TAA - German Traffic Sign Recognition Benchmark (GTSRB)

Diogo Magalhães - 102470
DETI
University of Aveiro
d.magalhaes@ua.pt
Tópicos de Aprendizagem Automática
Petia Georgieva

Emanuel Marques - 102565
DETI
University of Aveiro
emanuel.gmarques@ua.pt
Tópicos de Aprendizagem Automática
Petia Georgieva

***Abstract*—Nowadays autonomous driving is a hot topic and it is becoming increasingly evident that cars will become more and more automated in the future. To achieve this, it is necessary that cars can analyze the environment around them and make the best decisions based on that. One of the most important aspects of this is the ability to recognize the correct traffic sign. This report shows our approach to the machine learning GTSRB - German Traffic Sign Recognition Benchmark challenge. The GTSRB dataset consists of 43 classes of traffic signs, commonly found on German roads, and has over 5 thousand images. In this report, we will analyze how different convolutional neural networks architectures perform in this task and review their results. We are also applying preprocessing to the images to enhance their quality as some of them reveal very poor lighting and contours. Both students contributed equally to the elaboration of this project.**

***Index Terms*—Machine Learning, CNN, GTSRB**

I. Introduction

More and more we see an increased interest in automated vehicle driving both in the manufacturers as well as in common citizens wanting to buy such products. The recognition of traffic signs is an essential task for ensuring the safety and efficiency of this type of vehicle driving.
With the increasing number of vehicles on the road, the need for accurate, fast, and reliable way of traffic sign recognition is becoming more pressing, challenging, and important than ever. In recent years, with the development of machine learning techniques, great potential for solving this problem has been shown.
In this project, we are using the GTSRB dataset, a very popular problem in the field, which contains over 5 thousand images of 43 different types of traffic signs commonly found on German roads. This is a very challenging problem due to the variation in lighting, weather, and other factors that can affect the appearance of the signs. The classes presented in our data and to which we must develop a model to predict upon new unseen traffic signs, are shown in figure 1 in an ordered way.

Our goal is to explore the usage of convolution neural networks, a deep machine learning algorithm that is very useful when dealing with images, as this method can recognize patterns and extract new features by itself.
We will explore various convolutional neural network architectures and how different data prepossessing methods influence the overall results on new samples not used during the training phase. Moreover, we will also analyze the strengths and weaknesses of each model and provide insights into how to improve their performance.
The findings of this report have an important impact on real-world applications, such as autonomous driving as seen before, where accurate and reliable detection and recognition of traffic signs is crucial for ensuring the safety of passengers and pedestrians.
By offering profound insights into the performance of diverse machine learning models in traffic sign recognition, this report can effectively guide the advancement of precise and dependable systems for autonomous driving and other transportation applications.

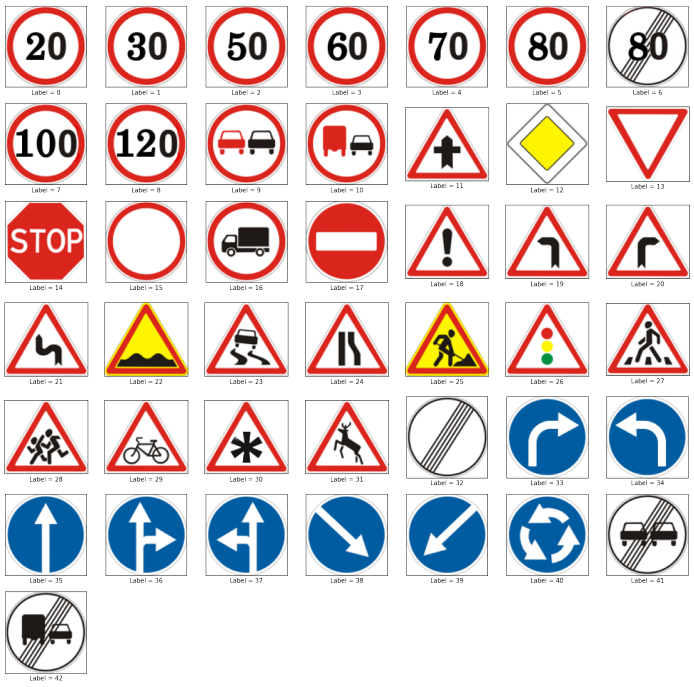


Fig. 1: All 43 different types of signals (classes).

II. Data Visualization

To better visualize the images we have at hand, we plotted 30 random samples from our dataset in a grid of 6 by 5, which is show in figure 2.

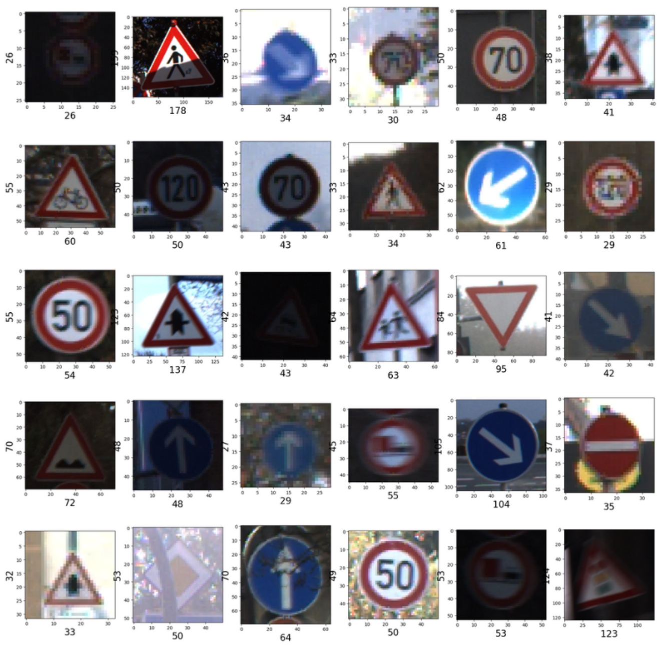


Fig. 2: Random samples of 30 images

As mentioned before, we have 43 classes in total, corresponding to a different traffic sign. Our entire dataset is subdivided into a training folder and testing folder. On the training folder, we have 39209 images spread across the different classes and the testing folder contains 12630 images, this makes around 24.36% of the entire dataset to test. However, each class does not have the same amount of samples, some classes have 200 images and others have 10 times more. Figure 3 shows this class imbalance across the complete dataset.
As this is a multiclasss classification problem, we had to use one hot encoding to represent the categorical classes in binary vectors.

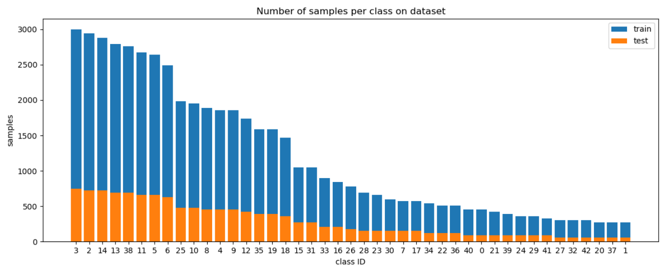


Fig. 3: Quantity of images by class in train and test datasets

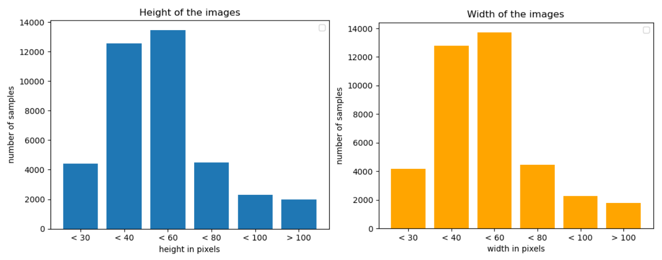


Fig. 4: Bar graphs with image size distribution

Analysing figure 4, one can see the distribution of images across different dimensions as their sizes are not the same. Most images are small, in the range of around 40 by 40 pixels but others are bigger reaching 160 pixels width by 160 pixels height, although this represents a small portion of the dataset.

III. Data Preprocessing

The particularity of the different images size is a problem as machine learning models required the same amount of features across all samples, and in our case, each feature will be a pixel value. The solution was to resize all images to the same size. We chose 30 by 30 pixels as it was the most used proportion, and dimensions bigger than that would be too much to handle and require too much time to train.

On some models with trained with the images with RGB channels and then trained again but with the images in grey scale to analyse the outcomes. Sometimes, the accuracy of the model was a worse, but with other models the accuracy had improved. The reason for this may be that the images on grey scale already have a large quantity of information and the additional color channels would not add much to the image itself. However, some convolutional layers may take advantage of the colors and extract features from that.

Since we were working with pixels, the value of each feature would only vary between 0 and 255. Even though these weren't very big numbers, we decided to always normalize the features. Sometimes we normalized the features by dividing each feature by 255, and some other times we normalized the features with the featureNormalization function of the practical classes that used Z-Score Normalization that is expressed by $Xnorm = (X - Xmean)/Xstd$.

Together with the images and their metadata, we also have access to values representing the region of interest of the represented sign. With the information of the coordinates of the left upper point and the right lower point, we can create a bounding box of the sign within that image and crop the original to contain only the relevant pixels.

Fig. 5: Example of and image cropped

Other data augmentation methods we used include rotations, scaling, shearing, brightness, and color adjustment in the least representative classes, which help to overcome the problem of class imbalance. We believe that these augmentation techniques significantly contributed to improving our models performance.

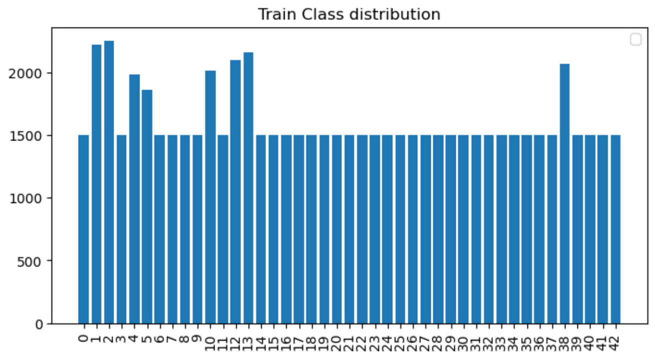


Fig. 6: Quantity of images by class after Data Augmenting

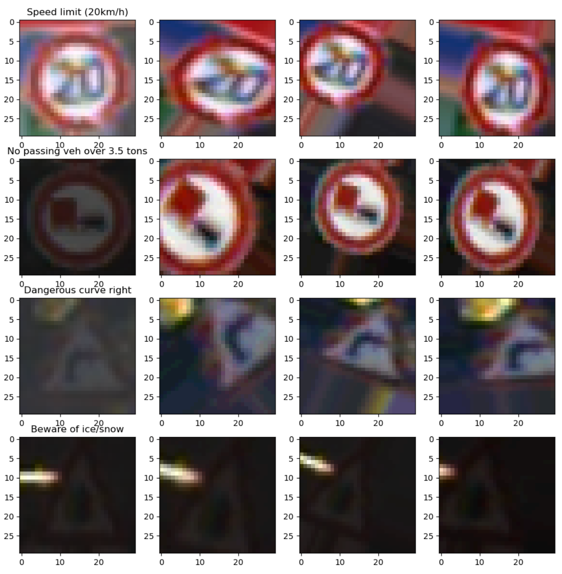


Fig. 7: Example of images augmented (Column 1 are the originals)

All of these types of image preprocessing were not used simultaneously in the models, even though a mix of preprocessing techniques was used, and later on in this report, it will be discussed why that happened.

IV. State of the Art

The German Traffic Sign Recognition Benchmark (GTSRB) dataset, was provided by the Real-Time Computer Vision research group at Institut für Neuroinformatik (INI) [1], in Germany. Along side with the German Traffic Sign Detection Benchmark (GTSDB), which consisted of detecting the bounding box of traffic sings within a broader image, these datasets were used to held a competition at the International Joint Conference on Neural Networks (IJCNN) in 2011.

The team who won the competition had a total of 99.46% recognition rate with new unseen examples and used the Committee of CNNs method. Through the analysis of their paper [2] we learned that they applied data preprocessing such as image adjustment and image equalization making high contrast and good quality images to work with and then used 3 sets of one convolutional layer and one max-pooling layer.
Years latter another contestant appear who achieved 99.71% with the method of CNN with 3 Spatial Transformers [3]. They used an architecture with 3 CNN layers and 14,629,801 trainable parameters, but the inovation here was the introduction of the spatial transformer module which is a layer incorporated into the neural network that can learn the appropriated transformations to apply to the input data in order to make the important content of the images have the same proportions. Then, together with data augmentation, the images passed by a network of 3 sets of one convolutional layer and relu activation and next a max-pooling layers and local contrast normalization, ended in a fully connected layer with 400 neurons.
On IEEE Explore website, a paper appears regarding this competition at the International Joint Conference on Neural Networks [4]. It explains how the data was gathered, and later labeled by humans. The dataset was provided with the histograms of oriented gradients (HOG), the haar-like features and color histograms like hue values for the least experience data scientists to use. The findings of making humans classify images of the traffic signs shows that not even a biological person can get it right all the time, miss labeling every now and then, which is interesting.
In Kaggle platform, many enthusiasts have also attempted to create the best model they can get to complete this challenge. Convolutional neural networks is without a doubt the most used approach. An architecture with 4 convolutionals layers and one fully connected layer with 512 neurons can easily achieve 98% test accuracy with data augmentation [5].
On the free open-source world, we could find engineers that have also embraced this problem. One GitHub repository was particular interesting as it explored the use of LeNet-5 and VGGNet architectures and concluded that the VGGNet got the best performance and was faster to converge [6].
The bottom line is ”Traffic sign detection is a challenging computer vision task of high industrial relevance.” [7] and any contribution is welcome for the growth of the society.

V. Models

In this report, we investigate the performance of several machine learning models for recognizing traffic signs in the GTSRB dataset.
By comparing the performance of different machine learning neural networks on the traffic sign recognition task, this report can provide valuable insights into the strengths and weaknesses of these models for real-world applications such as autonomous driving.
In the previous project [8], we developed several different algorithms and models that we will not focus on explaining the step by step, but for comparasion reasons, we will use them for comparations in the Results Section VII.
Popular CNN architectures like LeNet-5, AlexNet and VGGNet and its variants are not being explored by us because that has already been studied by state of the art data scientists and we doing the exactly same thing would not add much to our contribution on the subject.

A. Convolutional Neural Networks

The heart of deep learning algorithms are the Neural Networks. Neural networks are a computational model that are inspired by the structuring and functioning of the human brain and tries to mimic it. Neural networks are made up of neurons that are connected to each other in which the connections have weights associated that are used to multiply the values before passing through an activation function and sending them to the next layer, which are adjusted during the training process, with them and are arranged in layers. The first layer is responsible for receiving the data and is called the input layer, the last layer is responsible for returning the output and is called the output layer, and all the layers in between are responsible for extracting the features and processing the data and are called hidden layers.
Convolutional Neural Networks are a type of neural networks that are specialized in processing data that has a grid-like topology, such as an image that is a series of pixels arranged in a grid-like fashion. They are very effective in reducing the number of parameters that are required to be trained by extracting features from the data and classifying them based on those features. They are composed of convolutional layers, that apply filters that detect edges, textures, and other important patterns in the data extracting features from the data, of pooling layers, that are responsible for reducing the dimensionality of the data maintaining important features, and of fully connected layers, that are responsible for classifying the data based on the features extracted by the convolutional layers.
As in Neural Networks, during the training process, CNNs learn to optimize their filters and parameters by adjusting their weights using a technique called backpropagation. This process involves iteratively comparing the network's output with the desired output and updating the weights accordingly, minimizing the difference between them.
The key advantage of a CNN is its ability to automatically learn relevant features from raw data capturing both low-level and high-level features, enabling it to recognize complex patterns and objects in the data due to the hierarchical nature of the network.

Before moving on to how Convolutional Neural Networks are structured, a short explanation of the layers used is necessary. Along this project we used Input Layers, that are responsible for receiving the data, we used ZeroPadding layers, that are responsible for adding rows and collumns of zeros around the data that we want to process in order to preserve the spatial information of the input during convolutional operations, we used Convolutinal Layers, that convolves and processes data with a set of filters, we used Max Pooling layers, that are responsible for reducing the spatial dimensions of the feature maps, extracting important features, and improving the computational efficiency of CNNs, we used Dropout layers, that are a regularization technique that helps prevent overfitting by randomly setting to zero a portion of the neurons in the previous layer during training, we used Flatten layers, that reshapes the multidimensional feature maps into a one-dimensional vector preserving the depth, we used Dense layers, that are fully connected layers meaning that it connects every neuron in the current layer to every neuron in the subsequent layer and we used Batch Normalization layers, that is another technique of regularization that normalizes the inputs of a neural network by adjusting and scaling the activations of each mini-batch during training.

In this project, we created several Convolution Neural Networks with the main objective of obtaining the best model possible that could output the maximum accuracy. We experimented with several different quantity of layers and types of layers.
All of the following models were tested using different strategies of data preprocessing in other to be able to compare the results, or improve results.

1) One Convolutional Layer: The first model created was a model with only one convolutional Layer, that was composed of a Input Layer, followed by a Zero Padding layer with 3x3 padding, then was a convolutional Layer with 32 filters, kernel size of 3x3 and stride of 1x1, and a ReLU activation function. Next it was followed by a max pooling layer with a pool size of 2x2, followed by a Dropout of 0.2, followed by a Flatten layer, followed by a Dense layer with 256 neurons and a ReLU activation function. Next, was a Dropout layer of 0.3, followed by a Dense layer with 43 neurons and a softmax activation function since we have 43 classes and we want to output the probability of each class.
This model was overall our worst model, with training accuracies of approximately 90% in the test dataset.

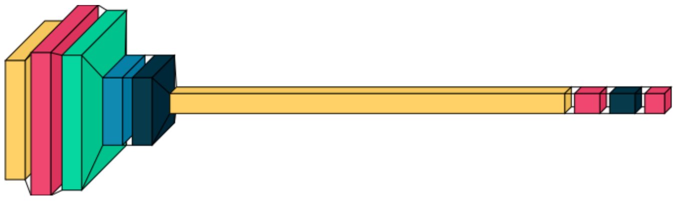


Fig. 8: Architecture of the first developed CNN

2) Three Convolutional Layer: The next three models share the same ammount of convolutional layers, having all three convolutional layers, but differ in the regularization and normalization techniques used. The first of those three models was composed of an input layer, followed by a zero padding layer with 3x3 padding, followed by a convolutional layer with 32 filters, kernel size of 3x3 and stride of 1x1, with the ReLU activation function. Followed was a max pooling layer with a pool size of 2x2, that preceded a dropout layer of 0.25. Next, was another convolutional layer with 64 filters, same kernel size, and again the ReLu activation function. Next was a max pooling layer with a pool size of 2x2, followed by a dropout of 0.25, followed by the last convolutional layer but time time with 128 filters, followed by a max pooling layer with a pool size of 2x2, followed by a dropout of 0.25, followed by a flatten layer, a dense layer with 256 neurons and a ReLU activation function, followed by a dropout of 0.5. After all of this, was a dense layer with 43 neurons and a softmax activation function since we have 43 classes and we want to output the probability of each class.

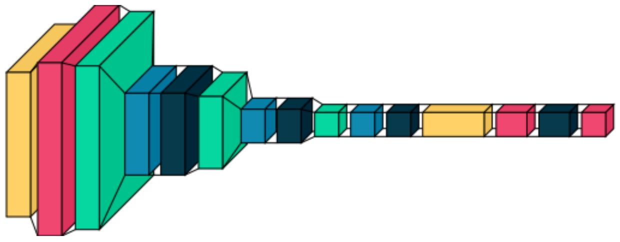


Fig. 9: Architecture of the first CNN with 3 layers

The second of the three models, was basically the same as the first one, but instead of using Dropout layers, we used Batch Normalization layers.

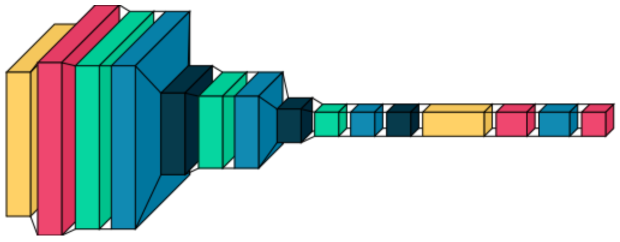


Fig. 10: Architecture of the second CNN with 3 layers

The third of the three models was again, basically the same as the first and second ones, but instead of using Dropout layers, we used Batch Normalization layers and instead of using ReLU activation functions, we used Leaky ReLU activation functions.

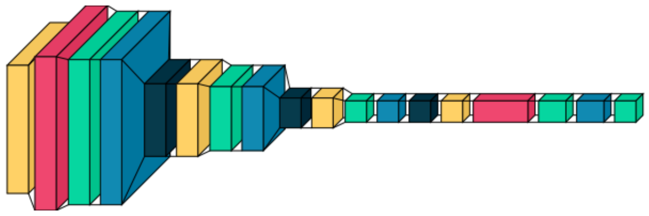


Fig. 11: Architecture of the third CNN with 3 layers

The first both models, had improved results, when compared with the no use of normalization and regularization techniques, and we obtained approximately 93% with the first model and 95% in the second one, which led use to try the third one that was a combination of both and we got a score of near 97.5%, which we found very good and led us to try to improve it even more by using even more Convolutional Layers.

3) Four Convolutional Layer: The last model, had a more complex structure, having four convolutional layers and it was composed by a input layer, followed by a zero padding layer with 3x3 padding, a convolutional Layer with 32 filters, kernel size of 3x3, stride of 1x1, and a ReLU activation function. Followed that, was a Batch Normalization layer, followed by an identical convolutional layer but with 32 filters, followed by a batch normalization layer, a max pooling layer with a pool size of 2x2, and at the end of this, a dropout layer of 0.25. The next chunk was another convolutional layer with 64 filters, followed by a batch normalization layer, and then the last convolutional layer with 128 filters, same kernel size and activation function, next another batch normalization layer, followed by a max pooling layer with a pool size of 2x2, and a dropout of 0.25. After this deep learning phase, we used a flatten layer to change the matrix shape of the data and then followed by a dense layer with 512 neurons and a relu activation, a batch normalization layer, and a dropout of 0.5. Then, again a fully connected layer with 256 neurons, which preceded a batch normalization layer, and a dropout layer of 0.5, and finally followed by a dense layer with 43 neurons and a softmax activation function since we have 43 classes and we want to output the probability of each class.

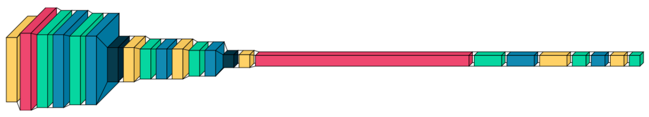


Fig. 12: Architecture of the CNN with 4 layers

Out of all the models created, this was the best one, with training accuracies of approximately 98.5% when testing in the

test dataset with sometimes getting accuracies over 98.9%. Our best model had 98.9% accuracy, and after analyzing the loss and accuracy values over the course of training the model, we found that the model was stagnant and that it was not possible to improve its performance by increasing the amount of epochs.
Even though it improved when comparing with using 3 convolutional layers, the increment was not very substancial and we did not tried to introduce more layers since it would not have that much of impact and we wanted to explore more solutions.

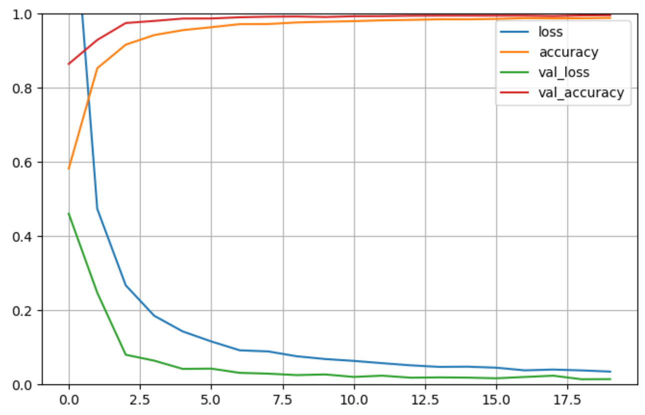
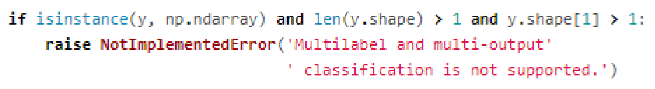


Fig. 13: Accuracy and Loss evolution over Epochs of the Best Model

As we were not yet satisfied with the obtained result, because some state of the art models had accuracies almost 1% better, we went looking for solutions and found something that we didn't see being done in any state of the art report, which was the use of the ensembling of models. Since keras didn't have any built in function to do this in multilabel classification problems because it was not yet implemented, we had to do it manually, and we did it by averaging/summing the predictions of the models.



```
if isinstance(y, np.ndarray) and len(y.shape) > 1 and y.shape[1] > 1:
    raise NotImplementedError('Multilabel and multi-output'
                              ' classification is not supported.')
```

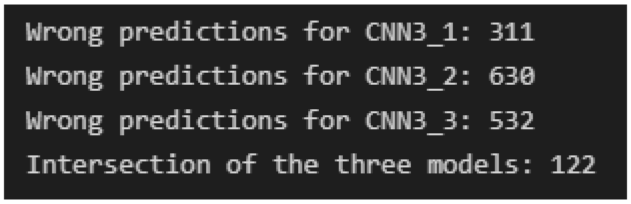
Fig. 14: Ensembling Multilabel Classification not yet implemented by keras library

B. Ensemble Learning

Ensemble learning is a machine learning technique that combines several base models in order to produce one optimal predictive model. The idea behind ensemble learning is that when weak models are correctly combined, they can achieve better predictive performance and be more accurate than a single strong model alone. The main principle of ensemble learning is the idea that combining the strengths of different models can help compensate for their individual weaknesses, being particularly effective when the individual models used in the ensemble are diverse and make independent errors. The diversity can arise from using different algorithms, different subsets of the data, or introducing randomness during the training process.

The ensemble base models can be trained in parallel and the final prediction can be made by counting the votes of the base models or by averaging or summing the predictions of the base models and choosing the class with the highest probability. The base models can be trained using different algorithms or the same algorithm with different hyperparameters.
In our case we didn't train the models using specific esemble methods, such as Bagging, Boosting, Stacking, or Random Forest, but we trained the models individually, always making some small adjustements in the training data of each one, with the objective of making them more diverse, and reduce the bias and variance that a single model could have.

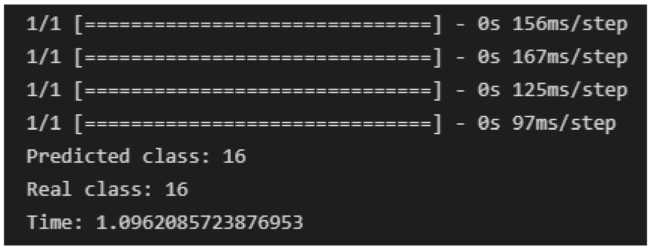
Since our models were almost at their limit and we wanted to improve, we had the idea to analyse if the best models, with similar accuracies were failing in the same images, that could be because of those images being hard to analyse even by a human.
After intersecting the predictions of the three best models we checked that only a few part of images were failing in all three models simultaneously.



```
Wrong predictions for CNN3_1: 311
Wrong predictions for CNN3_2: 630
Wrong predictions for CNN3_3: 532
Intersection of the three models: 122
```

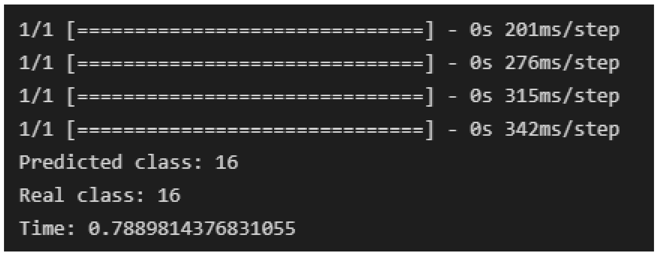
Fig. 15: Predictions that occur in all of three best models simultaneously

We chose our 4 best models to ensemble and predict simultaneously. Our first concern was about the time it would take to analyse a single image, since this is suposed to work in a smart car.
We notice that predicting a single sample took a long time in comparasion with predicting all of the examples, but assuming that the time with good algorithms to predict a sample is the same as predicting a class when predicting all examples we have reasonable times since we are only using 4 models. Taking advantage of threads, we can reduce that time even more, since the 4 models work simultaneously, and in smart cars the processor cannot be a problem.



```
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 97ms/step
Predicted class: 16
Real class: 16
Time: 1.0962085723876953
```

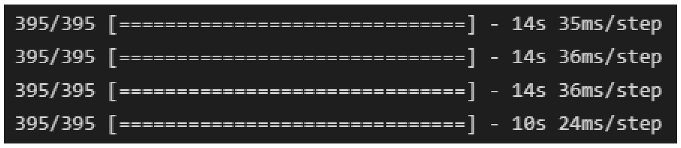
Fig. 16: Prediction time without threads



```
1/1 [==============================] - 0s 201ms/step
1/1 [==============================] - 0s 276ms/step
1/1 [==============================] - 0s 315ms/step
1/1 [==============================] - 0s 342ms/step
Predicted class: 16
Real class: 16
Time: 0.7889814376831055
```

Fig. 17: Prediction time with threads

As we can see, even though in the figure 16 and figure 17 it took more time to calculate the correct label when using threads, the overall time was faster.



```
395/395 [==============================] - 14s 35ms/step
395/395 [==============================] - 14s 36ms/step
395/395 [==============================] - 14s 36ms/step
395/395 [==============================] - 10s 24ms/step
```

Fig. 18: Prediction time in each step

As we can see in the figure 18, when calculating several predictions at the same time, each step takes approximately 35 milliseconds.

With that problem cleared up, we moved on to the accuracies of these ensembled models and obtained an excellent improvement in the results, ending up with an accuracy of 99.42% in the test dataset, which is quite close to the State of the Art models.

Figure 20 shows that some classes was correctly predicted almost all the time.

VI. Real Images

To test our model in the real world, we went and took some pictures of traffic signs, taking precaution to use different angles, different distances, different lighting conditions and different states of the traffic signs.

We took 21 pictures of the traffic signs, and our model was able to correctly classify 19 of them, which is a 90.47% accuracy.

The most impressive samples that our model was able to correctly classify were the following:

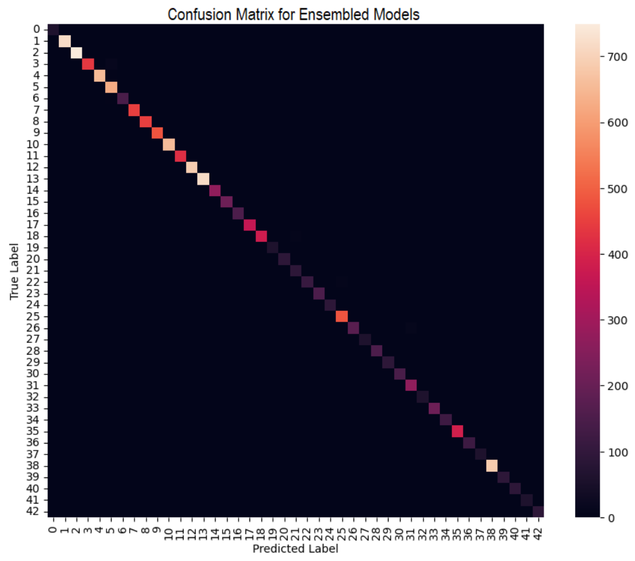


Fig. 19: Confusion Matrix of ensembled models

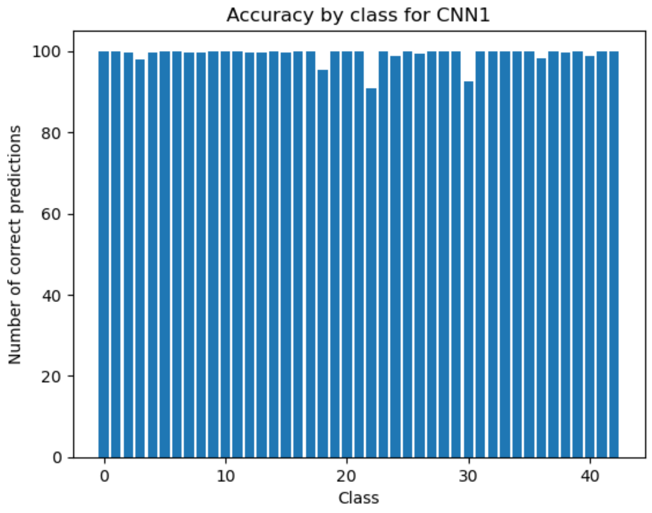
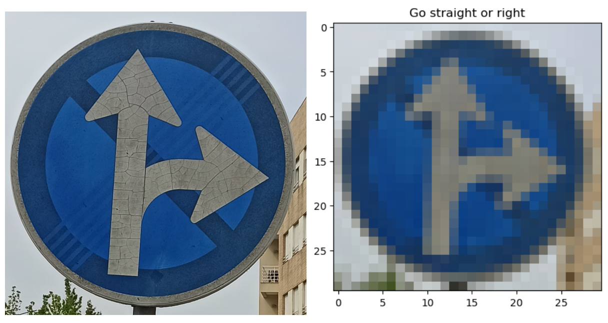


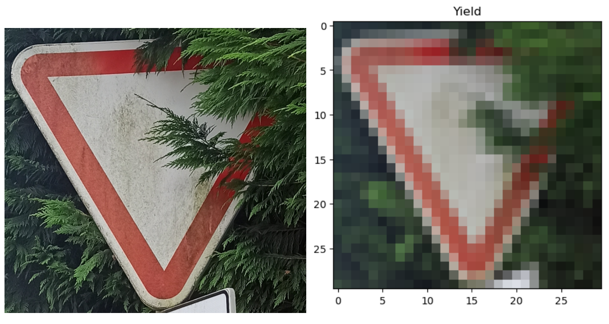
Fig. 20: Test accuracy by class



(a) Original Image (b) Image and Prediction

Fig. 21: Original and Processed images of Go Straight or Right sign with correct prediction

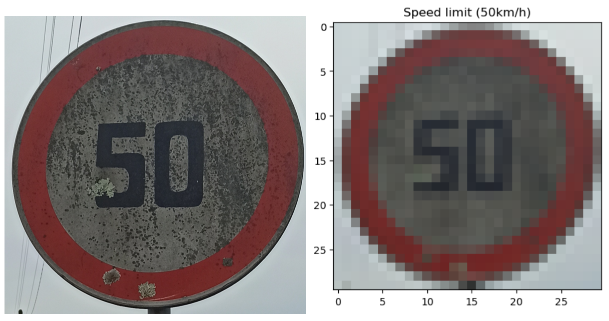
In Figure 21, our model correctly predicted that the correct sign was a 'Go Straight or Right' sign, even though the sign was a sticker on top of another sign.



(a) Original Image (b) Image and Prediction

Fig. 22: Original and Processed images of Yield sign with correct prediction

In Figure 22, our model correctly predicted that the correct sign was a 'Yield' sign, even though it had some trees in front of it.

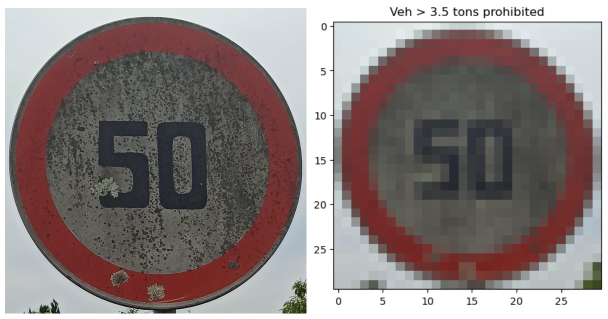


(a) Original Image (b) Image and Prediction

Fig. 23: Original and Processed images of Speed Limit (50 km/h) sign with correct prediction

In Figure 23, our model correctly predicted that the correct sign was a 'Speed Limit (50 km/h)' sign, even though it was an old model of the sign, that probably was not used in Germany where the dataset was collected, and it was full of dirt and rust.

Our least impressive sample was the following:



(a) Original Image (b) Image and Prediction

Fig. 24: Original and Processed images of Speed Limit (50 km/h) sign with wrong prediction

In Figure 24, our model incorrectly predicted that the correct sign was a 'Speed Limit (50 km/h)' sign, even though it was the same sign as in figure 22, but with a slightly different angle. It ended up predicting 'vehicle 3.5 tons prohibited' sign which is disappointing since the only similarities in the sign is both being round and red in the borders.
Overall this experiment was really fun and interesting to check that in real world, things might work differently than expected.

VII. Comparation of Results

Based on the models we have developed, which include variations with one, three, and four convolutional neural layers, as well as an ensemble classifier combining multiple models, we conducted an evaluation to compare their performance. To provide a clear overview, we constructed the table I showcasing the results of each model and our analysis leads us to the conclusion that the ensemble classifier outperformed the individual models, emerging as the superior choice.

TABLE I: Comparation of our models

	One Convolutional Layer	Best of Three Convolutional Layers	Four Convolutional Layers	Ensemble Classifier
Test Accuracy	∼90%	∼97.5%	∼98.5%	∼99.42%

In previous project, we also worked with this problem but explored different and more simpler algoritms. We have used logistic regression with the best score of 72.88%. K-Nearest Neighbors which got the worst performance with around 37%. Support Vector Machine which, after hyper parameters tunning gave us a final score of 81.08%, and, Artificial Neural Network with an architecture of 1 fully connected hidden layer with 800 neurons which achived 84% accuracy. The comparation between these models and our new best can be visualized in table II. We are happy that the second version of this work surpassed the first by a large amount.

TABLE II: Comparation with our previous work

	Logistic Regression	Artificial Neural Network	K-Nearest Neighbors	SVM	CNNs with Ensemble Classification
Test Accuracy	72.88%	∼84%	∼37%	∼81.08%	∼99.42%

When comparing our best model, the ensemble classifier, with the official results of the top performers [9], we find that we did not achieve the highest rank. However, our performance was remarkably close, with a marginal difference of just a few hundredths of a percentage point from the second-place position. This information is clearly presented in the table III, substantiating our claims.

TABLE III: Comparation with the state of the art

	CNNS with Ensemble Classification (ours)	CNN with 3 Spatial Transformers (best)	Committee of CNNs (second best)	Multi-Scale CNNs
Test Accuracy	99.42%	99.71%	99.46%	98.31%

VIII. Conclusion

This topic, and this dataset in specific, is very popular and a bit complex, however, we are satisfied with our results as we come close to the state of the art performance and could overpass the 99% threshold with ensemble classification. Our model being able to recognize some traffic sign from pictures taken by us, also demonstrates the power of machine learning and the ability of our models to capture sensitive information from pixels. Data preprocessing was what contributed the most for the excellent performance of the convolutional layers and was the biggest change introduced relative to our previous work. Indeed, by rotating, cropping, and shearing the images, and at the same time increase the number of samples in the least representative classes, allowed the models to be more robust to recognize new images from any class. Transfer Learning which is the process of using pre-trained models to capture high level features, like shape and edges and then fine-tuning to match our goal could be another technique to use but we found that data processing was more crucial that the inside of the network and its training. This project was a good opportunity to apply and experiment with various techniques and machine learning models that we learned in class. This field has captivated our interest, and being able to visualize and work with different models has allowed us to gain valuable experience. Both students contributed equally to the elaboration of this project.

References

[1] Institut für Neuroinformatik
https://www.ini.rub.de/

[2] Multi-column deep neural network for traffic sign classification
https://www.sciencedirect.com/science/article/pii/S0893608012000524

[3] Deep neural network for traffic sign recognition systems
https://www.sciencedirect.com/science/article/pii/S0893608018300054

[4] The German Traffic Sign Recognition Benchmark: A multi-class classification competition
https://ieeexplore.ieee.org/document/6033395

[5] German Traffic Sign Recognition, Amitabh Priyadarshi Notebook
https://www.kaggle.com/code/amitabhpriyadarshi/german-traffic-sign-recognition

[6] German-Traffic-Sign-Classification-Using-TensorFlow, by Mohamed Ameen
https://github.com/mohamedameen93/German-Traffic-Sign-Classification-Using-TensorFlow

[7] Detection of traffic signs in real-world images: The German traffic sign detection benchmark
https://ieeexplore.ieee.org/abstract/document/6706807

[8] Project 1 of GTSRB, by Emanuel Marques and Diogo Magalhães
https://github.com/EmanGM/TAA-project1

[9] Official results for GTSRB
https://benchmark.ini.rub.de/gtsrb_results.html