Table 1.

Their solution also includes dynamic preprocessing per epoch before feeding the samples to the classification CNNs. In the preprocessing phase, the images are cropped to the respective *Region of Interest (ROI)*, excluding its margins, resized to 48×48 pixels, and

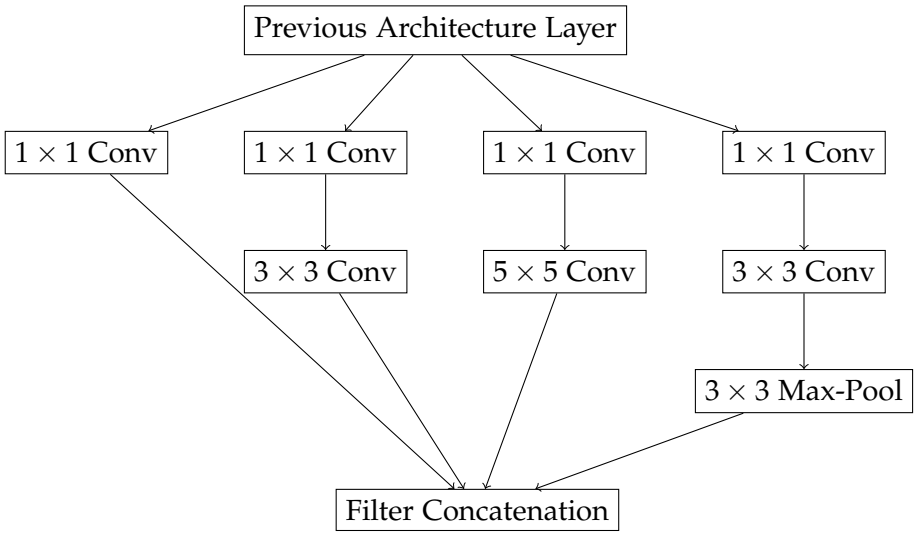


Figure 9.: Example of Inception module for traffic sign classification. Adapted from Haloi (2015).

randomly distorted in a stochastic manner, meaning that each sample suffers a distinct distortion each epoch. Finally, before training a CNN, the data also undergoes a normalization process which includes four image adjustment methods such as contrast stretching, histogram equalization, adaptative histogram, and contrast normalization. When the validation error reaches 0, the training process concludes for that CNN. The final predictions are obtained by averaging individual predictions of each CNN. Each CNN takes approximately 25 epochs to train.

In 2015, Haloi (2015) was able to achieve a state-of-the-art performance of 99.81% on the GTSRB dataset using a CNN consisting in sequential STN layers along with a modified version of GoogLeNet, which in turn is based on an Inception architecture (Szegedy et al., 2014).

That is, to be able to cope with traffic signs deformations such as blurring, translations, rotations, or skewing, the STN acts as a transformation invariant (Jaderberg et al., 2015). Another appealing property of spatial transformers is that it can be inserted into an existing CNN due to the fact the parameters of the transformations are also learned by back-propagation algorithms. In addition, in order to increase the robustness of the CNN, the modified version of GoogLeNet allows a more precise traffic sign classification even under deformations due to the fact the Inception layer allows its internal layers to choose which filter size considers more relevant to learn the required features. For instance, one of the Inception modules used can be seen in Figure 9. Thus, the Inception module, instead of having just a single convolution layer, has a composition of convolutional or max-pooling layers in width while the more traditional approach just employs convolutions in depth.

Hence, an Inception module can infer which internal layer is more beneficial for the architecture and global accuracy. After processing all the layers, the Inception module

Table 2.: Simplified version of the Haloi (2015) architecture for the GTSRB dataset classification with approximately 10.5 million parameters. The STN and Inception modules definitions have been omitted for illustration purposes.

Layer	Type	Kernel/Stride	Output Shape	Parameters $\times 10^3$
0	Input		$128 \times 128 \times 3$	
1	STN1			3000
2	Convolution	$5 \times 5/2$	$64 \times 64 \times 64$	1.6
3	Max Pooling	$3 \times 3/2$	$32 \times 32 \times 64$	
4	STN2			891
5	Convolution	$3 \times 3/1$	$32 \times 32 \times 192$	110
6	Max Pooling	$3 \times 3/2$	$16 \times 16 \times 192$	
7	STN3			1000
8	Inception		$16 \times 16 \times 288$	206
9	STN3			1000
10	Inception		$16 \times 16 \times 480$	436
11	Max Pooling	$3 \times 3/2$	$8 \times 8 \times 480$	
12	Inception		$8 \times 8 \times 512$	395
13	Inception		$8 \times 8 \times 512$	467
14	Inception		$8 \times 8 \times 512$	546
15	Inception		$8 \times 8 \times 528$	624
16	Inception		$8 \times 8 \times 832$	880
17	Max Pooling	$3 \times 3/2$	$4 \times 4 \times 832$	
18	Inception		$4 \times 4 \times 832$	965
19	Inception		$4 \times 4 \times 1024$	1200
20	Average Pooling	$4 \times 4/1$	$1 \times 1 \times 1024$	
21	Dropout		$1 \times 1 \times 1024$	
22	Fully Connected		43	44

concatenates the result and proceeds for the next layer. Due to the nature of this module, it increases the computational demands and increases the architectural complexity. However, it allows a much more flexible CNN architectural definition.

Regarding the training phase, all traffic signs were resized to 128×128 using cubic interpolation. As optimization algorithm, *Stochastic Gradient Descent (SGD)* with momentum was applied along with a batch size of 20 samples. The fully connected layer suffered a dropout of 40%, being the only dropout layer employed in the entire architecture. Finally, as for the activation function, they used *Parametric Rectied Linear Unit (PReLU)* instead of *Rectified Linear Unit (ReLU)*, as they noticed an increase in performance using PReLU.

The weights are not randomly initialized, the MSRA method was preferred which was proven useful for PReLU activation unit based networks (He et al., 2015). Their architecture had a total of 10.5 million parameters while Cireşan et al. (2012) had approximately 90 million parameters. Thus, reducing the computational cost. An adapted and simplified version is shown in Table 2.

In 2015, Jurišić et al. (2015) achieved an accuracy of $98.17 \pm 0.22\%$ on the BTSC dataset and $99.53 \pm 0.10\%$ on the rMASTIF dataset using the same CNN architecture, which they named OneCNN. Their model was based on the CNN used by Cireşan et al. (2012). They included width branches (also named scales) of convolutional layers in order to create a multi-scale architecture in a similar approach to the work of Sermanet and LeCun (2011). A simplified representation of the OneCNN architecture is shown in Figure 10.

After each convolution layer, a fully connected layer is added. Later, the output of each scale is concatenated and combined into a fully connected layer. Their architecture aimed to avoid the use of multiple networks in a committee or in an ensemble manner while approaching state-of-the-art results with a single CNN. As activation function, ReLU was applied after the fully connected layers. Finally, SGD was used as optimization algorithm and *Smooth approximation of maximum (Softmax)* as loss function. The training set suffered data augmentation in form of pixel intensity histogram equalization. For each sample, a padding of 10% was added on top of the annotated ROI and resized to 53×53 pixels. Later, a random crop of 48×48 pixels was applied to each training sample.

In 2016, with the publication of the DITS dataset, Youssef et al. (2016) achieved an accuracy of 97.20% using a simple architecture consisting in only 2 convolutional layers followed by pooling layers. Similarly, only 1 fully connected layer is used alongside Softmax. The samples are resized to 28×28 pixels. Data augmentation is also used in form of jittering and 3 geometric operations such as rotations, scales and translations. The maximum values for rotations is $[15, 15]$, for translations is $[-4, 4]$, and as for scaling a factor of $[0.8, 1.2]$ is used. The augmented data set is 4 times larger than the original.

In 2017, Arcos-García et al. (2018) accomplished a performance of 99.71% on the GTSRB dataset and 98.87% on the BTSC dataset. Their architecture consists in a single CNN that combines convolutional layers and 3 STNs modules. Similarly to the previous work of (Cireşan et al., 2012), these layers perform explicit geometric transformations on feature maps, removing progressively background detail and geometric noise. From the various architecture they tested, the best consisted of convolutional, sub-sampling, and 3 spatial transformers layers while the worst did not include any STN module. This was due to CNNs being particularly sensible to scales and rotations.

Finally, this architecture has a total of 14.6 million parameters while achieving the maximum accuracy with 21 epochs in the GTSRB dataset. The best result was obtained with the SGD without momentum as the loss function optimizer.

In 2018, the performance on the BTSC was improved by Saha et al. (2018) achieving a state-of-the-art accuracy of 99.17%. The result was achieved through the application of a CNN with dilated convolutions (Yu and Koltun, 2015) as shown in Table 3. The samples were resized to 56×56 pixels and batches of 32 samples fed to the CNN. The best result

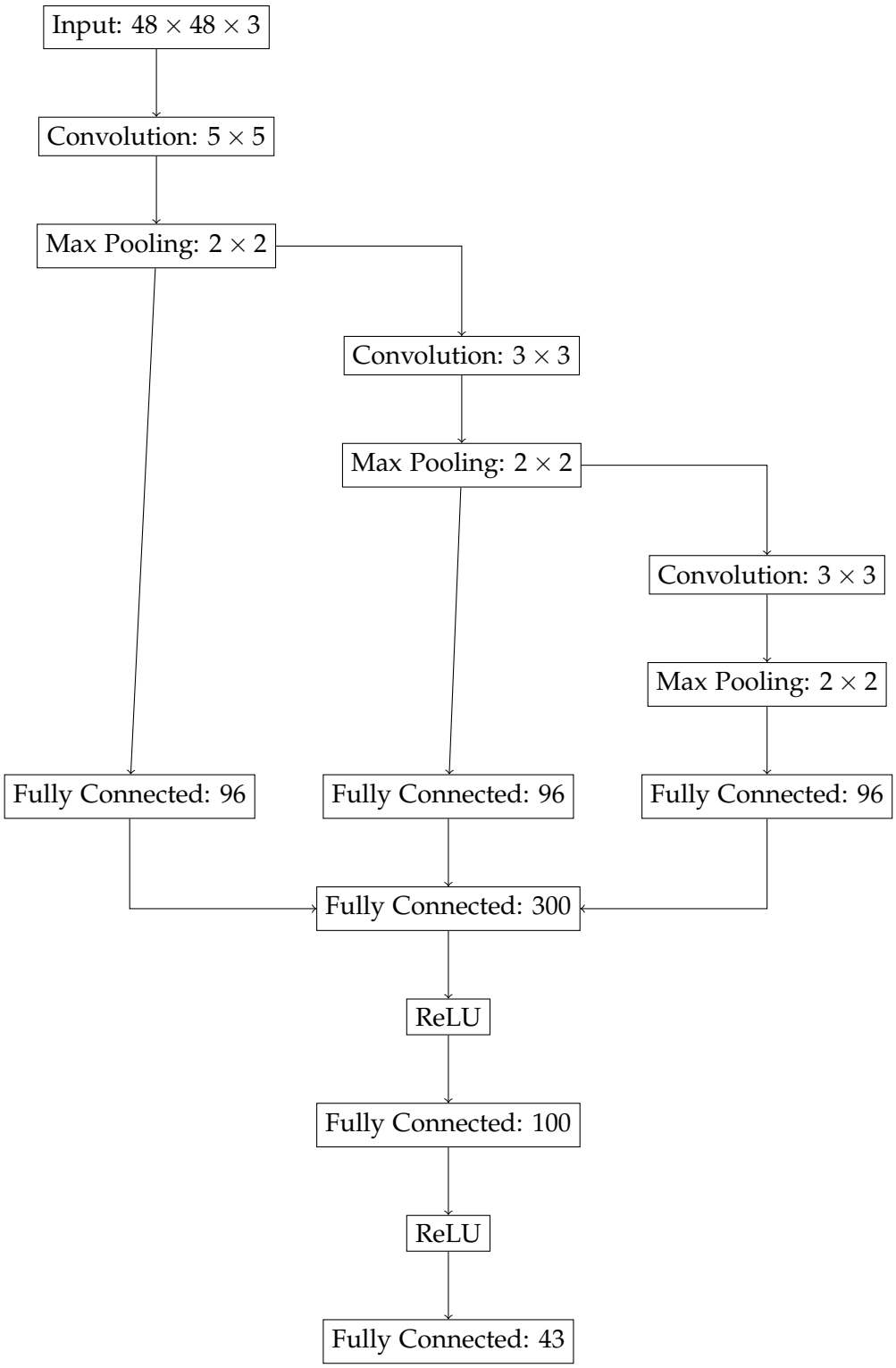


Figure 10.: Architecture devised by Jurišić et al. (2015) for traffic sign classification. The pooling layers have a stride of 2. Simplified representation from original work.

Table 3.: The CNN architecture used for the BTSC dataset classification by Saha et al. (2018). The number of parameters is 6.256 million. Table reproduced from original work.

Layer name	Output Size	Kernel Operation
Conv1	$56 \times 56 \times 64$	1×1, 64, stride = 1
Conv2a,b	$6 \times 56 \times 64$	$D_{64} \times 2$
Pool1	$28 \times 28 \times 64$	3×3 Max Pool, stride = 2
Conv2c	$28 \times 28 \times 64$	$D_{64} \times 1$
Conv3a	$28 \times 28 \times 128$	$D_{128} \times 1$
Pool2	$14 \times 14 \times 128$	3×3 Max Pool, stride = 2
Conv4a	$14 \times 14 \times 256$	$D_{256} \times 1$
Pool3	$7 \times 7 \times 256$	3×3 Max Pool, stride = 2
Conv4b	$7 \times 7 \times 256$	$D_{256} \times 1$
AvgPool	$1 \times 1 \times 256$	7×7
Dense1	512	Fully Connected
Dense2	Number of Classes	Fully Connected + Softmax

was achieved in 30 epochs. The total number or parameters of the architecture is 6.256 million.

2.3 DATA AUGMENTATION

Neural networks are heavily reliant on data to be effective. Usually, when data is scarce there is a tendency for overfitting to occur. Hence, overfitting occurs when there is too much reliance on the training data, and, as a result, a poor performance is obtained in unseen data. However, it is still possible to extract useful information with limited volume of samples by adding new data derived from the original dataset in the form of data augmentation. Therefore, increasing not only the diversity of the data but also the generalization capabilities of the neural network.

Data augmentation can be made a priori, and then using the fixed resulting dataset to train the network. A more promising approach is dynamic data augmentation, where new data is generated in each epoch. In this later case the network does not see the same samples twice, therefore being forced to cope with a larger set of circumstances.

In this section we present the most common data augmentation methods, based on color and geometric operations applied to each sample.

2.3.1 *Geometric transformations*

Geometric transformations consists of using a combination of affine transformations to manipulate the training data (Perez and Wang, 2017). Both the original image and the

(a) Original (b) Horizontal flip (c) Rotation of $-15°$

(d) Translation of 10×10 pixels (e) Center crop of 20% (f) Horizontal shear of 15 pixels

Figure 11.: Geometric data augmentation examples for class 34 of the GTSRB dataset.

augmented can be fed to the neural network, thus, increasing the volume of data in the same amount as the original for each transformation. Data augmentation has been shown to be a promising approach to increase the accuracy of object classification tasks. Nevertheless, some image transformations are unsafe depending in the domain they are being applied. The safety property refers to its likelihood of preserving the label post-transformation (Shorten and Khoshgoftaar, 2019).

Flip

Images can be flipped in the horizontally axis, in the vertically axis, or in both. Horizontal flipping is most commonly adopted due to the nature of traffic sign pictograms. This operation does not preserve the label post-transformation for all classes as shown in Figure 11b. In some cases, a valid traffic sign is still obtained, and labelling may be changed accordingly.

Rotation

Images can also be rotated with limited angles to ensure the preservation of a valid sign. Thus, considering traffic signs, rotating it by finer angles is preferable. An example of a rotation is shown in Figure 11c. Rotations are required due to the angle of the camera look direction with the direction to the sign, as well as in cases where, due to wind or vandalism, the sign itself is no longer perpendicular to the street.

Translation

Image translation consists in moving the image along the x-axis and y-axis in order to avoid positional bias in the data. This operation forces the CNN to search features along the entire sample frame. The remaining space can be filled with either a constant color, such as white or black, or with random noise. Spatial dimension of the augmented samples is preserved by inserting padding. An example of a translation is shown in Figure 11d. The requirement for the translation operation is derived from the method employed to crop the traffic signs to the bounding box, i.e. when using automated methods to detect and crop traffic signs, such as *You Only Look Once (YOLO)* (Redmon and Farhadi, 2018), the resulting image does not necessarily contains a centered traffic sign.

Crop

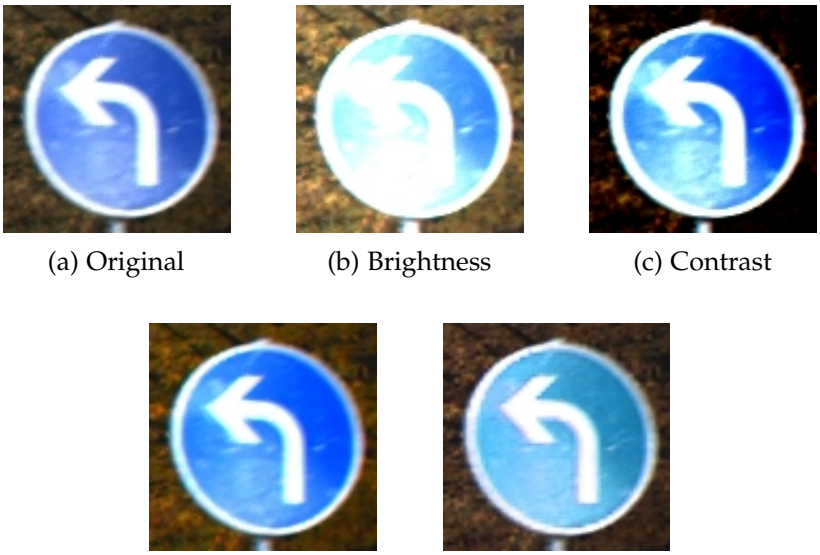
This operation consists in cropping a central patch from the original image and then resize the patch to the original image size. Additionally, besides central cropping, random cropping can also be used. The difference between random cropping and image translation lays in the input size reduction. While cropping will undoubtedly reduce the size of the input, translating will preserve the original spatial dimensions of the image. An example of cropping is shown in Figure 11e. Cropping in this sense is similar to scaling up. This operation is required for reasons similar to those pointed in the translation case.

Shear

Image shearing, also known as skewing, is an affine transformation that distorts the image plane displacing each point in a fixed direction. When shearing horizontally, the y-axis coordinates of each point are fixed meaning that a rectangle becomes a parallelogram, and a circle becomes an ellipse. Under shear, the area of the image is the same as the original (Figure 11f). This operation benefit generalisation due to the distortion on the traffic sign captured images.

2.3.2 *Color transformations*

A widely adopted color augmentation technique consists of color jittering (Shijie et al., 2017). Color jittering can be performed by adjusting the brightness, contrast, saturation, or hue of an image using random factors as shown in Figure 12. Other common color transformations consists in converting to the image to grayscale or apply gamma correction Wang et al. (2017).



(a) Original (b) Brightness (c) Contrast

(d) Saturation (e) Hue

Figure 12.: Color jittering examples of class 34 from GTSRB dataset. Hue jittered with a factor of 0.5 while the remaining color operations used a factor of 2.

These transformations aim at increasing the diversity of lighting situations, thereby potentially increasing the generalisation ability for unseen signs captured under different lighting conditions and camera exposures.

In other particular domains, other color augmentation techniques are also applied, such as color casting, which consists in tinting the sample with a particular color (Galdran et al., 2017), or vignetting, which consists in darkening the periphery of the sample compared to the image center (DBL, 2015).

2.3.3 *Random erasing*

Besides essentials geometric and color data augmentation, other complementary techniques have also been used. For instance, random erasing aims to improve the robustness of the neural network to noise and occlusions (Zhong et al., 2017). Usually, the collected training samples exhibit a limited number of object occlusions. In the worst case, when all training samples are clearly visible, the accuracy of the CNN may drop significantly when partially occluded test samples are classified. Therefore, a critical influencing factor on the generalization capability of the neural network is object occlusion.

In order to address the occlusion problem and, consequentially, improve the generalization ability of the neural network, random erasing is applied to the original samples. This process consists in simulating object occlusion by applying to the original samples rectan-

Figure 13.: Random erasing applied to class 11 of the GTSRB dataset.

gle regions of arbitrary size and position. The pixels values within the selected region are re-assigned with random values, which can be viewed as adding block noise to the sample.

Due to the fact the position and area of occlusion can be randomly chosen, several occlusion levels can be generated. In a similar fashion to random cropping, random erasing also incurs in information lost during augmentation. However, due to the fact only part of the object is occluded, the object structure is preserved. An example of random erasing applied to traffic signs can be seen in Figure 13.

Also, random erasing is similar to dropout at the image level by blocking part of the image with a defined probability. This way, decreasing the possibility of overfitting. In other words, like other types of data augmentation, random erasing is also acts as an explicit form of regularization.

In order to assess the performance of this approach, Zhong et al. (2017) applied random erasing to the CIFAR-10, CIFAR-100, and Fashion-MNIST datasets with several different neural network topology. Each test consisted in 5 runs. The baseline result was compared to the data set using new data erasing to each training iteration. They obtained a reasonable improvement on object detection. In CIFAR-10, CIFAR-100, and Fashion-MNIST they were able to improve the accuracy by 0.72%, 0.76, and 0.36% using a Wide Residual Network (ResNet) architecture, respectively.

2.4 SYNTHETIC DATA

The efficiency of the CNNs relies heavily on data. Therefore, a large amount of accurately labeled data is required. The data should be as representative as possible, covering a large number of possible scenarios. Therefore, a network that trains in high quality data will more likely exhibit high accuracy in unseen data. However, in particular domains, such as in traffic sign recognition, collecting the needed data is not an easy task.

As mentioned before, in an attempt to overcome this problem, data augmentation is commonly applied to the training data, increasing the original number and diversity of samples. However, the increased data is derived from the original set and will always be conditioned by the original samples. Although data augmentation has proven to be an

Figure 14.: Sample scenes of the office, living room and kitchen models used for rendering of the synthetic dataset. Source: Lee and Moloney (2017).

efficient method for improving the robustness of neural networks, it fails to succeed when the unseen samples fall far from the range present in the training set. Therefore, it is often necessary to reproduce the samples in different circumstances from those in which they were collected as in the context of traffic signs.

An alternative approach that derives from the scarcity of samples is data synthetisation. Synthetic data can not only augment real data but also simulate data under specific conditions that are not easily found in real data. That is, synthetise traffic signs under distinct conditions such as occlusions, reflections, vandalism, aging, noise, weather patterns, or time of day.

A synthetic data driven approach has been at least partly successful in other domains. For instance, Lee and Moloney (2017) crafted a complex synthetic dataset for training and evaluation of stereo vision algorithms. Stereo vision is a way of extracting the 3D structure of a scene using at least two images of that same scene, ensuring that each one is acquired from a different viewpoint in space. For this domain, a CNN model was trained with synthetic data (Figure 14) and compared the accuracy against real world data in order to assess the validity of their approach as a mean to train neural networks. They found that the accuracy drop induced with the use of synthetic data was low in comparison with real world data. Nonetheless, they assessed that the synthetic data was modeling, albeit not completely, the parameters of the real world data. They concluded it was a viable method for training models within real world application.

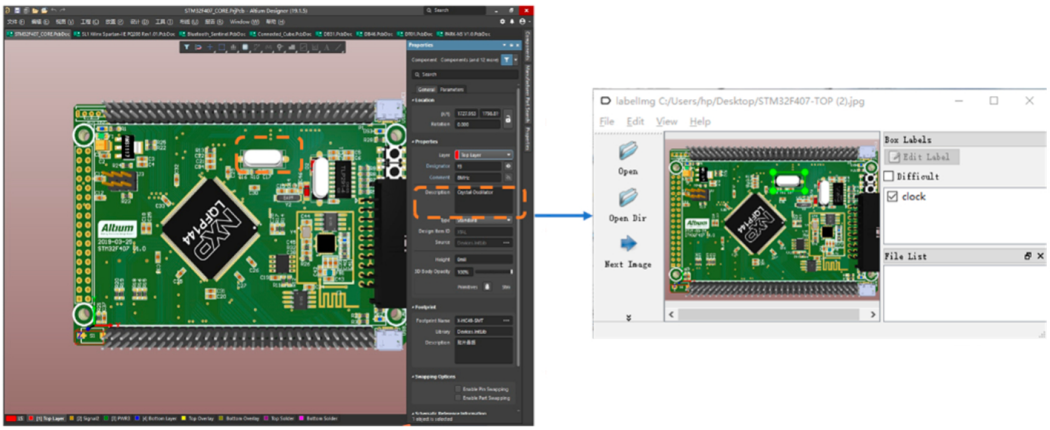


Figure 15.: Display query and label of a synthesized PCB. Source: Li et al. (2019).

In other domains, synthetic data was able to actually surpass the usage of real world data. For instance, Jiang et al. (2019) proposed an automatic infrared spaceborne ship detection approach relying solely in synthetic generated data. Infrared images allows all-weather detection capability which is important for both military and civil applications. The detection module was based on CNNs. Their approach consisted in converting optical ship samples from Google Earth into infrared ship samples as synthetic training data. In order to generate synthetic infrared training images, they designed an image generator to convert the style of Google Earth optical images into the style of infrared's. This is a domain where actual training data of infrared spaceborne ships is very difficult to obtain. Their method was able to surpass some of previous results based in *Histogram of Oriented Gradients (HOG)* features and transfer learning, achieving a precision of 94.50%.

Other domain of great importance is the detection of several components on PCBs in order for the 3C industry (Computer, Communication and Consumer Electronic) manufacturing companies to achieve quality control and an intelligent assembly by robots. The quantity of electronic components on PCBs is high, and consisting in different shapes. It is very difficult to collect real photos of PCBs containing most electronic components to build a training dataset. Besides, identifying all device categories for labeling requires a lot of knowledge and time. Therefore, Li et al. (2019) proposed an improved detection algorithm based on YOLO, which uses real and synthetic PCBs pictures as a joint training dataset. In order to produce synthetic PCBs, they used Altium Designer, a PCB designer software that provides a component search engine which accesses a huge number of manufacturers and electronic components. The synthesization of a PCB can be seen in Figure 15. Their approach was able to achieve a mean average precision of 93.07%.

Furthermore, research showed that it is not always ideal to simulate all the realistic features found in the environment of the object of interest. For instance, Tremblay et al. (2018) employed domain randomization, in which the parameters of the simulator, such as light-

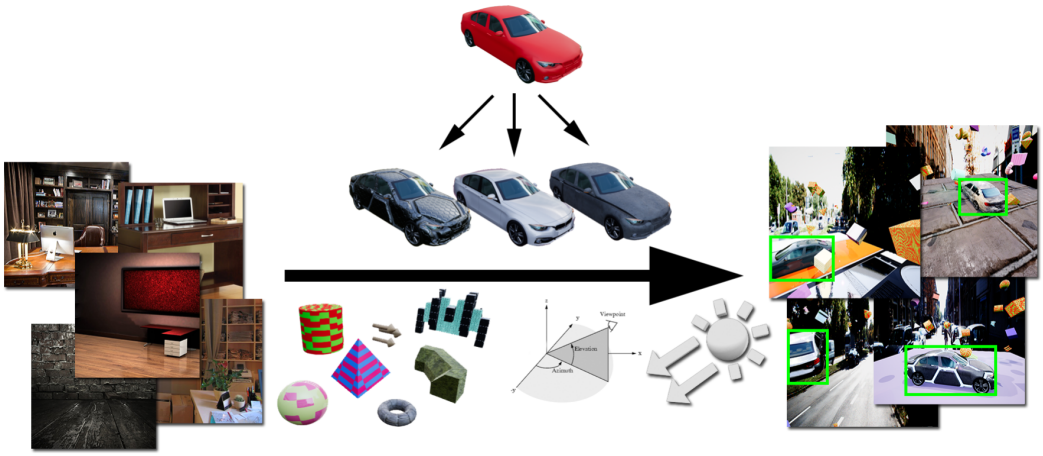


Figure 16.: Domain randomization for object detection. Synthetic objects are rendered on top of a random background along with random geometric shapes (next to the background images) in a scene with random lighting from random viewpoints. Source: Tremblay et al. (2018).

ing and object textures, are randomized in non-realistic ways. In other words, domain randomization intentionally abandons photo-realism by randomly perturbing the environment in order to force the neural network to learn the essential features of the object to be detected, as exemplified in Figure 16.

To evaluate their approach they used the KITTI dataset Geiger et al. (2012b) as a real-world test set with three different state-of-the-art object detectors: Faster R-CNN (Ren et al., 2015), R-FCN (Dai et al., 2016), and SSD Liu et al. (2015). The detectors were trained in both domain randomized data and in the Virtual KITTI (VKITTI) dataset (Geiger et al., 2012a). From the three employed detectors, R-FCN and SSD provided best results with domain randomized data.

2.4.1 *Traffic sign detection*

A synthetic data approach has already been explored in conjunction with neural networks in traffic sign detection which consists in detecting the position and bounding box of a traffic sign in a street picture. The detected traffic sign is later classified. This section only addresses traffic sign detection.

In order to avoid the time and effort consuming task of hand labelling traffic signs, synthesized data was used for road signs detection tasks by Møgelmose et al. (2012), Greenhalgh and Mirmehdi (2012), and Hanel et al. (2018).

In 2012, Møgelmose et al. (2012) generated synthetic training images templates in an attempt to substitute real-world training data for traffic sign machine-learning based detection systems. In order to simulate the traffic sign real characteristics, several transforma-

tions are applied, such as hue variations, lighting variations, rotations, real backgrounds, Gaussian blur, Gaussian noise, and occlusions. In their study, the real-world data performed significantly better than the synthetic data. They noticed that providing more synthetic training data did improve the detector accuracy, but even more than a double increase of training data does not make the synthetic data perform comparably to the real-word data. They concluded that there is no substitute for real world images in the context of traffic sign detection.

However, Greenhalgh and Mirmehdi (2012) generated training data from synthetic templates of traffic sign images, avoiding the use of real footage road signs as training data. Their detector was trained in fully synthesized data. Randomized geometric distortions, randomized brightness, contrast, noise, blurring, and pixelation were applied to the German templates. A total of 1200 synthetic traffic signs were generated for each class. They benchmark both the synthetic data as well as the real data on a linear *Support Vector Machine (SVM)* classifier. Using the GTSRB test set, the classifier that was trained on the synthetic data resulted in an accuracy of 85.70% while the classifier that was trained on real-world images resulted in an accuracy of 85.90%. They concluded that the synthetic data set produced results comparable to the data set real-world samples.

2.4.2 *Traffic sign classification*

A synthetic data approach can be found applied in traffic sign classification tasks as well. In 2018, Stergiou et al. (2018) proposed the use of synthetic training data which was based in traffic sign templates. The goal was to overcome the problems of existing traffic sign datasets, which are only subject to specific sets of traffic signs found explicitly in specific countries.

They processed 50 distinct templates resulting in 50 classes of British traffic signs. As for the background, 1000 samples of British roads in several scenes were used. Half of the backgrounds are examples of urban areas and roads, while the other half is depicted in rural environments. The synthesized traffic signs are created to simulate different lighting conditions with the intent of closely simulate real-world scenarios. Part of the normalization process, non-local means denoising is applied to the images, blurring them. Also, 20 distinct affine transformations were applied in conjunction with rotations, scaling, and translations. The final data set was constructed with 4 brightness variations generating a total of 2388397 samples divided by 50 classes. From the total of synthetic images generated, only 3400 were retrieved for each class totaling in 170000 training images.

To evaluate their approach they use a single CNN architecture which consists of 6 convolutional layers with zero padding. The architecture is shown in Table 4. They used *Exponential Linear Unit (ELU)* as activation function (Clevert et al., 2015) which was applied

Table 4.: The CNN architecture used for traffic sign classification with synthetic data by Stergiou et al. (2018) with a total number of parameters of 2676306.

Layer	Type	Kernel	Output Shape	Parameters
0	Input		$48 \times 48 \times 3$	
1	Convolution	3×3	$48 \times 48 \times 32$	896
2	Batch Normalization		$48 \times 48 \times 32$	128
3	Convolution	3×3	$48 \times 48 \times 32$	9248
4	Batch Normalization		$48 \times 48 \times 32$	128
5	Max Pooling	2×2	$24 \times 24 \times 32$	
6	Dropout		$24 \times 24 \times 32$	
7	Convolution	3×3	$24 \times 24 \times 64$	18496
8	Batch Normalization		$24 \times 24 \times 64$	256
9	Convolution	3×3	$24 \times 24 \times 64$	36928
10	Batch Normalization		$24 \times 24 \times 64$	256
11	Max Pooling	2×2	$12 \times 12 \times 64$	
12	Dropout		$12 \times 12 \times 64$	
13	Convolution	3×3	$12 \times 12 \times 128$	73856
14	Batch Normalization		$12 \times 12 \times 128$	512
15	Convolution	3×3	$12 \times 12 \times 128$	147584
16	Batch Normalization		$12 \times 12 \times 128$	512
17	Max Pooling	2×2	$6 \times 6 \times 128$	
18	Dropout		$6 \times 6 \times 128$	
19	Fully Connected		512	2359808
20	Batch Normalization		512	2048
21	Fully Connected		50	25650

to the output of every convolutional layer. In addition, the dropout was also inserted in the architecture in form of regularization method. The rate of the dropout is 20% in the convolution phase, 50% after the fully connected layer. Finally, they were able to obtain a peak accuracy of 92.20%. However, they do not mention the data set used for evaluation neither the test conditions.

2.5 SUMMARY

Traffic sign classification research is an ongoing process. There is, clearly, still room for improvement. Currently, CNN based classifiers are the preferred approach to recognize traffic signs and were able, in fact, of bringing impressive results but still imperfect.

In the domain of ADAS, every imperfection must be addressed to ensure the safety of all public road actors. Thus, CNN architectures can become incrementally complex in an attempt to improve the state-of-the-art in traffic signs recognition. It is crucial to note that

the necessity of increasing the architectural complexity of the neural networks stems from the scarcity of traffic sign samples and, therefore, the need to compensate for the lack of it.

For instance, distinct approaches were used for the GTSRB dataset that ranged from employing a committee of several CNNs (Cireşan et al., 2012) to, in spite of keeping just a single architecture, inserting various Inception and STN modules (Haloi, 2015).

It was possible to notice in Section 2.1 that the datasets have clear differences between them. Those differences ranges from the total number of classes and its selection, and between the pictograms of equivalent classes. It is common for each country to employ its own pictograms even in equal classes based in the Vienna Convention on Road Signs and Signals.

To reach the required level of training data quality, a high volume of traffic signs has to be collected. It is, however, a task that requires a lot of human effort, time, and resources. For instance, the biggest European data set has roughly 39000 training samples while, even tough it can be considered a reasonable amount, the second largest has only approximately 8000 training images. The difficulty of creating an abundant data set ultimately influences the accuracy of the CNNs, given that they are highly sensitive to input data differences, even if slight. For that same reason, non acquired features of unseen training data tends to be poorly classified.

A commonly adopted approach to address the low data volume problem is based on augmention of the the original samples. Data augmentation may be able to improve the accuracy, to an extent, by increasing the volume and diversity of the training data. This approach has limitations, however, due to the high dependency on the original samples.

Hence, a different approach to circumventing the scarcity of training samples is based on data synthesization. That is, to produce a synthetic set of road signs based only in templates, similar to those found in the real-world. Unlike real traffic signs datasets, a synthetic dataset does not rely on actual data. Certain classes of traffic signs are, in fact, very abundant in the real world while others are less frequent. In particular, road signs that are more common in cities and a lot less in villages, and vice versa. For this reason, capturing certain classes can be hard. This class-level representativity problem is not found in synthetic data as all classes are treated uniformly and the number of samples generated is conceptually unlimited. Ultimately, any class can be added to the set by simply providing the corresponding templates.

A synthetic data approach is heavily domain dependent. In traffic sign classification, the usage of synthetic training data is highly scarce. In other domains it was proven a successful method but far from effortless. Due to the nature of image synthetization, it is crucial to understand the necessary geometric and color transformations so that the synthetic traffic signs samples resembles real-world data. Recent results with synthetic data, while far from state-of-the-art results with real data, are, nonetheless, encouraging.

3

TRAFFIC SIGN SYNTHETISATION TECHNIQUES

Synthetise traffic signs is a complex process. This chapter aims to review the distinct approaches employed in this work for data synthesization. Those are divided in geometric transformations, color transformations, and transformations aiming to disturb the traffic sign in form of noise or blur.

This incremental process consists in approximate the synthetic traffic signs to real-world samples taking into account the necessity of generalization capabilities. In order to develop an accurate synthetic dataset it is crucial to understand and reproduce the essential features of real-world samples which may not be present in the synthetically generated images. As such, incorrectly predicted real traffic signs are interpreted continually in order to understand which traces they could contain that the synthetic data was not able to reproduce.

It is essential to evaluate the developed approach against real-world traffic sign datasets and assess its effectiveness. The main dataset used for testing throughout this work is the GTSRB dataset as it is the largest European set publicly available. If the developed technique is sufficiently generic, it should be able to be applied to several distinct traffic sign datasets of another countries.

3.1 SYNTHETIC RESOURCES

A synthetic sample is the composition of a foreground and a background. The foregrounds are computed based on sign templates. Each class consists in, at least, one template. More than one template may be required when there are different pictograms in use for the same traffic sign. Backgrounds may be obtained from real images or also created synthetically, as explored in this section.

3.1.1 *Templates*

Handcrafting a synthetic traffic sign dataset requires the respective class templates. As mentioned before, the German dataset was chosen to evaluate the effectiveness of the different

Figure 17.: Templates used for the GTSRB synthetic data set representing each class labeled from left-to-right and top-to-bottom. Source: Wikipedia (2019).

transformations applied to the synthetic samples. The main foregrounds for each class employed in the template-based synthetic dataset are shown in Figure 17. These templates were collected from Wikipedia (2019) corresponding to the original 43 classes of the GTSRB dataset.

Some classes of the German dataset have more than one type of traffic sign pictogram as shown in Figure 18:

- Class 1, the speed limit of 30 km/h, has three distinct templates with different fonts and text (Figure 18a);
- Classes 3 and 5, the speed limits of 60 km/h and 80 km/h, respectively, have two slightly different fonts (Figure 18b);
- Class 25, men at work, can be seen with two different pictograms styles (Figure 18c);
- Classes 35 and 38, directional arrows, have slightly different pictograms and hue in comparison to old versions found in the road (Figure 18d).
- Class 40, the roundabout vertical sign, is installed in two different orientations which represent a rotation too large to be considered as data augmentation (Figure 18e).

The acquired templates are in *Scalable Vector Graphics (SVG)* format providing a transparency mask. Originally, The templates had a thin black line on the border of the sign that had to be removed as it is not part of real signs.

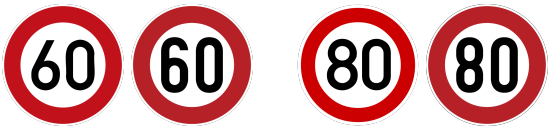
3.1.2 *Backgrounds*

Backgrounds are also an essential in building synthetic images. As such, both real and synthetic backgrounds were evaluated, ensuring identical test conditions.

Backgrounds from street views

The usage of real backgrounds in synthetic samples is found in the work contribution of other authors (Møgelmose et al., 2012; Stergiou et al., 2018). Images of street scenarios

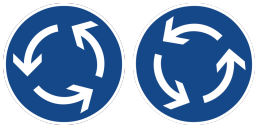
(a) Class 1



(b) Classes 3 and 5

(c) Class 25

(d) Classes 35 and 38



(e) Class 40

Figure 18.: Templates added to complete German traffic sign dataset classes. The leftmost sign of each class is the main template.

(a) Class 9 (b) Class 33 (c) Class 0

Figure 19.: Examples of synthetic templates with real backgrounds for the German, Belgium, and Croatian datasets, respectively. Samples with a resolution of 32×32 pixels.

from Google Street Views were used as real backgrounds. The street samples are negative backgrounds meaning that traffic signs are not present in the image. The backgrounds are segmented in squares generating a higher volume of samples.

For each synthetic traffic sign sample, the background was selected randomly from the loaded set. In Figure 19 is shown an example of synthetic samples with actual backgrounds for the GTSRB, BTSC, and rMASTIF datasets, respectively.

Using real backgrounds seemed a promising approach, as there would be a close background match with real images. However, this approach was not able to capture all the diversity of situations that could be found in the GTSRB dataset, or created an uneven set of background types.

Due to the fact the entire street sample was segmented in several backgrounds, these contained a large amount of sky backgrounds (Figure 19b) as well as a high prevalence of greenish colors (Figure 19a) derived from vegetation.

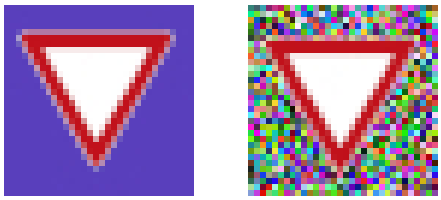
Tests performed with these backgrounds hinted that the uneven background distribution was affecting the accuracy of CNNs trained with synthetic data and tested on the GTSRB test dataset. It was noticed that the neural network was not being able to recognize a few traffic signs in optimal visibility conditions whose background contained color and shapes not found in the training synthetic dataset. This was confirmed by replacing manually the background of some misclassified samples with known backgrounds. In most cases, the change in background was enough for the sample to be correctly classified as Figure 20 suggests.

Synthetic Backgrounds

In order to improve the classification of traffic signs it is crucial to distinguish them from its background. In other words, ignore the background details entirely by forcing the CNN to focus only on features present on traffic signs.

Two distinct approaches were employed to synthesize backgrounds as shown in Figure 21. The first approach is the simplest and consists in coloring each background with a random color (Figure 21a). A second approach consists in coloring the background with

Figure 20.: Test traffic signs (on the top) correctly classified only after altering the background (on the bottom).



(a) Solid color (b) Pixelated

Figure 21.: Synthetic background examples of class 13 from the German dataset with a resolution of 32×32 pixels.

random pixels (Figure 21b). The aim was to force the neural network to understand that the background is noise and that it should be discarded. The background dimensions are calculated based in the traffic sign resolution and given margin as Equation 1 evidences.

$$background\_size = sign\_size/(-2 * margin + 1) \tag{1}$$

From these two approaches, using a background with a random color for each pixel presented the worst accuracy. Furthermore, using solid backgrounds increased the overall accuracy slightly in comparison to real backgrounds as reported in Section 4.3. Hence, the synthetic dataset backgrounds consists in randomly colored squares for every sample.

3.2 GEOMETRIC TRANSFORMATIONS

Real-world traffic sign samples are not perfectly shaped. Those can be seen with different scales both regarding the total occupied frame area and the overall traffic sign resolution.

Collected samples from roads and streets contains, commonly, traffic signs in different scales, shifted both horizontally and vertically, and slight rotations in both directions. Therefore, those essential transformations are performed to the templates.

After being loaded, the templates are resized to random resolutions drawn from a continuous interval. Each synthetic sample will contain a traffic sign with a different resolution

(a) Class 8

(b) Class 28

Figure 22.: Pictogram quality differences depending in the order of resize and rotation for 32×32 pixels traffic signs. The middle sign is rotated after resized while the rightmost sign is rotated first.

from that of the input template. The background dimensions are calculated after resizing the template. This way, increasing variability even further.

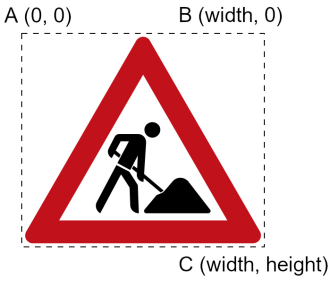
Every sign is shifted freely in the x-axis and y-axis respecting an imposed translation margin. The translation occurs when pasting the resized traffic sign onto the background. This constraint was to ensure that the template remained inside the sample frame and, similar to real samples, that the translations are around the center. Very aggressive translations, those which touch the edge of the frame sample, for example, are problematic for the CNN to handle. Extreme translations are also uncommon in real-world traffic signs samples as well.

Finally, the traffic signs undergoes random rotations in a given range. The rotation is performed before resizing the template. Rotating in small resolutions creates defects in the pictogram as the Figure 22 evidences. Smaller resolutions than those of the figure, magnify the pictogram deformation. Note that the interpolation algorithm is the same in both cases.

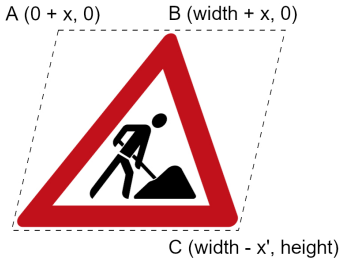
These three operations constitute the essential geometric transformations and are able to achieve 91% of accuracy by themselves on the GTSRB dataset alongside synthetic backgrounds of randomized dimensions (Section 4.4 presents a more detailed report on the implementation and results).

3.2.1 *Shear*

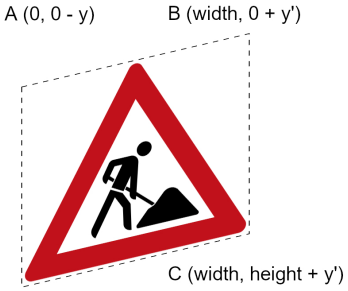
Recently, the work of Stergiou et al. (2018) accomplished an accuracy of 92.20% with purely synthetized traffic signs. Note that, at the time, this was the state-of-the-art result. An important reference in this work was the use of 20 fixed, and distinct, affine transformations



(a) Original template



(b) Horizontal shear



(c) Vertical shear

Figure 23.: Horizontal and vertical shear applied to template of class 25 from the GTSRB dataset.

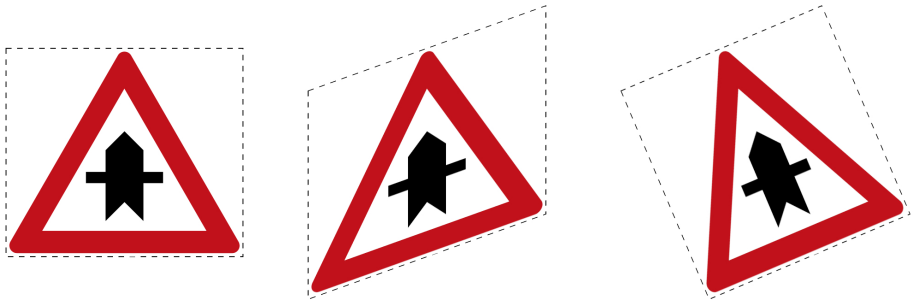
in form of shearing in an attempt of recreating different viewpoints for the same traffic sign template.

Instead of 20 fixed transformations, we generate a random shear transformation for each target traffic sign sample. The shear can be performed both horizontally or vertically. An example of shear performed onto a template is shown in Figure 23.

First, three points are chosen such as those in Figure 23a. In the case of horizontal shear, a quantity x is added or subtracted to the x-axis meaning that the traffic sign can be tilted to the left or to the right (Figure 23b). Thus, x is determined as Equation 2 suggests. This way, we ensure that any input template resolution is admitted. In the case of vertical shear, the operation is analogous to that of horizontal shear but with a quantity y (Figure 23c) regarding the height.

$$\text{Let } \{u \in \mathbb{R} : 0.03 \leq u \leq 0.10\},\ x' = x + width * u \tag{2}$$

Note that horizontal shear and vertical shear are never performed in the same sample, just one of the two, arbitrarily. This is due to the fact that performing both could have the undesirable effect of rotate the traffic sign as seen in Figure 24. Inverting the order of the operations produces the same result.



(a) Original template (b) Vertical shear (c) Horizontal shear

Figure 24.: Resulting rotation by applying vertical shear followed by horizontal shear to class 11 template of the GTSRB dataset.

(a) Original (b) Adjusted

Figure 25.: Manually perspective adjusted test sample from class 25 of the GTSRB dataset.

3.2.2 *Perspective*

Although most traffic signs face the camera in a frontal manner, a few collected samples can be observed in distorted perspective views. This can be due to strong winds, even vandalism, or the proximity of the camera to the traffic sign.

For instance, in class 25 of the GTSRB dataset is possible to observe an example of such perspective deformation in Figure 25. This particular sample (Figure 25a) was initially incorrectly classified. However, after performing a manual perspective adjustment it became successfully recognized (Figure 25b), hinting that the perspective transform could be the cause for the initial misclassification.

This occurrence affects other datasets as well, namely the BTSC and DITS datasets. In the BTSC dataset, some of the perspective distorted traffic sign samples are correctly classified without resorting to synthesization of such transformations. Some of those samples are shown in Figure 26. Note that since these images are not squared, when resized to the final square size of 32×32 pixels, the perspective effect is somewhat attenuated.

In other instances, the samples are not correctly classified, in particular if the distortion is greater, as shown in Figure 27.

Figure 26.: Correctly classified perspective distorted samples from classes 2, 8, and 18 of the BTSC dataset, respectively.

Figure 27.: Misclassified perspective distorted samples from classes 32 and 47 of the BTSC dataset.

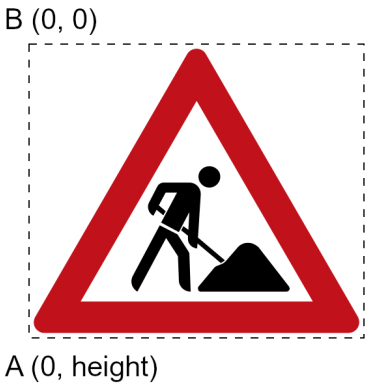
Although with less frequency than in the BTSC dataset, both the classes 37 and 45 of the DITS dataset contains samples positioned in a lateral perspective manner (Figure 28). As for the rMASTIF dataset, no sample was incorrectly classified due to perspective distortions.

Therefore, in order to improve the recognition of perspective distorted samples, a perspective adjustment was performed as shown in Figure 29.

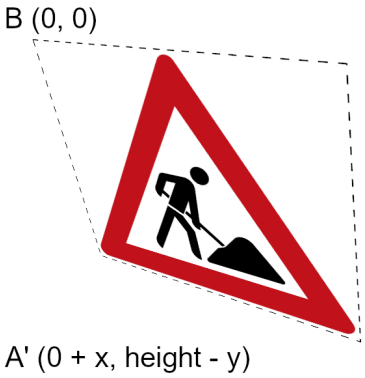
The perspective transform operation is applied in just one step. For demonstration purposes, however, it is shown as a separate operation for each chosen vertex. Thus, firstly, a point is chosen randomly from the set of four points coinciding with the vertices of the image. In this case, the bottom-left vertex was chosen as shown in Figure 29a. In order to complete the operation, two points are necessary. For this reason, the second point is always the clockwise neighbor of the first. Having the two points, the next operation consists determine the resulting points of the transformation.

For each point, a random factor in the range $[5, 15]\%$ is multiplied by the width, in the case of the x-coordinate (Equation 3), or by the height, in the case of the y-coordinate. Then, the determined value is added or subtracted, depending on the point, to the respective

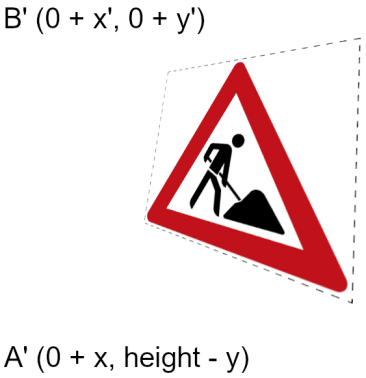
Figure 28.: Lateral perspective distorted traffic signs of classes 37 and 45 of the DITS dataset.



(a) Original template



(b) Perspective applied to point A



(c) Perspective applied to point B

Figure 29.: Perspective transformation applied to the template of class 25 from the GTSRB dataset.

Figure 30.: Examples of different generated viewpoints of class 25 template from GTSRB dataset.

coordinate. The remaining two opposite points undergo the operation in the opposite direction, widening further the perspective.

$$\text{Let } \{u \in \mathbb{R} : 0.05 \leq u \leq 0.15\},\ x' = x + width * u \tag{3}$$

For example, if the image size is 600×600 pixels and the factor is 10% for both the x-coordinate and y-coordinate, the initial point in Figure 29b becomes $(60, 540)$. The process is repeated for the second point in Figure 29c.

Due to the fact that any point can be chosen alongside random factors per coordinate, this approach generates a high volume of traffic signs in different viewpoints (Figure 30).

3.3 COLOR TRANSFORMATIONS

Real-world traffic sign samples have different intensities of brightness between them, whether by direct sunlight exposure or when placed in the shade. Actual signs also undergo slight changes in hue and saturation caused by external factors such as, for example, weather phenomena or camera sensor quality.

This should be taken into account when producing synthetic traffic signs. By default, the templates used in the synthesization process are highly saturated and bright. On the other hand, real samples are present in harsh conditions such as low light exposure, or worn out due to prolonged exposure to the natural environment.

3.3.1 *Hue and saturation jitter*

By adjusting the hue and the saturation we can increase the range of admissible traffic sign colors in the produced dataset. While jittering the hue increases the traffic sign color range, as seen in Figure 31, jittering the saturation is able to simulate worn out or damaged signs, as seen in Figure 32. Note that both hue and saturation jitter can be applied to the same sample amplifying the range of colors that the CNN is able to see.

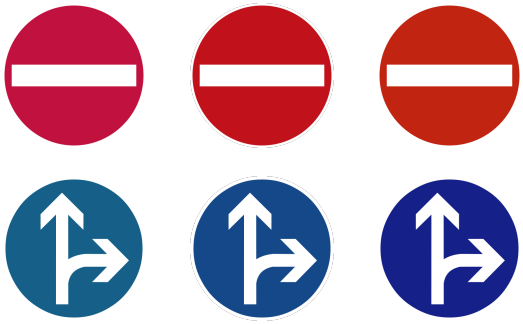


Figure 31.: Admissible hue variations for classes 17 and 36 of the GTSRB dataset. The middle sign represents the original template.

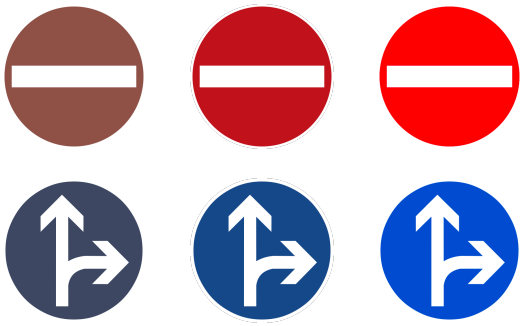


Figure 32.: Admissible saturation variations for classes 17 and 36 of the GTSRB dataset. The middle sign represents the original template.

3.3.2 *Brightness distribution*

Actual data contains valuable information for its synthetic replication. Among the many properties that can be extracted, luminosity is one of them. Instead of jittering the brightness, a different approach was to base the brightness values to be used on synthetic data on real-world data. This approach requires determining how they are distributed among the traffic sign samples. Therefore, the initial step was to calculate the brightness of an image. The computation follows the standard definition of the Luma equation which is used in most digital cameras (Rec. 601) as seen in Equation 4.

$$Y'_{601} = 0.299R' + 0.587G' + 0.114B' \tag{4}$$

The Luma is calculated for each pixel of the image. The resulting image brightness is the mean of the individual previous calculations.

An optimization was performed after noticing that the brightness value of the image, using the previous method, was very close to that of calculate the mean of the pixels from the value component after converting the image to HSV color space. This last method

Table 5.: Fitted Johnson distribution parameters of the German, Belgian, and Croatian datasets brightness.

	Parameter			
Dataset	γ	δ	ξ	λ
GTSRB	0.747	0.907	7.099	259.904
BTSC	0.727	1.694	2.893	298.639
rMASTIF	0.664	1.194	20.527	248.357

was preferable since it is slightly faster to calculate, which translated into a faster synthetic dataset generation time.

In order to reproduce the brightness of the real traffic sign samples it is necessary to determine the distribution of the entire data set. Hence, the next step is to calculate the brightness distribution of diverse traffic sign datasets. For this purpose, the German, the Belgian, and the Croatian datasets were used since they have a higher number of samples than other smaller and, therefore, less representative data sets.

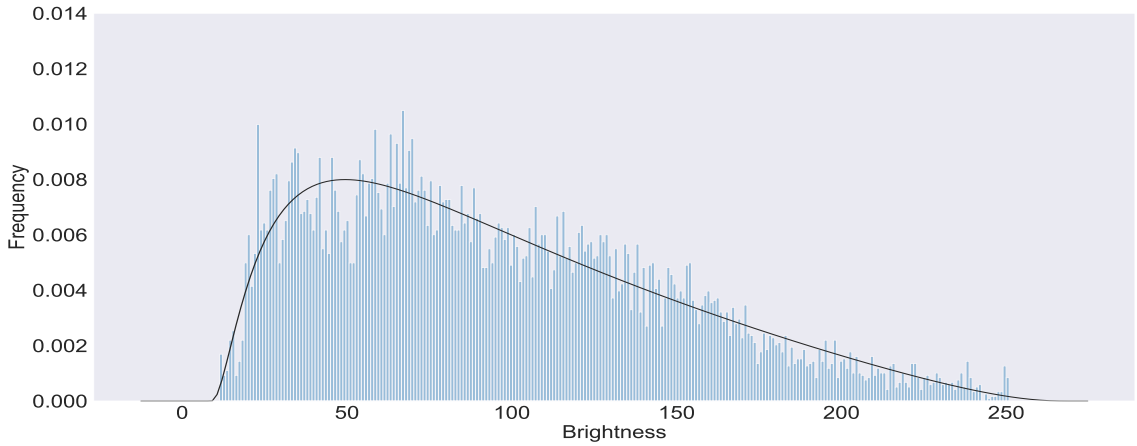
Therefore, to assess which distribution fits the dataset best, the Kolmogorov–Smirnov test (K-S test) is performed. The K-S test is used to test whether two samples are drawn from identical distributions. Thus, for the goodness of fit test, several distributions are tested and ordered by discrepancy from the best fit to worse fit. The parameters of the best fit distribution are used to generate the new brightness values for each synthetic sample in the dataset generation process.

After running the K-S test in all available samples from the three datasets it was found that the Johnson distribution with bounded values was able to fitted them correctly. The histogram plot of the distributions for each country are represented in Figure 33 with the same scale. The parameters of the distributions are presented in Table 5. For any distribution value x: $\xi \leq x \leq \xi + \lambda$.

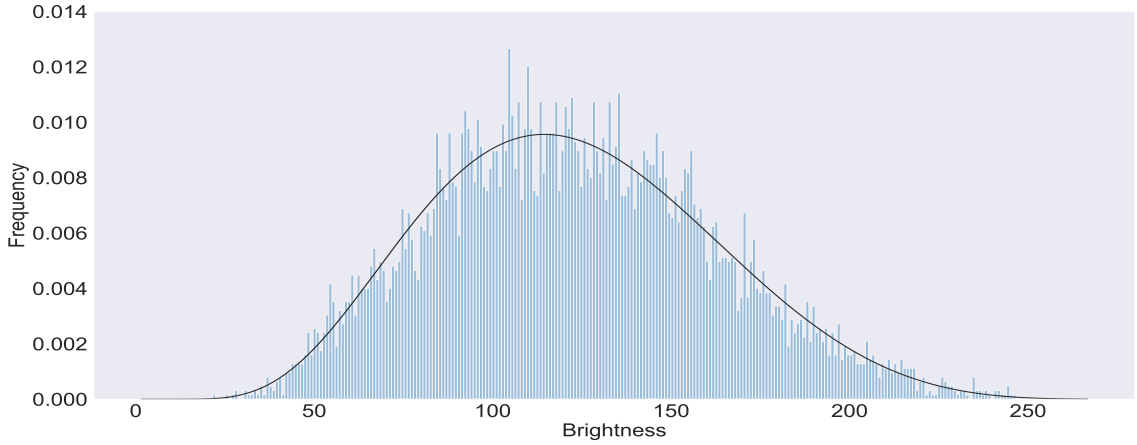
The brightness of the traffic signs varies from sample to sample within the data set itself which increases the recognition effort by the neural network. From the three represented datasets, the German has a higher volume of darkened samples in comparison to the remaining two. The Johnson distribution was initially proposed as a transformation of the normal distribution. In fact, it is possible to visualize a mean brightness value for each data set.

To test the potential benefit of using this information, the samples of the synthetic dataset were adjusted to the Johnson distribution with the parameters determined for the GTSRB dataset, and a CNN trained with this synthetic dataset was tested against the GTSRB test set.

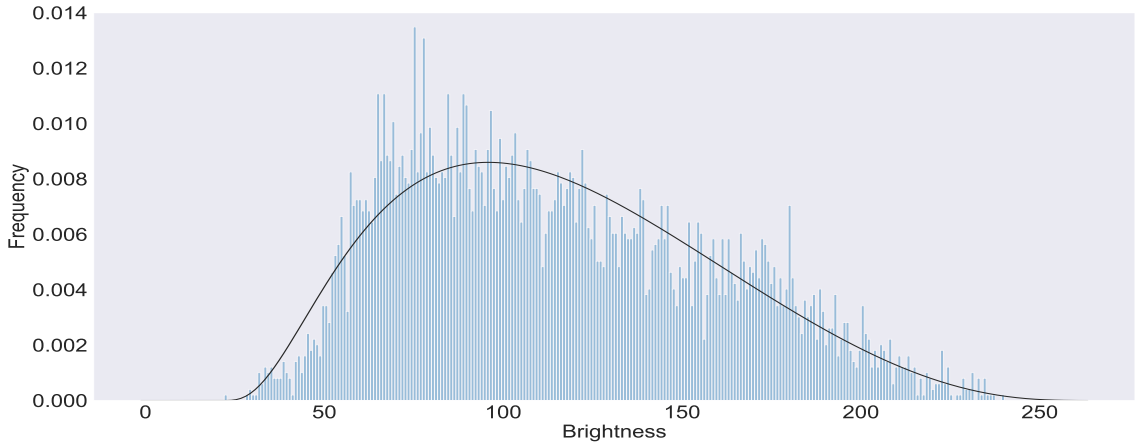
To adjust the brightness it is necessary to first generate values that conform the determined distribution. Thus, each synthetic sample will have the traffic sign adjusted in the



(a) Germany



(b) Belgium



(c) Croatia

Figure 33.: Brightness frequency for the German, Belgian, and Croatian datasets. The curve represents the Johnson fitted distribution. The brightness is normalized between $[0, 255]$ with both ends representing a black and white image, respectively

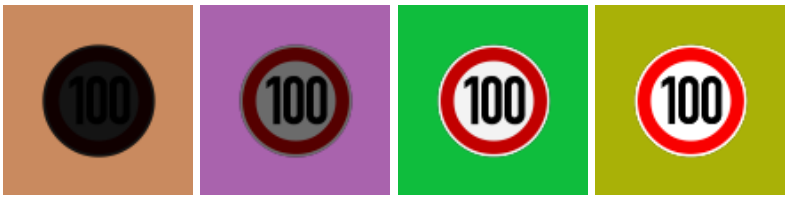


Figure 34.: Brightness adjusted synthetic samples of class 7 from the GTSRB dataset.

final step of the operations pipeline. The background is kept untouched as its an invariant. Changing the background brightness proved to be an unsuccessful strategy given that we are hindering the sign recognition task. In addition, we want to ensure a discrepancy in brightness and color between background and the traffic sign.

Having the new brightness value, the next step consists in calculate the brightness of the traffic sign template. Then, the brightness ratio is determined by dividing the new brightness value by the traffic sign brightness. Finally, each position of the value component of the HSV color space of the traffic sign template is multiplied by the ratio. The only constraint is to make sure the values never exceed 255 since the brightness ranges from 0 to 255. The end result can be seen in Figure 34.

3.4 TEMPLATE DISTURBANCES

The traffic sign templates are, by default, very clean and homogeneous exhibiting a total absence of any type of color disturbance. From this stems a necessity of increasing the graphic fidelity in order to resemble real-world sign samples. The disturbances applied to templates are in form of blur and noise.

3.4.1 *Motion blur*

Due to the fact that traffic signs are extracted from video clips and usually collected on a moving car, some samples contain blur introduced by the motion of the camera, namely, samples that are closer to the car. This artifact usually occurs along the direction of the camera movement.

The blur is found only in one direction in a wide range of samples with different levels of aggressiveness. In Figure 35 the test samples can be seen with slight blur from the leftmost sign to an increasingly more noticeable motion blur.

Therefore, motion blur was introduced in the synthetic samples in order to mimic this defect as seen in Figure 36.

Figure 35.: Examples of GTSRB test samples with a high presence of motion blur.

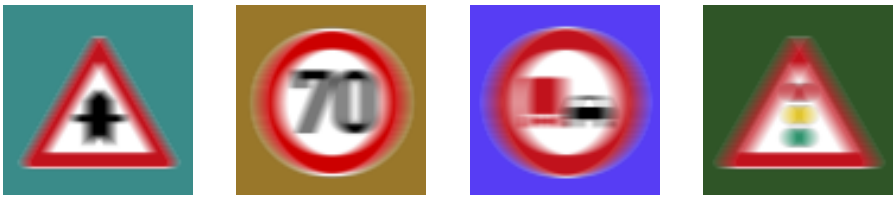


Figure 36.: Examples of synthetic samples with recreated motion blur.

3.4.2 *Confetti noise*

A large portion of the small samples of the GTSRB dataset have abrupt pixel color deviations between adjacent pixels. The most affected classes are speed limits whose numeric text is commonly pixelated in very small samples (Figure 37). Some of these samples affect the overall accuracy.

A first attempt to improve the classification of small samples, due to these defects, was introducing Gaussian noise (more details in Section 4.7.2). However, the visual resulting effect proved to be difficult for the CNN to deal with. Therefore, another algorithm was developed that focused in small samples. The algorithm simulates an impulsive noise which is resistant to resizing. This noise, confetti noise, modifies the value of the pixels in a random fashion while being only projected to small samples. The algorithm needs an image with a resolution higher than the target image size as input. Afterwards, instead of a pixel noise, a rectangle is applied with a random color with a probability into the original template. Each window is applied with a fixed minimum spacing to avoid an uneven noise and overlapping rectangles. Finally, the template is resized to the desired resolution.

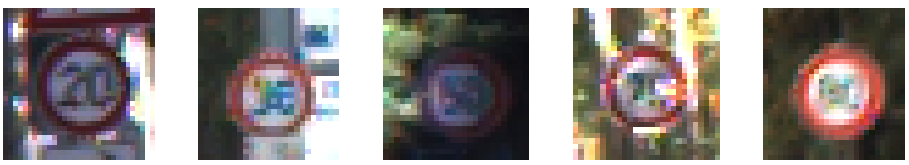


Figure 37.: Examples of noisy traffic sign samples of classes 0, 1, 2, 3, and 4 from the GTSRB dataset, as presented.

(a) Original template (b) Noise before resize (c) Noise after resize

Figure 38.: Confetti noise applied to class 5 of the GTSRB dataset.

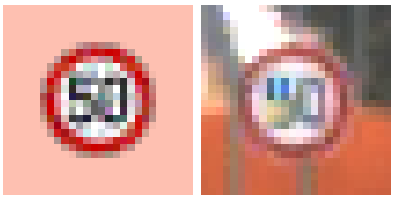


Figure 39.: Comparison of confetti noised template of class 5 (at the left) and an actual traffic sign from the GTSRB test set (at the right).

An example of how the algorithm affect the templates can be seen in Figure 38. A comparison with an actual traffic sign sample and a noised synthetic sample can be observed in 39.

3.4.3 *Perlin noise*

The synthesized traffic signs do not include any shade or color intensity variation within the same color whereas natural samples are highly heterogeneous, consisting of differences produced by an uneven light exposure, color fade, or noise. In order to simulate variations of color shade found in natural traffic signs, Perlin noise was added to the templates.

The process of applying Perlin noise onto the template consists in alpha compositing with a ratio of 60% for the template and 40% for the noise for each generated synthetic traffic sign. This ratio emphasizes the characteristics of the traffic sign relative to the Perlin noise sample. The noise is applied to all samples.

Since this type of gradient noise is computational expensive, it is not viable to generate a new Perlin noise sample for each traffic sign while being able to finish the generation of the synthetic data in useful time. Therefore, a sample with a resolution of 2048×2048 pixels is created once at the beginning of the process, as seen in Figure 40.

To apply noise to a template, a window of the noise image is extracted with 512×512 pixels which is later resized to the template resolution. The position of the extracted sample from the noise is random in order to increase variability. Figure 41 shows examples of Perlin noise applied to different templates.

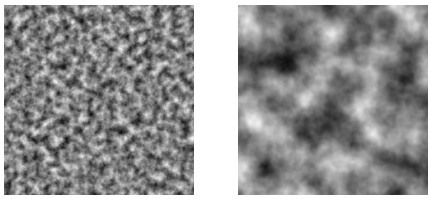


Figure 40.: Generated Perlin noise sample with 2048 × 2048 pixels and a randomly extracted sample of 512 × 512 pixels to be applied to templates.

Figure 41.: Perlin noise sample applied to classes 1, 36, and 41 from the GTSRB dataset.

Note that the intensity of the noise is also influenced by the brightness when its adjustment occurs. For instance, increasing the brightness does attenuate the noise in already bright areas as Figure 42 evidences.

Introducing Perlin noise proved to be a successful decision which is then complemented with brightness adjustment to increase even further the diversity of the dataset.

Shadows

In order to simulate the effect of shadows produced by tree leaves, vegetation, or buildings found in real traffic sign samples, synthetic shadows based in Perlin noise were introduced in the dataset.

To create a shadow pattern, the Perlin noise sample alpha channel is altered to just two values, meaning that consists in fully transparent zones and fully opaque zones. Then, the opacity of the extracted shadow sample is reduced. Finally, before being pasted to the template, the shadow is resized for the target resolution.

Figure 42.: Perlin noise sample applied to class 1 of the GTSRB dataset followed by a brightness increase.

Figure 43.: Perlin shadow sample applied to classes 10 and 14 of the GTSRB dataset.

To be able to create a variety of shadows, different windows are extracted from the processed Perlin noise sample with a resolution of 512×512 pixel. Therefore, each shadowed synthetic sample has a different shadow pattern. The final effect can be seen in Figure 43.

4

EXPERIMENTS ANALYSIS AND RESULTS

This chapter reports on the extensive testing performed during this thesis in order to devise a pipeline of transformations and data augmentation required to successfully craft a synthetic dataset that achieves a state-of-the-art performance on several levels.

Initially, in section 4.1 the neural network architectures used throughout this thesis are detailed.

Prior to digging in the work related to the synthetic dataset, a cross-test with datasets from three countries is presented in Section 4.2. This test aims to demonstrate the practical requirement to develop datasets specific for each country.

The remaining sections deal with the construction and testing of the synthetic dataset. The tests are performed cumulatively, i.e., each section provides a dataset to be tested in the next section.

First, the issue of selecting the backgrounds is discussed and tested in Section 4.3. Geometric transformations are presented next. First, the basic operations: translation, rotation, and scale are introduced (Section 4.4), allowing to define a control dataset that will allow the impact assessment of each operation introduced afterwards. Then shear and perspective transform (Section 4.5) are introduced.

Color transformation is addressed in Section 4.6, assessing the impact of hue and saturation jittering, and brightness adjustment.

To close the gap between synthetic and real data, the following two sections explore and assess methods to increase the similarity between synthetic and real samples. In Section 4.7 noise is introduced, and Section 4.8 deals with the simulation of occlusion and shadows.

The crafting of the final synthetic dataset is presented in Section 4.9, and the results achieved are detailed in Section 4.10.

Regarding the tests, for each, a set of three runs was performed, and the average and standard deviation accuracy are presented.

4.1 NEURAL NETWORK ARCHITECTURES

In order to evaluate the performance of the synthetic dataset and employed techniques, two different network architectures were devised. The first CNN architecture is simpler in order to train in useful time but does not produce state-of-the-art results (Section 4.1.1). It is a topology aimed for a fast training time while still being able to produce satisfactory results.

In contrast, the second architecture (Section 4.1.2) is close to state-of-the-art approaches, allowing to evaluate the full potential of synthetic data. This network requires a far longer time to train, hence it will only be used in the final tests.

Both architectures have a low number of parameters when compared to the state-of-the-art approaches reported in Section 2.2. The purpose of having a low number of parameters is twofold. First we demonstrate that even with a low number of parameters it is possible to achieve state-of-the-art results, while having a network that is far simpler than those used to obtain current state-of-the-art accuracies. Second, the number of parameters has a direct influence on the applicability of such networks in embedded and mobile systems.

4.1.1 *Fast training*

As said before, in order to be able to perform exhaustive tests on the techniques employed in the synthesis process, it is necessary to design a CNN architecture with a low training time. It is also necessary to achieve a reasonable performance, otherwise, the results will not be reliable.

As such, the CNN architecture consists of only 3 convolutional layers with only a single fully connected layer. In order to provide a degree of generalization and avoid overfitting, dropout layers were added as a regularization mechanism. In addition, max-pooling was also added in order to reduce the dimensionality of the input representation.

The CNN architecture is shown in Table 6. Some parameters have been omitted. The dropout rate is 15% for all convolutional layers and 40% for the fully connected layer. Finally, all the layers use ReLU as activation function except the last, where Softmax is applied. The total number of parameters is 1559211. This architecture is based on the work of Novais and Fernandes (2017).

Finally, in order to evaluate the performance of the architecture, the GTSRB dataset is used for benchmark purposes. For this test, the conventional partition of the GTSRB was used consisting in 39209 samples for training and the remaining 12630 samples for testing. The samples were resized to 32×32 pixels and kept in RGB color mode. No data augmentation was applied. The samples were not cropped to the ROI. A batch size of 128 images was used. A set of 10 runs was performed in order to assess the average accuracy and

Table 6.: Fast training CNN architecture used for synthetic data training. The total number of parameters is 1559211.

Layer Type	Kernel	Output Shape	Parameters
Input		$32 \times 32 \times 3$	
Convolution	5×5	$28 \times 28 \times 64$	4864
Convolution	5×5	$24 \times 24 \times 128$	204928
Max Pooling	2×2	$12 \times 12 \times 128$	
Dropout		$12 \times 12 \times 128$	
Convolution	5×5	$8 \times 8 \times 256$	819456
Max Pooling	2×2	$4 \times 4 \times 256$	
Dropout		$4 \times 4 \times 256$	
Fully Connected		128	524416
Dropout		128	
Fully Connected		43	5547

standard deviation of the architecture which achieved $98.25 \pm 0.17\%$. From the total of runs performed, the best result was 98.50%.

4.1.2 *Improved*

The improved architecture is more complex and includes a STN, a module that can be inserted into a standard CNN architecture to provide spatial transformations increasing spatial invariance. This allows affine transformations such as cropping, translating, rotating, scaling, and skewing the samples dynamically in execution time in a non-supervised manner improving the robustness of the architecture to geometric sample variation (Jaderberg et al., 2015).

However, when training with the synthetic dataset this module is not used as it proved to have a negative impact in the accuracy levels obtained. Nevertheless, as the best performance obtained when training with the GTSRB training set is obtained with the STN, and to provide a fair comparison, we report the results with the STN when considering this training set.

The network is based on the architecture found in the GitHub repository Hiranandani (2018) with some adaptations that proved successful when training the synthetic dataset and did not affect negatively the training process with the GTSRB dataset.

The architecture is presented in Table 7. The main difference regarding the original architecture is the use of 5×5 kernels in every convolutional layer, as opposed to 3×3 kernels in the last two convolutional layers as in the original work.

This architecture is able to deliver an average accuracy of 99.70% (12592/12630) on the GTSRB data set which is close to the state-of-the-art accuracy of 99.81%. Dynamic data

Table 7.: Best neural network with a total of 2736943 trainable parameters.

Layer Type	Output Shape	Parameters
Input	$32 \times 32 \times 3$	
Convolution	$28 \times 28 \times 100$	7600
Batch Normalization	$28 \times 28 \times 100$	200
Dropout	$28 \times 28 \times 100$	
Convolution	$24 \times 24 \times 150$	375150
Max Pooling	$12 \times 12 \times 150$	
Batch Normalization	$12 \times 12 \times 150$	300
Dropout	$12 \times 12 \times 150$	
Convolution	$8 \times 8 \times 250$	937750
Max Pooling	$4 \times 4 \times 250$	
Batch Normalization	$4 \times 4 \times 250$	500
Dropout	$4 \times 4 \times 250$	
Fully Connected	350	1400350
Fully Connected	43	15093

augmentation in form of geometric and color transformations is used during training. The data augmentation operations applied can be seen in more detail in Section 4.10.1.

4.2 Cross Test Between European Datasets

Considering a particular dataset, real traffic signs are usually collected in the same conditions for both the training and testing sets. Unintentionally, all the samples may have similar features due to the fact they are collected by the same camera(s), in the same day, or in geographically close areas. For instance, the weather influences greatly as the samples brightness varies depending on the sunlight exposure. In fact, depending on the time of the day, the samples may have a great predisposition to shadows or even contain none. Another factor to take in account is the fact that, in the GTSRB dataset, the samples are extracted from video tracks of, usually, 1 second. That is, the same traffic sign does appear several times but with different scales, orientations, perspectives, or camera exposure conditions.

A cross test was performed between the German, Belgian and Croatian datasets. The improved CNN architecture was trained with the training set of a specific country but evaluated against the test set of the other countries. Two different approaches were used such as maintaining only the equal classes, and admitting not only equal but also classes with slight pictogram differences. For example, in Figure 44 are shown examples of small differences that are admitted (Figure 44b) and just equal classes (Figure 44a). All the classes that were considered equal and similar for testing purposes are detailed in Appendix Table A.1.

(a) Equal class

(b) Similar classes

Figure 44.: Examples of classes from the German, Belgian, and Croatian datasets, respectively, considered equal and similar.

Table 8.: Resulting accuracy (%) of cross evaluating the test set between European countries restricted to only equal classes.

	Evaluated test set		
Country	Germany	Belgium	Croatia
Germany	99.70	98.28	99.93
Belgium	92.13	98.57	80.15
Croatia	95.57	98.72	99.20

For each country, three models were trained with different random seeds. The results presented are an average of the correct predictions obtained with the three models.

In Table 8 is reported a test with just equal classes where is possible to observe that the accuracy decreases in almost every test set from other countries. The German training set sustains the best average decreased accuracy between the different countries while the CNNs trained with the Belgium dataset shows the poorest performance on the Croatian test set.

When admitting test sets from other countries with, not only equal, but also classes with small differences in pictograms as well, the accuracy decreases even further as presented in Table 9. For instance, the Belgium dataset suffers a decrease of approximately 36% when it is being tested on the Croatian test set.

Table 9.: Resulting accuracy (%) of cross evaluating the test set between European countries admitting equal and classes with small differences.

	Evaluated test set		
Country	Germany	Belgium	Croatia
Germany	99.70	98.20	98.49
Belgium	74.34	98.57	62.70
Croatia	89.91	76.57	99.20

This is an indicator that the data sets are unprepared for other countries due to the lower sample count and lack of diversity. Even considering the German dataset, which is by far the largest, it is also clearly visible a decrease in accuracy.

Note that this test should not be seen as a comparison between datasets. Further testing would be required to be able to do so. Nonetheless, there seems to be a correlation between the number of training samples and the accuracy achieved in foreign test sets. Another issue that could impact the results is the number of classes. The Belgian dataset has far more classes than the other two sets, which could overburden the network, and make it more susceptible to class confusion when faced with foreign datasets.

These tests show the need for a training set for each country, contemplating each countries pictograms, fonts and font size, among other variations.

4.3 BACKGROUNDS

The background is an essential part of the synthetic sample. Three different type of backgrounds were tested. Real backgrounds, extracted from street views samples, are loaded in 64×64 pixels and resized for the target resolution to be composed with the template. An amount of around 38000 real backgrounds were segmented from 600 street views samples. The remaining two types of backgrounds are synthetic. An advantage of using synthetic backgrounds consists in the fact that it is costless to create any desired amount. The two approaches consisted in generating a solid colored or per pixel colored background.

For this test, the templates just undergo simple geometric transformations such as scales, translations, and rotations. Using solid colored backgrounds revealed to be the best approach with a difference of around 1% in accuracy in comparison to using street views samples as shown in Table 10. The worst approach was using a background in noise format. Thus, all synthetic samples are generated with solid colored synthetic backgrounds.

Table 10.: Accuracy for different types of traffic sign backgrounds composed with templates and simple geometric transformations.

Background	Accuracy (%)
Street views	89.90 ± 0.68
Solid color	91.01 ± 0.58
Pixelated	82.72 ± 1.06

Figure 45.: Small resolution traffic sign examples from the GTSRB test set.

4.4 SIMPLE GEOMETRIC TRANSFORMATIONS

The traffic signs present in real-world training samples contains varying scales which alters the free space in the frame depending on its resolution.

This section presents tests that consists in simple geometric transformations applied to the traffic sign templates. The templates are resized, translated, and rotated. This reproduces the geometric relation between the traffic sign and the canvas found in real images.

The input template resolution has no impact in the final result. The template is resized randomly from 12×12 to 48×48 pixels. In this context, traffic signs in the range [12×12, 17×17] pixels are considered small or low resolution, and those with resolution in the range [18×18, 48×48] pixels are considered large or high resolution.

The margin level also varies randomly. For example, considering low resolution signs these tend to have slightly larger margins, similar to some real samples. A few examples are shown in Figure 45. The background size is the sign dimension plus the margins.

Also, it is relevant to note that the smallest traffic signs of the GTSRB dataset have a resolution of 15×15 pixels. However, due to the fact that templates do not have noise, blur, or any type of artifacts by default, in comparison to an actual traffic signs, templates have more defined and clear features. Reducing its resolution improves the visual similarities to small real traffic signs.

High resolution traffic signs have margins in the range [7, 21]% while smaller traffic signs have margins in the range [20, 25]%. The ratio of high resolution traffic signs to the total amount of samples per class is 80%. The remaining samples are smaller traffic signs. The dataset is balanced with a total of 2000 synthesized samples per class.

Without performing any other transformation, that is, keeping the traffic signs centered, the resulting average accuracy was 85.20 ± 0.02%.

Figure 46.: Examples of traffic signs from the GTSRB dataset correctly classified after inserting rotations.

The next step added translations to every sample in a random manner. A limit was imposed to prevent the sign from exiting the sample. This limit is 15% lateral space of the entire frame for larger traffic signs and 20% for smaller ones. The traffic sign can translate freely in the admissible space. The accuracy improved, as expected, to $90.43 \pm 0.61\%$.

The source code for scales and translations can be seen in Appendix B.1.

Finally, rotations were performed centered on the center of the sign on 70% of all traffic signs. A total of 30% was kept at $0°$. The rotations are performed uniformly in the range $[-15, 15]°$. The accuracy did improve, albeit slightly, achieving an average of $91.01 \pm 0.58\%$.

An example of traffic signs that were incorrectly classified prior to the insertion of rotations can be seen seen in Figure 46.

The implementation details of rotations can be seen in Appendix B.2.

4.5 Further Geometric Transformations

Common transformations are already performed to the templates such as scales, translations and rotations. In addition, other geometric transformations are also applied to the templates consisting in shear and perspective transformations.

4.5.1 *Shear*

Traffic sign shear is applied to 60% of the samples either horizontally or vertically. Shear in form of shrinking is not performed, only stretching. A random factor in the range $[3, 10]\%$ is multiplied to each one of the three chosen points to perform the calculation. A low factor of, for example, 3% performs a subtle shear while a high factor exhibits a noticeable distortion. After adding this transformation alongside the simple geometric transformations the accuracy improved to $91.93 \pm 0.26\%$.

In the GTSRB dataset most skewed samples have a subtle distortion to human eyes. However, for the CNN even subtle shear can have impact in the classification performance. Some of the samples which were incorrectly classified prior to adding shear are shown in Figure

Figure 47.: Traffic signs from the GTSRB dataset correctly classified after performing shear.

Figure 48.: Traffic signs from the GTSRB dataset after performing shear correction manually for demonstration purposes.

47. In the right traffic sign, the shear is more visible in comparison to the traffic sign of the left.

Performing shear correction manually, the difference is more noticeable as shown in Figure 48. In the left traffic sign, the speed limit of 100 km/h, an horizontal is performed while in the right traffic sign, no entry for vehicular traffic, a vertical shear is performed instead.

The implementation details of the shear transformation can be seen in Appendix B.3.

4.5.2 *Perspective*

A perspective transformation is performed to 60% of the samples. This implies that some of the images that suffered shear may also suffer from perspective transformation.

This transformation was able to achieve an accuracy of $91.38 \pm 0.25\%$, just an increase of 0.37% with respect to only adding simple geometric transformations. Note, however, that there are very few real samples that can benefit from this transformation.

The synthesisation method must be as generic as possible. The perspective transformation method addresses mainly aggressive viewpoints distortions.

Even tough perspective transforms may not contribute as much in the German dataset due to the lack of this type of deformations present in its training set, in other datasets it has a clear impact. For instance, in the Belgian dataset some traffic signs suffers aggressive viewpoint distortions. Some of this traffic signs can be correctly classified after inserting perspective transform. In Figure 49 is shown examples of traffic signs that are incorrectly classified in absence of perspective transformations inserted in the synthetic traffic sign dataset.

Figure 49.: Examples of viewpoint distorted traffic signs from the BTSC dataset correctly classified after performing perspective transform.

Figure 50.: Examples of traffic signs from the GTSRB dataset correctly classified after jittering the hue and saturation.

The implementation details can be seen in Appendix B.4.

4.6 COLOR TRANSFORMATIONS

By default, templates exhibit limited hue and saturation diversity. Experiments carried out in the domain of color transformations consisted in color jitter and brightness adjust in form of its value replacement. While jittering hue and saturation aims to simulate aging and faded signs, encompassing a larger range of colors, the brightness adjustment goal is to alter the traffic sign to simulate different light exposures.

4.6.1 *Hue and saturation jitter*

The hue and saturation are performed separately with a probability of 80%. That is, they are not necessarily applied to the same sample. The hue can be adjusted in the range $[-12, 20]^\circ$ assuming a color wheel of 360°. The saturation can be reduced or increased by an uniformly generated factor in the range of $[0.4, 2]$. A factor of 2 will double the saturation while a factor of 0.4 will reduce the saturation of the template by 60%.

Jittering the image color alongside performing geometric transformations was very successful achieving an accuracy of $92.41 \pm 0.34\%$. In Figure 50 is possible to see traffic signs which were not being correctly classified with only geometric transformations and the corresponding templates.

The entire source code for color jittering is shown in Appendix B.5.

Figure 51.: Examples of traffic signs from the GTSRB dataset correctly classified after adjusting the brightness.

4.6.2 *Brightness jitter*

The templates are bright, as expected, while real-world samples present a variety of brightness values. The brightness was adjusted accordingly to generated values that follows a bounded Johnson distribution from the GTSRB dataset with the same parameters of Table 5. Due to the fact the brightness is in the range $[0, 255]$, values higher than 255 are truncated.

Changing the templates brightness by randomly generated values following a distribution and performing, also, geometric transformations was a very successful approach achieving $95.01 \pm 0.56\%$.

Finally, instead of using a distribution from real-world samples, a uniform distribution was employed instead. The generated brightness values were in the range $[7, 255]$ based in the bound values of the previous distribution. The accuracy improved slightly, achieving $95.12 \pm 0.59\%$. Note, however, that this difference can not be deemed significant with only three runs.

Due to the fact the German brightness distribution has an emphasis on darker samples, which are more difficult to classify, the brightness values are generated using that approach. Note, however, that the background is not affected by the brightness replacement method, meaning that the brightness value of the entire final synthetic sample is not equivalent to that generated from the distribution.

It is easy to spot test samples that benefit from this approach, namely, highly dark samples. In Figure 51 are shown some dark samples that are now correctly classified.

Finally, joining color jitter, brightness adjustment, and keeping previously defined geometric transformations was able to improve the accuracy, reaching $96.42 \pm 0.35\%$.

The implementation details are presented in Appendix B.6.

4.7 TEMPLATE DISTURBANCES

This section presents further approaches applied to templates in order to noise them with the intent of closing the gap between synthetic traffic sign and real-world samples.

Figure 52.: Examples of traffic signs from the GTSRB dataset correctly classified after applying motion blur.

4.7.1 *Motion blur*

In order to improve blurred traffic sign recognition, horizontal motion blur is applied to 30% of all traffic signs with high resolutions above and equal to 18×18 pixels on top of previous geometric and color transformations. The motion blur kernel is uniformly generated in the range of $[2 \times 2, 5 \times 5]$ pixels. Blurring is very aggressive to small traffic signs, even with a low kernel, so this transformation was not applied to those samples below the referred resolution.

Performing motion blur was able to improve the recognition rate achieving an accuracy of $96.85 \pm 0.16\%$. Although the improvement in accuracy seems low, this transformation was successful as shown in Figure 52.

The source code for the motion blur routine is detailed in Appendix B.7.

4.7.2 *Confetti noise*

Initially, Gaussian noise was inserted in the synthesization process with the intention of improving low resolution traffic sign classification. A few small samples exhibits artifacts in form of pixel color deviation in relation to their neighbor pixel colors. This samples were being incorrectly classified.

It was possible to assess that the real noise that this type of samples presents were being an obstacle to the CNN as other, even smaller, samples were being correctly classified.

The noise was added to 50% of the small samples, with a kernel size of 3×3 pixels. The previous transformations were kept, that is, geometric, color, and motion blur transformations. However, the accuracy decreased slightly. It is possible to notice that Gaussian noise is very noticeable in the image pictogram in some cases (Figure 53).

As an alternative, confetti noise was developed. Confetti noise is less aggressive than Gaussian noise and blends easier with the template decreasing the abrupt color deviance in the traffic sign pictogram.

The confetti noise was also applied to half of the small samples. The noise function has three parameters. The kernel size ratio is, by default, 3% of the original template height and width. For instance, for a template with a dimension of 512×512 pixels, the kernel has

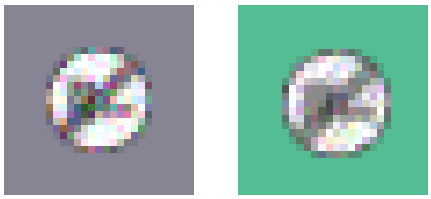


Figure 53.: Visual differences between Gaussian noise (left sample) and confetti noise (right sample) applied to class 42 from the GTSRB dataset.

Figure 54.: Examples of traffic signs from the GTSRB dataset correctly classified after applying confetti noise. Samples from classes 1 and 5, respectively.

a dimension of 16×16 pixels. The second parameter is the probability of applying noise using the kernel area which by default is 3%. Finally, the last parameter is the minimum spacing between kernels which is 1.5% the template height and width.

Even tough the accuracy did not show a clear increase, maintaining an accuracy of $96.90 \pm 0.02\%$, it did, in fact, improved the classification of a few, very specific cases as shown in Figure 54. As it is possible to confirm, the samples exhibit a high level of noise deforming the pictogram. Notice that these samples were not classified correctly in any of previous runs.

The source code for the confetti noise generation routine is detailed in Appendix B.8.

4.7.3 *Perlin noise*

Perlin noise was added to the templates in order to improve their texture. Templates do not contain any type of pixel color variation within the same hue. In fact, even when jittering the hue of the traffic sign, the resulting color is still uniform.

The process of applying Perlin noise to the templates consists in alpha blending with $\alpha = 0.60$ in order to highlight the template in comparison to the Perlin texture. As mentioned before, due to hardware limitations, a sample of Perlin noise of 2048×2048 pixels is generated in the beginning of the synthesization process and for each template, a window

Figure 55.: Examples of traffic signs from the GTSRB dataset that benefited from Perlin noise insertion.

of 512×512 pixels is extracted with random position. The Perlin noise parameters are as follows: 6 octaves, a persistence of 0.5, and a lacunarity of 2.0.

Adding Perlin noise was able to achieve an accuracy of $98.15 \pm 0.16\%$. The best best result obtained was 98.42%.

There are traffic sign samples that clearly benefited from Perlin noise as shown in Figure 40. Usually traffic signs with texture irregularities due to dirt, fading, or simply uneven light exposure.

The implementation details can be found in Appendix B.9.

4.8 Occlusions and shadows

Real traffic sign samples are susceptible to both occlusions and strong shadows. In an attempt to reproduce these phenomena, random erasing is studied as a mean to simulate occlusions; and Perlin noise based shadows are crafted in order to cope with this problem.

4.8.1 *Random erasing*

This technique aims to improve occlusions and vandalism invariance, by selecting a rectangle region in the traffic sign sample and erasing its pixels with random values.

Random erasing was applied to 70% of the traffic sign samples with dimensions higher and equal than 18×18 pixels. Only a block was erased per sample.

This approach, however, did not seem to improve the accuracy when testing on the GTSRB dataset. The resulting accuracy was $98.13 \pm 0.18\%$. Complicated occluded traffic signs continued being incorrectly classified, e.g., two commonly cases are shown in Figure 56.

The application of this technique requires further study both on the shape and dimensions of the area to be "erased" as well as the erasing pattern.

The implementation details can be found in Appendix B.10.

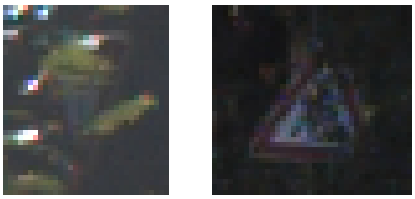


Figure 56.: Incorrectly classified traffic sign from classes 13 and 25 of the GTSRB dataset due to a presence of strong occlusions.

Figure 57.: Misclassified traffic sign from class 27 of the GTSRB dataset due to a presence of a strong shadow.

4.8.2 *Shadows*

Observing traffic sign samples can be noticed that shadows are highly present. A large portion of samples that exhibits light shadows are correctly classified. However, in some signs this effect is too large for the CNN to cope, as seen in Figure 57. In this particular case, the entire track is misclassified.

The synthetic shadows are Perlin based. The noise is converted to a 4 channel image. Resorting to the alpha channel, the intensity of the noise is altered for just two values. Pixels with an alpha value higher than a defined threshold of 120 have that same value set to 255, that is, fully opaque. The remaining pixels are replaced with 0, that is, fully transparent. For each sample, a factor in the range $[0.50, 0.75]$ is multiplied by the alpha channel in order to create variable transparency. Finally, the shadow is pasted onto the the traffic sign. Shadows were applied to a third of the samples of high resolution.

Inserting shadows as described above decreased the accuracy to $97.98 \pm 0.18\%$. This could be due to the percentage of samples being too high, or the Perlin noise parameterization. Further study is required to decide whether this transformation is useful in the context of the generation of a synthetic dataset.

The implementation details can be found in Appendix B.11.

4.9 SYNTHESISATION PIPELINE

Based on the previous sections a set of transformations was selected based on their impact on performance. The transformations that contribute to the synthesization process are

Figure 58.: Generated synthetic traffic sign sample for each class of the GTSRB dataset. The classes are labeled from left-to-right and top-to-bottom.

Table 11.: Average accuracy for the the combined transformations of the synthesization pipeline.

Techniques	Accuracy (%)
Simple geometric transformations	91.01 ± 0.58
Further geometric transformations	91.93 ± 0.26
Color jitter	92.42 ± 0.34
Brightness adjust	96.42 ± 0.35
Motion blur	96.85 ± 0.16
Confetti noise	96.90 ± 0.02
Perlin noise	98.15 ± 0.16

combined in Table 11. A diagram representing the pipeline of transformations with the probabilities for each stage is presented in Figure 59.

Finally, in Figure 58 is shown an example of a generated synthetic traffic sign sample for each class of the GTSRB dataset.

The implementation of the full pipeline can be found in Appendix B.12.

4.10 Final Training Procedure and Results

To evaluate the full potential of synthetic data against real-world data, the more complex architecture is used in this section. This section covers training related aspects as the dynamic data augmentation (Section 4.10.1), and usage scenarios for a synthetic dataset with the final results presented (Section 4.10.2).

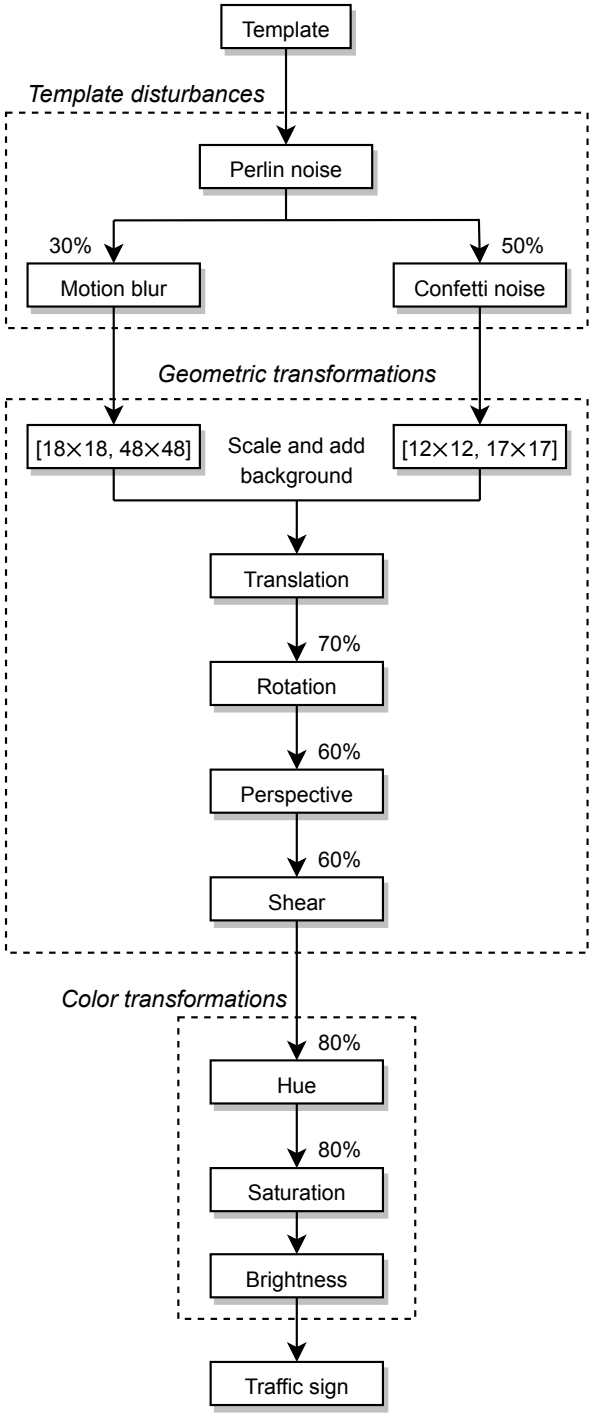


Figure 59.: Synthetic traffic sign generation pipeline.

4.10.1 *Dynamic data augmentation*

Besides upgrading the CNN topology for the most complex, close to state-of-the-art results architecture, dynamic data augmentation in form of geometric transformations and color jittering were applied to the already created synthetic dataset. Dynamic data augmentation has the advantage of being generated per epoch meaning that the neural network is seeing new data and, therefore, being force to cope with new features. By feeding new data per epoch, the neural network is forced to learn more representative features.

Therefore, nine different types of data augmentation are applied to each sample of the original set representing an increase in 9 times the volume of samples per epoch. Due to the fact the synthetic data set consists in 2000 samples per class or a total of 86000 samples, the CNN is trained with a set of 774000 samples. From those samples, only 86000 are repeated each epoch since they represent the original synthetic dataset. The new volume of samples increases immensely the training time.

Finally, both the geometric transformations and color jittering have their parameters randomized each time new data is created. The geometric transformations that consists in affine transformations kept the center invariant. The transformations are made up of:

- Small rotations with a maximum of 5° to each direction;
- Shear in the range of $[-2, 2]$ pixels composed with rotations;
- Translations in a range of $[-3, 3]$ pixels, also composed with rotations;
- Center cropping of 28×28 pixels.

The color transformations consists in jittering. The brightness, saturation, and contrast can be increased or decreased in the range of $[0, 3]$ times the original value while the hue can be jittered 0.4 times the original value.

It is important to notice that due to the fact the number of transformations and its parameters is high, the neural network will not be able to perform so well in the training set. However, the overwhelming quantity of new data per epoch is actually beneficial as it improves the invariance to, for instance, aggressive colors or geometric deformities.

Full source code for the data augmentation can be found in Appendix B.13.

4.10.2 *Usage scenarios and final results*

The synthetic dataset can be used as a standalone dataset. When tested with templates for the GTSRB achieved an accuracy of 99.406%, or 12555/12630 samples correctly classified. This clearly surpasses the current state-of-the-art of 92.20% (Stergiou et al., 2018) regarding synthetic datasets. The value is close to what is achievable with the same network with

Table 12.: Best resulting accuracy (%) of training the German, Belgium, and Croatian datasets with synthetic data and evaluated them with the corresponding real-world test set.

Source	Architecture	GTSRB	BTSC	rMASTIF
Real	State-of-the-art	99.81 Haloi (2015)	99.17 Saha et al. (2018)	99.53 Jurišić et al. (2015)
	Fast training	98.50	98.33	98.26
	Improved	99.70	98.61	99.33
Synthetic	Fast training	98.42	97.82	98.43
	Improved	99.41	98.96	98.88

real data, 99.70, or 12592/12630 samples correctly classified. Therefore, in total absence of actual traffic sign samples, the synthetic approach is capable of delivering satisfactory results.

We also assess the performance of synthetic data extending the tests to the Belgian and Croatian datasets. The following accuracies refer to the best run on both the fast training and the improved neural networks. The state-of-the-art accuracy is also mentioned for direct comparison. The accuracy is round to two decimal places. The summary results can be seen in Table 12.

The exact accuracy for the Belgian test set is 2494/2520 and for the Croatian test set is 1764/1784 with the improved network. Note that the results for the mentioned datasets may not embrace all possible templates. The generation parameters were the same for each dataset.

Another usage scenario is when there is real data available and the synthetic data is used as a complement to the training set, i.e. the creation of a mixed training set. To evaluate this scenario a merge of the train set of the GTSRB with the synthesised data was performed. The real-world traffic sign dataset was balanced to a minimum of 2000 samples per class, resorting to data augmentation in form of geometric transformations. On the other hand, the synthetic portion of the dataset consists in 2000 samples per class. Thus, the final dataset has a volume of 170856 samples. On top of that, dynamic data augmentation is also performed resulting in an 8 fold increase of the data. This approach matches the state-of-the-art accuracy of 99.809%, or 12606/12630, presented in Haloi (2015). Our results can be seen in Table 13.

Note that this result is achieved with a network with a much smaller number of parameters. While Haloi (2015) uses a network with 10.5 million parameters, the network used to achieve our result has less than 2.8 million which is clearly a considerable simpler network.

Finally, this approach was extended for the Belgian and Croatian datasets as seen in Tables 14 and 15. As in the previous case, the datasets were also balanced in the same proportion before merging with synthetic data. For both these datasets, the state-of-the-art

Table 13.: State-of-the-art classification results for the GTSRB dataset.

Dataset	Literature	Architecture	Parameters $\times 10^6$	Correct /12630	Acc (%)
Real	Cireşan et al. (2012)	25 CNNs committee	1.1 p/ CNN	12562	99.46
	Arcos-García et al. (2018)	3 STNs	14.6	12594	99.71
	Haloi (2015)	4 STNs, 9 Inception	10.5	12606	99.81
Merged	Ours	Single CNN	2.7	12606	99.81

Table 14.: State-of-the-art classification results for the BTSC dataset.

Dataset	Literature	Architecture	Parameters $\times 10^6$	Correct /2520	Acc (%)
Real	Arcos-García et al. (2018)	3 STNs	14.6	2492	98.87
	Saha et al. (2018)	Single CNN w/ dilated convolutions	6.3	2499	99.17
Merged	Ours	Single CNN	2.7	2509	99.56

results reported in Section 2.2 were surpassed. Furthermore, the number of parameters in our network is again clearly inferior to the previous reports in these datasets. Although we do not have the number of parameters for the network used in Jurišić et al. (2015), our network is far simpler, both regarding the input shape (32×32 versus 48×48), as well as not containing branches, and the number of fully connected layers.

As conclusion it is clear that synthetic signs are a promising approach to create traffic sign training sets, both as standalone, or as a complement when real data is available.

Table 15.: State-of-the-art classification results for the rMASTIF dataset.

Dataset	Literature	Architecture	Parameters $\times 10^6$	Correct /1784	Acc (%)
Real	Jurišić et al. (2015)	Multi-scale CNN	–	–	99.53
Merged	Ours	Single CNN	2.7	1784	100

5

CONCLUSION

Extensive research has been done on traffic sign classification. With a few exceptions, the usage of real samples prevails in the range of the works discussed in Section 2.2 whose main focus is based mostly on the architecture of the neural networks. Although data augmentation is commonly used, no effort is clearly visible to explore the particularities of the traffic sign recognition problem.

The few works focusing on synthetic data show that crafting a successful dataset is not a trivial task. In practice, there are several pictograms for each traffic sign in use, and collecting them was not simple due to the language barrier and absence of less frequent pictograms. However, this is not a real problem for the automotive industry, and national road traffic agencies.

The main challenge relates to creating synthetic samples that are sufficiently similar to real ones so that the network can successfully classify real samples. In the proposed synthesisation pipeline, besides the traditional color and geometric operations performed on data augmentation, we introduced Perlin Noise to increase the "realism" of the synthetic samples. Furthermore, we account for shadows and motion blur which are not found in previous works.

After synthesising the training set, we proposed two usage scenarios: using the synthetic dataset as a standalone training set, or, when real data is available, perform a merge of the real and synthetic training sets.

While our tests show that there is still room for improvement, we were able to achieve results close to the state-of-the-art with a synthetic standalone training set on three different national test sets.

When considering the merging of real and synthetic data we surpassed state-of-the-art results, with the major advantage of achieving these results with a much simpler architecture.

In summary, creating synthetic datasets is an approach worth exploring since it can allow the creation of datasets without having to collect and label thousands of samples of real traffic signs. Furthermore, synthetic datasets deal much better with the issue of new pictograms being released for the same signs, or even new traffic signs, as in this case get-

ting a significant amount of data is not feasible until a large number of signs is placed on the streets.

On the other hand, when merged with real training sets, the synthetic dataset can be seen as a procedure for data augmentation that is more successful than current approaches, as demonstrated by our state-of-the-art results, and with simpler architectures.

Yet, we believe that the full potential of synthetic training sets is still unreached. There is clear room for improvement and we hope to pursue this work further to enhance the synthesisation pipeline.

5.1 FUTURE WORK

In order to further improve the synthetic dataset we propose a deeper study of the transformations which were not successful in our first approach. Shadows simulation and random erasing are two examples that require further study.

Exploring further transformations to simulate night time, rain, snow, and fog are also required if these datasets are ever to be used in real scenarios.

Finally, exploring a usage scenario that was not covered in this thesis: ensembles of networks. These can be ensembles with pure synthetic data or both real and synthetic data.

BIBLIOGRAPHY

Deep image: Scaling up image recognition. *CoRR*, abs/1501.02876, 2015. URL http://arxiv.org/abs/1501.02876. Withdrawn.

Álvaro Arcos-García, Juan Alvarez-Garcia, and Luis M. Soria-Morillo. Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods. *Neural Networks*, 99, 01 2018. doi: 10.1016/j.neunet.2018.01.005.

Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333 – 338, 2012. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2012.02.023. URL http://www.sciencedirect.com/science/article/pii/S0893608012000524. Selected Papers from IJCNN 2011.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015. URL http://arxiv.org/abs/1511.07289.

Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016. URL http://arxiv.org/abs/1605.06409.

Adrian Galdran, Aitor Alvarez-Gila, Maria Inês Meyer, Cristina López Saratxaga, Teresa Araujo, Estíbaliz Garrote, Guilherme Aresta, Pedro Costa, Ana Maria Mendonça, and Aurélio J. C. Campilho. Data-driven color augmentation techniques for deep skin image analysis. *CoRR*, abs/1703.03702, 2017. URL http://arxiv.org/abs/1703.03702.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012a.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. pages 3354–3361, 05 2012b. ISBN 978-1-4673-1226-4. doi: 10.1109/CVPR.2012.6248074.

J. Greenhalgh and M. Mirmehdi. Real-time detection and recognition of road traffic signs. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1498–1506, Dec 2012. ISSN 1524-9050. doi: 10.1109/TITS.2012.2208909.

Mrinal Haloi. Traffic sign classification using deep inception based convolutional networks. *CoRR*, abs/1511.02992, 2015. URL http://arxiv.org/abs/1511.02992.

A. Hanel, D. Kreuzpaintner, and U. Stilla. Evaluation of a traffic sign detector by synthetic image data for advanced driver assistance systems. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2:425–432, 2018. doi: 10.5194/isprs-archives-XLII-2-425-2018. URL https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-2/425/2018/.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

Pooja Hiranandani. Pytorch implementation of gtsrb classification challenge. https://github.com/poojahira/gtsrb-pytorch, 2018.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. URL http://arxiv.org/abs/1506.02025.

Bitao Jiang, Xiaofeng Ma, Yao Lu, Yang Li, Li Feng, and Zhenwei Shi. Ship detection in spaceborne infrared images based on convolutional neural networks and synthetic targets. *Infrared Physics Technology*, 97:229 – 234, 2019. ISSN 1350-4495. doi: https://doi.org/10.1016/j.infrared.2018.12.040. URL http://www.sciencedirect.com/science/article/pii/S1350449518306352.

F. Jurišić, I. Filković, and Z. Kalafatić. Multiple-dataset traffic sign classification with onecnn. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 614–618, Nov 2015. doi: 10.1109/ACPR.2015.7486576.

Fredrik Larsson and Michael Felsberg. Using fourier descriptors and spatial models for traffic sign recognition. In Anders Heyden and Fredrik Kahl, editors, *Image Analysis*, pages 238–249, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-21227-7.

K. Lee and D. Moloney. Evaluation of synthetic data for deep learning stereo depth algorithms on embedded platforms. In *2017 4th International Conference on Systems and Informatics (ICSAI)*, pages 170–176, Nov 2017. doi: 10.1109/ICSAI.2017.8248284.

Jing Li, Jinan Gu, Zedong Huang, and Jia Wen. Application research of improved yolo v3 algorithm in pcb electronic component detection. *Applied Sciences*, 9:3750, 09 2019. doi: 10.3390/app9183750.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. URL http://arxiv.org/abs/1512.02325.

M. Mathias, R. Timofte, R. Benenson, and L. Van Gool. Traffic sign recognition — how far are we from the solution? In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Aug 2013. doi: 10.1109/IJCNN.2013.6707049.

Andreas Møgelmose, Mohan Manubhai Trivedi, and Thomas B. Moeslund. Learning to detect traffic signs: Comparative evaluation of synthetic and real-world datasets. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3452–3455, 2012.

A. Møgelmose, M. M. Trivedi, and T. B. Moeslund. Learning to detect traffic signs: Comparative evaluation of synthetic and real-world datasets. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3452–3455, Nov 2012.

H. Novais and A. R. Fernandes. Community based repository for georeferenced traffic signs. In *2017 24º Encontro Português de Computação Gráfica e Interação (EPCGI)*, pages 1–8, Oct 2017. doi: 10.1109/EPCGI.2017.8124297.

Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017. URL http://arxiv.org/abs/1712.04621.

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

Sourajit Saha, Sharif Amit Kamran, and Ali Shihab Sabbir. Total recall: Understanding traffic signs using deep hierarchical convolutional neural networks. *CoRR*, abs/1808.10524, 2018. URL http://arxiv.org/abs/1808.10524.

P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale convolutional networks. In *The 2011 International Joint Conference on Neural Networks*, pages 2809–2813, July 2011. doi: 10.1109/IJCNN.2011.6033589.

J. Shijie, W. Ping, J. Peiyi, and H. Siping. Research on data augmentation for image classification based on convolution neural networks. In *2017 Chinese Automation Congress (CAC)*, pages 4165–4170, Oct 2017. doi: 10.1109/CAC.2017.8243510.

Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, Jul 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0197-0. URL https://doi.org/10.1186/s40537-019-0197-0.

J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0):–, 2012. ISSN 0893-6080. doi: 10.1016/j.neunet.2012.02.016. URL http://www.sciencedirect.com/science/article/pii/S0893608012000457.

Alexandros Stergiou, Grigorios Kalliatakis, and Christos Chrysoulas. Traffic sign recognition based on synthesised training data. *Big Data and Cognitive Computing*, 2(3), 2018. ISSN 2504-2289. doi: 10.3390/bdcc2030019. URL http://www.mdpi.com/2504-2289/2/3/19.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

R. Timofte, K. Zimmermann, and L. V. Gool. Multi-view traffic sign detection, recognition, and 3d localisation. In *2009 Workshop on Applications of Computer Vision (WACV)*, pages 1–8, Dec 2009. doi: 10.1109/WACV.2009.5403121.

Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. *CoRR*, abs/1804.06516, 2018. URL http://arxiv.org/abs/1804.06516.

Shui-Hua Wang, Yi-Ding Lv, Yuxiu Sui, Shuai Liu, Su-Jing Wang, and Yu-Dong Zhang. Alcoholism detection by data augmentation and convolutional neural network with stochastic pooling. *Journal of Medical Systems*, 42(1):2, Nov 2017. ISSN 1573-689X. doi: 10.1007/s10916-017-0845-x. URL https://doi.org/10.1007/s10916-017-0845-x.

Wikipedia. Bildtafel der Verkehrszeichen in der Bundesrepublik Deutschland seit 2017 — Wikipedia, the free encyclopedia. http://de.wikipedia.org/w/index.php?title=Bildtafel%20der%20Verkehrszeichen%20in%20der%20Bundesrepublik%20Deutschland%20seit%202017&oldid=183841356, 2019. [Online; accessed 15-January-2019].

Ali Youssef, Dario Albani, Daniele Nardi, and Domenico Daniele Bloisi. Fast traffic sign recognition using color segmentation and deep convolutional networks. In Jacques Blanc-Talon, Cosimo Distante, Wilfried Philips, Dan Popescu, and Paul Scheunders, editors,

Advanced Concepts for Intelligent Vision Systems, pages 205–216, Cham, 2016. Springer International Publishing. ISBN 978-3-319-48680-2.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. 11 2015.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *CoRR*, abs/1708.04896, 2017. URL http://arxiv.org/abs/1708.04896.

S. Šegvic, K. Brkić, Z. Kalafatić, V. Stanisavljević, M. Ševrović, D. Budimir, and I. Dadić. A computer vision assisted geoinformation inventory for traffic infrastructure. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 66–73, Sep. 2010. doi: 10.1109/ITSC.2010.5624979.

A

TABLES

A.1 CROSS TEST BETWEEN EUROPEAN DATASETS

The classes considered equal and similar between the German, Belgian, and Croatian datasets, for cross testing purposes, are detailed in Table 16.

Table 16.: Classes considered equal and slightly different from German, Belgium, and Croatian traffic sign datasets.

Considered	Germany	Belgium	Croatia
Equal	12	61	16
	13	19	14
	14	21	15
	15	28	
	17	22	
	18		0
	23		9
	26	11	
	31		12
	38		26
		14	8
		41	23
Similar	1		17
	2		19
	3		20
	4		21
	9	31	22
	11	17	1
	16	25	
	19	3	4
	20	4	5
	21	5	6
	22	0	
	24	16	
	25	10	
	35	34	
	36	36	
	40	37	
	41		27
		6	7
		38	24
		39	25
		45	29

B

ROUTINES IMPLEMENTATION DETAILS

B.1 SCALE, TRANSLATE, AND ADD BACKGROUND

Code for template scale, translation, and the process of adding a background, see Section 4.4.

```
# all necessary imports
import datetime
import glob
import multiprocessing as mp
import os
import re
from random import randint, uniform, random

import cv2
import noise
import numpy as np
import skimage
from PIL import Image, ImageEnhance
from scipy import stats

# paste a foreground to a background
def paste(foreground, background, position=(0, 0)):
    output = Image.fromarray(background)
    input = Image.fromarray(foreground)
    output.paste(input, position, mask=input)
    return np.asarray(output)

def add_background(traffic_sign, shape=(18, 48), margin_range=(0.07, 0.21),
    t_margin=0.15, gtsrb_brightness=True, random_erasing_prob=0.0):
    # resize the traffic sign template
    s = randint(shape[0], shape[1])
    h, w = s, s
    image = cv2.resize(traffic_sign, (h, w), interpolation=cv2.INTER_AREA)

    # generate new brightness value
```

```
    if gtsrb_brightness:
        brightness = stats.johnsonsb.rvs(0.746678, 0.9066458, 7.098753,
            259.904883)
    else:
        brightness = uniform(7, 255)
    image = replace_brightness(image, brightness)

    # define the background shape
    margin = uniform(margin_range[0], margin_range[1])
    height, width = round(h / (- 2 * margin + 1)), round(w / (- 2 * margin +
        1))

    # create background with previous calculated dimensions
    background = np.zeros((height, width, 3), np.uint8)
    background[:] = tuple((randint(0, 255), randint(0, 255), randint(0, 255)))

    # calculate the new position to translate the traffic sign
    if (1 - (h / height)) / 2 < t_margin or (1 - (w / width)) / 2 < t_margin:
        y = round((height - h) / 2)
        x = round((width - w) / 2)
    else:
        y = randint(round(height * t_margin), round(height - height * t_margin
            - h))
        x = randint(round(width * t_margin), round(width - width * t_margin -
            h))

    # paste the traffic sign into the background
    output = paste(image, background, (y, x))

    # perform random erasing
    return random_erasing(output, t_margin) if random() < random_erasing_prob
        else output
```

B.2 ROTATION

Code for template rotation, see Section 4.4.

```
def rotate(traffic_sign, angle):
    h, w = traffic_sign.shape[:2]
    M = cv2.getRotationMatrix2D((h / 2, w / 2), randint(0, angle * 2) - angle,
        1)
    return cv2.warpAffine(traffic_sign, M, (h, w))
```

B.3 SHEAR TRANSFORMATION

Code for shear transformation, see Section 4.5.1.

```
def shear_transformation(traffic_sign, factor=(0.03, 0.10)):
    h, w = traffic_sign.shape[:2]

    # source points of the transformation
    src = np.float32([[0, 0], [w, 0], [w, h]])

    operation = randint(0, 3)
    u1 = uniform(factor[0], factor[1])
    u2 = uniform(factor[0], factor[1])

    # horizontal shear
    if operation == 0:
        dst = np.float32([[w * u1, 0], [w + w * u1, 0], [w - w * u2, h ]])
    elif operation == 1:
        dst = np.float32([[- w * u1, 0], [w - w * u1, 0], [w + w * u2, h]])

    # vertical shear
    elif operation == 2:
        dst = np.float32([[0, h * u1], [w, - h * u2], [w, h - h * u2]])
    else:
        dst = np.float32([[0, - h * u1], [w, + h * u2], [w, h + h * u2]])

    M = cv2.getAffineTransform(src, dst)
    return cv2.warpAffine(traffic_sign, M, (w, h))
```

B.4 PERSPECTIVE TRANSFORMATION

Code for perspective transformation, see Section 4.5.2.

```
def perspective_transformation(template, full=True):
    y, x = template.shape[:2]
    src = np.float32([[0, 0], [x, 0], [x, y], [0, y]])

    # source  transformation  vertices
    h = [0, 0, 0, 0]
    w = [0, 0, 0, 0]

    # choose a random  point
    v = randint(0, 3)
```

```
    if v == 0 or v == 2:
        mult = [-1, 1]
    else:
        mult = [1, -1]

    for i in range(0, 2):
        # determine new y-coordinate
        h[(v + i) % 4] = round(y * uniform(0.05, 0.15))
        # determine new x-coordinate
        w[(v + i) % 4] = round(x * uniform(0.05, 0.15))

        # transforms opposite vertices
        if full:
            h[(v + 3 - i) % 4] = mult[1] * h[(v + i) % 4]
            w[(v + 3 - i) % 4] = mult[0] * w[(v + i) % 4]

    dst = np.float32([[w[0], h[0]], [x - w[1], h[1]], [x - w[2], y - h[2]], [w
        [3], y - h[3]]])
    M = cv2.getPerspectiveTransform(src, dst)
    return cv2.warpPerspective(template, M, (x, y))
```

B.5 COLOR JITTER

Code for color jitter, see Section 4.6.1.

```
#  brightness is not jittered by default
def hsv_jitter(image, hue=(-12, 20), saturation=(0.4, 2), brightness=(0.5, 1),
     probs=(0.8, 0.8, 0)):
    # jitter hue with input probability
    if random() < probs[0]:
        # the input hue is in the range [0, 360] while OpenCV uses 180 degrees
        shift = uniform(0, hue[1] / 2 - hue[0] / 2) + hue[0] / 2
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        hsv[..., 0] = (hsv[..., 0] + shift) % 180
        hue_shifted = cv2.cvtColor(cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR), cv2.
            COLOR_BGR2BGRA)
        # preserve original alpha channel
        hue_shifted[:, :, 3] = image[:, :, 3]
        image = hue_shifted
    # jitter saturation
    if random() < probs[1]:
        enhanced = ImageEnhance.Color(Image.fromarray(cv2.cvtColor(image, cv2.
            COLOR_BGRA2RGBA))).enhance(uniform(saturation[0], saturation[1]))
        image = cv2.cvtColor(np.asarray(enhanced), cv2.COLOR_BGRA2RGBA)
```

```
    if random() < probs[2]:
        enhanced = ImageEnhance.Brightness(Image.fromarray(cv2.cvtColor(image,
            cv2.COLOR_BGRA2RGBA))).enhance(uniform(brightness[0], brightness
            [1]))
        image = cv2.cvtColor(np.asarray(enhanced), cv2.COLOR_BGRA2RGBA)
    return image
```

B.6 Replace Brightness

Code for template brightness replacement, see Section 4.6.2.

```
def get_image_brightness(image):
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv_image)
    return np.mean(v)

def replace_brightness(image, brightness):
    old_brightness = get_image_brightness(image)
    ratio = brightness / old_brightness

    # convert image to hsv color space and extract the value channel
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            b = round(v[i][j] * ratio)

            # replace brightness and avoid overflow
            v[i][j] = b if b < 255 else 255

    output = cv2.cvtColor(cv2.cvtColor(cv2.merge((h, s, v)), cv2.COLOR_HSV2BGR
        ), cv2.COLOR_BGR2BGRA)
    # keep original alpha channel
    output[:, :, 3] = image[:, :, 3]
    return output
```

B.7 Motion Blur

Code for motion blur, see Section 4.7.1.

```
def motion_blur(image, size):
    kernel_motion_blur = np.zeros((size, size))
    kernel_motion_blur[int((size - 1) / 2), :] = np.ones(size)
    kernel_motion_blur /= size
    return cv2.filter2D(image, -1, kernel_motion_blur)
```

B.8 CONFETTI NOISE

Code for confetti noise creation, see Section 4.7.2.

```
def confetti_noise(image, kernel=(0.03, 0.03), probability=0.03, spacing
    =0.015):
    h, w = image.shape[:2]
    kh, kw = round(kernel[0] * h), round(kernel[1] * w)
    s = round(spacing * min(kernel[0], kernel[1]))
    y, x = 0, 0
    for i in range(h - kh):
        if y > 0:
            y -= 1
            continue
        for j in range(w - kw):
            if x > 0:
                x -= 1
                continue
            if random() <= probability:
                # insert colored block
                image[i: i + kh, j: j + kw, :3] = randint(0, 255), randint(0,
                    255), randint(0, 255)
                y, x = kh + s, kw + s
    return image
```

B.9 PERLIN NOISE

Code for Perlin noise creation and application, see Section 4.7.3.

```
# create Perlin noise sample
def perlin_noise(shape=(2048, 2048), octaves=6):
    matrix = np.zeros(shape)
    for i in range(shape[0]):
```

```
        for j in range(shape[1]):
            matrix[i][j] = noise.pnoise2(i / 100, j / 100, octaves=octaves,
                persistence=0.5, lacunarity=2.0)
    # normalize Perlin noise from range [-1, 1] to [0, 255]
    return cv2.cvtColor(np.uint8(np.round(np.interp(matrix, (matrix.min(),
        matrix.max()), (0, 255)))), cv2.COLOR_GRAY2BGR)

# extract an arbitrarily positioned sample from an image with a given shape
def extract_from(image, shape=(512, 512)):
    h, w = image.shape[:2]
    y = randint(0, h - shape[0])
    x = randint(0, w - shape[1])
    return image[y: y + shape[0], x: x + shape[1]]

def add_perlin_noise(traffic_sign, perlin_noise, alpha=0.6):
    y, x = traffic_sign.shape[:2]
    mask = cv2.inRange(traffic_sign, (0, 0, 0, 255), (255, 255, 255, 255))
    perlin = cv2.resize(extract_from(perlin_noise), (x, y), interpolation=cv2.
        INTER_AREA)
    dst = np.zeros((y, x, 3), np.uint8)
    cv2.addWeighted(cv2.cvtColor(traffic_sign, cv2.COLOR_BGRA2BGR), alpha,
        perlin, 1 - alpha, 0, dst)
    dst = cv2.cvtColor(dst, cv2.COLOR_BGR2BGRA)
    traffic_sign[mask > 0] = dst[mask > 0]
    return traffic_sign
```

B.10 RANDOM ERASING

Code for random erasing, see Section 4.8.1.

```
# the margin ensures the erased block is applied to the sign
def random_erasing(image, margin=0.1):
    position = randint(0, 2)
    # factors for square shape
    if position == 0:
        hr = (0.1, 0.25)
        wr = (0.1, 0.25)
    # factors for horizontal rectangle
    elif position == 1:
        hr = (0.1, 0.15)
        wr = (0.15, 0.4)
    # factors for vertical rectangle
    else:
        hr = (0.15, 0.4)
```

```
        wr = (0.1, 0.15)

    hratio = uniform(hr[0], hr[1])
    wratio = uniform(wr[0], wr[1])
    height, width = image.shape[:2]

    # dimensions of erased area
    h, w = round(hratio * height), round(wratio * width)

    # position to be erased
    y = randint(round(height * margin), round((height - h) * (1 - margin)))
    x = randint(round(width * margin), round((width - w) * (1 - margin)))

    # erase with noise
    image[y: y + h, x: x + w, :3] = (np.random.rand(h, w, 3) * 255).astype(np.
        uint8)  #randint(0, 255), randint(0, 255), randint(0, 255)
    return image
```

B.11 ADD SHADOW

Code for shadow creation and application onto templates, see Section 4.8.2.

```
# generate a shadow from a Perlin noise sample
def shadow_from_pnoise(perlin_noise):
    h, w = perlin_noise.shape[:2]
    # add alpha channel
    perlin_shadow = cv2.cvtColor(perlin_noise, cv2.COLOR_BGR2BGRA)
    for i in range(h):
        for j in range(w):
            # the Perlin noise is in 3 channel grayscale
            if perlin_shadow[i][j][0] > 120:
                # transparent
                perlin_shadow[i][j] = [255, 255, 255, 0]
            else:
                # opaque
                perlin_shadow[i][j] = [0, 0, 0, 255]
    return perlin_shadow

# add shadow to an image from an already processed Perlin noise sample
def add_shadow(image, perlin_shadow):
    h, w = image.shape[:2]
    shadow = cv2.cvtColor(extract_from(perlin_shadow), cv2.COLOR_BGR2BGRA)
    # reduce transparency to simulate a shadow
    shadow[:,:, 3] = shadow[:,:, 3] * uniform(0.50, 0.75)
```

```
    shadow = cv2.resize(shadow, (h, w), interpolation=cv2.INTER_AREA)
    return paste(shadow, image)
```

B.12 OPERATIONS PIPELINE

Code for the full pipeline, see Section 4.9.

```
# load templates from given path
def load_templates(path):
    num_classes = 0
    for template in os.listdir(f'{path}'):
        match = re.match('[0-9]+(_0)?.png', template)
        if match:
            num_classes += 1

    templates = []
    for i in range(num_classes):
        template = cv2.imread(f'{path}/{i}.png', cv2.IMREAD_UNCHANGED)
        if template is not None:
            templates.append([template])
        else:
            templates.append([cv2.imread(image, cv2.IMREAD_UNCHANGED) for
                image in sorted(glob.glob(f'{path}/{i}_[0-9]*.png'))])
    return templates

# return a template from the given set
def choose_template(templates):
    n = len(templates)
    if n > 1 and random() < 0.3:
        return templates[randint(1, n - 1) if n > 2 else 1]
    else:
        return templates[0]

# pipeline
def gen(classes, templates, n_per_class, output_path, perlin_noise):
    for class_id in classes:
        # process high resolution samples
        for i in range(round(n_per_class * 0.8)):
            template = choose_template(templates[class_id])

            template = hsv_jitter(template)

            if random() < 0.7:
                template = rotate(template, 15)
```

```
            if random() < 0.6:
                template = perspective_transformation(template)
            if random() < 0.6:
                template = shear_transformation(template)

            template = add_perlin_noise(template, perlin_noise)

            template = add_background(template, (18, 48), (0.07, 0.21), 0.15)

            if random() < 0.3:
                template = motion_blur(template, randint(2, 5))

            cv2.imwrite(f'{output_path}/{format(class_id, "05d")}/{i}.png',
                template, [cv2.IMWRITE_PNG_COMPRESSION, 0])

        # process small samples
        for i in range(round(n_per_class * 0.2)):
            template = choose_template(templates[class_id])

            template = hsv_jitter(template)

            if random() < 0.7:
                template = rotate(template, 15)
            if random() < 0.6:
                template = perspective_transformation(template)
            if random() < 0.6:
                template = shear_transformation(template)

            template = add_perlin_noise(template, perlin_noise)

            if random() < 0.5:
                template = confetti_noise(template)

            template = add_background(template, (12, 17), (0.20, 0.25), 0.20)

            cv2.imwrite(f'{output_path}/{format(class_id, "05d")}/{i + round(
                n_per_class * 0.8)}.png', template, [cv2.
                IMWRITE_PNG_COMPRESSION, 0])


if __name__ == '__main__':
    templates_path = 'C:/templates_path'
    output_path = 'C:/synthetic_dataset_path'
    n_per_class = 2000
    # optimization for faster generation time
    num_proc = 4
```

```
    templates = load_templates(templates_path)
    num_classes = len(templates)

    # create dataset directory and respective class subfolders
    os.makedirs(output_path)
    [os.makedirs(os.path.join(output_path, format(class_id, "05d"))) for
        class_id in range(num_classes)]

    # generate the Perlin noise sample
    perlin_noise = perlin_noise((2048, 2048))

    processes = [mp.Process(target=gen, args=(range(round(i * (num_classes /
        num_proc)), round((i + 1) * (num_classes / num_proc))), templates,
        n_per_class, output_path, perlin_noise)) for i in range(num_proc)]

    for process in processes:
        process.start()

    for process in processes:
        process.join()
```

B.13 DATA AUGMENTATION

Code for the dynamic data augmentation, see Section 4.10.1.

```
import torchvision.transforms as transforms

# resize all images to (32, 32) and normalize them to mean = 0 and standard-
    deviation = 1 based on statistics collected from the training set

data_transforms = transforms.Compose([
  transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])

# resize, normalize, and jitter image brightness
data_jitter_brightness = transforms.Compose([
  transforms.Resize((32, 32)),
    transforms.ColorJitter(brightness=2),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])
```

```
# resize, normalize, and jitter image saturation
data_jitter_saturation = transforms.Compose([
  transforms.Resize((32, 32)),
    transforms.ColorJitter(saturation=2),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])

# resize, normalize, and jitter image contrast
data_jitter_contrast = transforms.Compose([
  transforms.Resize((32, 32)),
    transforms.ColorJitter(contrast=2),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])

# resize, normalize, and jitter image hues
data_jitter_hue = transforms.Compose([
  transforms.Resize((32, 32)),
    transforms.ColorJitter(hue=0.4),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])

# resize, normalize, and rotate image
data_rotate = transforms.Compose([
  transforms.Resize((32, 32)),
    transforms.RandomRotation(5),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])

# resize, normalize, and shear image
data_shear = transforms.Compose([
  transforms.Resize((32, 32)),
    transforms.RandomAffine(degrees=5, shear=2),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])

# resize, normalize, and translate image
data_translate = transforms.Compose([
  transforms.Resize((32, 32)),
    transforms.RandomAffine(degrees = 5,translate=(0.1,0.1)),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])
```

```
# resize, normalize, and crop image
data_center = transforms.Compose([
  transforms.Resize((36, 36)),
    transforms.CenterCrop(32),
    transforms.ToTensor(),
    transforms.Normalize((0.3337, 0.3064, 0.3171), (0.2672, 0.2564, 0.2629))
])

import torch

# load training set and apply data augmentation
train_loader = torch.utils.data.DataLoader(
   torch.utils.data.ConcatDataset([
        datasets.ImageFolder(path, transform=data_transforms),
        datasets.ImageFolder(path, transform=data_jitter_brightness),
        datasets.ImageFolder(path, transform=data_jitter_hue),
        datasets.ImageFolder(path, transform=data_jitter_contrast),
        datasets.ImageFolder(path, transform=data_jitter_saturation),
        datasets.ImageFolder(path, transform=data_translate),
        datasets.ImageFolder(path, transform=data_rotate),
        datasets.ImageFolder(path, transform=data_center),
        datasets.ImageFolder(path, transform=data_shear)]),
        batch_size=args.batch_size, shuffle=True,
        num_workers=4, pin_memory=use_gpu)
```