Neural Network-based Multi-Class Traffic-Sign Classification with the German Traffic Sign Recognition Benchmark

Csanád Ferencz, Máté Zöldy

Department of Automotive Technologies, Faculty of Transportation and Vehicle Engineering, Budapest University of Technology and Economics, Stoczek 6, 1111 Budapest, Hungary; E-mail: csanadferencz@edu.bme.hu, zoldy.mate@kjk.bme.hu

Abstract: Traffic-sign detection has an essential role in the field of computer vision, having many real-world applications more and more object recognition and classification task is being solved by using Convolutional Neural Networks (CNNs or ConvNets), especially in the field of intelligent transportation. In the present article, we offer an implementation chosen from several CNN-based traffic-sign recognition and classification algorithm architectures, using a ConvNet classifying 43 different types of road traffic signs in the TensorFlow framework, as part of the German Traffic Sign Recognition Benchmark (GTSRB) competition. A Deep ConvNet was trained end-to-end, aiming to improve the prediction performance of a DCNN-based autonomous driving system equipped with a front-facing digital camera, with as input a sequence of images, as output directly the prediction results. The results obtained on held-out data demonstrated the high accuracy of the model, matching the state-of-the-art multi-class recognition and classification accuracies, as well as related human-level recognition performances.

Keywords: convolutional neural networks; end-to-end classification model; German traffic-sign recognition benchmark (GTSRB); traffic-sign recognition; cognitive mobility

1 Introduction

As a result of, their fast execution and high recognition rate, Deep Convolutional Neural Networks (DCNN-s) have been pointed out in recent years to offer state-of-the-art traffic sign classification performance, beyond what could be accomplished by former state-of-the-art methods.

The emergence of modern discoveries and developments in deep learning gave rise to positive state-of-the-art results for traffic sign classification and recognition, with most of the research focusing on designing and developing cognitive techniques, such as DCNN-s for enhanced recognition accuracy. In the realm of mobility, cognitive methods can aid in making immediate decision. At the moment, nearly all

of the state-of-art network architectures for traffic-sign classification are based on Convolutional Neural Networks (or ConvNets) [14], [18], [19].

In the present article, the authors introduce a revised end-to-end traffic sign detection and recognition system, while thoroughly describing the design and development processes of the ConvNet through which the real-time processing of images, as well as classification of localized objects are implemented [20].

The presented end-to-end cognitive approach revealed a substantial efficiency decrease during testing and development for the recognition and classification of road signs in real conditions, a simple template matching classification algorithm was not able to achieve high-quality recognition results due to limited set of predefined templates, angle of rotation, contrast in images of localized traffic signs or too intense variations in the illumination. To improve the system's recognition performance, a possible solution might be that these widely applied convolutional neural networks could be combined with the localization algorithm that has shown good results [14], [18], [19].

The experiments conducted demonstrate that the perception module shows promising performance results, and the classifier is capable of detecting and sorting the various signs properly enough regardless of quality, orientation or size. The work shows promising results that push the complexity boundaries regarding navigating self-driving cars with a cognitive toolbox through towns [18], [19].

The final model was more than capable of a smooth detection and classification of a significant number of traffic signs on the German Traffic Sign Recognition Benchmark (GTSRB) dataset, with an accuracy as high as 97.98%, even approaching the related human-level recognition performance of 98.81%. It is interesting to note that the best 13 networks submitted for the GTSRB competition all use CNNs with a classification performance of at least 98.10%. Human-level accuracy is outperformed by 5 of these, indicating once again why it is the most commonly used state-of-the-art traffic-sign classification method [25].

In the following parts of the study beginning with *Section 2*, a detailed description will be given regarding the classification task and dataset pre-processing, in *Section 3* the layer architecture definition, building and training of the model provided in *Section 4*, and last but not least *Section 5* concluding with result validation and discussion.

2 Traffic Sign Classification Task, Dataset Pre-Processing and Loading

CNN-s are frequently used for different purposes in image processing, such as segmentation, classification, or detection [25]. The traffic-sign classification problem has been addressed with various well-known methods, including neural

networks (NNs), Support-vector machine model architecture (SVM), or Naive Bayes classification. Many approaches used for traffic-sign recognition are based on sliding window methods, which resolve the classification and recognition problems simultaneously, but require unfeasible computational resources. However, several recent literature state-of-the-art systems separate recognition from classification. Firstly, we would deal with the recognition with custom-built computationally inexpensive methods, e.g. color thresholding. After that, the classification will be conducted on the detected samples with algorithms having higher accuracy.

Traffic-sign recognition is a rather constrained problem, it has many direct real-world applications, road signs being unique with little variability in appearance, fixed, and intended to be displayed for the drivers. Even though the current task is exclusively classification, when designing a classifier the ultimate objective of detection is to optimize both efficiency and accuracy. With color thresholding and various heuristics, for instance, we first detect sign shapes, followed by multi-layer NN classification for each type of traffic sign outer shape. The method has the advantage of fast training, reduced chance of overfitting, needs fewer samples, and it is great for local information capturing and reducing the complexity of a given model [6].

The German traffic sign detection dataset consists of 39 209 image samples corresponding to 43 different classes (ranging from 0 to 42), each one of the signs having its folder of different image sizes and resolutions. The training dataset is made up of 34 799 images, the validation dataset of 4 410 images, and respectively the test set of 12 630 images. Because samples are divided unevenly between the related classes, our model could predict some of the classes with higher accuracy than others. The samples are composed of video sequence frames of 1 [s], and as the camera approaches a given sign each real-world sample produces 30 samples increasing in resolution [23], [24], [25]. The total number of samples present in our dataset exceeds 50 000 images.

Thus, we will categorize 40 × 40 RGB pixel space input images into 43 possible traffic sign categories: h: $\mathbb{R}^{4800} \mapsto \{0, 1, \ldots, 42\}$. Note that due to the three-color channels, we have a 40 × 40 × 3 = 4800 - dimensional input. In this task, convolutional neural networks (CNN-s) usually perform at around 90-95% accuracy [8], [24], [25]. We will try to train a network that performs better than humans — 98.9% on the original German Traffic Sign Recognition Benchmark (GTSRB).

Many images from the sample set given by the GTSRB competition present several difficult challenges, some of these are very hard to distinguish and classify even for a human (low-contrast, viewpoint variations, motion-blur, physical damage, occlusions, colors fading or input resolution as low as 15 × 15). However, we can assume that the ground truth that we have is exact.

Because the image samples present in the dataset dynamically varies between a wide range of dimensions from 15 × 15 × 3 to 250 × 250 × 3, it is not possible to pass them directly to the CNN model, hence this cannot be trained on different image dimensions, but we also need to avoid stretching the images too much when compressing them to a single dimension. All things considered, we will use an already pre-processed version of the images re-sized to 40 × 40 × 3.

Furthermore, because the present project's objective is to build a robust recognizer excluding temporal evidence accumulation, road signs in the training dataset will be available as video sequences, and temporal information will not be in the test dataset [6], [23], [24], [25]. Moreover, we will populate and diversify the original set of data as well with various image modifying techniques such as rotations, color distortions and blurring techniques. Therefore, at this stage we are performing data augmentation (see *Figure 1*).

Figure 1
Data augmentation creating modified input sample versions [5]

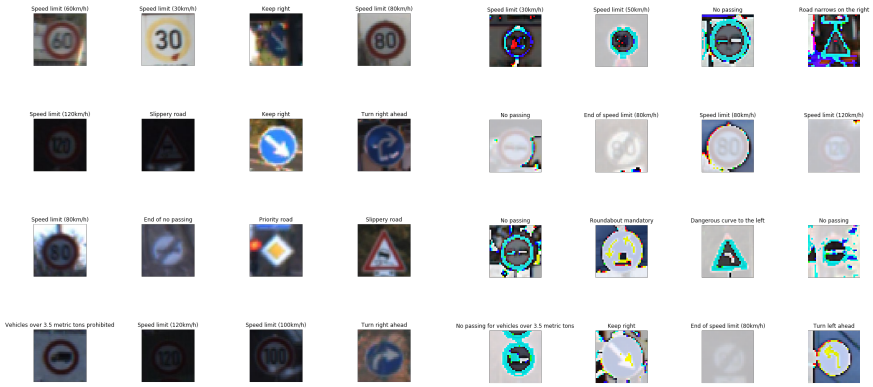

Figure 2
Labeled input dataset before and after augmentation, pre-processing and normalization [5]

With this solution basically we are adding more sample to our dataset, but without collecting any new one. The samples following data augmentation, pre-processing

and normalization will look like as presented in *Figure 2*. All of the dataset image samples were normalized, in order to help the model converge faster, the data having a mean of zero and equal variance.

After finding out the model's actual accuracy by training on the original dataset, in the next step by adding even more data and evening out the classes we will check the accuracy again [6]. The purpose of the test set would be to detect, during training, that the model has overfitted and to implement, e.g. early stopping, dropout or some other method to compensate. Once the model is completely trained, the validation set needs to be simply run through the model and the results reported. In order to have diverse datasets and higher prediction performance on unseen data, we will extract randomly one sample per class for validation, instead of mixing all images randomly and separating into similar training and validation datasets [16].

3 Network Layer Architecture

Generally speaking, similarly to regular NN, convolutional neural networks also consists of neurons having learnable biases and weights, organized in layers. Contrary to regular neural networks, however, CNNs make the most of the fact that they constrain the input architecture composed of images in a more sensible way, in particular having layers with neurons arranged in three dimensions [11], [16].

However, a ConvNet will take a two-dimensional image and progressively will process it with the convolutional layer. In the case of this classification task, the input volume samples of activations having the size of 40 × 40 × 3 (40 - width, 40 - height, 3 color channels), in the first hidden layer a single fully connected neuron would have 40 × 40 × 3 = 4 800 weights. This volume still looks computable, however, this structure certainly will not scale to larger images, e.g. size of 250 × 250 × 3, in the case of regular neural networks, because that would lead to neurons that have 187 500 weights, and because we generally want to have more than one of these neurons the parameters would add up quickly [11]. It is clear that full connectivity would not be particularly efficient, and these considerable amounts of parameters would swiftly give rise to overfitting.

As *Figure 3* illustrates, each of the ConvNet's layers will transform the three-dimensional input volume to a three-dimensional output volume with neuron activations. In the case of the left example, the red input layer's height and width will be the image dimensions, while the depth will correspond to the red, green, and blue channels. The right figure illustrates an example input volume of a 32x32x3 image and the volume of neurons in the first convolutional layer. In this case, there are 5 neurons along the depth, connected to or looking at the same region of the input volume. These 5 neurons do not represent the same weights, they are only associated with 5 different filters, sharing the same receptive field [25].

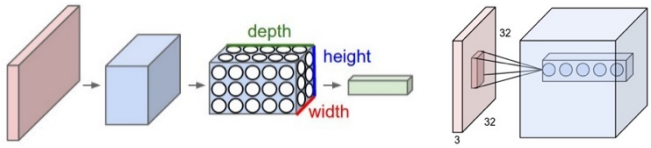

Figure 3
3-layer network vs CNN organizing neurons in three dimensions [25]

When dealing with deep CNN, normal CNNs generally have two or three layers, but deep CNNs will have multiple hidden layers usually more than 5, which are used to extract more features and increase the accuracy of the recognition. Therefore, the depth of the network shall correlate with the amount of data, a deep network with insufficient data will produce an overfitted model, while a shallow network with a lot of data will not have a high accuracy [11]. Moving from layer to layer, the low-level input feature vector is put into the first layer and transformed into a high-level feature vector. The neuron number in the output layer is equal to the classifying class number, while the vector of probability output is showing the possibility of the input vector belonging to a corresponding class. The output of a weighted adder is described by (1):

$$a_j^i = \sigma\left(\sum_k a_k^{i-1} w^i j_k\right) \tag{1}$$

where a_j^i - j^{th} neuron, i^{th} layer and w_k^{ij} - weight of synapse, connecting j^{th} neuron in the layer i^{th}, with the k^{th} neuron in the $i-1$ layer. As activation function the logistic function will be applied, frequently used in regression problems [2], [25].

In classification problems during training process the goal is to minimize the cost function by gradient decent with minimization methods. In this case, cross entropy is the most widely used cost function, $H(p,q)$ being the cross-entropy of the distribution q relative to a distribution p over a given set, presented in (2):

$$H(p,q) = -\sum_i Y(i) \log y(i) \tag{2}$$

Because in computer vision CNN classification is the state-of-the-art method for pattern recognition, our implemented network layer structure, illustrated in *Figure 4*, is also mostly based on convolutional layers, following the architecture Input — Convolutional Layer — ReLU — Pooling Layer — Fully Connected (FC) Layer. Image convolution is the simple sum of element-by-element matrix multiplication between weights and the input volume region they are connected to, followed by a summing of the elements together [25]. These layers operate in the form of sliding windows, and do not require a fixed image size, while the output of these will be the spatial arrangement of activations, or so-called feature maps of any size, which allow us to detect the same features in different locations [1], [2], [24].

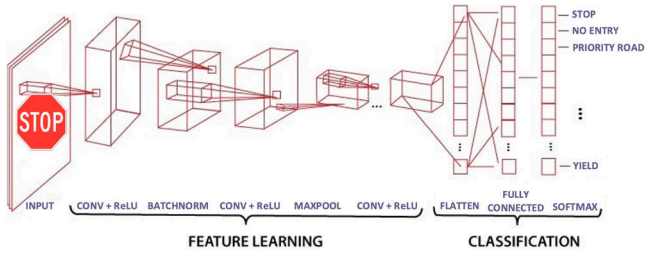

Figure 4
Applied classification layer structure (Own work, 2021)

In substance, the convolutional layer requires four hyperparameters (S - stride, K - number of filters, F - spatial extent, P - the amount of zero padding) and accepts a $W1 \times H1 \times D1$ volume size. A common setting of the hyperparameters is $F = 3$, $S = 1$, and $P = 1$. It produces a volume of size $W2 \times H2 \times D2$, where heights and widths are equally computed by symmetry, described by (3), (4), and (5):

$$W2 = \frac{(W1-F+2\cdot P)}{S} + 1 \tag{3}$$

$$H2 = \frac{(H1-F+2\cdot P)}{S} + 1 \tag{4}$$

$$D2 = K \tag{5}$$

Using sharing of parameters, for $(F \times F \times D1) \times K$ weights and K biases, $F \times F \times D1$ weights per filter will be introduced. Regarding the output volume, a $W2 \times H2$ sized d^{th} depth slice will result after performing a convolution of the d^{th} filter with a stride of S over the input volume, offset then by the d^{th} bias.

As a next layer, we implement batch normalization to stabilize training and facilitate a smoother hyperparameter tuning process, while substantially increasing the classification performance [1], [25]. This normalization operation normalizes the elements of x_i inputs by calculating the mean μ_B and variance σ_B^2 over the time, spatial and observation dimensions independently in the case of every channel, respectively also calculating the so-called normalized activations as in (6) [2]:

$$\widehat{x_i} = \frac{x_i-\mu_B}{\sqrt{\sigma_B^2+\epsilon}}, \tag{6}$$

where ϵ — constant, improving numerical stability when variance is small.

This operation additionally scales and shifts the activations, allowing those inputs with unit variance as well as zero mean to not be optimal for operations following batch normalization, using the transformation presented in (7) [1]:

$$y_i = \gamma\widehat{x_i} + \beta, \tag{7}$$

where $\widehat{x_i}$ - the resulting normalized activation having zero mean and unit variance, β — offset and γ — scale factor. These parameters are learnable and updated during

the training of the network. In order to make predictions after training, batch normalization assumes a fixed mean and variance, calculated from training data after or during training, to normalize data [2].

Between consecutive convolutional layers we can insert periodically an intermediate pooling layer, with the intention of operating independently on the input's each depth slice and successively reducing the spatial size, number of parameters, and network computation, leading also to controlling the overfitting. The most frequently used form of this is a 2×2 filter size pooling layer downsampling every input depth slice with a stride of 2 across both height and width, discarding two-thirds of the activations, but leaving unchanged the depth dimension, described by (8), (9) and (10). This pooling layer requires two hyperparameters, the S - stride and F - spatial extent, while accepting a $W1 \times H1 \times D1$ sized volume, and producing a $W2 \times H2 \times D2$ sized volume, where:

$$W2 = \frac{(W1-F)}{S} + 1 \tag{8}$$

$$H2 = \frac{(H1-F)}{S} + 1 \tag{9}$$

$$D2 = D1 \tag{10}$$

It should be also emphasized that only two predominantly used max pooling layer variation is found in practice, the more common one with $S = 2$ and $F = 2$, but also a configuration of $S = 2$ and $F = 3$, or the so-called overlapping pooling. Pooling sizes with bigger receptive fields could be too detrimental [1], [2]. *Figure 5* illustrates the most widely used max-pooling downsampling operation with a stride of 2. The $224 \times 224 \times 64$ sized input volume is pooled with stride 2 and filter size 2 into a $112 \times 112 \times 64$ sized output volume, while the volume depth is preserved [9], [10].

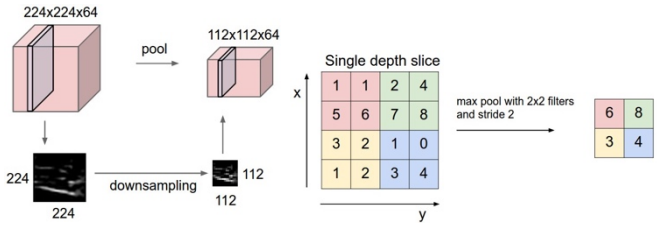

Figure 5
Illustration of pooling layer downsampling the input volume [25]

In the final four layers, we will perform a flatten operation on the last convolutional layer's output, a final batch normalization, as well as a dropout operation while entering into the last output dense layer — this will be the same as the number of classes that we have. The purpose of using dropout is to avoid overfitting and to generalize while also improving reliability. Neurons with probability p of the current layer will disconnect randomly from the next layer's neurons, going through the depth of the network, learning more filters as the network deepens.

Training deep networks using several layers with a sigmoid activation function will be complicated because of the vanishing gradient issue [9], [10], [13]. To resolve this difficulty, we will specifically use the activation *Rectified Linear Unit (ReLU)*, which applies $max(0, x)$ elementwise non-linearity activation function thresholding at zero, x being the input to a neuron, presented in (11):

$$f(x) = x^+ = max(0, x) \tag{11}$$

Using ReLU, compared with other similar activation functions such as the sigmoid function, enables a more efficient, but also faster training process of complex and large datasets attributed to deep neural architectures, leaving the volume size unchanged [12]. A *SoftMax* classifier is also added to the network to normalize the output of the previous layer, so the output of this will contain the probability values of belonging to a recognizable class (the layer's output basically will be the prediction values). The standard (unit) SoftMax function σ: $\mathbb{R}^K \rightarrow [0,1]^K$ for $i = 1, \ldots, K$ respectively $z = (z_1, \ldots, z_k) \in \mathbb{R}^K$ is defined by (12):

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{12}$$

where σ - softmax, z input vector, e^{z_i} - standard exponential function for input vector, K - number of classes in the multi-class classifier, e^{z_j} - standard exponential function for output vector. Essentially what this does is that it takes a vector z as input with K real numbers and normalizes it. The probability distribution containing K probabilities will be proportional to the input numbers' exponentials.

At the end of the network, we add a fully connected layer as well, thus realizing class score computing, leading to volumes of a 1 × 1 × 43 size, where numbers will be matched to one of the 43 class category scores of the GTSRB dataset. The network will learn filters that activate when some type of visual feature can be seen on the first layer (e.g. edge of some orientation) and ultimately entire patterns on the network's higher layers [29].

4 Building and Training the Model

An illustration of the classifying process is presented in *Figure 6*. At this point, the built network processed from the training dataset a 50-image batch over one iteration. The so-called intermediate accuracy was calculated every 100 iterations, with a batch of 50 images from the test set to report progress. After successful training, the accuracy is calculated again, using all samples from the test set.

The input image of 40 × 40 × 3 is a multi-dimensional matrix, holding raw pixel values, having, just like a traditional matrix, a width of 40 (number of columns), a height of 40 (number of rows), as well as a depth of 3 for a standard RGB image,

representing the image channel number. Plotting the histogram for the sample images in our dataset for different road traffic signs will result in *Figure 7*.

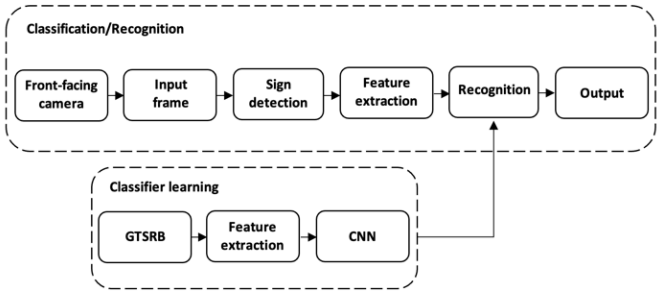

Figure 6
Simplified functional diagram of the classifying process (Own work, 2021)

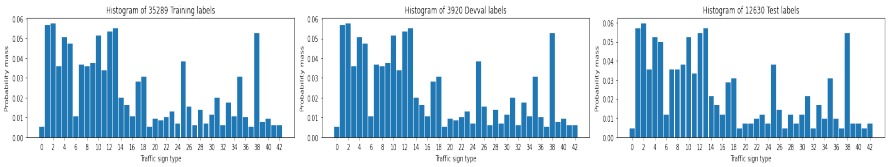

Figure 7
Histogram distribution of the augmented training image categories (Own work, 2021)

The sample numbers for each road sign are unevenly distributed between classes, not having the same number of samples for every class (also shown in the histogram distribution), leading to a biased model that recognizes and classifies some of the traffic signs more accurately than others. Thus, sample images had to be augmented for some of the classes to reach a minimum of 250 images in each class, while avoiding replicates in the input dataset [3], [26].

All coding of data arguments along with the training model is created in the *Jupyter Notebook* environment. The detailed CNN layer architecture and specification used, built with [21] and [22], is summed up in *Table 1*, respectively shown in *Figure 8* and *Figure 9* [8]. Among the types of layers used are two-dimensional convolution layers (Conv2D), batch normalization, two-dimensional max pooling, flatten, dropout, and dense - with detailed descriptions in the previous section [4].

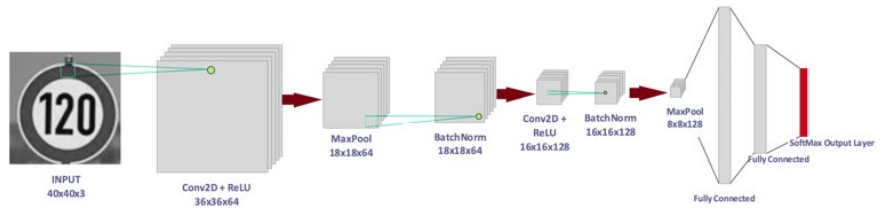

Figure 8
Implemented deep convolutional neural network layer architecture (Own work, 2021)

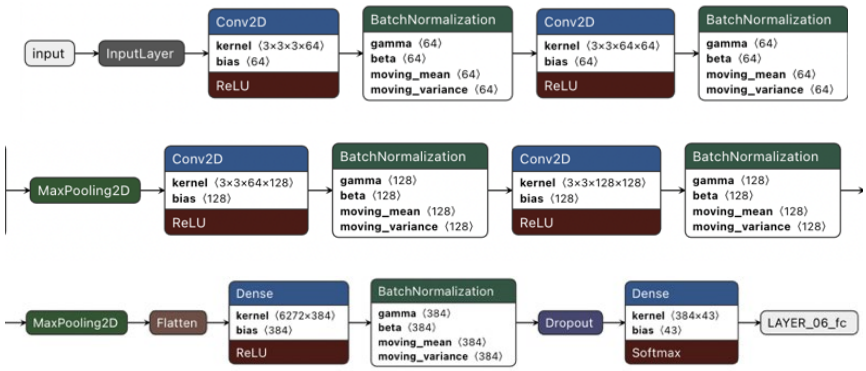

Figure 9

Deep convolutional neural network architecture specifications layer-by-layer (Own work, 2021)

The entire procedure for model training requires about two hours over a single GPU, the total number of parameters is 2 688 619 (from which 2 687 083 are trainable and 1 536 non-trainable), the training process is carried out with an Intel Core i7-4790K - 32 GB RAM, 4.00 GHz processor and Ubuntu-64 bit operating system [1], [2]. We will be using *Python 3.x* with *TensorFlow*, the scripts only reference standard libraries available over the 'pip' package manager, such as *os*, *time*, *numpy*, *zipfile*, or *matplotlib* backend.

To maximize likelihood (MLE) we have to minimize the sum/mean/root-mean-squared error (SSE/MSE/RMSE) [3], [4], [6]. Consequently, for result evaluation we adopt this MSE, the default loss for regression problems used to predict continuous target values, calculated as the average of the squared differences between the predicted ($\hat{y}_i$) and actual (y_i) values, shown in (13):

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \tag{13}$$

A supervised training gradient-based method is updating each one of the filters in each layer, in each filter bank in such a way that it minimizes the loss function, while the last stage's output is fed to a classifier. In the case of classification problems, we approximate $p(Y|X)$ k-class discrete conditional distribution with $q(\hat{Y}|X)$ modeling distribution $\mathbb{R}^n \rightarrow [0,1]^k$ [8], [15].

Table 1

Network layer architecture, parameters and sizes

Layer type	**Filter nr.**	**Kernel size**	**Activ. fc.**	**Output shape**	**Param. nr.**
Conv1	64	3x3	RELU	(None, 38, 38, 64)	1 792
BatchNorm	-	-	-	(None, 38, 38, 64)	256
Conv2	64	3x3	RELU	(None, 36, 36, 64)	36 928

Layer type	Filter nr.	Kernel size	Activ. fc.	Output shape	Param. nr.
BatchNorm1	-	-	-	(None, 36, 36, 64)	256
MaxPool	-	2x2	-	(None, 18, 18, 64)	0
Conv3	128	3x3	RELU	(None, 16, 16, 128)	73 856
BatchNorm2	-	-	-	(None, 16, 16, 128)	512
Conv4	128	3x3	RELU	(None, 14, 14, 128)	147 584
BatchNorm3	-	-	-	(None, 14, 14, 128)	512
MaxPool1	-	2x2	-	(None, 7, 7, 128)	0
Flatten	-	-	-	(None, 62 72)	0
Dense	384	-	RELU	(None, 384)	2 408 832
BatchNorm4	-	-	-	(None, 384)	1 536
Dropout	-	-	-	(None, 384)	0
Dense	-	-	Softmax	(None, 43)	16 555

To maximize likelihood, we have to minimize cross-entropy $H(p,q)$ or Kullback-Leibler (KL) divergence $D_{KL}(p||q)$. The entropy formula is given in (14), while the cross-entropy formula is given in (15), $H(p,q) \geq H(p)$, asymmetric [15], [25], [28]. KL Divergence is also known as relative entropy or information gain, given in (16), $D_{KL}(p||q) \geq 0$, asymmetric.

$$H(p) = -\sum_k p_k \ln p_k \tag{14}$$

$$H(p,q) = -\sum_k p_k \ln q_k \tag{15}$$

$$D_{KL}(p||q) = H(p,q) - H(p) \tag{16}$$

Cross-entropy loss function is given in (17) and (18), where we minimize the amount of surprise suffered between our expectation (q) and reality (p) [15]:

$$L_{xH} = -\frac{1}{m}\sum_i \sum_k p_{ik} \ln q_{ik}, \tag{17}$$

$$p_{ik} = p(Y_i = k|X_i) \tag{18}$$

Our first hyperparameter (see *Table 2*), the number of epochs, indicates how many times should the network go through a full training process. In this case, the network will go over all the 50 000 images, as well as validate itself with 12 000 test images exactly 20 times. The number of batches in epoch is the training set size over the batch size. In case of setting this batch size to a larger value, the quality of our model could deteriorate, eventually leading to a point where the model is unable to generalize well on previously unseen data [4].

The hyperparameters controlling the output volume's size are the stride, zero-padding and depth. Stride is wherewith we slide the filter, if it is equal to 1, we are moving one pixel at a time for every filter. The zero-padding has a role in controlling spatial sizes of output volumes, and the output volume depth will correspond to the number of filters we are planning to use [25]. Another aspect

worth noting is the impracticality of connecting neurons to all previous volume neurons, rather connecting each neuron to only a local region, leading to the spatial extent called the receptive field — or otherwise called the filter size, summarized for each specific layer in *Table 1*.

The learning rate, defined between 0 and 1, in this case 0.001, describes our weights' update rate. Because cycling through all of the 50 000 samples at the same time would not be computationally feasible, the batch size will express the number of image samples our neural network will cycle through at once. Our optimizer will be created as the Adam optimizer for the stochastic gradient solver, while the weight decaying parameter is also set to 0.001 — this reduces overfitting [25], [26].

Table 2
Training the network - hyperparameters

Parameter	**Value [-]**
Learning rate	0.001
Decay	0.001
Batch size (Training)	256
Batch size (Validation)	256
Epochs	2
Verbose	1

Regarding batch sizes, each batch size is composed of 256 frame inputs for the training, as well as in the case of the validation phase. Each batch trains the network in successive order, taking into account the updated weights coming from the appliance of the previous batch, each sample passed through to the network at one time. In the case of the hyperparameter batch size, we have to test and adjust it as per how our specific model performs during training. This hyperparameter also must be tested concerning how our machine is operating in respect of resource utilization [5]. Setting the verbose to 1 will mean that the progress of the model being trained will be shown during development time.

5 Validation Results and Optimization

The result validation phase during development gives us the improvement directions on how the precision accuracy could be increased. *Figure 10* presents some results during optimization with erroneous sample predictions, as well as the accuracy/loss variation of the model.

In the case of running the network without image augmentation, there are indications that the model's validation accuracy was rather high compared to the training accuracy, as *Figure 11* illustrates. At this point, we additionally defined two levels of dropout, one for convolutional layers with a rate of 0.75 and one for

fully connected layers with 0.5. The initial convolutional depth of 32 was also modified due to obtain better results with 64 [25]. Additionally, the right image in *Figure 11* presents the convolutional working process over some sample traffic signs, visually revealing the network's recognition principle.

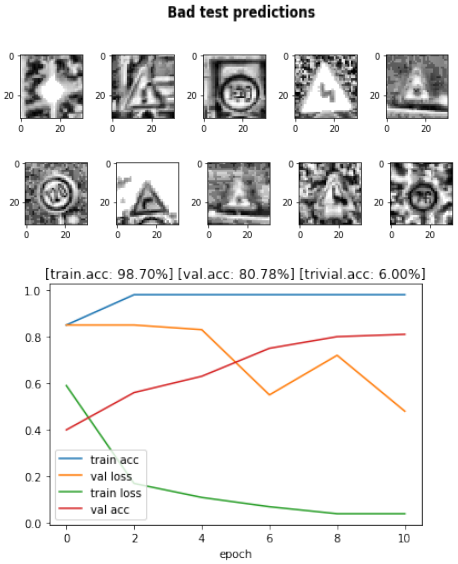

Figure 10

Samples with erroneous predictions and development accuracy/loss of the model (Own work, 2021)

The values of both loss function and precision accuracy variation for our model for 10 epochs are illustrated graphically in *Figure 12*. We can observe that the adopted CNN reduces smoothly the values of these performance metrics over the epochs, there is an apparent efficiency in the learning process, and these values tend to be flat and convergent, ultimately approaching the human-level recognition performance goal of 98.81% set in the first place, with a final recognition performance of 97.98%.

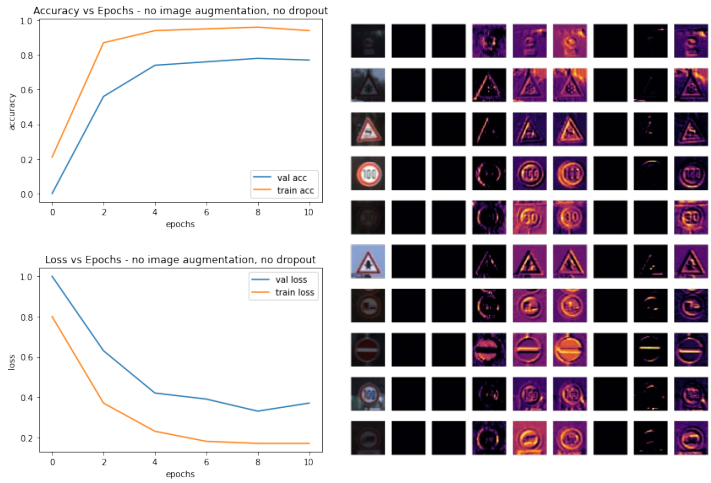

Figure 11
Accuracy and loss before optimization, and convolution visualization (Own work, 2021)

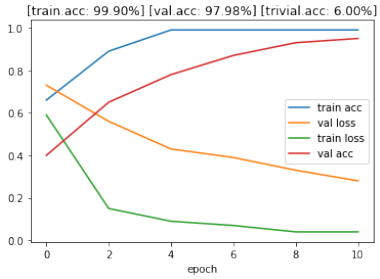

Figure 12
Final training and validation accuracy of the model (Own work, 2021)

Conclusions

In this paper the authors presented a ConvNet system and its architecture with state-of-the-art results on the GTSRB dataset, investigated a real-world traffic-sign recognition and classification problem, built a highly configurable network, as well as developed a flexible method to assess multiple architectures. The research provides evidence of practicality of the presented applications, highlighting the enhanced efficiency brought by the cognitive methods [18], [19], [27].

The validation of the network showed promising and smooth results that hold up against existing literature findings and outcomes in the field of ML-based classification problems. The network is working effectively with the preprocessed images and produces good results. The remaining errors are due to either too low-resolution inputs or physically degraded road signs for which classification is not possible with just a single image instance.

Concerning possible future development directions, one could be the problem of perception input, the traditional ConvNet architecture could be modified by feeding features to the classifier of 1st stage besides features of 2nd stage, as well as by using greyscale driving images having strong correlation features among continuous frames instead of color and by increasing the network capacity [17]. Regarding the color image samples, by visualizing the erroneous predictions we can assume that normalized color channels could be more informative than raw color.

Future studies should examine the effect of input resolution to enhance processing speed and classification accuracy, in addition to processing with multiple networks, which might further improve accuracy [16]. Finally, the influence and effect of unsupervised pre-training of feature-extracting stages should be investigated as well, which could be more easily learned than with a strictly supervised method, explicitly with an increased number of features at each stage.

Acknowledgement

Within the framework of the New Széchenyi Plan, the project "Development of talent management and researcher supply in the field of autonomous vehicle control technologies (EFOP-3.6.3-VEKOP-16-2017-00001)" provided funding for the study. The research was supported by the European Union and co-financed by the European Social Fund.

References

[1] A. Rosebrock, "Convolutions with OpenCV and Python," 2016 (Online) Available at: https://www.pyimagesearch.com/2016/07/25/convolutions-with-opencv-and-python/

[2] A. Rosebrock, "Keras Conv2D and Convolutional Layers," 2018 (Online) Available at: https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

[3] A. Shustanov and P. Yakimov, "CNN Design for Real-Time Traffic Sign Recognition," Procedia Engineering, Vol. 201, pp. 718-725, 2017, ISSN 1877-7058, DOI: https://doi.org/10.1016/j.proeng.2017.09.594

[4] A. Wong, M. J. Shafiee and M. St. Jules, "MicronNet: A Highly Compact Deep Convolutional Neural Network Architecture for Real-Time Embedded Traffic Sign Classification," IEEE Access, Vol. 6, pp. 59803-59810, 2018, DOI: https://doi.org/10.1109/ACCESS.2018.2873948

[5] Benchmark - Institut für Neuroinformatik (INI), "Dataset - German Traffic Sign Recognition Benchmark," 2021 (Online) Available at: https://benchmark.ini.rub.de/gtsrb_dataset.html

[6] L. Kovacs, L. Lindenmaier, H. Nemeth, V. Tihanyi and A. Zarandy, "Performance Evaluation of a Track to Track Sensor Fusion Algorithm," CNNA 2018: The 16th International Workshop on Cellular Nanoscale Networks and their Applications, Budapest, Hungary, 2018, pp. 1-2

[7] C. Ferencz and M. Zöldy, “End-to-end autonomous vehicle lateral control with deep learning,” 12th International Conference on Cognitive Infocommunications (CogInfoCom), IEEE Xplore, September 23-25, 2021, Budapest, ISBN 978-1-6654-2495-0

[8] J. Credi, “Traffic sign classification with deep convolutional neural networks,” Master’s thesis in Complex Adaptive Systems, Department of Applied Mechanics, Chalmers University of Technology, 2016, Gothenburg, Sweden. Available at: https://publications.lib.chalmers.se/records/fulltext/238914/238914.pdf

[9] D. Sanket, “Convolutional Neural Network: Learn And Apply,” 2019 (Online) Available at: https://medium.com/@sdoshi579/convolutional-neural-network-learn-and-apply-3dac9acfe2b6

[10] D. Sanket, “Traffic Sign Detection using Convolutional Neural Network,” 2019 (Online) Available at: https://towardsdatascience.com/traffic-sign-detection-using-convolutional-neural-network-660fb32fe90e

[11] E. Forson, “Recognising Traffic Signs With 98% Accuracy Using Deep Learning,” 2017 (Online) Available at: https://towardsdatascience.com/recognizing-traffic-signs-with-over-98-accuracy-using-deep-learning-86737aedc2ab

[12] F. Nushaine, “Building a Road Sign Classifier in Keras,” 2020 (Online) Available at: https://towardsdatascience.com/building-a-road-sign-classifier-in-keras-764df99fdd6a

[13] Google Colaboratory, “Welcome to Colaboratory,” 2021 (Online) Available at: https://research.google.com/colaboratory/ (Accessed 10 08 2021)

[14] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” Science, Vol. 313, No. 5786, pp. 504-507, 2006.

[15] I. Sergey, and C. Szegedy “Batch Normalization: Accelerating eep Network Training by Reducing Internal Covariate Shift,” Preprint, 2015, https://arxiv.org/abs/1502.03167

[16] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” Neural Networks, Vol. 32, pp. 323-332, 2012, ISSN 0893-6080, DOI: https://doi.org/10.1016/j.neunet.2012.02.016

[17] Keras - Simple. Flexible. Powerful, “API docs,” 2021 (Online) Available at: https://keras.io/ (Accessed: 01 08 2021)

[18] M. Zöldy and P. Baranyi, “Cognitive Mobility – CogMob,” 12th IEEE International Conference on Cognitive Infocommunications (CogInfoCom) pp. 915-919, 2021

[19] M. Zöldy and P. Baranyi, “The Cognitive Mobility Concept,” Infocommunications Journal, Vol. 2023/01, pp. 35-40, 2022

[20] Netron App, "Lutz Roeder's Netron," 2021 (Online) Available at: https://netron.app (Accessed: 05 08 2021)

[21] NN-SVG, "Publication-ready NN-architecture schematics," 2021 (Online) Available at: https://alexlenail.me/NN-SVG/LeNet.html (Accessed: 05 08 2021)

[22] P. Baranyi and Y. Yam, "Singular value-based approximation with Takagi-Sugeno type fuzzy rule base," Proceedings of 6th International Fuzzy Systems Conference, Vol. 1, pp. 265-270, 1997, DOI: https://doi.org/10.1109/FUZZY.1997.616379

[23] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale Convolutional Networks," The 2011 International Joint Conference on Neural Networks (IJCNN), 2011, pp. 2809-2813, DOI: https://doi.org/10.1109/IJCNN.2011.6033589

[24] R. Uppala, "Traffic Signs Classification with a Convolutional Neural Network," 2017 (Online) Available at: https://medium.com/@techreigns/traffic-signs-classification-with-a-convolutional-neural-network-75911a1904

[25] Stanford University CS231n, "Convolutional Neural Networks for Visual Recognition," 2021 (Online) Available at: https://cs231n.github.io/convolutional-networks/

[26] S. Saha, S. Amit Kamran and A. Shihab Sabbir, "Total Recall: Understanding Traffic Signs Using Deep Convolutional Neural Network," 2018 21st International Conference of Computer and Information Technology (ICCIT), 2018, pp. 1-6, DOI: https://doi.org/10.1109/ICCITECHN.2018.8631925

[27] T. Péter, A. Hary, F. Szauter, K. Szabó, T. Vadvári, I. Lakatos, "Analysis of Network Traversal and Qualification of the Testing Values of Trajectories," Acta Polytechnica Hungarica, Vol. 18, pp. 151-171, 2021, https://doi.org/10.12700/APH.18.10.2021.10.8

[28] Wikipedia - The Free Encyclopedia, "Kullback–Leibler divergence," 2021 (Online) Available at: https://en.wikipedia.org/wiki/Kullback-Leibler_divergence = shaded area

[29] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, S. Hu, "Traffic-Sign Detection and Classification in the Wild," Proceedings of CVPR, 2016, pp. 2110-2118