Automatic Traffic Sign Recognition

Artificial Inteligence - Deep Learning Algorithm

Mihai Daniel RADU
University Politehnica of Bucharest, Transports Faculty, Electronics Department, Bucharest, ROMANIA
mikeradu96@yahoo.ro

Ilona Madalina COSTEA
University Politehnica of Bucharest, Transports Faculty, Electronics Department, Bucharest, ROMANIA
ilona.costea@upb.ro

Valentin Alexandru STAN
University Politehnica of Bucharest, Transports Faculty, Electronics Department, Bucharest, ROMANIA
valentin.stan@upb.ro

***Abstract* - Due to the large number of deaths and car accidents caused by the driver's lack of attention, car manufacturers are trying to integrate ADAS systems with artificial intelligence and CV.**

One function that helps the driver is traffic sign recognition (TSR). This is a technology with which a vehicle is able to recognize road signs placed on the road, e.g. "speed limit" or "give way" or "stop" all this being possible with the help of computer vision and Convolutional Neural Networks. In this article we propose an implementation based on LeNet architecture for traffic sign recognition using CNN.

We preferred the deep learning approach for this challenge as methods like shape based or color based share a common weakness in factors as light changes, scale change, rotation.

The paper presents implementation of the network architecture based on LeNet5 using Keras and TensorFlow library, how we trained the CNN using the GTSRB dataset, and what results we obtained training and using the network for real time applications using only CPU power, with Intel i7 7700K. The experimental results showed a 95% accuracy in recognizing traffic signs.

***Keywords* – Artificial Intelligence, AI, Convolutional neural network, deep learning, LeNet architecture, traffic signs recognition**

I. Introduction

Traffic signs represent an important part of our road infrastructure because it provides important information for road users, which in turn requires them to adjust their driving behavior to make sure they comply with road regulation enforced on the road sector.

Usually, traditionally computer vision methods are used to detect and classify traffic signs, but these required considerable manual work to handcraft important features in images which is very time-consuming. Also methods like shape or color based can be used but they share a common weakness in several factors such as lighting changes, occlusions, scale change, rotation, and translations. However, these problems could also be treated using machine learning, but it requires a large database of annotated data[6]. The other more efficient method is deep learning which has acquired general interest in recent years due to its high performance of classification and the power of representational learning from raw data. However, there are some challenges in order for a TSR to be successful, as performance of the system is affected by the surrounding environment or road sign visibility. The main problems remain light conditions, bad weather, vandalism or bad postage of signs.

Figure 1. Vandalizes Signs

This paper provides a comprehensive guide of model arhitecture, images and distribution, pre-processing and the results obtained using an HD camera and CPU power.

II. Data set

The dataset from GTSRB (German Traffic Sign Recognition Benchmark) was used for training and testing the developed network.

The images in the dataset have a dimension of 32 pixels (width) x 32 pixels (height) x 3 (RGB color channels), and it contains 34799 images. The data is split in 43 different classes (e.g. Speed Limit 20km/h, stop, give way, etc.), each class corresponding to a traffic sign. For training we used 22271 images, for validation we used 5568 images and for testing we used 6960 images.

Figure 2. Dataset examples

The dataset is also imbalanced across classes, as some classes have less than 200 images and others have over 1200.

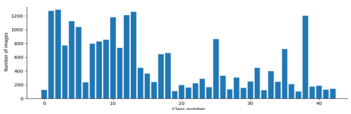


Figure 3. Data Distribution

III. Preprocessing Images

The preprocessign step is very important because the images in the dataset have 3 channels (RGB) and undistributed light and in order to classify the image we need grayscale images with standard lighting. The first step is converting the image from the dataset from 3 channels (RGB) to a single channel, grayscale . The conversion from RGB to grayscale also helps reducing computing, also an advantage would be that we have reduced the training time a lot by only processing grayscale images as they have 3 times less data than RGB images.

The next step is to apply histogram equalization, which is a computer processing technique used to improve contrast in images which effectively spreads out the most frequent intensity values, so it stretches out the intensity range of the image. This method helps increasing the global contrast of images when its usable data is represented by close contrast values, so areas of lower local contrast gain a higher contrast.

The last step of the preprocessing is normalizing the images so that each pixel has a value between 0 and 1 instead of 0 and 255.

The end result after preprocess is this:

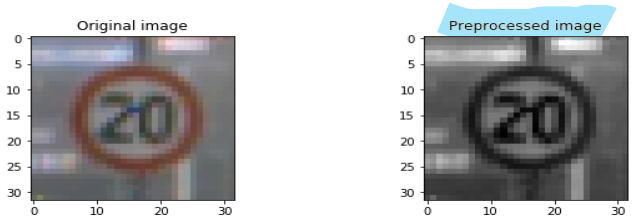


Figure 4. Original vs Preprocessed Images

As presented in the data distribution chart, there is imbalance across the 43 classes, but this is not a problem as we are able to reach very high accuracy despite the class imbalance. This is why se use data augmentation techniques in an attempt to:

- Enlarge the dataset and provide additional pictures in different lighting settings and orientations
- Make the model more generic
- Improve test and validation accuracy, especially on distorted images

Data augmentation is a procedure where images become transformed versions of the same images in the training dataset that belongs to the same class.

The operations we used to manipulate images were shifts, zooms, and rotation. The reason we use data augmentation is because on the streets, traffic signs may be slightly rotated, or it may be inclined, so this way we can still recognize them. Below is an example of traffic signs after augmentation.

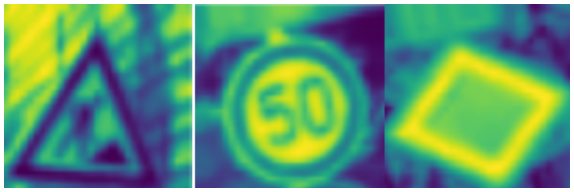


Figure 5. Preprocessed Images

IV. Model Arhitecture

A Convolutional Neural Network (CNN) architecture can be split in three main parts:

- A convolutional layer which extracts features from an input image also it helps with blurring, sharpening, edge detection, noise reduction, or other operations that help the algorithm to learn specific characteristics of an image.
- A pooling layer which helps reducing the image dimension without losing important features or patterns.
- A fully connected layer also known as the dense layer, in which the results of the convolution process are fed through one or more neural layers to generate a prediction.

The model used is a deep neural network model, more exactly a convolutional neural network(CNN), the reason we preferred a CNN is that they have very good results in image recognition, and also we don`t have to worry about filters and weights initialization as they are randomly initialized using gaussian distribution and then fine-tuned using gradient descend. The main advantage of a CNN is the convolution operation. The model scans the input image many times to find certain features. This scanning which is the convolution operation can be set with 2 main parameters: stride and padding type. After the first convolution step we end up with a set of new frames, which can be found in the second layer. Each frame contains an information about one feature and its presence in the input image. The resulting frame will have larger values in places where a feature is strongly visible and lower values where the features are low or there are not any.

The model architecture is inspired from Yann Le Cun paper on classification of traffic signs[1].With the help of Keras we managed to build our model sequentially.

The model we used is based on LeNet architecture, the input shape is 32x32x1. First convolution layer will have a depth of 60, a filter size of (5, 5), and a stride of (1, 1) and it is composed of two identical convolution operators. The main advantage when using a second convolution layer is that it has more flexibility in expressing non-linear transformations without losing information. MaxPool removes information from the signal, dropout forces distributed representation, thus both effectively make it harder to propagate information.

A convolution sweeps the window through images then calculates its input and filter dot product pixel values. This process emphasize the relevant features of the image.

With this process, we detect a particular feature from the input image and produce feature maps, resulted from the convolution process which emphasizes the important features. These resulted features will always change depending on the filter values affected by the gradient descent to minimize prediction loss.

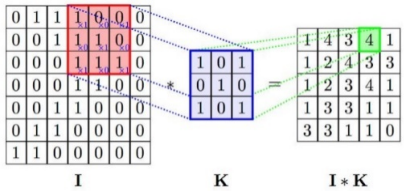
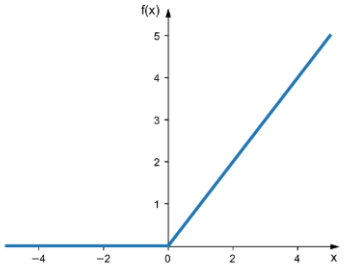


Figure 6. Example of convolution data-flow

For traffic sign recognition, highly non-linear transformation has to be applied on raw data, so stacking multiple convolutions (with ReLU) will make it easier to learn.

As presented in [2], we observe that if we remove the nonlinear operation, the model performance decreases.

Rectified linear unit also known as ReLU and it receives a real number as a parameter and returns the received number if it is greater than 0 otherwise it returns 0.



$$f(x) = \max(0, x)$$

Figure 7. – ReLU function graph

The next layer is called max polling layer and effectively downsizes the data by only selecting the max value pixel for adjacent pixels. LeNet uses a (2, 2) filter size.

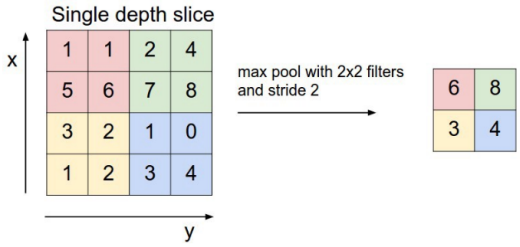


Figure 8. – The MaxPool operation

Next LeNet has a second convolution layer with depth of 30, filter size of (3, 3) and ReLU as an activation function, together with a max-pool layer.

After the second stack of convolutions we use dropout, which is a form of regularization where weights are kept with a probability p; the unkept weights are thus "dropped". This prevents the model from overfitting.

The data resulted from the convolution layers is flattened before the fully connected layers. The flattening process takes a two-dimensional matrix of features and transforms it into a single vector that can be fed to the fully connected neural network classifier.

The last part of the model consists of two fully connected layers. The first one uses ReLU as activation function than dropout layer is added to reduce overfitting. The second layer is represented by a fully connected layer with size of 43 representing the number of classes.

The output of convolution and pooling layers is flattened into a single vector of values, each value representing a probability that a certain feature belongs to a traffic sign class.

Input values go into the first layer of neurons where are multiplied by weights and pass through a ReLU activation function.

In the fully connected layer an input image goes through the network and it returns as a stack of matrixes of features which are compressed into a single vector. When the input image is a traffic sign, certain values in the vector tend to be higher, so the network knows to categories the sign to the corresponding class.

We used SoftMax function as the final activation function in the network as this function normalizes an input vector into a range that often leads to a probabilistic interpretation.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 60)	1560
conv2d_2 (Conv2D)	(None, 24, 24, 60)	90060
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 60)	0
conv2d_3 (Conv2D)	(None, 10, 10, 30)	16230
conv2d_4 (Conv2D)	(None, 8, 8, 30)	8130
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 30)	0
dropout_1 (Dropout)	(None, 4, 4, 30)	0
flatten_1 (Flatten)	(None, 480)	0
dense_1 (Dense)	(None, 500)	240500
dropout_2 (Dropout)	(None, 500)	0
dense_2 (Dense)	(None, 43)	21543

Total params: 378,023
Trainable params: 378,023

Figure 9. Model Summary

The fully connected part of the neural network goes through its own backpropagation process in order to determine the most accurate weights. Each neuron receives weights that prioritize the most appropriate traffic sign class so that neurons decide which class has the highest probability and that is the network answer for the input image.

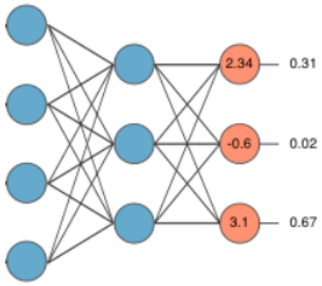


Figure 10. Example of Flatten- Fully Connected Layer

V. Model Training

In the training step an input is fed to the network and all the layers elements effectively constitute a transformation of this input to a predicted output. In the process of training all filters and weights are randomly assigned using gaussian distribution. The variation between this predicted output and the actual output is defined as loss. The loss value is then passed backwards through filters and used to adjust the values of the filters and neuron weights. This process is called backpropagation and it what it does is to minimize the difference between predicted and actual output. This way the value of the filters are constantly adjusted during training and system is considered trained when the loss is minimized.

The training process can be divided into four steps:

- Step 1: Initializing filters and weights with random values

- Step 2: The network takes the training set as input, goes through the forward propagation step, which means going through convolution layers, ReLU and pooling operations along with forward propagation in the Fully Connected layer and finds the output probabilities for each class.
- Step 3: We compute the total error at the output layer. The formula for total error is:

$$Total\ Error = \sum \tfrac{1}{2}\ (target\ probability - output\ probability)^2$$

- Step 4: We compute the gradients of the error with respect to all weights in the network using backpropagation and use gradient descent to update all filter values and weights to minimize the global error.

These steps are mandatory for each image, and the data set must be fed through the network multiple times in order to have good results. We passed through the network the dataset 30 times in order to obtain 97% accuracy.

The main advantage when using Backpropagation is that the error in the final answer is used to adjust how much the network changes its parameters.

For example, if we fed the network an image representing the „50 km/h speed limit sign" the network will say it is a probability of 0.92 that the sign is part of the no.2 class, and a 0.51 probability that the sign is part of the no.4 class which represents „70 km/h speed limit".

Class no.	Right answer	Actual answer	Error
2	1	0.92	0.08
4	0	0.51	0.49
		Total	0.57

So, in order to correct this error we use gradient descend. Each weight value is increased or decreased in order to minimize the error, until we have the right answer and the error is at minimum.

The results for 30 epochs are presented below. An epoch is when the whole dataset is passed forward and backward through the network but only once. The reason we choose 30 epochs is that after 15 epochs we obtained an accuracy of 89-90% so it was room for improvement and if we increased the epochs number more than 30 the accuracy would not be higher than 98% but training time will increase.

The model accuracy was obtained with Keras metrics, and the formula for accuracy is:

$$\text{Accuracy} = \frac{\text{number of correct prediction}}{\text{total number of predictions}}$$

The accuracy of the model is 97%

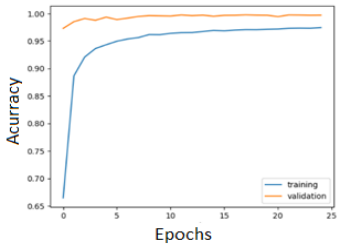


Figure 11. Algorithm efficiency per dataset pass

VI. Conclusion

The reason we choose a convolutional neural network is that now the best option for computer vision applications, as in terms of accuracy and easiness of implementation compared to other networks it stands above all. The main advantage of CNN compared to its predecessor is that it automatically detects features without any human supervision. In our case, we served the network many pictures of traffic signs and the algorithm learned distinctive features for each traffic sign class, with an accuracy of 97%.

As presented in the images below the network recognizes images rotated with an angle of 30° or zoomed in or out. The average time of recognition for an image was 0.2 seconds with a maximum distance from camera of two meters.

Figure 12. Experimental results

VIII. References

[1] Pierre Sermanet and Yann LeCun "Traffic Sign Recognition with Multi-Scale Convolutional Networks" Courant Institute of Mathematical Sciences, New York University

[2] C.-C. Jay Kuo "Understanding Convolutional Neural Networks with A Mathematical Model" Ming-Hsieh Department of Electrical Engineering University of Southern California, Los Angeles, CA 90089-2564, USA

[3] CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, spring 2020

[4] Vision-Based Traffic Sign Detection and Recognition
Systems: Current Trends and Challenges Safat B. Wali, Majid A. Abdullah, Mahammad A. Hannan, Aini Hussain , Salina A. Samad, Pin J. Ker and Muhamad Bin Mansor, 6 May 2019.

[5] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[6] Saadna, Yassmina & Behloul, Ali. (2017). An overview of traffic sign detection and classification methods. International Journal of Multimedia Information Retrieval. 6. 1-18. 10.1007/s13735-017-0129-8.