Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform

Maurice Peemen, Bart Mesman and Henk Corporaal

Eindhoven University of Technology, The Netherlands
m.c.j.peemen@tue.nl

Abstract. This paper proposes an algorithmic optimization for the feature extractors of biologically inspired Convolutional Neural Networks (CNNs). CNNs are successfully used for different visual pattern recognition applications such as OCR, face detection and object classification. These applications require complex networks exceeding 100,000 interconnected computational nodes. To reduce the computational complexity a modified algorithm is proposed; real benchmarks show 65 - 83% reduction, with equal or even better recognition accuracy. Exploiting the available parallelism in CNNs is essential to reduce the computational scaling problems. Therefore the modified version of the algorithm is implemented and evaluated on a GPU platform to demonstrate the suitability on a cost effective parallel platform. A speedup of 2.5x with respect to the standard algorithm is achieved.

Keywords: Convolutional Neural Networks, Feature Extraction, GPU.

1 Introduction

Visual object recognition is a computationally demanding task that will be used in many future applications. An example is surveillance, for which multiple human faces have to be detected, recognized and tracked from a video stream. The classical approach to achieve visual object recognition using a computer is to split the task into two distinct steps [9]: feature extraction and classification. During the first step, the original input is typically preprocessed to extract only relevant information. Classification with only the relevant information makes the problem easier to solve and the result becomes invariant to external sources like light conditions that are not supposed to influence the classification. Classical approaches use matching algorithms for classification. These compute the difference between the feature vector and a stored pattern to distinguish different object classes. To construct a pattern which gives robust results is a very difficult task. Therefore, it is desirable to train a classifier for a certain task by using a labeled set of examples.

Convolutional Neural Networks (CNNs) are fully trainable pattern recognition models that exploit the benefits of two step classification by using feature extraction [7]. CNN models are based on Artificial Neural Networks (ANNs) [4] but their network structure is inspired by the visual perception of the human

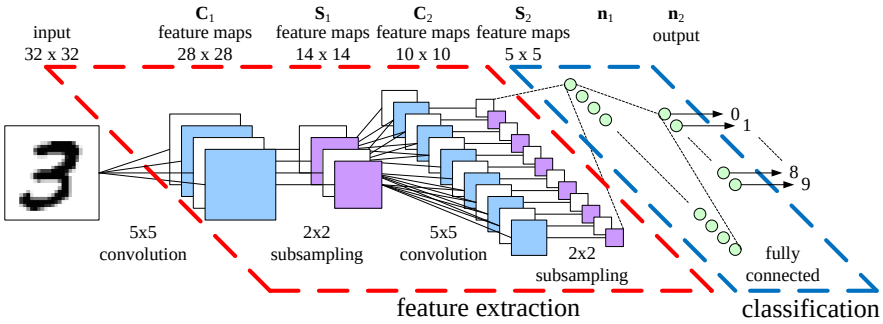


Fig. 1. An Example CNN architecture for a handwritten digit recognition task.

brain. The network architecture of an example CNN is depicted in Fig. 1. The processing starts with feature extraction layers and is finished by fully connected ANN classification layers. Using different layers delivers robust recognition accuracy and is invariant to small geometric transformations of the input images. The robust recognition accuracy makes that CNN are successfully used for classification tasks on real world data [3][7][14].

It is a challenge to implement these CNNs for real-time recognition; this is due their large computational workload, especially on high resolution images. Consider for example the results that are published for face recognition applications on programmable architectures (see Table 1). These results not yet meet real-time requirements, and assume a relative low resolution. To reach recognition results with 20 frames per second for 1280x720 HD video streams the processing speed must be improved considerably. The processing problem gets even worse when the implementation platform is a low cost consumer platform as used in smart phones. So a more efficient algorithm is needed.

The contribution of this work is a modified CNN architecture to reduce the computational workload and data transfer. Training rules for the modified architecture are derived and the recognition accuracy is evaluated with two real world benchmarks. An Intel CPU and an Nvidia CUDA-enabled Graphics Processing Unit (GPU) are used to demonstrate the performance improvement of the modified feature extraction layers.

The content of the paper is as follows. Section 2 contains an overview of the CNN model introduced in [6]. Section 3 describes the algorithmic optimization and training rules are derived. In Section 4, the recognition accuracy is evaluated. Section 5 describes the mapping of the feature extractors and the speedup of the modification is evaluated. Section 6 describes related work and in Section 7, the paper is summarized and concluded.

Table 1. Frame rate for a face recognition CNN on three programmable platforms.

platform	input pixels	frames per second
1.6 GHz Intel Pentium IV [3]	384x288	4.0
2.33 GHz 8-core Intel Xeon [1]	640x480	7.4
128-Core GPU Nvidia Tesla C870 [1]	640x480	9.5

2 CNN Algorithm Overview

An example architecture of a CNN is shown in Fig. 1. This one is used for handwritten digit recognition [7]. The last two layers $\mathbf{n}_1$ and $\mathbf{n}_2$ function as an ANN classifier. The first layers of the network $\mathbf{C}_1$ up to $\mathbf{S}_2$ function as a trainable feature extractor. These are ANN layers with specific constrains to extract position invariant features from two-dimensional shapes. The different layers in this architecture can be described by:

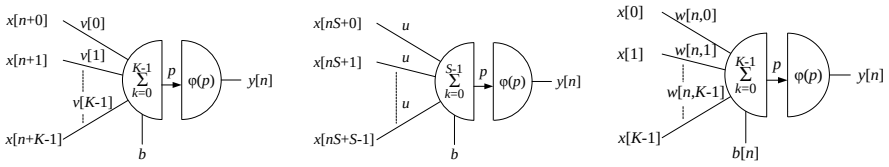
1) Convolution Layers (CLs): The feature maps of CLs, such as $\mathbf{C}_1$ and $\mathbf{C}_2$ in Fig. 1, contain neurons that take their synaptic inputs from a local receptive field, thereby detecting local features. The weights of neurons in a feature map are shared, so the exact position of the local feature becomes less important, thereby yielding shift invariance. The schematic overview of a convolution neuron for a one-dimensional input is shown in Fig. 2(a). The schematic shows the names for the different variables, x for the input window, v for the shared trainable kernel weights and b for the trainable bias value. The inputs are used to compute a weighted sum with kernel size K; this is represented as the neuron potential p. To generate an output value y, the potential value is passed through an activation function $\phi(p)$. For a two-dimensional feature map the model is rewritten, and the neuron operation can be described by

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

where,

$$\phi(x) = \frac{1}{1+\exp(-x)}. \tag{2}$$

The sigmoid activation function of (2) is used in this work, but many other functions can be used [4]. The kernel operation is a two-dimensional convolution on the valid region of the input. This 2d-convolution is done multiple times with different kernels to generate multiple feature maps that are specialized to extract different features. Some of these feature maps as in layer $\mathbf{C}_2$ of Fig. 1 are fed by multiple inputs, in this case multiple kernels are used, one for each input and the results are summed.



(a) CL (b) SL (c) NL

Fig. 2. Schematic models of different neuron types in a CNN. (a) Convolution Layer (CL) neuron. (b) Subsample Layer (SL) neuron. (c) Neuron Layer (NL) neuron.

2) Subsampling Layers (SLs): A CL is succeeded by a SL to carry out a data reduction operation of the CL output. The data reduction operation is done by local averaging over a predefined, non-overlapping window; the size is described by the subsample factor S. The result of local averaging is multiplied by a shared trainable coefficient u and a shared bias coefficient is added to the result before it is passed through the activation function. The schematic model of a one-dimensional subsampling neuron is depicted in Fig. 2(b). The mathematical model of a two-dimensional feature map gives

$$y[m,n] = \phi(p) = \phi(b + u \sum_{k=0}^{S-1} \sum_{l=0}^{S-1} x[mS+k, nS+l]). \tag{3}$$

3) Neuron Layers (NLs): The output layers of a CNN such as $\mathbf{n}_1$ and $\mathbf{n}_2$ in Fig. 1 contain classical neuron models or perceptrons [4]. The perceptron model that is depicted in Fig. 2(c) has a unique set of weights w and bias b for each neuron. With the unique set of weights each neuron can detect a different pattern; this is used to make the final classification. In most NLs the result of the preceding layer is used as a one-dimensional fully connected input. When K equals the number of neurons in the preceding layer the expression to compute the NL output 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 [7].

3 Algorithm Optimization

As is shown in Section 1 the high computational complexity of CNNs restricts their applications to high performance computer architectures. To enable CNN applications for cheap consumer platforms a reduction of the computational workload would be very desirable. This reduction is achieved by a high level modification that reduces the number of Multiply Accumulate (MACC) operations and the amount of data movement in the feature extractor. High level modifications to an algorithm can have a huge impact on performance, but in most cases it is a trade-off between recognition accuracy and computational complexity. Therefore changes in the algorithm must be analyzed carefully to verify that the classifier does not lose the learning and classification abilities it had before. To analyze the recognition performance new training rules are derived and two real world benchmarks are used to validate the recognition performance.

3.1 Merge Convolution and Subsampling

The data dependencies between a CL and a SL; as depicted in Fig. 3 show that the SL output can be calculated directly from the input. Therefore the succeeding

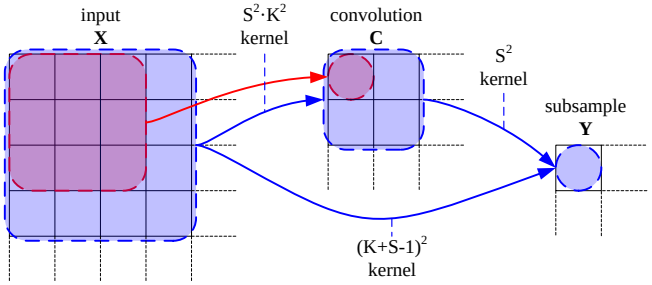


Fig. 3. Feature extraction layer example with 2d-convolution kernel size K=3 and subsample factor S=2, data dependencies are visualized from input to CL and SL. For the merged method of operation there is no need for a middle CL.

operations are merged; this is only possible if the activation function of the CL is linear. The merged expression with corresponding coefficients is derived by substitution of the CL expression (1) with a linear activation function into the SL expression (3).

$$\begin{aligned} y[m,n] &= \phi_s(b_s + u \sum_{i=0}^{S-1} \sum_{j=0}^{S-1} c[mS+i, nS+j]) \\ &= \phi_s(b_s + u \sum_{i=0}^{S-1} \sum_{j=0}^{S-1} \phi_c(b_c + \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} v[k,l]x[mS+i+k, nS+j+l])) \\ &= \tilde{\phi}(\tilde{b} + \sum_{k=0}^{K+S-2} \sum_{l=0}^{K+S-2} \tilde{v}[k,l]x[mS+k, nS+l]) \end{aligned} \quad (5)$$

The enlarged kernel $\tilde{v}$ is constructed from all coefficients that are multiplied with each input value x. The new bias $\tilde{b}$ is the CL bias multiplied by u and added to the SL bias. From Fig. 3 and (5) is concluded that merging a linear CL and a SL result in a reduction of MACC operations while retaining the functional correctness. With the significant reduction of MACC operations the number of memory accesses is also reduced because there is no intermediate storage of a CL. Table 2 shows expressions for the number of kernel weights, MACC operations and memory accesses that are required to calculate a feature map output. The reduction of MACC operations for multiple merged CL and SL configurations is depicted in Fig. 4.

Table 2. For a feature map output the required number of weights, MACC operations and memory accesses depend on kernel size K and subsample factor S.

feature extractor	# kernel weights	# MACC operations	# mem. accesses
CL and SL	K^2+1	$S^2(K^2+1)$	$S^2(K^2+2)+1$
merged	$(K+S-1)^2$	$(K+S-1)^2$	$(K+S-1)^2+1$

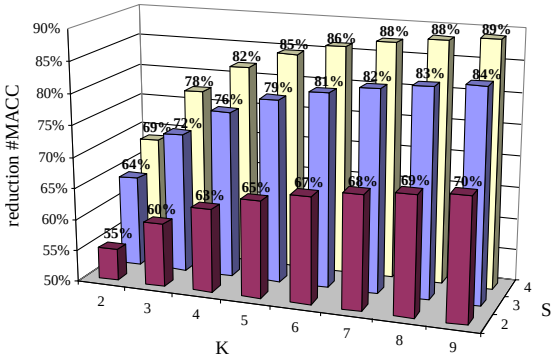


Fig. 4. Reduction of the #MACC operations to calculate a merged feature map compared to the original algorithm for multiple kernel sizes K and subsample factors S.

It is not possible to derive the coefficients of $\tilde{v}$ and $\tilde{b}$ if a non-linear activation function is used in the CL. This is not a problem; the learning algorithm is adapted such that it can train the coefficients for the merged configuration. After merging the CL and SL the weight space is changed; therefore training could find better solutions in the weight space which makes derivation from the old weight space suboptimal. The recognition performance of such a trained kernel is evaluated in Section 4.

During the remaining part of this paper the merged configuration is used. The merged layers are named Feature Extraction Layers (FELs). For completeness the expression is given for a variable number of input feature maps,

$$y[m,n] = \phi(b + \sum_{q \in Q} \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} v_q[k,l] x_q[mS+k, nS+l]). \tag{6}$$

As depicted in Fig. 5(a) the number of input feature maps can vary for each feature map. The set Q in (6) contains the indices of the connected input feature maps. The constant K describes the new kernel size and S describes the step size of the input window on an input feature map as depicted in Fig. 5(b).

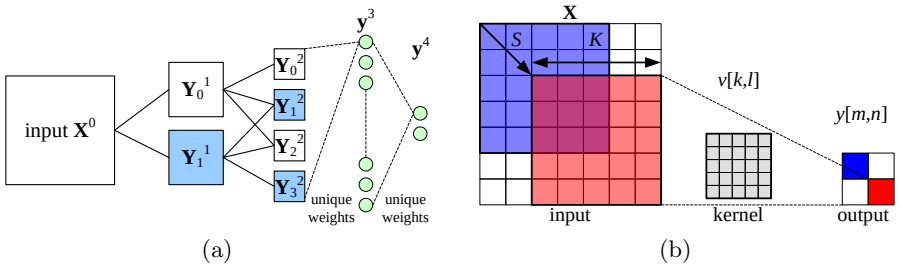


Fig. 5. Variables and indices required for feed-forward computation. (a) Feature map naming with connections. (b) Variables for computation of feature map neurons.

3.2 Training with Error Back-Propagation

The training algorithm that is used to learn the coefficients of the merged FELs is the on-line mode of error back-propagation [12]. The derivation of the training rules is described in detail because the merged FELs change the published CNN training expressions [7]. The new contributions to the training procedure are the steps involving the merged FELs.

The basic idea of error back-propagation is to calculate the partial derivatives of the output error in function of the weights for a given input pattern. The partial derivatives are used to perform small corrections to the weights to the negative direction of the error derivative. This procedure is split into three parts; feed-forward processing, compute partial derivatives and update the weights. For clarity the network depicted in Fig. 5(a) is used to explain the training procedure.

1) Feed-forward processing: Before training all weights are initialized to a small random value. Then a pattern of the training sample is processed in feed-forward mode through the FELs (6) and the NLs (4). The feed-forward propagation results in an output vector that is compared with the desired output in the cross-entropy (CE) error-function [5],

$$E_{CE} = -\sum_{n \in N} d_n \log(y_n) + (1 - d_n) \log(1 - y_n). \tag{7}$$

In (7) the set N contains all output neuron indices and d_n the target values.

2) Compute partial derivatives: In the previous expressions x is used as input and y as output, for the error derivatives more variables are required. The remaining expressions use λ to describe in which layer variables are positioned. The partial derivatives are found by applying the chain rule on the CE error function of (7), which results in

$$\begin{aligned} \frac{\partial E_{CE}}{\partial w_n^\lambda[k]} &= \frac{\partial E_{CE}}{\partial y_n^\lambda} \frac{\partial y_n^\lambda}{\partial p_n^\lambda} \frac{\partial p_n^\lambda}{\partial w_n^\lambda[k]} \\ &= \frac{y_n^\lambda - d_n}{y_n^\lambda (1 - y_n^\lambda)} \dot{\phi}(p_n^\lambda) y_k^{\lambda-1} \\ &= \delta_n^\lambda y_k^{\lambda-1} \end{aligned} \tag{8}$$

where,

$$\dot{\phi}(x) = \phi(x)(1 - \phi(x)) = y_n(1 - y_n), \tag{9}$$

$$\delta = \frac{\partial E_{CE}}{\partial y} \frac{\partial y}{\partial p}. \tag{10}$$

Efficient computation of the partial derivatives for NLs that are not positioned at the output is performed by reusing the local gradient δ of the succeeding layer.

$$\begin{aligned} \frac{\partial E_{CE}}{\partial w_n^{\lambda-1}[k]} &= \sum_{i \in D} \frac{\partial E_{CE}}{\partial y_i^\lambda} \frac{\partial y_i^\lambda}{\partial p_i^\lambda} \frac{\partial p_i^\lambda}{\partial y_n^{\lambda-1}} \frac{\partial y_n^{\lambda-1}}{\partial p_n^{\lambda-1}} \frac{\partial p_n^{\lambda-1}}{\partial w_n^{\lambda-1}[k]} \\ &= \sum_{i \in D} \delta_i^\lambda w_i^\lambda[n] y_n^{\lambda-1} (1 - y_n^{\lambda-1}) y_k^{\lambda-2} \\ &= \delta_n^{\lambda-1} y_k^{\lambda-2} \end{aligned} \tag{11}$$

The set D in (11) contains all neurons of the succeeding layer that are connected to neuron $y_n^{\lambda-1}$ or y_n^3 in Fig. 5(a). To compute the partial derivatives for multiple NLs (11) is used recursively.

The calculation of the gradients for weights in the FELs is done in two steps. First the local gradients are computed by back-propagation from the succeeding layer. Second the local gradients are used to compute the gradients of the weights. Computation of the local gradients for an FELs succeeded by an NL such as for y^2 in Fig. 5(a) is expressed as

$$\delta^{\lambda}[m,n] = \sum_{i \in D} \delta_i^{\lambda+1} w_i^{\lambda+1}[m,n] \dot{\phi}(p^{\lambda}[m,n]). \tag{12}$$

If the succeeding layer is an FEL a select set of neurons is connected which makes computation of the local gradients complex. The connection pattern is influenced by the current neuron indices, the subsample factor and the kernel size as depicted in Fig. 6. For the two-dimensional case the local gradient is

$$\delta^{\lambda}[m,n] = \sum_{q \in Q} \sum_{k=K_{\min}}^{K_{\max}} \sum_{l=L_{\min}}^{L_{\max}} (\delta_q^{\lambda+1}[k,l] v_q^{\lambda+1}[m-Sk, n-Sl] \dot{\phi}(p^{\lambda}[m,n])) \tag{13}$$

where,

$$K_{\max} = \lfloor \frac{m}{S} \rfloor, \quad K_{\min} = \lfloor \frac{m-K+S}{S} \rfloor, \quad L_{\max} = \lfloor \frac{n}{S} \rfloor, \quad L_{\min} = \lfloor \frac{n-K+S}{S} \rfloor.$$

Border effects restrict $K_{\min}$, $K_{\max}$, $L_{\min}$ and $L_{\max}$ to the featuremap indices.

The obtained local gradients are used to compute the gradients of the FEL coefficients. The bias is connected to all neurons in a feature map therefore the gradient is computed by summation over the local gradients in a feature map.

$$\frac{\partial E_{CE}}{\partial b} = \sum_{m=0}^{M} \sum_{n=0}^{N} \delta[m,n] \tag{14}$$

The gradients for the kernel weights of a FEL are computed by

$$\frac{\partial E_{CE}}{\partial v^{\lambda}[k,l]} = \sum_{m=0}^{M} \sum_{n=0}^{N} \delta^{\lambda}[m,n] y^{\lambda-1}[mS+k, nS+l]. \tag{15}$$

3) Update the coefficients of the network: The delta rule of the error back-propagation algorithm is used to keep the training algorithm simple and easy to reproduce with η as single learning parameter. The update function is given as

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial E_{CE}}{\partial W_{\text{old}}}. \tag{16}$$

In the update function (16) W represent the weights w, kernels v and bias b for all possible indices in the network.

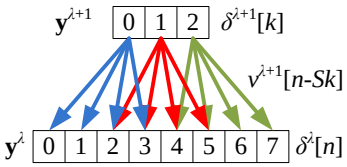


Fig. 6. Two one dimensional FELs with K=4 and S=2. To compute the local error gradient the error is back-propagated from the succeeding layer.

4 Validate the Recognition Performance

Evaluation of the recognition performance of the merged feature extractors is performed by a training task on two published datasets. The availability of published training results for CNN implementations is the main motivation to use these datasets for a fair comparison.

The first training task is performed on the MNIST handwritten digit dataset [7]. This dataset consist of 28x28 pixel images of handwritten digits as shown in Fig. 7(a). For evaluation of feature extraction based on separated or merged CLs and SLs, a MATLAB implementation of a CNN based on LeNet-5 [7] is trained for both configurations. For fair training and testing the original separation of the MNIST data into 60,000 training and 10,000 test samples is used. The classification performance is expressed as the percentage of the test set that is misclassified. Classification of a pattern of the test set is performed by selecting the output neuron that is activated the most (winner takes all). These outputs represent the digits zero to nine. The classification score for the original and the merged network for the MNIST dataset is summarized in Table 3.

The second dataset is the small-NORB stereoscopic object classification dataset [8]. This dataset consists of 96x96 pixel image pairs which belong to one of five object classes; a subset is shown in Fig. 7(b). The dataset contains 50 different objects, 25 for training and 25 for testing which are equally distributed over the five classes. These objects are shown from different angles and with different lightning conditions, which makes that each set consist of 24,300 image pairs. For comparison of the recognition performance, implementations with separated or merged CLs and SLs which are based on the LeNet-7 [8] are trained. The classification scores for the small-NORB dataset are also shown in Table 3.

(a) MNIST (b) small-NORB

Fig. 7. Subset of the visual patterns that are used for training.

Table 3. Comparison of the training results for MNIST and NORB data set.

benchmark	misclassification	# MACC ops.	# coefficients FELs
MNIST LeNet-5 [7]	0.82%	281,784	1,716
separated CLs and SLs	0.78%	281,784	1,716
merged CLs and SLs	0.71%	97,912	2,398
reduction	8.97%	65%	-40%
NORB LeNet-7 [8]	6.6%	3,815,016	3,852
separated CLs and SLs	6.0%	3,815,016	3,852
merged CLs and SLs	6.0%	632,552	6,944
reduction	0%	83%	-80%

Important to conclude from the results of the experiments is that merging the convolution and subsample layers of the feature extractor do not negatively influence the networks ability to generalize. The number of MACC operation to do feed-forward detection is significantly reduced. As mentioned before there is also a non-favorable property, this is the increase of the number of coefficients which is due to the increased kernel sizes as described in Section 3.1. The networks that are implemented for this experiment do not need extra preprocessing of input patterns, such as mean removal. The training procedures used in [7] and [8] use this extra preprocessing on the input data to improve recognition results.

5 Implementation

Demonstration of the recognition speedup as result of merged FELs is performed with a real application. The application is an internally developed implementation of a road sign detection and classification CNN [11]. The CNN is build with merged FELs and with separated CLs and SLs to compare the processing speed. Both CNNs are trained to classify road signs on a 1280x720 HD input image.

First a C implementation of the two feature extractor configurations are executed on a 2.66 GHz Core-i5 M580 platform as a reference. The implementation is optimized to exploit data locality by loop interchanges. Compilation of the code is done with MS Visual Studio 2010, all compiler flags are set to optimize for execution speed. The timing results for the two feature extractors are shown in the first column of Table 4. The speedup of a factor 2.8x after merging matches the expectation. In Fig. 4 is shown that the workload of the feature extractor with kernel size $K = 5$ and subsample factor $S = 2$ is reduced with 65%.

Table 4. Timing comparison of the FELs for the standard and the merged configuration.

configuration	CPU	GPU	speedup
CLs and SLs	577 ms	6.72 ms	86 x
merged FELs	203 ms	2.71 ms	75 x
speedup	2.84 x	2.48 x	

The processing in the feature extractor contains a huge amount of parallelism. A cost effective platform to exploit parallelism is a GPU. Therefore the standard and the merged feature extractors are mapped to a GPU platform to test the impact of the proposed optimization on a parallel implementation. The platform that is used for the experiment is an Nvidia GTX460. First the GPU implementation is optimized to improve data locality by loop interchange and tiling. Second GPU specific optimizations described in the CUDA programming guide [10] 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. The following intrinsics from the CUDA programming guide are used to perform a fast but less accurate evaluation of the sigmoid activation function.

```
__fdividef(1,1+__expf(-x));
```

The kernel execution times for the two GPU implementations are shown in Table 4. The GPU speedup after merging is close to the CPU speedup, this shows that the performance gain of merging is not reduced much due to a parallel implementation.

6 Related work

Acceleration of CNNs is not a new field of research. Since a few years the first dedicated hardware implementations of CNNs for FPGA platforms are published in [1] and [2]. These implementations are based on hand crafted systolic implementations of the convolution operation to speed up execution time. Non of these implementations explore high level trade-offs to the CNN algorithm.

A different simplified CNN is given in [13]. Instead of averaging with subsampling using (3), they only calculate convolution outputs for S^2 pixels. This also reduces computational complexity, but likely at a severe recognition quality loss. However the paper does not report on this. Furthermore, the work in [13] differs from this work because no analysis is published that shows how kernel size K and subsample factor S influence the reduction of computational complexity. Performance measurements of the simplified algorithm on real platforms such as a GPU are not published.

7 Conclusion

In this work a high level algorithm modification is proposed to reduce the computational workload of the trainable feature extractors of a CNN. The learning abilities of the modified algorithm are not decreased; this is verified with real world benchmarks. These benchmarks show that the modification results in a reduction of 65-83% for the required number of MACC operations in the feature extraction stages.

To measure the real speedup that is gained by the algorithm modification; an implementation of a road sign classification system is performed. This application is mapped to a CPU and a GPU platform. The speedup of the CPU implementation is a factor 2.7 where the GPU implementation gains a factor 2.5, compared to the original convolution and subsample feature extractor. These speedups on real platforms prove that the proposed modification is suitable for parallel implementation.

The modifications that are proposed in this paper enable implementations of CNNs on low cost resource constrained consumer devices. This enables a legacy of applications that use trainable vision systems on mobile low-cost devices such as smartphones or smart cameras.

References

1. Chakradhar, S., Sankaradas, M., Jakkula, V., Cadambi, S.: A dynamically configurable coprocessor for convolutional neural networks. In: ISCA '10: Proceedings of the 37th annual international symposium on Computer architecture. pp. 247–257. ACM, New York, NY (2010)
2. Farabet, C., Poulet, C., Han, J., LeCun, Y.: Cnp: An fpga-based processor for convolutional networks. Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on pp. 32–37 (aug 2009)
3. Garcia, C., Delakis, M.: Convolutional face finder: A neural architecture for fast and robust face detection. IEEE Trans. Pattern Anal. Mach. Intell. 26(11), 1408–1423 (2004)
4. Haykin, S.: Neural Networks and Learning Machines. Prentice Hall, 3 edn. (2008)
5. Hinton, G.E.: Connectionist learning procedures. Artif. Intell. 40(1-3), 185–234 (1989)
6. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. 1(4), 541–551 (1989)
7. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (nov 1998)
8. Lecun, Y., Huang, F.J., Bottou, L.: Learning methods for generic object recognition with invariance to pose and lighting. In: In Proceedings of CVPR04 (2004)
9. Nixon, M., Aguado, A.S.: Feature Extraction & Image Processing, Second Edition. Academic Press, 2nd edn. (2008)
10. Nvidia: NVIDIA CUDA C Programming Guide 3.2. NVIDIA Corporation (2010)
11. Peemen, M., Mesman, B., Corporaal, C.: Speed sign detection and recognition by convolutional neural networks. In: Proceedings of the 8th International Automotive Congress. pp. 162–170 (2011)
12. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1. pp. 318–362. MIT Press, Cambridge, MA (1986)
13. Simard, P., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: ICDAR. pp. 958–962 (2003)
14. Szarvas, M., Yoshizawa, A., Yamamoto, M., Ogata, J.: Pedestrian detection with convolutional neural networks. In: Proceedings IEEE Intelligent Vehicles Symposium. pp. 224–229. Las Vegas, NV (Jun 2005)