**TRAFFIC-SIGN RECOGNITION (TSR)**

Gabriel Dorobanțu, Narcis Ioniță

# **Abstract**

Identifying traffic signs has become an important issue when debating about vehicle safety applications. Traffic-Sign Recognition (TSR) is currently an important feature for the automotive industry, as self-driving cars are no longer just a futuristic dream and new cars need to be safer and safer in order to minimize accidents produced by human errors. In this paper, we propose a solution for this modern problem based on existing approaches of traffic-sign recognition. Related work focuses on different methods like template matching, convolutional neural networks (CNN), Haar-like features, support vector machines (SVM) method and a few more others, however these methods are not perfect, and each method comes with different downsides. Using deep learning and computer vision preprocessing, the proposed method tries to overcome as many of these disadvantages as possible providing a real-time solution that can be a core part of advanced driver-assistance systems (ADAS). The evaluation of the proposed method consists of training the model using two datasets: the German Traffic Sign Recognition Benchmark (GTSRB) and the Belgium Traffic Sign Dataset (BTSD).

# **Introduction**

Considering an interesting statistic about road traffic injuries brought by World Health Organization [[1](#fno_WHO)], there are around 1.3 million people that die each year as a result of road crashes, from which 93% of these crashes occur in less developed countries and that can be due to a large percentage of older cars that are in circulation, with less equipped safety technology. There are also between 20 to 50 million more people that get injured and from an economic point of view, this causes a cost of 3% of a country's gross domestic product. Keeping in mind this idea there is safe to say that the real-world applicability of detection and recognition of road signs is undeniable. As all new cars sold in the EU will be expected to be mandatory equipped with this type of technology in the near future [[2](#fno_EULaw)], and since the European New Car Assessment Programme (Euro NCAP) place great value on car safety and they also conducted surveys and safety campaigns regarding ADAS, stating that cars of the future need “readable” roads [[3](#fno_EuroNCAP)], this is seen as a challenge to detect different signs not only in different weather or daytime conditions, but also in different road conditions produced by various external factors. Even though the current advanced driver assistance systems use traffic sign recognition, they only have a defined subset of possible signs. It is surprising that there has not been an implementation of an extensive unbiased comparison of sign detection systems. One of the reasons for the slow development of this feature might be the lack of a large benchmark data set that is freely available. The recognition process can be divided into two steps, detection and classification. It is safe to say that the detection takes priority when comparing the two, due to the fact that the state-of-the-art classification methods have a human competitive performance at best. Therefore, the classification can be regarded as solved, at least for the time being.[[4](#fno_mvc)][[5](#fno_dccs)] While most of the attention of sign detection is on particular shapes, such as rectangles and circles, and the type of the sign (speed limit traffic signs), when not focused on a single type of road signs, an additional system should be put in place. In most of the cases, the system uses a color based segmentation, which is followed by a recognition stage. For this approach to work, a large training database with plenty of road signs is needed. In order to minimize the size of the database and get the intended learning process results, the color based segmentation can be replaced by a combination of color and shape detection.

# **Related work**

Early methods had a set of rules in place that restricted color and shape and required that signs appear only in certain regions of an image, these regions are considered to be candidates, which then they are recognized based on a template matching method using other images; such method was used by Michael Shneier [[6](#fno_TemplateMatching)] in his article about road sign detection, where his algorithm performed fast enough to be used in real-time, but it only addressed warning signs and a few regulatory signs, also for blurry or affected images, the algorithm had a lower performance and the candidates couldn’t be properly detected. Further, things have advanced with the emergence of the machine learning concept, and many articles came up with different approaches that use support-vector machines or convolutional networks. In an article by David Soendoro and Iping Supriana, a SVM method is proposed for classifying binary images with localized traffic signs, which are resulted from a color-based method with CIELab + hue [[7](#fno_SVM)]. A more recent take used a CNN with fewer parameters, smaller models and easier training which performed a high accuracy, close to 97%, better than a classical convolutional network [[8](#fno_smallCNN)]. A completely new and bolder approach in the field is a CNN method that uses GPGPU [[9](#fno_CNNGPGPU)] and Nvidia's latest solution in the automotive industry for autonomous vehicles which is called Nvidia DRIVE [[10](#fno_Nvidia)]. This method focuses on solving severe illumination problems regarding low light or wide variance of light like reflection, in images captured from real-world.  
Regarding traditional methods, a wide variety of hand-crafted systems, including specific colors and shapes, such as HOG [[16](#fno_tc47)][[17](#fno_tc17)] or SIFT [[18](#fno_zt13)][[19](#fno_gt13)] were used for classification with machine learning models, like SVM, tree classifiers and boosting.

Various traffic sign recognition competitions were held, with the sole purpose to allow scientists from different fields to contribute their results. One of these competitions, GTSRB, had the goal to produce a paper regarding benchmarking learning algorithms for traffic sign recognition. The official results offer an overview of the up-to-date recognition performance [5].  
IDSIA achieved an error rate of 0.54% and it combined several deep convolutional neural network columns and preprocessed the input images as many small blocks [12]. Meanwhile, COSFIRE, used multi-scale CNNs and fused local and global features, achieving an error rate of 1.03%. [14] It should be noted that the GTSRB dataset contains images in which the traffic sign occupies a large proportion of the image, and that in the real world, classifying images in which the traffic signs only occupy a small proportion of the traffic scene is more important and should be the main focus of the researchers.

*Table 1: Performance of various methods in the IJCNN2011 Competition*

|  |  |  |
| --- | --- | --- |
| TEAM | METHOD | ACCURACY |
| DeepKnowledge Seville [[11](#fno_dks)] | CNN with 3 Spatial Transformers | 99.71% |
| IDSIA [[12](#fno_idsia)] | Committee of CNNs | 99.46% |
| COSFIRE [[13](#fno_cosfire)] | Color-blob-based COSFIRE filters for object recognition | 98.97% |
| INI-RTCV [5] | Human Performance | 98.84% |
| sermanet [[14](#fno_sermanet)] | Multi-Scale CNNs | 98.31% |
| CAOR [[15](#fno_caor)] | Random Forests | 96.14% |
| INI-RTCV [5] | LDA on HOG 2 | 95.68% |
| INI-RTCV [5] | LDA on HOG 1 | 93.18% |
| INI-RTCV [5] | LDA on HOG 3 | 92.34% |

# **Beyond State of the Art**

Most of the datasets used in the state-of-the-art algorithms are focused either on one type of sign or have its images with a great focus on the area that the sign is in. Our goal is to start by feeding the algorithm a traffic sign focused dataset, then as we consolidate it, to increase the detection of panoramic images. The datasets used in some challenges have different flaws, mainly because they focus on a specific target, rather than the real life situations.

Another challenge that we are keen to explore is doing the final identification in a time efficient manner. We will take into account that the final solution might be ported on an image capturing device to identify the traffic signs that the driver may not be aware of, in a way that the result of the processing will still be relevant to the user.

We will dive into the user experience side of the problem too. As mentioned before, displaying the result in a way that it does not become a distraction is an issue which has yet to be approached. This not only means that the result has to be presented in a certain way, but a selection process has to take place before deciding if the data is worthy of displaying.

Even though it might sound that we have unreachable goals in mind, we are looking forward to exploring them, and at least making the foundation for the next generation of traffic sign recognition software.

# **Proof of concept**

In our first prototype of our application, we succeeded in building a CNN model based on the GTSRB dataset [[20](#fno_GTSRB)]. In order to accomplish the task, we used Python programming language, along with Numpy module for mathematical calculations, OpenCV for image processing, Tensorflow module and Keras API for neural networks and deep learning support and also Scikit-learn library for easy training and testing a machine learning model.

## Preliminary architecture

The GTSRB dataset is loaded using the images and their labels, described in code as X and y. Since the input images are fairly small (all images are resized to 30x30 pixels), the CNN will run over each image very quickly. The images and their corresponding labels will be splitted into a training model and test model. The shape of X\_train will be (62734, 30, 30, 3) , where the first number represents the number of images on which the model is trained, and the shape of X\_test will be (15684, 30, 30, 3) , where the first number represents the number of images that are being tested on, the next two numbers in each variable is the size of an image, and the last number is the number of color channels, in this case 3 for RGB model . The model is built using Keras Sequential function, which allows us to build the model layer by layer. The layers that will be using are Conv2D layers, which are convolutional layers that uses the input images, seen as 2D matrices, MaxPool2D layers for down-sampling, Dropout layers, which uses a technique of ignoring random neurons based on a rate to better train a model, Flatten layers to make a connection between a Conv2D layer and a Dense layer, latter’s being the output layer in the case of neural networks. The activation method used in Conv2D layers and some Dense layers (except last) will be ReLU, which stands for Rectified Linear Activation and has been proven to work well in neural networks. The last Dense layer will be using the ‘softmax’ activation to transpose the results into probabilities, and a number of 43 nodes, one for each possible class outcome - describing the road signs.

Compiling the model will take as input the Adam optimizer for adjusting the learning rate throughout training, the loss parameter will be ‘sparse\_categorical\_crossentropy’ in order to skip the manual encoding of y variable. The ‘fit’ function will be used for training the model, where the epochs parameter will specify the number of the model run cycles through the data.

In order to classify a traffic sign, it’s mandatory to first detect the sign. An easy way to understand how the application works is presented in the figure below through a pipeline diagram.



*Figure 1: Pipeline diagram of our implementation*

The presumed context is simple: a vehicle equipped with a camera can record the road and capture essential data for detection and recognition of traffic signs, which then can be displayed to the driver for warnings, alerts and informative notifications about the surroundings. From the set of frames of the video input, a carefully selected frame is chosen for detection, the detected regions are then validated before being fed up as input for the recognition classifier, which is based on the model trained using CNN. The recognized signs and based on some other parameters are being analyzed in order to process what message should be provided to the driver as feedback.

There were two approaches tried in order to detect the signs: first one using Fast R-CNN method, which took a long time to train and didn’t give a concludent and satisfactory result, and second one using Maximally Stable Extremal Regions (MSER) which requires some preprocessing of the input image.

For the MSER method, we need to split detection by color (currently we only look for red or blue colored traffic signs), and keeping in mind this idea, we use two different routines for each color in order to achieve best results. Firstly, to address the problem of computing the mask of red colored signs, we need to process the image by performing a contrast normalization over each channel and then normalizing the red channel intensity. The red mask is obtained by binary thresholding with a threshold value close to the maximum intensity value. Secondly, the blue mask is computed by enhancing the contrast of the original image and then converting it to HSV color model for ease of segmentation of the blue color. This way, we can augment out the blue area by defining a lower and upper limit of the blue mask (see figure 2). The resulting red mask and blue mask are merged by a bitwise operation, afterly the merged mask is dilated to enhance its features. Using this mask, the MSER method is applied to detect the regions of presumed road signs. Lastly, some of the regions of interest (ROIs) are dropped out by specifying a minimum area, the bounding boxes detected needs to be square-like and also we are considering the case were boxes are intersecting with each other or included one in another, and if the intersection is large enough, then we can unite one with another. The output of the detection algorithm will be the cropped final ROIs that will be passed on to the recognition algorithm as shown in the middle of the figure 2.



*Figure 2: First line - blue mask and red mask*

*Second line - enhanced merged mask and original image with sign detected in a bounding box*

*In middle - cropped output traffic sign*

## Preliminary results

For training our model we used 3 epochs, which took around 160 seconds per epoch to train and then tried different input and compiling parameters.

*Table 2: Performance of various CNN models in the preliminary development process*

|  |  |  |
| --- | --- | --- |
| CNN LOSS FUNCTION | COLOR MODEL | ACCURACY |
| sparse\_categorical\_crossentropy | RGB | 96.31% |
| sparse\_categorical\_crossentropy | BGR | 97.69% |
| sparse\_categorical\_crossentropy | Grayscale | 97.72% |
| categorical\_crossentropy | RGB | 97.19% |
| categorical\_crossentropy | BGR | 98.61% |
| categorical\_crossentropy | Grayscale | 99.05% |

We were satisfied with our first prototype accuracy, the best model (see Table 2) being the model with the images input converted into grayscale and compiled with ‘categorical\_crossentropy’ loss, that produced an accuracy of 99.05%, and not only has the highest accuracy, but it performs 4 seconds faster than the second most accurate that uses OpenCV’s BGR model. We expected that the grayscale model would perform better after studying other papers, and since it has only one channel of color, this explains why it is faster than the other two, but it’s surprising that the BGR model actually came closer to the grayscale model than the RGB model.

Based on the results for detection, which didn’t perform well in low light or high light conditions, we concluded that in order to achieve best results, it’s ideal to have some other inputs than the camera input, that describes the current road condition. These parameters can be obtained from either application integration with the car, which will be equipped with rain and light sensors or other new technology, or they can be obtained from a Bluetooth connection with the driver’s smartphone device that can access the internet and the current location in order to retrieve weather data, which will be passed on as input for our pipeline.

# **Implementation versus State-of-the-Art**

In this chapter, we aim to make a comparison between our implementation and the results from previous competitions with a given dataset. This time we will be looking at the German Traffic Sign Detection Benchmark and the German Traffic Sign Recognition Benchmark.

## Detection

The German Traffic Sign Detection Benchmark is a single-image detection assessment for researchers with interest in the field of computer vision, pattern recognition and image-based driver assistance. It consists of 900 images, divided into 600 training images and 300 evaluation images, and it is divided into three categories that suit the properties of various detection approaches with different properties.

Initially there were three baseline approaches, trained independently. These were the Viola-Jones detector, the HOG feature approach, and a model-based method, Hough-like voting.

It turned out that the detection rate of the Viola-Jones approach was the highest from the pool of independent methods of the chosen category, and the HOG classifier performed comparably well too. It is also notable that the general performance dropped for the mandatory (blue circular) and danger signs (red triangular). Both the model-based and the HOG method could handle this difficulty better due to the use of higher-order shape features.

*Table 3: The detection rate of all the preliminary algorithms [21]*

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Prohibitive | Danger | Mandatory |
| HOG | 91.3% | 90.7% | 69.2% |
| Hough-like | 55.3% | 65.1% | 34.7% |
| Voila-Jones | 98.8% | 74.6% | 67.3% |

The preliminary conclusion was that the classic general-purpose detectors yielded very permission results and clearly outperformed a state-of-the-art model-based approach. However, the performance on the special subsets, such as the mandatory signs, was yet too low for a possible industrial approach.

Therefore a challenge was issued, and teams all around the globe came with new ideas that would prove to be more efficient than expected.

There are seven teams that managed to differentiate themselves from the others by their astonishing results. These teams managed to achieve perfect results in a category.

*Table 4: Competition Ranking by Area-Under-Curve (Average Overlap) [21]*

|  |  |  |  |
| --- | --- | --- | --- |
| Team | Prohibitive | Danger | Mandatory |
| wgy@HIT501 | 100% | 99.91% | 100% |
| visics | 100% | 100% | 96.98% |
| LITS1 | 100% | 98.85% | 92% |
| BolognaCVLab | 99.98% | 98.72% | 95.76% |
| NII-UIT | 98.11% | - | 86.97% |
| wff | - | 99.78% | 97.62% |
| milan | - | 96.55% | 96% |

One thing that can be observed is that the mandatory traffic signs are harder to detect. This can also be attributed to their blue color shades, that seem to be hard to distinguish in natural scenes, and the fact that they are installed near the ground, which makes them prone to deterioration or vandalism.

## Recognition

The German Traffic Sign Recognition Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks. The dataset is large and it is considered a lifelike database. It consists of more than 40 classes, and contains more than 50,000 images in total.

In the final competition stage of GTSRB, four teams managed to differentiate themselves from the rest.

*Table 5: The final ranking of GTSRB IJCNN [20]*

|  |  |  |  |
| --- | --- | --- | --- |
| Team | Representative | Method | Correct recognition rate |
| IDISA | Dan Ciresan | Committee of CNNs | 99.46% |
| INI | - | Human Performance | 98.84% |
| sermanet | Pierre | Sermanet | 98.31% |
| CAOR | Fatin Zaklouta | Random Forests | 96.14% |

During the GTSRB competition, various upgraded algorithms were presented, and a detailed comparison of the traffic sign recognition performance of state-of-the-art machine learning algorithms and humans was made.

The GTSRB conclusion was that, even though the best individual in the human performance experiment achieved a close-to-perfect accuracy of 99.22%, it was outperformed in this challenging task by the best-performing machine learning approach, a committee of convolutional neural networks, with 99.46% correct classification rate. In contrast to traditional computer vision, where hand-crafted features are common, convolutional neural networks are able to learn task-specific features from raw data.[4] However, in return, “finding the optimal architecture of a ConvNet for a given task remains mainly empirical”.[22]

We managed to be as close to the human performance experiment as the latter got to the machine learning approach. Therefore, the difference between our approach, which used categorical crossentrop loss on a grayscale color model and has a correct recognition rate of 99.05%, and the human performance experiment is only 0.17%. Comparing our result with the ones from the GTSRB final ranking would place us in second place, with 0.21% above the third place, and 0.41% under the first place.

# **Fine tuning and final application**

After some testing and other considerations, we decided to replace the MSER method used for detection, which had a low accuracy of detection, around 60%, and many unwanted regions were seen as candidates because of the different lighting scenarios. Since the Fast R-CNN wasn’t convincing enough at a first try, we turned ourselves towards a deep neural network (DNN) state of the art solution which seems promising for this field.

You only look once (YOLO) is a system for real-time object detection. It can process images at 30 FPS, and even at 45 FPS, higher than real time, with CUDA acceleration.[28] There are several versions of YOLO, the one which suits best our interests is YOLOv3, which in comparison with YOLOv4, trades speed for accuracy, since we will relay on a GPGPU acceleration later on to improve speed detection.

## Final improved architecture

Pretrained YOLO uses Darknet architecture and comes with 80 classes of the COCO (Common Objects in Context) dataset, but in our case we will need only 4 classes for the GTSDB dataset in YOLO format, which are prohibitory, danger, mandatory and other, and for each of these classes, there will be grouped all 43 subclasses, from speed limit signs to priorities and directions signs. The model is trained in 8000 iterations in order to achieve an accuracy of 97.20%, which took around 8 hours to train and gives a much better result than the MSER implementation.

A better result can be obtained for detection when training, by iterating twice through the dataset for each epoch. The accuracy in this case increases to 99.11%, but of course, this kind of workaround takes twice the time to train.

The average data loss during the detection and recognition process is around 3.69%, mainly because of different light conditions or camera artifacts, like motion blur and out of focus signs. The low light performance is quite good, the data loss in this case is not that meaningful.

## Unifying models

The first step taken for testing, was to try the application on the images from the GTSDB dataset. In order to unify the models in a single pipeline, the application loads the saved YOLO weights, the YOLO configuration file and the custom CNN trained model. For each image, a set of regions of interest are selected using the YOLO model and the candidates with the best confidence scores and above a threshold are kept for recognition. Based on a prediction with maximum probability, computed using softmax, which returns distributed probabilities, a bounding box with the class description and confidence percentage is shown on the image for the recognized sign.



*Figure 3: Applications detection and recognition results on the GTSDB dataset*

In figure 3, the output of the demonstrator application is shown, where signs detected by YOLO trained model are outlined in a rectangle shape with a text description, consisting of the class prediction and it’s confidence level of the recognition model, that are displayed in various colors. The detection and recognition rates perform best when the supposed traffic signs are not too far and not too close to the camera that is capturing, meaning the ideal distance between the road sign and the camera is approximately 10 to 20 meters.

On the same level, a streamed video can be fed as input into the pipeline for a real-time traffic sign recognition. Based on some parameters like display time or accuracy variation, and the output of the recognition output or the class type output, different outcome cases can be made, for example, how much time the sign is displayed on the screen of a car, or how or when the driver is notified about the road sign or conditions and so on. For these particular cases, we would like to try to create a graphical user interface using Python library tkinter, for users to test, where based on a video input, a traffic sign is detected and classified in order to create a notification that can be displayed to the user, who can provide feedback about which notifications are useful or not and under what circumstances.

## Bug tracking and code refactoring

Mainly the bugs that we encountered were about miss interpretation of the output files cases and conversion type errors due to human errors. For example, for the video stream, the OpenCV library breaked during a detection and recognition session because a frame wasn’t able to be read or because the last frame wasn’t detected and the loop kept reading none type variables. The solution for these kinds of bugs was to introduce new case statements that will either break the loop or pass to another frame if it exists.

Another matter that we faced was about the Python’s OpenCV library not detecting the CUDA toolkit, in order to obtain real time performance for our demonstrator application. The OpenCV, CUDA and cuDNN (CUDA DNN) are not very easy to configure for Windows. The not-friendly at all workaround is to install the OpenCV library using CMake, and for this there is no precompiled version of the OpenCV for Windows, which to create the compiled version is another time consuming task.

# **Future work**

## Application Integration

The final and perfected application would provide options for both online and offline users.

The offline application should work well with the proposed algorithm implementation, while the online part would be served using a web API for handling the detection and recognition tasks.

An example of a framework that could be used in this case, is Flask, a micro web framework written in Python. It does not require particular tools or libraries and has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

The flask dependencies are as follows:

* Request: Used to process the request object coming to the API. It will be used for accepting the image file sent by the application.
* Jsonify: As expected, it's used for creating json objects as responses from the API to the application.
* Render Template: A method that takes the html file provided from the templates folder and renders it on the viewport. It’s used for providing a simple home page for the API in case someone tries to access it from a web browser.

**Application Workflow**

The client device will be used mainly for image capturing. It then will send a post request to the server, with the image attached. The server will take care of the prediction stage, which will end with sending a post request back to the client application, containing the prediction results. Depending on the application, it could access a dictionary to display the result in different ways, for example as an image or as sound effects or different types.

## NVIDIA AV

Tapping into decades-long experience in high-performance computing, imaging, and AI, NVIDIA has built a software-defined, end-to-end platform for the transportation industry that enables continuous improvement and deployment through over-the-air updates. It delivers everything needed to develop autonomous vehicles at scale.[23]

**Artificial Intelligence Infrastructure**

NVIDIA DRIVE Infrastructure is a complete workflow platform for data ingestion, curation, labeling, and training plus validation through simulation. Building autonomous vehicles requires massive amounts of data. Managing and curating this data takes high-performance computation, as well as intelligent training methods.

NVIDIA DRIVE DGX Systems and advanced training tools enable streamlined, large-scale training and optimization of deep neural networks (DNNs). Using the power of GPUs and AI, developers can comprehensively train DNNs for autonomous vehicle perception, planning, driving, and more.

It’s impossible for an autonomous vehicle to encounter every possible traffic situation while testing on public roads. In DRIVE Sim, virtual vehicle fleets can drive millions of miles across a broad range of scenarios, from routine driving to rare or even dangerous situations, with greater efficiency, cost-effectiveness, and safety than in the real world. The DRIVE Constellation simulation platform comprises two side-by-side servers that generate the sensor output from the virtual car and streams that data into the DRIVE AGX AI car computer running the AV stack to make real-time decisions. Vehicle control commands are then sent back to the simulator. This closed-loop process enables bit-accurate, timing-accurate, hardware-in-the-loop testing.

As autonomous driving software develops and improves, it’s vital that new versions can be tested against previously captured sensor data to avoid regression. With the DRIVE Constellation hardware-in-the-loop platform, developers can replay driving data and compare the performance of the latest self-driving system to past versions. When combined with simulation testing, the DRIVE Constellation platform provides a comprehensive solution to cloud-based validation of autonomous driving technology.[24]

**Self-Driving Hardware and Software**

The scalable, software-defined NVIDIA DRIVE AGX platform delivers industry-leading performance, enabling autonomous vehicles to process large volumes of sensor data and make real-time driving decisions. The open NVIDIA DRIVE Software stack also helps developers build perception, mapping, planning, and driver monitoring capabilities with redundant and diverse DNNs. The platform is always becoming more capable through continuous iteration and over-the-air updates.

The foundation of the DRIVE Software stack, DRIVE OS is the first safe operating system for in-vehicle accelerated computing. It includes NvMedia for sensor input processing, NVIDIA CUDA libraries for efficient parallel computing implementations, NVIDIA TensorRT for real-time AI inference, and other developer tools and modules to access hardware engines.

NVIDIA DriveWorks provides middleware functions on top of DRIVE OS that are fundamental to autonomous vehicle development. These consist of the sensor abstraction layer (SAL) and sensor plug-ins, data recorder, vehicle I/O support, and a deep neural network (DNN) framework. It’s modular, open, and designed to be compliant with automotive industry software standards.

In the planning and control layer, the NVIDIA Safety Force Field computational module keeps a vehicle out of harm’s way and ensures that it won’t contribute to or cause an unsafe situation.

NVIDIA Map is built to be safe, scalable, and always up-to-date. It uses perception results from the DRIVE Hyperion sensor set to identify intersection details, traffic lights, parking spots, and road boundaries and determine safe driveable paths. It also uses the NVIDIA DGX SuperPOD infrastructure to maintain these maps at a global scale. These AI systems ingest terabytes of data to create and update maps worldwide.

**Production-Ready AV Platform**

NVIDIA DRIVE Hyperion is a production-ready platform for autonomous vehicles. This AV reference architecture accelerates development, testing, and validation on the path to production by integrating DRIVE Orin-based AI compute with a complete sensor suite that includes 12 exterior cameras, three interior cameras, nine radars, 12 ultrasonics, and one front-facing lidar, plus one lidar for ground truth data collection. DRIVE Hyperion features the full software stack for autonomous driving (DRIVE AV) as well as driver monitoring and visualization (DRIVE IX), which can be updated over-the-air, adding new features and capabilities throughout the life of the vehicle.[25]

The NVIDIA DRIVE Orin SoC (system-on-a-chip) delivers 254 TOPS (trillion operations per second) and is the central computer for intelligent vehicles. It’s the ideal solution for powering autonomous driving capabilities, confidence views, digital clusters, and AI cockpits. The scalable DRIVE Orin product family lets developers build, scale, and leverage one development investment across an entire fleet, from Level 2+ systems all the way to Level 5 fully autonomous vehicles.

NVIDIA DRIVE AGX Pegasus uses the power of two NVIDIA Xavier SoCs and two Turing GPUs to achieve 320 TOPS of super compute capability. The platform is designed and built for all types of autonomous systems, including robotaxis.

NVIDIA DRIVE AGX Xavier delivers 30 TOPS for Level 2+ and Level 3 automated driving. At its core is the first-ever production auto-grade Xavier SoC, which incorporates six different types of processors, including a CPU, GPU, Deep Learning Accelerator (DLA), Programmable Vision Accelerator (PVA), Image Signal Processor (ISP), and stereo/optical flow accelerator.

The next generation SoC NVIDIA DRIVE Atlan will be a data center on wheels, complete with the performance, safety, and security found in data center technology. Atlan is designed to achieve 1,000 TOPS of computing performance, the basis for a safe and secure autonomous vehicle development platform in the future.[26]

**Intelligent Assistants**

NVIDIA DRIVE Chauffeur is built on NVIDIA DRIVE Orin and the NVIDIA DRIVE SDK. It features the perception, mapping, and planning layers, as well as diverse DNNs trained on high-quality, real-world driving data and synthetic data, to handle both highway and urban traffic scenarios. These perception outputs can be used for both autonomous driving and mapping, delivering a personal chauffeur for your daily drive.[27]

DRIVE IX is an open software platform that delivers interior sensing for innovative AI cockpit solutions. It provides perception applications to access features and DNNs for advanced driver and occupant monitoring, AR/VR visualization, and natural language interactions between the vehicle and passengers.

Built on NVIDIA DRIVE IX and Omniverse Avatar for real-time conversational AI, NVIDIA DRIVE Concierge gives vehicle occupants access to new, always-on intelligent services. Omniverse Avatar lets DRIVE Concierge serve as everyone’s digital assistant, making recommendations, helping book reservations, making phone calls, accessing vehicle controls, and providing alerts using natural language. DRIVE Concierge also provides a dashboard view into what the DRIVE Chauffeur sees around the car and what it’s planning. Plus, it serves as a valet, automatically parking and summoning the car.

# **Conclusions**

## Results

We managed to score some very precise results in the recognition stage (see Table 2). The highest precision was from the categorical cross entropy loss function, on the grayscale color mode, with a value of 99.05%.

Comparing our results with the ones from GTSRB IJCNN (see Table 5), we can easily say that we could have been a great match for first place during the competition.

*Table 6: The final precision comparison*

|  |  |
| --- | --- |
| Source | Precision |
| Best GTSRB Machine Learning Algorithm | 99.46% |
| GTSRB Human Precision | 99.22% |
| Us | 99.05% |

We are extremely close to the measured human precision, and as mentioned in numerous articles, any recognition software that goes beyond 98% precision, is a good one to be further developed and implemented into Level 2+ systems, which are those that do not drive the automobile, but offer support to the driver.

Further development doesn’t only mean making it more precise, but finding spots where it does not shine in, and finding solutions for said areas. For example, taking into account road conditions, weather conditions, and driver actions.

Implementations could be made by fully integrating the recognition software into the automobile’s core system, such as an integrated camera, along with surrounding sensors, or by making it into an accessory, such as a bluetooth dashboard camera connected to the driver’s mobile phone, which will serve as a display endpoint.

**References**

[[1](#fnote_WHO)] Road traffic injuries, 2021, Available at: <https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>, Accessed at: 02.11.2021

[[2](#fnote_EULaw)] Cars of the future need "Readable" roads now - press release, 2013, Available at: <https://www.euroncap.com/en/press-media/press-releases/cars-of-the-future-need-readable-roads-now/>, Accessed at: 02.11.2021

[[3](#fnote_EuroNCAP)] Procedure 2018/0145/COD, 2019, Available at: <https://eur-lex.europa.eu/procedure/EN/2018_145>, Accessed at: 02.11.2021

[[4](#fnote_mvc)] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, Man vs. computer: Benchmarking machine learning algorithms for trafﬁc sign recognition, Neural Networks, vol. 32, 2012  
[[5](#fnote_dccs)] D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, A committee of neural networks for trafﬁc sign classiﬁcation, IEEE International Joint Conference on Neural Networks, 2011,pp. 1918–1921

[[6](#fnote_TemplateMatching)] M. Shneier, Road Sign Detection and Recognition, IEEE Computer Society International Conference on Computer Vision and Pattern Recognition, 2005

[[7](#fnote_SVM)] D. Soendoro, I. Supriana, Traffic sign recognition with Color-based Method, shape-arc estimation and SVM, Proceedings of the 2011 International Conference on Electrical Engineering and Informatics, IEEE, Bandung, Indonesia, 2011, pp. 1-6, 10.1109/ICEEI.2011.6021584

[[8](#fnote_smallCNN)] W. Li, D. Li, S. Zeng, Traffic Sign Recognition with a small convolutional neural network, IOP Conference Series: Materials Science and Engineering, vol. 688, issue 4, IOP, 2019

[[9](#fnote_CNNGPGPU)] K. Lim, Y. Hong, Y. Choi, H. Byun, Real-time traffic sign recognition based on a general purpose GPU and deep-learning, PLoS ONE, 2017, 10.1371/journal.pone.0173317

[[10](#fnote_Nvidia)] Self-Driving Cars, Available at: <https://www.nvidia.com/en-us/self-driving-cars/>, Accessed at: 01.11.2021

[[11](#fnote_dks)] Álvaro Arcos-García, Juan A. Álvarez-García, Luis M. Soria-Morillo, CNN with 3 Spatial Transformers, Neural Networks, vol. 32, 2012

[[12](#fnote_idsia)] D. Ciresan, U. Meier, J. Masci, J. Schmidhuber, Multi-column deep neural network for traffic sign classification, Neural Networks, vol 32, 2012

[[13](#fnote_cosfire)] Baris Gecer, George Azzopardi, Nicolai Petkov, Color-blob-based COSFIRE filters for Object Recognition, Image and Vision Computing (57), 2017

[[14](#fnote_sermanet)] P. Sermanet, Y. LeCun, Traffic sign recognition with multi-scale Convolutional Networks, International Joint Conference on Neural Networks (IJCNN), 2011

[[15](#fnote_caor)] Zaklouta, B. Stanciulescu, O. Hamdoun, Traffic sign classification using K-d trees and Random Forests, International Joint Conference on Neural Networks (IJCNN), 2011

[[16](#fnote_tc47)] Huang Z, Yu Y, Gu J, Liu H, An efficient method for traffic sign recognition based on extreme learning machine, IEEE Trans Cyber 47, 2018

[[17](#fnote_tc17)] Yang Y, Luo H, Xu H, Wu F, Towards real-time traffic sign detection and classification. IEEE Trans Intell Transp Syst 17, 2016

[[18](#fnote_zt13)] Zaklouta F, Stanciulescu B, Real-time traffic-sign recognition using tree classifiers. IEEE Trans Intell Transp Syst 13, 2012

[[19](#fnote_gt13)] Greenhalgh J, Mirmehdi M, Real-time detection and recognition of road traffic signs. IEEE Trans Intell Trans Syst 13, 2012

[[20](#fnote_GTSRB)] German Traffic Sign Benchmarks, 2019, Available at: <https://benchmark.ini.rub.de/>, Accessed at: 10.11.2021

[21] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, Christian Igel, Detection of Trafﬁc Signs in Real-World Images: The German Trafﬁc Sign Detection Benchmark, ResearchGate, 2013

[22] Traffic sign recognition with multi-scale convolutional networks, Proceedings of the IEEE international joint conference on neural networks, IEEE Press, 2011

[23] NVIDIA at CES 2022, Available at: <https://www.nvidia.com/en-us/events/ces/>, Accessed at: 15.01.2022

[24] Nefi Alarcon, NVIDIA Developer Blog: GTC Autonomous Vehicle Presentations , Available at: <https://developer.nvidia.com/blog/gtc-autonomous-vehicle-presentations/>, Accessed at: 15.01.2022

[25] End-to-end solutions for autonomous vehicles, Available at: <https://developer.nvidia.com/drive>, Accessed at: 15.01.2022

[26] Nvidia GPU Technology Conference, Available at: <https://www.nvidia.com/gtc/>, Accessed at: 16.01.2022

[27] Dave Altavilla, NVIDIA Blitzes Markets With Powerful New Tech For AI, The Metaverse, Autonomous Vehicles And More, Forbes, 09.11.2021

[28] YOLO: Real-Time Object Detection, Available at: <https://pjreddie.com/darknet/yolo/>, Accessed at: 26.01.2022