SPEED SIGN DETECTION AND RECOGNITION BY CONVOLUTIONAL NEURAL NETWORKS

Peemen, Maurice*, Mesman, Bart, Corporaal, Henk
Eindhoven University of Technology, the Netherlands,

KEYWORDS – Convolutional Neural Networks, Feature Extraction, Automated Recognition, Speed Sign, GPU.

ABSTRACT – From the desire to update the maximum road speed data for navigation devices, a speed sign recognition and detection system is proposed. This system should prevent accidental speeding at roads where the map data is incorrect for example due to construction work. Multiple examples of road sign classification systems already exist but none uses a fully trainable solution. This feature enables the "vendor" to easily add new speed signs by training with a set of examples instead of designing a new system. To meet the above requirements a fully trainable Convolutional Neural Network (CNN) is used for the detection and recognition of speed signs.

The system is trained with a labelled set of examples of speed sign images. Training of the total classification system is done off-line with the of error back-propagation algorithm. A trained system is used to collect new training data from road scene images to learn from previous errors, this technique is known as boosting. After the boosting step 0.19% of the images in our online available test set are misclassified. For the detection application the search window of the trained CNN is scaled to a 1280x720 HD image size to detect speed signs at multiple scales and positions in front of a vehicle. Because of the massive amount of parallelism in the computations of a CNN the algorithm is mapped to a Graphics Processing Unit (GPU). The GPU implementation demonstrates the abilities of the recognition system on a low cost consumer platform with a real-time frame rate of 35 fps.

1. INTRODUCTION

In the past decade the automobile industry has made a shift towards intelligent vehicles equipped with driving assistance systems. GPS navigation is an example of a commercially successful driving assistance system. The current state-of-the-art navigation devices do not only assist for correct navigation but also warns the driver to prevent accidental speeding. Most navigation systems use their static map with road type information to warn about the maximum allowed speed. In some situations the static map data alone is not sufficient, for example during road works that temporally change the maximum allowed speed. A solution to the dynamically changing road situation is the usage of vision systems.

Recently the automobile industry has introduced vision systems in their high end cars. Examples are the BMW 7er, Mercedes S-Class, Audi A8, Opel Insignia and VW Phaeton. Also in the research community vision systems for speed sign recognition are published [1][2][3]. All these systems are designed for their specific application as a pipeline of carefully tuned algorithms. In the first step of the pipeline the input is pre-processed with fixed algorithms such as Lightning correction, Histogram stretch, Colour segmentation, Edge detection, Hough transform, etc. The result of these steps is used to perform the classification by a matching algorithm or by machine leaning techniques such as Artificial Neural Networks

(ANNs) or Support Vector Machines (SVMs). Designing such a pipeline of algorithms is very time consuming and must be redone to support new road signs or other objects.

The mapping of these computationally complex algorithms to onboard vision platforms is a time consuming task. This is especially true if the system is already deployed and the "vendor" wants to sell new functionalities to existing platforms. This is a realistic business case because car lifetime is estimated to 15 years but new developments in driving assistance systems return every 2 years. Therefore automotive "vendors" should have a flexible platform to anticipate on new developments. But we cannot expect all "vendors" to standardize on their hardware platforms. Some sort of a virtual platform such as a virtual machine for vision algorithms is a far more flexible solution.

Our vision solution differs from the current state-of-the-art systems in that it uses a platform independent flexible approach. The proposed vision system is based on a fully trainable Convolutional Neural Network (CNN). The inputs of the CNN are the raw pixel values and the outputs are direct confidence values representing the possibility of a speed sign. Usage of a fully trainable solution enables the feature to modify the objects of interest by training with a set of examples. After training the "vendor" can support speed signs from multiple countries by updating the set of network coefficients of the user. The CNN algorithm works as a virtual machine that can imitate all kind of vision systems. After the system is deployed new features such as Pedestrian detection [4], Lane detection and Car to car distance detection can be added by a simple update of the network coefficients. This makes that a CNN solution is very flexible and therefore reusable with the small effort of training and updating coefficients. With this system "vendors" can make money even long after the car has been sold.

Our contributions of this work are: ***1)*** The collection and labelling of a large training dataset consisting of speed sign images. ***2)*** The designing and off-line training of a CNN for a speed sign recognition application. ***3)*** Mapping of the recognition algorithm to a GPU that performs real-time detection at 35 fps.

The content of the paper is as follows. Section 2 contains an overview of the CNN model as introduced in [5]. Section 3 describes the collection procedure and characteristics dataset that is used for training and testing. Section 4 describes the total training procedure. Section 5 describes the mapping of the speed sign detection and recognition procedure for 1280x720 HD video frames on a GPU platform. Section 6 concludes with discussion.

2. ALGORITHM OVERVIEW

An example architecture of a CNN is shown in Fig. 1. This one is used for handwritten digit recognition [5]. The first layers of the network $\mathbf{C_1}$ up to $\mathbf{S_2}$ function as a trainable feature extractor. All the network layers contain neuron models as in a classical Multi Layer Perceptron (MLP) network [6]. The feature extraction layers $\mathbf{C_1}$ up to $\mathbf{S_2}$ have specific constraints such as local connectivity and weight sharing. With these constraints the first layers are able to extract position invariant features from two-dimensional shapes. The classification layers $\mathbf{n_1}$ and $\mathbf{n_2}$ at the output are fully connected MLPs. These layers use the extracted local features to perform classification of the input image. The details of the three different types of layers are described in the next subsections.

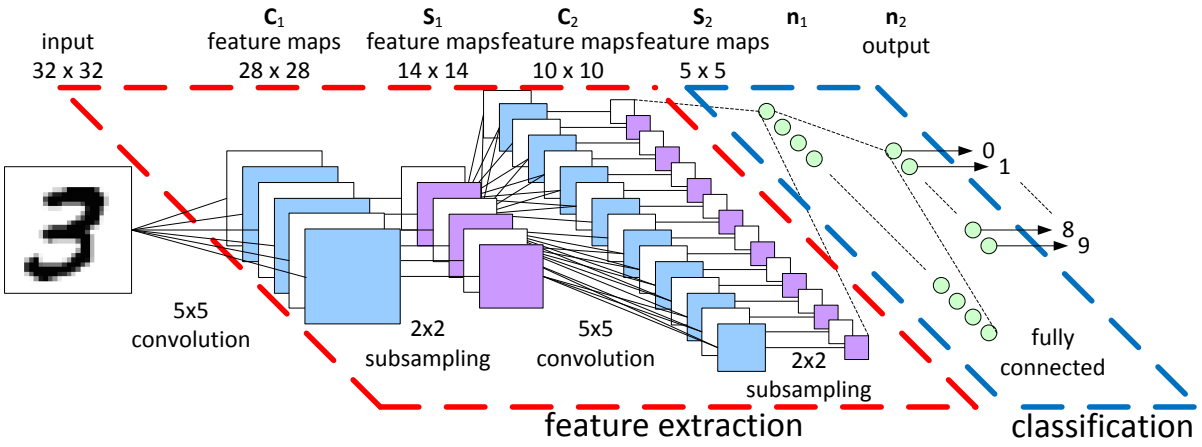

Fig. 1 CNN architecture for a handwritten digit recognition task.

2.1 Convolution Layers (CLs)

The feature maps of CLs, such as **C_1** and **C_2** in Fig. 1, contain neurons that take their synaptic inputs from a local receptive field, thereby detecting local features. The weights of the convolution neurons within a feature map are shared, so the position of the local feature becomes less important, thereby yielding shift invariance. The expression to compute a convolution neuron output is given as:

$$y[m,n] = b + \sum_{k=0}^{K-1}\sum_{l=0}^{K-1} v[k,l]x[m+k,n+l] \tag{1}$$

This expression describes the convolution operation on input image *x* with convolution kernel *v.* The only difference with a standard convolution is the threshold value *b* which is added to the result.

2.2 Subsample Layers (SLs)

CLs are succeeded by a SL to perform a data reduction operation of the CL result. This data reduction operation is done by local averaging over a predefined, non-overlapping window. The size of the averaging window is described by the subsample factor *S.* The expression for computation of a subsample neuron is given as:

$$y[m,n] = \phi(p) = \phi(b + u\sum_{k=0}^{S-1}\sum_{l=0}^{S-1} x[mS+k,nS+l]) \tag{2}$$

where,

$$\phi(p) = \frac{1}{1+\exp(-p)} \tag{3}$$

2.3 Neuron Layers (NLs)

The classification of the input image of the CNN is done at the output layers such as **n_1** and **n_2** in Fig. 1. In these layers all neurons have a unique set of weights, this enables them to detect complex features and perform a classification. The expression for the computation of these classical perceptrons [6] is given as:

$$y[n] = \phi(p) = \phi(b[n] + \sum_{k=0}^{K-1} w[n,k]x[k]) \tag{4}$$

An important property of the CNN architecture is that all synaptic weights and bias values can be trained by cycling the simple and efficient stochastic mode of the error back-propagation algorithm through the training sample [5].

3. DATASET CONSTRUCTION

The detection and classification of the speed signs is performed by a fully trainable CNN. Instead of focussing on improved features and algorithms we focused on learning from a large dataset. In other published work is already observed that the use of large data sets for trainable classifiers shows very promising results for generalization and recognition accuracy [7]. Therefore the collection of a representative training dataset is very important because it has a big impact on the recognition performance.

The construction of the initial training dataset is performed by manually collecting real-world images of the investigated speed sign classes on roads in the Netherlands. One part is collected by using Google search commands for images the other part is collected by using images from Google Street View. To increase the training set, selections of speed sign images that are published in [8] are added. The constructed dataset contains 713 images of speed signs that are all cropped to show the full speed sign with a small border. A subset of the training images is depicted in Fig. 2. Training with real-world images with natural variations results in invariance to light conditions, small scale differences and small rotations.

Fig. 2 Example images from the collected traffic sign dataset.

Using only traffic sign image is not sufficient to train the classifier; a good set of background images is also required. A representative background training class prevents false detections and is therefore very important. One part of the background class is gathered from random image patches that are collected form road scene images. The remaining part is constructed by traffic sign images that look similar to the speed sign classes as depicted in Fig. 3.

Fig. 3 Example images from the background class

To test the generalization of the trained classifier with data that is not used for training; 1/7 part of the images of each class is separated into a testing set. A complete overview of the dataset organization is depicted in Table 1. Collecting a speed sign dataset of this size is time consuming but necessary for training. To our knowledge little effort is done to publicize real-world speed sign image databases. Also the fact that speed signs differ from country to country makes it very valuable to publicize our dataset [9].

Table 1. Dataset configuration with the desired output coding for each class

class	output label	# patterns		
		training	testing	total
background	0 0000000	2489	415	2904
30 km/h max	1 1000000	77	13	90
50 km/h max	1 0100000	120	20	140
60 km/h max	1 0010000	62	10	72
70 km/h max	1 0001000	129	21	150
80 km/h max	1 0000100	75	13	88
90 km/h max	1 0000010	79	13	92
100 km/h max	1 0000001	69	12	81
		3100	517	3617

4. TRAINING

The training algorithm that is used to learn the coefficients of the CNN is the on-line mode of error back-propagation [10]. The main reason to choose for this training algorithm is the fact that it scales well with large datasets of training data [6]. The basic idea of error back-propagation is to efficiently calculate the partial derivatives of the output error in function of the weights for a given input pattern. These partial derivatives are used to perform small corrections to the weights to the negative direction of the error derivatives. Because of space limitation the details of the derivation of the update rules are not described, a good description is found in [6] and [11].

4.1 Preparation of the training data

Before the training procedure is started the data set is prepared for training. First the dataset is converted to greyscale images; the main motivation is the fact that colour representations are not consistent between day and night conditions. For training of the CNN a fixed size window as input layer is required. In the experiment a 32x32 pixel input is used; this results in a final application that can detect speed signs of 32x32 pixels and more.

Inspired by the good results for handwritten digit recognition published in [5] the amount of different training examples is artificially increased by selecting small modifications to each image during training. Adding small distortions to the training set results in more invariance to the applied distortion. The following modifications are used to expand the training set:

- Light intensity modification, pixel values multiplied by `[0.8 0.9 1.0 1.1 1.2]`
- Shift in x and y position `[-2 -1 0 +1 +2]` pixel positions
- Scaling the image `[0.93 1 1.05]`

4.2 Network design

Inspired by the good results on hand written digit recognition by the Lenet-5 CNN [5], this architecture is used as a starting point of the exploration. The configuration of Lenet-5 is similar to the configuration displayed in Fig 1. Layer $\mathbf{C_1}$ contains 6 feature maps and layer $\mathbf{C_2}$ contains 16 feature maps with a specific connection scheme. This interconnection scheme forces the feature maps to learn different features. The fully connected neural layer $\mathbf{n_1}$ contains of 120 neurons and $\mathbf{n_2}$ has 8 output neurons. The 8 outputs correspond to the output coding depicted in Table 1.

In a paper under submission we propose efficiency optimization methods that reduce the workload of the feature extraction layers significantly. For the optimized Lenet-5 CNN used on a 1280x720 pixel input image the computational workload for layer $\mathbf{n_1}$ is 92% of the total execution workload. Therefore the parameters of layer $\mathbf{n_1}$ are varied to select a good performing and computationally efficient network architecture. Three test configurations are trained:

1). The original configuration with 120 neurons in layer $\mathbf{n_1}$ is trained as a reference.
2). The size of layer $\mathbf{n_1}$ is reduced to 80 fully connected neurons.
3). The first 40 $\mathbf{n_1}$ neurons of network 2 are connected to feature map 1 to 8 and the other 40 neurons are connected to feature map 9 to 16.

The training progress for the three configurations over 2000 epochs of training is depicted in Fig. 4. The recognition performance for the last 100 epochs is magnified in Fig. 5. From this graph we conclude that network 1 and 3 have the best recognition accuracy. The reduced number of neurons in network 2 has degraded the accuracy on the test set. On the other hand the more specific connections used in network 3 have compensated for a reduced number of neurons. The reduction of the number of neurons and local connection in layer $\mathbf{n_1}$ result in a reduction of the computational workload of 67% for this computation intensive layer. By taking these observations into account we conclude that network 3 is the best configuration.

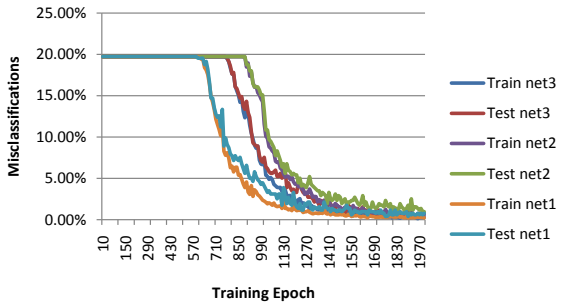

Fig. 4 Classification score over 2000 training epochs

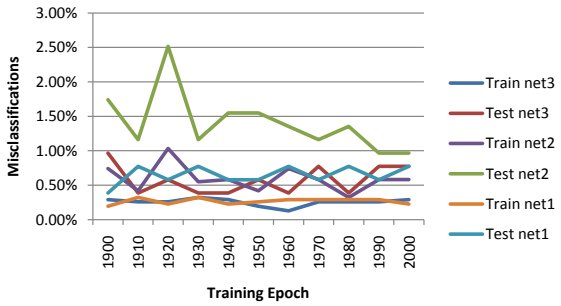

Fig. 5 Classification score at the end of training

4.3 Iterative boosting

During the first tests with the trained CNN for captured video material a large number of false detection are measured. The reason for the high amount of false detections is that the background class has an infinite variety of patterns which is impossible to include in the training set. To efficiently train the system to suppress all sorts of background patterns an iterative boosting algorithm as published in [12] is used. The basic idea behind the algorithm is to add only patterns to the training set which resulted in false detections. This forces the network learn from his previous errors.

To automate the boosting procedure 292 road scene images without speed signs are collected. First the recognition algorithm is executed on the new dataset with a high detection threshold. This step selects only detections with a confidence value above 0.9. The resulting patterns are the image patches for which the classifier is very sensitive. These examples are added to the training set and training is continued for 200 epochs. Next these steps are repeated with a decreased acceptance threshold. The results of the iterations are depicted in Table 2. After 6 boosting iterations is concluded that the number of false detections above a threshold of 0.5 is acceptable.

Table 2. Overview of the rejection score after each boost iteration

boost iteration	fixed score threshold	false detections	selection threshold	detections selected	train set
1	0.5	12595	0.9	361	3461
2	0.5	232	0.8	18	3479
3	0.5	419	0.7	85	3564
4	0.5	18	0.6	7	3571
5	0.5	103	0.5	103	3674
6	0.5	3	0.4	15	3689
7	0.5	1	0.3	19	

The network that is obtained after boosting scores very well on the test set that is used in section 4.2. For 517 test images only one misclassification is measured. The percentage of misclassifications is reduced from 0.77% without boosting, to 0.19% with boosting. Therefore is concluded that boosting improves the recognition accuracy of the CNN by a significant amount.

The obtained network functionality is stored as 24,958 coefficients. To load this functionality into an existing online CNN detection system requires a 100 KB software update. Mapping of new algorithms is not necessary because the network operations are not modified.

5. GPU IMPLEMENTATION

In this section the conversion of the trained Convolutional Neural Network (CNN) into a real-time detection system is outlined. The operations of a CNN contain a huge amount of parallelism. A cost effective platform to exploit parallelism is a GPU. Therefore the algorithm is mapped to a GPU. The platform that is used for the mapping is a consumer grade Nvidia GTX460.

The first step of the algorithm that is mapped is the construction of an image pyramid. This step subsamples the input image with steps of 1.25 to detect also large speed signs which are close to the camera. To perform this step efficient, the hardware optimized texture units of the GPU are used. The CUDA programming guide [13] describes how bilinear filtering is implemented for this task.

The next step is the processing of the layers of the CNN. First the implementation is optimized to improve data locality by loop interchange and tiling. Second GPU specific optimizations described in the CUDA programming guide [13] are applied. Memory accesses to the images are grouped to have coalesced memory accesses and the used kernel coefficients are stored in the fast constant memory. As final optimization the non-linear sigmoid activation function is evaluated fast using the special function units of the GPU. Fig 6. shows that the detection step of the CNN consumes most of the processing time. Summing the total processing time required for a 1280x720 HD video frame results in 28 ms of processing. This results in a practical frame rate of 35.7 frames per second for real-time detection.

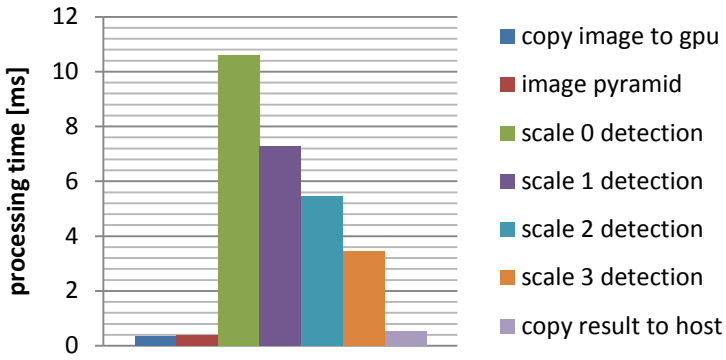

Fig. 6 Processing time of the CNN algorithm on the GPU platform

Example output video frames for the CNN speed sign detection application are depicted in Fig. 7 and 8. In these video frames it is shown that a speed sign results in multiple overlapping detection at different positions and scales. It is assumed that all visualize speed signs will have these overlapping detections over multiple frames. Therefore the recognition accuracy of the detector is improved by temporal and spatial integration of overlapping detections.

Fig. 7 Detector output for a 50 km/h speed sign

Fig. 8 Detector output for a 70 km/h speed sign

6. CONCLUSION

This paper proposes a speed-sign detection and recognition application based on a fully trainable Convolutional Neural Network (CNN). To train the application a dataset containing 713 real speed sign images from roads in the Netherlands are collected and published. The dataset is completed with 2904 background images of other road signs, threes, cars and other non-speed sign images. The speed sign recognition application is trained with greyscale images to classify speed signs based on their shape of at least 32x32 pixel values. During the training phase a network configuration exploration is performed. The exploration resulted in a computationally efficient network architecture which has the same recognition accuracy as the computationally complex configurations.

After the training phase the rejection performance is improved by applying an iterative boosting algorithm. Using the boosting algorithm on a set of road scene images reduces the amount of false detections tremendous. After training the application misclassifies only 0.19% percent of the test set. The resulting functionalities of the CNN are stored as 24,958 weights that can be trained by the “vendor” and are easy spread to the “consumer” by a 100

KB software update. No new mapping is required for this step because the elementary operations of the network do not change.
To demonstrate the real-time recognition speed of the application the algorithm is mapped to a consumer grade GPU. The GPU implementation has a recognition speed 35.7 fps for 1280x720 HD video. With this mapping example is shown that a fully trainable CNN based detection system scale very well for parallel GPU platforms. Multiple new low-cost embedded platforms for smartphones are already equipped with a mobile GPU e.g. (TI OMAP 3/4 or Nvidia Tegra 2). The next generation On Board Units (OBUs) for in car navigation are designed around these cost effective parallel platforms. Therefore is expected that real-time recognition systems can be deployed in the automotive market directly. This should improve safety on the roads and the driver receives less speeding tickets if he obeys to the directions of the OBU.

AKNOWLEGEMENT – We would like to thank TomTom for their input and collaboration.
This research is funded by the SPITS project (spits-project.com).

REFERENCES

(1) M.L. Eichner, T.P. Breckon, “Integrated Speed Limit Detection and Recognition from Real-Time Video”, Proc. IEEE Intelligent Vehicle Symposium, The Netherlands, 2008
(2) K.A. Ishak, M.M. Sani, N.M. Tahir, S.A. Samad and A. Hussain, “A Speed limit Sign Recognition System Using Artificial Neural Network”, 4th Student Conference on Research and Development, pp. 127-131, 2006
(3) A. Bargeton, “Improving pan-European speed-limit signs recognition with a new “global number segmentation” before digit recognition”, In Proc. Conf. IEEE Intelligent Vehicles Symposium, pp. 349-354, 2008
(4) M. Szarvas, U. Sakai and J. Ogata, “Real-time pedestrian detection using LIDAR and convolutional neural networks”, in Proc. IEEE Intell. Veh. Symp., pp. 224-229, 2005
(5) Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition”, Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998
(6) S. Haykin, “Neural Networks and Learning Machines”, 3rd ed., Prentice Hall, 2008
(7) D. Nistér, H. Stewénius, “Scalable recognition with a vocabulary tree”, CVPR, issue 2, pp. 2161-2168, 2006
(8) Y.-Y. Nguwi and S.-Y. Cho, “Emergent self-organizing feature map for recognizing road sign images”, Springer, Neural Computing & Applications, vol. 19, no. 4, pp. 601-615, 2009
(9) http://parse.ele.tue.nl/research/tools
(10) D.E. Rumelhart, G.E. Hinton and R.J. Williams, “Learning internal representations by error propagation”, MIT Press, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, pp. 318-362, 1986
(11) J. Bouvrie, “Notes on convolutional neural networks”, MIT CBCL Tech Report, pp. 38-44, 2006
(12) C. Garcia and M. Delakis, “Convolutional Face Finder: A neural architecture for fast and robust face detection”, IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1408-1423, 2004
(13) Nvidia Corporation, “NVIDIA CUDA C Programming Guide 3.2”, 2010