Convolutional Neural Networks for Croatian Traffic Signs Recognition

Vedran Vukotić, Josip Krapac and Siniša Šegvić
University of Zagreb - Faculty of Electrical Engineering and Computing
Zagreb, HR-10000, Croatia
Email: vevukotic@gmail.com

***Abstract*—We present an approach to recognition of Croatian traffic signs based on convolutional neural networks (CNNs). A library for quick prototyping of CNNs, with an educational scope, is first developed[1]. An architecture similar to LeNet-5 is then created and tested on the MNIST dataset of handwritten digits where comparable results were obtained. We analyze the FER-MASTIF TS2010 dataset and propose a CNN architecture for traffic sign recognition. The presented experiments confirm the feasibility of CNNs for the defined task and suggest improvements to be made in order to improve recognition of Croatian traffic signs.**

I. Introduction

Traffic sign recognition is an example of the multiple classes recognition problem. Classical approaches to this problem in computer vision typically use the following well-known pipeline: (1) local feature extraction (*e.g.* SIFT), (2) feature coding and aggregation (*e.g.* BOW) and (3) learning a classifier to recognize the visual categories using the chosen representation (*e.g.* SVM). The downsides of these approaches include the suboptimality of the chosen features and the need for hand-designing them.

CNNs approach this problem by learning meaningful representations directly from the data, so the learned representations are optimal for the specific classification problem, thus eliminating the need for hand-designed image features. A CNN architecture called *LeNet-5* [1] was successfully trained for handwritten digits recognition and tested on the MNIST dataset [2] yielding state-of-art results at the time. An improved and larger CNN was later developed [3] and current state-of-the-art results on the GTSRB dataset [4] were obtained.

Following the results by [3], we were motivated to evaluate a similar architecture on the Croatian traffic signs dataset FER-MASTIF TS2010 [5]. To do so, we first developed a library that would allow us to test different architectures easily. After different subsets were tested for successful convergence, an architecture similar to *LeNet-5* was built and tested on the MNIST dataset, yielding satisfactory results. Following the successful reproduction of a handwritten digit classifier (an error rate between 1.7% and 0.8%, where LeNet-X architectures yield their results), we started testing architectures for a subset of classes of the FER-MASTIF TS2010 dataset.

In the first part of this article, CNNs are introduced and their specifics, compared to classical neural networks, are presented. Ways and tricks for training them are briefly explained. In the second part the datasets are described and the choice of a subset of classes for the FER-MASTIF TS2010 dataset is elaborated. In the last part of the paper, the experimental setup is explained and the results are discussed. Finally, common problems are shown and suggestions for future improvements are given.

[1] Available at https://github.com/v-v/CNN/

II. Architectural specifics of CNNs

Convolutional neural networks represent a specialization of generic neural networks, where the individual neurons form a mathematical approximation of the biological visual receptive field [6]. Visual receptive fields correspond to small regions of the input that are processed by the same unit. The receptive fields of the neighboring neurons overlap, allowing thus robustness of the learned representation to small translations of the input. Each receptive field learns to react to a specific feature (automatically learned as a kernel). By combining many layers, the network forms a classifier that is able to automatically learn relevant features and is less prone to translational variance in data. In this section, the specific layers (convolutional and pooling layers) of CNNs will be explained. A CNN is finally built by combining many convolutional and pooling layers, so the number of output in each successive layer grows, while size of images on the output is reducing. The output of the last CNN layer is a vector image representation. This image representation is then classified using a classical fully-connected MLP [3], or another classifier, *e.g.* an RBF network [1]).

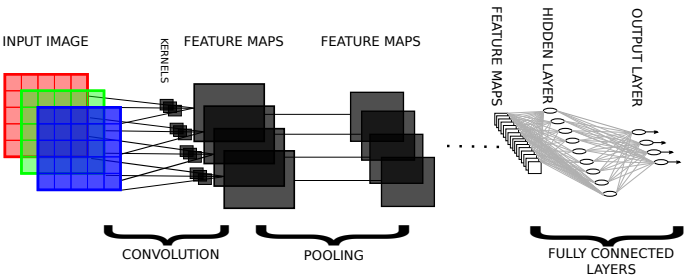

Fig. 1: Illustration of the typical architecture and the different layers used in CNNs. Many convolutional and pooling layers are stacked. The final layers consist of a fully connected network.

A. Feature maps

Fig. 2 shows a typical neuron (a) and a feature map (b). Neurons typically output a scalar, while feature maps represent

the two-dimensional output of an operation performed by a CNN unit. Typical operations performed in a CNN are convolutions and pooling operations. The former learns a feature (convolution kernel), while the latter only reduces the dimensionality by aggregation. These operations will be discussed in the following subsections.

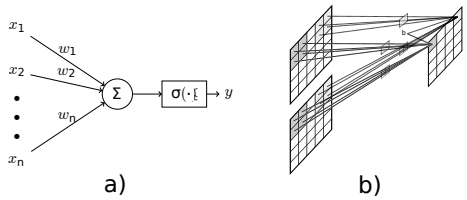

Fig. 2: CNN elements: a) a classical neuron and its connectivity b) a feature map in a convolutional operation (the two output pixels are computed with the same kernels)

B. Convolution

Convolutional layers compute feature maps by convolving the previous layer with a specific kernel. Let M^l be a feature map of layer l and M^{l-1} a feature map of the previous layer. The width and height of a feature map is indicated with M_w and M_h while the width and hight of kernels are indicated with K_w and K_h. Let S_w and S_h represent the horizontal and vertical steps of the kernel during a convolution operation. The sizes of the output feature maps (of the current layer) are then dependent on the sizes of feature maps from the previous layer, kernels and stepping factors. The output width and height are given by Eq (1) and Eq. (2), correspondingly.

$$M_w^l = \frac{M_w^{l-1} - K_w}{S_w} + 1 \tag{1}$$

$$M_h^l = \frac{M_h^{l-1} - K_h}{S_h} + 1 \tag{2}$$

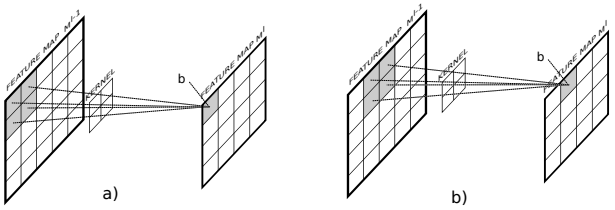

Fig. 3: Illustrated convolution a) first step b) next step, after moving the receptive field for S_w locations

After each convolution, a bias is added to each element and the result is passed through an activation function (see II-D).

Convolutional layers can either have full connectivity or sparse connectivity. In case of full connectivity, each feature map of the current layer is connected with every feature map from the previous layer. Each connection is represented by a kernel, so a convolutional layer that is fully connected will have $|M^l| \cdot |M^{l-1}|$ kernels. Fully connected convolutional layers are used by most authors, *e.g.* [7] and [3].

Sparse connectivity is a way of connecting feature maps in convolution operations where each feature map from the current layer is connected only to a specific subset of feature maps from the previous layer. The benefits of this approach are reduced computational complexity and improved generalization, as the network is forced to learn different features. When using fully connected convolutional layers there is a chance that the network will learn a less diverse set of features [1], [8].

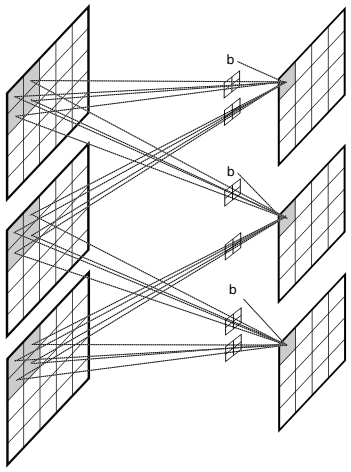

Fig. 4: An example of sparse connectivity of a convolutional layer. Each feature map is connected to only a subset of feature maps from the previous layer

C. Pooling

Pooling layers reduce dimensionality of feature maps from the previous layer by aggregating and representing the grouped features by one feature. An illustration of a generic pooling operation is shown in Fig. 5 b) where the features of the previous layer are grouped in 2×2 areas and are represented in the current map with one element. There are many different pooling operations but the most common ones are mean-pooling and max-pooling. The former represents the group with the average value of all the features withing the group, while the latter represents the group with the maximum element found within the group. Mean pooling was used in earlier works [1], but in recent works max pooling is mostly used, as it always outperforms mean pooling [9] and is additionally faster than mean pooling.

There are a few modern parametric pooling operations that can outperform max-pooling in terms of the quality of representation [10], [11]. However they are more computational expensive, require fine-tuning of additional hyper-parameters and were thus not used in this work.

D. Activation functions

An activation function is a sigmoid-shaped function that maps an input to its output, constrained to a defined range. Just as in classical multilayer perceptrons, they are also used in convolutional neural networks where they are applied to each element of a feature map. To be able to use the error backpropagation algorithm for training of CNN the activation function should be derivable. The two most commonly used activation functions [8], [12] are the logistic function, defined

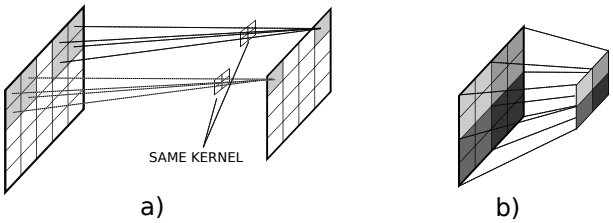

Fig. 5: a) Weight sharing in convolutional layers, the same kernel is used for all the elements within a feature map b) a generic pooling operation

in Eq. (3) and a scaled tanh sigmoidal function, defined in Eq. (4)

$$f(x) = \frac{2}{1 + e^{-\beta x}} - 1 \tag{3}$$

$$f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right) \tag{4}$$

Weights and kernel elements are randomly initialized by using a uniform distribution [13]. However, the sampling interval depends on the choice of the activation function. For the logistic function, the interval is given in (5), while for the scaled tanh sigmoidal function the interval is given in (6). In those equations n_{in} indicates the number of neurons (or feature map elements) in the previous layer, while n_{out} indicates the number of neurons (or feature map elements) in the current layer. Improper initialization may lead to poor convergence of the network.

$$\left[\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right] \tag{5}$$

$$\left[-4\sqrt{\frac{6}{n_{in} + n_{out}}}, 4\sqrt{\frac{6}{n_{in} + n_{out}}}\right] \tag{6}$$

III. Training CNNs

In supervised training, convolutional neural networks are typically trained by the Backpropagation algorithm. The algorithm is the same as for multilayer perceptrons but is extended for convolutional and pooling operations.

A. Backpropagation for convolutional layers

For convolutional layers, the backpropagation algorithm is the same as for multilayer perceptrons (if every element of a feature map is treated as a neuron and every element of a kernel is treated as a weight) with the only exception that weights are shared inside a feature map. Fig. 5 a) illustrates weight sharing in a convolution between two feature maps. The weight sharing easily fits into backpropagation algorithm: because multiple elements of the feature map contribute to the error, the gradients from all these elements contribute to the same set of weights.

B. Backpropagation for pooling layers

Pooling layers require a way to reverse the pooling operation and backpropagate the errors from the current feature map to the previous one. For the case of max-pooling, the error propagates to the location where the maximal feature is located while the other locations receives an error of zero. For mean-pooling, the error is equally distributed within the locations that were grouped together in the forward pass and can be expressed as $\mathbf{E}' = \mathbf{E} \otimes \mathbf{1}$, where $\mathbf{E}'$ is the error of the previous layer, $\mathbf{E}$ the error of the current layer and $\otimes$ represents the Kronecker product.

IV. Improving learning

A. Backpropagation with momentum

The classical Backpropagation algorithm uses a global learning rate η to scale the weight updates. This modified version scales the learning rate dynamically depending on the partial derivative in the previous update. The method is defined in the Eq. (7), where α represents the amortization factor (typical values between 0.9 and 0.99).

$$\Delta w(t) = \alpha \Delta w(t-1) - \eta \frac{\partial E}{\partial w}(t) \tag{7}$$

B. Adding random transformations

Generalization can be improved by increasing the number of training samples. However that usually requires additional human effort for collection and labelling. Adding random transformations can increase the generalization capability of a classifier without additional effort. There are two main ways for deploying random transformations into a system: (1) by integrating them into the network, after the input layers [3] or (2) by generating additional samples and adding them to the training set [14]. In this work we opted for generating additional samples since the code is currently not optimized for speed, and adding an additional task to be performed for each iteration would further slow the learning process.

C. Dropout

A typical approach in machine learning when improving generalization consists of combining different architectures. However, that can be computationally quite expensive. The dropout method suggests randomly disabling some hidden units during training, thus generating a large set of combined virtual classifiers without the computational overhead [15]. For a simple multilayer perceptron with N neurons in one hidden layer, 2^N virtual architectures would be generated when applying dropout. Usually, half the units are disabled in each learning iteration and that is exactly what we used in this work.

V. Datasets

Two datasets were used in this work. The MNIST dataset of handwritten digits [2] and the FER-MASTIF TS2010 dataset of Croatian traffic signs [5] [16].

A. *MNIST*

The MNIST dataset consists of 10 classes (digits from 0 to 9) of 28×28 grayscale images. It is divided into a training set of 60000 samples and a testing set of 10000 samples. The dataset was unaltered except for preprocessing it to a mean of zero and unit variance.

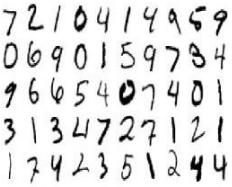

Fig. 6: Samples from the MNIST dataset

B. *FER-MASTIF TS2010*

This dataset consists of 3889 images of 87 different traffic signs and was collected with a vehicle mounted video camera as a part of the MASTIF (Mapping and Assessing the State of Traffic InFrastructure) project. In [17], images were selected by the frame number and split in two different sets, a training set and a test set. This method ensured that images of the same traffic sign do not occur in both sets. The sizes varies from 15 pixels to 124 pixels. We opted for classes containing at least 20 samples of sizes greater or equal to 44×44. Fig. 8 shows the nine classes that met that criteria.

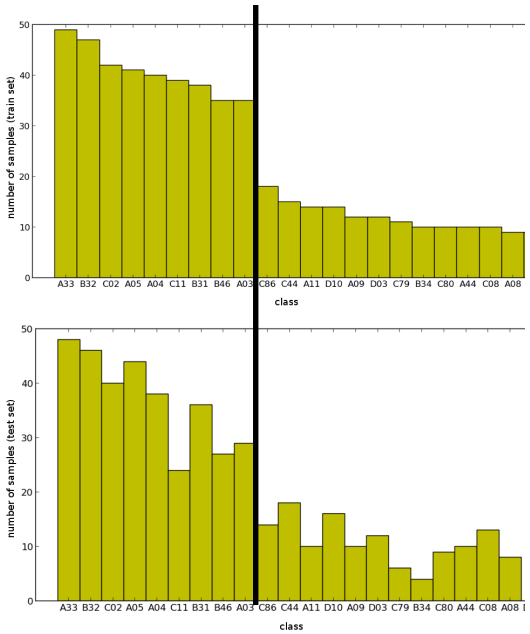

Fig. 7: Part of the histogram showing the number of samples bigger or equal to 44×44 for two sets. The black line separates the classes that we chose (more than 20 samples) for our experiments.

Each sample was first padded with 2 pixels on each side (thus expanding their size to 48×48) to allow the convolutional layers to relevant features close to the border. The selected subset of samples was then expanded by applying random transformations until 500 samples per class were obtained. The random transformations applied are: (1) rotation sampled from $\mathcal{N}(0, 5°)$ and (2) translation of each border (that serves the purpose of both scaling and translation), sampled from $\mathcal{N}(0, 1px)$. In the end, every sample was preprocessed so that it has a mean of zero and unit variance.

Fig. 8: The selected 9 classes of traffic signs.

VI. Experiments and Results

Different architectures were built and evaluated for each dataset. In the following section the architectural specifics of convolutional neural networks for each experiment are defined, their performance is illustrated and results are discussed.

A. *MNIST*

The following architecture was built:

- **input layer** - 1 feature map 32×32
- **first convolutional layer** - 6 feature maps 28×28
- **first pooling layer** - 6 feature maps 14×14
- **second convolutional layer** - 16 feature maps 10×10
- **second pooling layer** - 16 feature maps 5×5
- **third convolutional layer** - 100 feature maps 1×1
- **hidden layer** - 80 neurons
- **output layer** - 10 neurons

The network was trained by stochastic gradient descent with 10 epochs of 60000 iterations each. No dropout and no random transformations were used. Such network yielded a result of 98.67% precision on the MNIST test set. Table I shows the obtained confusion matrix. The three most common mistakes are as shown in Fig. 9. It can be noticed that samples that were mistaken share a certain similarity with the wrongly predicted class.

		Predicted class									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual class	**0**	973	0	1	0	0	0	3	1	2	0
	1	0	1127	4	1	0	0	1	0	2	0
	2	3	0	1020	0	0	0	0	4	5	0
	3	0	0	2	992	0	6	0	3	5	2
	4	1	0	1	0	963	0	4	0	2	11
	5	1	0	0	3	0	884	1	1	0	2
	6	10	2	0	0	1	2	943	0	0	0
	7	0	1	7	2	0	0	0	1014	1	3
	8	2	0	1	0	1	1	1	3	962	3
	9	3	2	0	3	1	3	1	4	3	989

TABLE I: Confusion matrix for the MNIST dataset

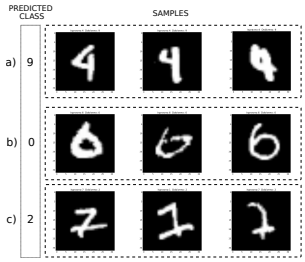

Fig. 9: The three most common errors on the MNIST dataset: a) number 4 classified as 9, b) number 6 classified as 0, c) number 7 classified as 2

B. FER-MASTIF TS2010

The following architecture was built:

- **input layer** - 3 feature maps 48×48
- **first convolutional layer** - 10 feature maps 42×42
- **first pooling layer** - 10 feature maps 21×21
- **second convolutional layer** - 15 feature maps 18×18
- **second pooling layer** - 15 feature maps 9×9
- **third convolutional layer** - 20 feature maps 6×6
- **third pooling layer** - 10 feature maps 3×3
- **fourth convolutional layer** - 40 feature maps 1×1
- **hidden layer** - 80 neurons
- **output layer** - 9 neurons

The network was trained with 20 epochs of 54000 iterations each by stochastic gradient descent. Random transformations were used (as defined in Section V) while dropout was not used. The trained network yielded a result of 98.22% on the test set. Table II shows the full confusion matrix on the test set and Fig. 10 shows the three most common errors made by the network.

		Predicted class								
		C02	A04	B32	A33	C11	B32	A05	B46	A03
Actual class	C02	100	0	0	0	0	0	0	0	0
	A04	0	99	0	0	0	0	1	0	0
	B32	0	0	97	3	0	0	0	0	0
	A33	0	0	0	100	0	0	0	0	0
	C11	0	0	0	0	100	0	0	0	0
	B31	0	0	0	0	0	100	0	0	0
	A05	0	0	0	0	0	0	98	0	2
	B46	0	0	0	0	0	0	0	100	0
	A03	0	3	0	0	0	0	7	0	90

TABLE II: Confusion matrix obtained on the FER-MASTIF TS2010 dataset

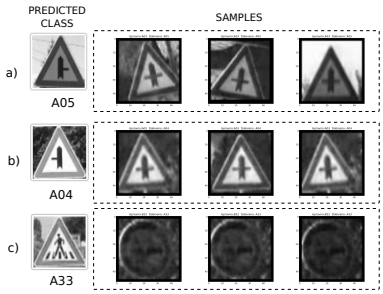

Fig. 10: The three most common errors on the FER-MASTIF TS2010 dataset a) A03 predicted as A05, b) A03 predicted as A04, c) B32 predicted as A33

C. FER-MASTIF TS2010 with smaller samples included

Fig. 11 shows the distribution of sample sizes (of the chosen 9 classes) in the FER-MASTIF TS2010 set. To determine the influence of sample size to the classification error, we scaled all samples to the same size (44×44 with padding leading to an input image of 48×48), matching the input of the network. In the previous experiment, samples smaller than 44×44 that were not used. However, in the following experiment they were included in the test set by upscaling to the required size.

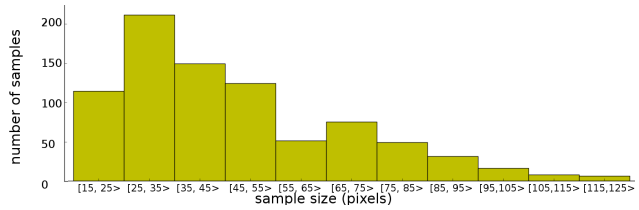

Fig. 11: Number of samples of different sizes (within the 9 chosen classes)

Fig. 12 shows the percentage of misclassified samples for different sizes. It can be seen that samples bigger than 45×45 produce significantly lower error rates than smaller samples, thus confirming our choice of the input dimensionality to the CNN.

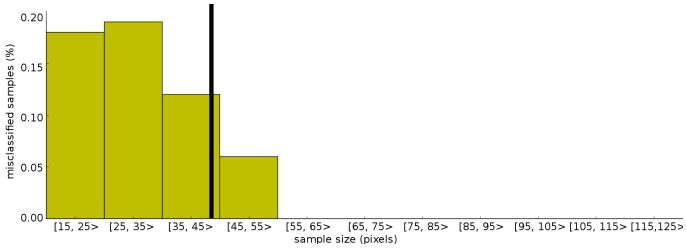

Fig. 12: Dependency of the error rate on the sample size. (The vertical line denotes the limit of 44px from where samples for the experiments in subsections A and B were taken.)

D. FER-MASTIF TS2010 with dropout

The same architecture was trained again with dropout in the final layers. We used the previously defined 9 classes and, again, only samples larger than 44×44. Using dropout improved the network generalization capability that yielded a result of 99.33% precision.

E. Comparison of learned kernels

Fig. 13 shows a few learned convolutional kernels from the first and second convolutional layers on the two different datasets. It is clearly visible that the network adapted to the dataset and learned different distinct features for each dataset.

VII. Common problems in training CNNs

CNNs are difficult to train because they have many hyper-parameters and learning procedure parameters, which need to be set correctly in order for learning to converge. In this section, we discuss a few issues that we had during the development of our library and the two networks.

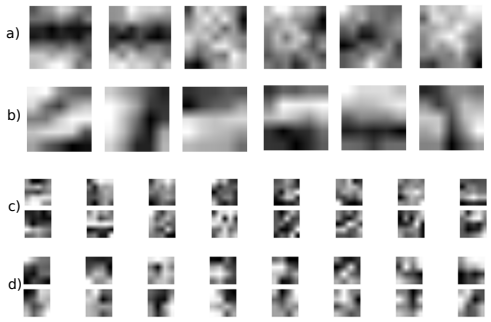

Fig. 13: Examples of learned kernels: a) first convolutional layer on MNIST b) first convolutional layer on FER-MASTIF TS2010 c) second convolutional layer on MNIST d) second convolutional layer on FER-MASTIF TS2010

Choosing architectural parameters like the number of layers and sizes of kernels and feature maps within each layer depend on the dataset. It makes sense to first choose the size of feature maps and kernels in the first layer according to the scale of the features in the dataset. Networks for dataset containing features of larger scale (like the FER-MASTIF TS2010 dataset) should use larger kernels and feature maps than networks that aim at classifying dataset with features of smaller scale (like the MNIST dataset). The same is valid for every other layer.

The **choice of an activation function** and number of neurons in each layer should match the **intervals for random initialization** for the weights and convolutional kernels. Also, it is suggested to preprocess the data to **zero mean and unit variance**.

CNNs are typically more difficult to train than MLPs, so when developing a new library it is recommended to **check the gradients** and **test for convergence** in all CNN layers. Like in classical NNs, whether a network will converge depends strongly on the **choice of the learning rate**. It is suggested to experiment with learning rates of different scales (*e.g.* 0.001, 0.01, 0.1, etc.) and to implement an adaptive algorithm that decreases the learning rate over time.

To improve the generalization capabilities of a CNN, it is suggested to use **dropout** or **sparse connectivity** between layers (dropout is a preferred method in modern state-of-the-art methods [3]) and including random transformations.

VIII. Conclusion and future work

We developed a library that enabled us to easily prototype and test different architectures of convolutional neural networks. After successfully testing the convergence of different elements of the library, we built a network similar to *LeNet-5* [1] and tested it on the MNIST dataset of handwritten digits where we obtained comparable results.

In the second part of this work, we analyzed and prepared the FER-MASTIF TS2010 dataset and built a convolutional neural network for recognizing a subset of 9 classes. Although limited to most numerous classes, the results were satisfactory.

Our library was developed having simplicity and clarity in mind, for educational purposes. It was not optimized for speed. Each training session lasted about a week. In case of building a bigger classifier for more classes of the FER-MASTIF TS2010 we suggest improving the speed of the convolutional operations and implementing mini-batch learning.

Regarding the FER-MASTIF TS2010 dataset, we suggest gathering more data. More data is suggested even for the 9 selected classes that had enough samples and yielded satisfactory results when used with random transformations, but especially for the remaining classes that didn't have enough samples to use (even when using random transformations to generate more samples).

References

[1] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, 1995.

[2] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits."

[3] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural Networks*, 2012.

[4] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*.

[5] S. Šegvić, K. Brkić, Z. Kalafatić, and A. Pinz, "Exploiting temporal and spatial constraints in traffic sign detection from a moving vehicle," *Machine Vision and Applications*.

[6] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda, "Subject independent facial expression recognition with robust face detection using a convolutional neural network," *Neural Networks*, 2003.

[7] P. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, 2003.

[8] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[9] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *ICANN 2010*. Springer, 2010.

[10] P. Sermanet, S. Chintala, and Y. LeCun, "Convolutional neural networks applied to house numbers digit classification," in *ICPR*. IEEE, 2012, pp. 3288–3291.

[11] Y. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in vision algorithms," in *ICML*, 2010.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[13] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.

[14] I. Bonači, I. Kusalić, I. Kovaček, Z. Kalafatić, and S. Šegvić, "Addressing false alarms and localization inaccuracy in traffic sign detection and recognition," 2011.

[15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[16] S. Šegvić, K. Brkić, Z. Kalafatić, V. Stanisavljević, M. Ševrović, D. Budimir, and I. Dadić, "A computer vision assisted geoinformation inventory for traffic infrastructure," *ITSC*, 2010.

[17] J. Krapac and S. Šegvić, "Weakly supervised object localization with large fisher vectors."

This research has been supported by the research project: Research Centre for Advanced Cooperative Systems (*EU FP7 #285939*).