Face Detection using GPU-based Convolutional Neural Networks

Fabian Nasse[1], Christian Thurau[2] and Gernot A. Fink[1]

[1] TU Dortmund University, Department of Computer Science, Dortmund, Germany
[2] Fraunhofer IAIS, Sankt Augustin, Germany

Abstract. In this paper, we consider the problem of face detection under pose variations. Unlike other contributions, a focus of this work resides within efficient implementation utilizing the computational powers of modern graphics cards. The proposed system consists of a parallelized implementation of convolutional neural networks (CNNs) with a special emphasize on also parallelizing the detection process. Experimental validation in a smart conference room with 4 active ceiling-mounted cameras shows a dramatic speed-gain under real-life conditions.

1 Introduction

The past years yielded increasing interest in transfering costly computations to Graphics Processing Units (GPUs). Due to parallel execution of commands this often results in a massive speedup. However, it also requires a carefull adaption and parallelization of the algorithm to be implemented. As noted in [1], convolutional neural networks (CNNs) [8, 2, 4, 3] offer state of the art recognizers for a variety of problems. However, they can be difficult to implement and can be slower than other classifiers, e.g. traditional multi-layer perceptrons. The focus of this paper resides within the implementation details of parallelizing CNNs for the task of face detection and pose estimation and evaluating its run-time performance. In contrast to [1], where GPU optimized CNNs were adapted for document processing, the considered face detection task requires additional considerations: In detail, the contributions of this paper are (a) extension of the face recognition (and pose estimation) system in [3] by parallelizing important parts of the computational process and implementing it on a graphics card, and (b) further enhancing the system by an optimized detector. Experimental validation takes place in a multi-camera environment and shows accurate detection and high performance under real-life conditions.

The remainder of this paper is organized as follows: In Section 2 we give a brief introduction to convolutional networks and how they are used in this work. Section 3 explains in detail the process of face detection using CVs, and Section 4 shows how this process can be efficiently parallelized. In Section 5 we present experimental results of the proposed optimized face detection approach. Finally we conclude this paper in Section 6.

2 Convolutional Neural Networks

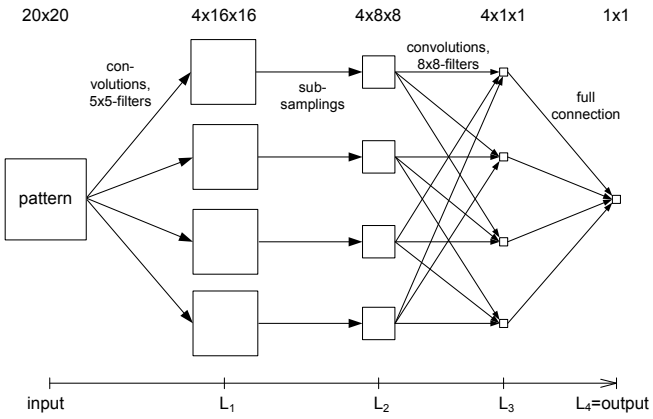

Fig. 1. structure of a simple convolutional net

In the following we will briefly introduce *convolutional neural networks (CNNs)* [8, 2, 4, 3]. In a nutshell, a CNN classifies an input pattern by a set of several concatenated operations, i.e. convolutions, subsamplings and full connections. Figure 1 shows a simple example for a CNN as it was used in [8]. For practical reasons the net is organized in successive layers (L_1 to L_4). On the left side we see an input image, in our case a monochromatic 20×20 pixel image. Each subsequent layer consist of several fields of the same size which represent the intermediate results within the net. Each directed edge stands for a particular operation which is applied on a field of a preceding layer and its result is stored into another field of a successive layer. In the case that more than one edge directs to a field, the results of the operations are summed. After each layer a bias is added to every pixel (which may be different for each field) and the result is passed through a sigmoid function, e.g. $s(x) = a \cdot tanh(b \cdot x)$, to finally perform a mapping onto an output variable.

Each convolution uses a different two-dimensional set of filter coefficients. Note that in case of a convolution the size of the successive field shrinks because the border cases are skipped. For subsampling operations a simple method is used which halves the dimensions of an image by summing up the values of disjunct 2x2-subimages and weighting each result value with the same factor. The term "'full connection"' describes a function, in which each output value is the weighted sum over all input values. Note that a full connection can be described as a set of convolutions where each field of the preceding layer is connected with every field of the successive layer and the filters have the same size as the input image. Thus, we do not treat full connections as a separate case here. The last layer forms the output vector. In the given example the output consists of a single value which finally classifies a given input image.

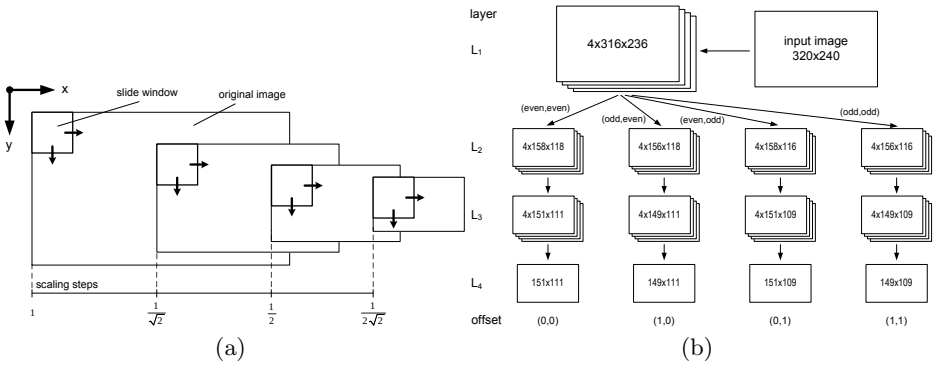

Fig. 2. Figure 2(a) shows a slide-window on an input image with different scaling factors. Figure 2(b) shows the scheme of the location process for the CNN from Figure 1 and an input image of size 320x240 px.

An important attribute of CNNs is the availability of an efficient training method. Since CNNs are based on the classical neural networks, each pixel of a field can be represented by a neuron and the all afore mentioned operations can be modeled by connections between neurons. For training, a modified version of a standard backpropagation algorithm can be applied. For further details on CNNs we recommend [2].

3 The Detection Process

For object/face detection we are usually interested in detecting all occurancies of an object in a given image. For a trained CNN, we can use a simple sliding window approach over a variety of scaled input images, see also Figure 2(a). The window is shifted above the image to get one result at each position. To search inside a specified size range the process is repeated with different scaling factors. In the given example the image is downscaled each time with the factor $1/\sqrt{2}$. By choosing this factor we make the assumption that the trained CNN is robust against variation in size at the range between two scaling steps.

One of the key advantages of CNNs are the inherent possibility for parallelizing the computational process when used as a detector. If a neural net is repeatedly applied on overlapping image areas redundancies in calculation occur. This is the case for convolutions as well as subsamplings in all layers. A significant speed gain is reached by avoiding these redundancies. We accomplished this by applying each operation within the net on the whole image at once instead repeatingly on all subimages. In the case of a subsampling operation four different offsets have to be considered depending on whether the 2x2-subimages start with odd or even coordinates in horizontal or vertical direction respectively. For the example given in figure 1 and an input image of size 320x240 this leads to a scheme as shown in figure 2(b). To assemble the four output images to one

the coordinates have to be multiplied by two and the spatial offset given in the last row must be added. Note that at the expense of precision the calculation of some paths in the tree may be skipped. The assumption by doing so is that the trained CNN is robust against small spatial shifts.

4 Parallelization of the Detection

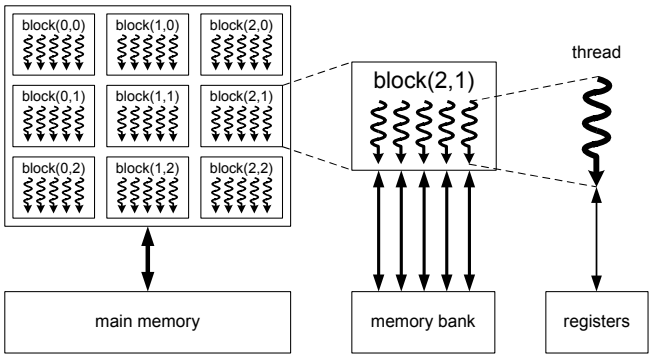

Fig. 3. Memory hierarchy for the CUDA architecture.

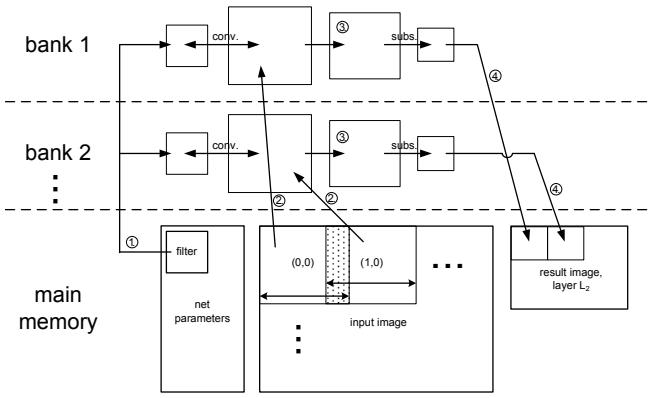

Fig. 4. Parallelizazion by dividing the input image into several rectangles.

The shown detection process was implemented and evaluated for the Nvidia GeForce 8800 GT GPU using the CUDA-architecture, which allows using the processing power of the graphics card for the purposes of scientific research. Note that also the following method is dedicated to CUDA-devices, similar approaches

can be applied to other multi-core devices. According to the number of cores the architecture consists of multiple threads each with its own set of registers. An important task for the design of the parallel algorithm is to take the memory hierarchy into account. Thus, figure 3 shows an overview. The host system (i.e. the pc) as well as the threads have access to the global main memory of the graphics card. Furthermore the threads are grouped into so called blocks. The threads of a block share a small and fast memory bank.

The different bandwidths of the memory interfaces leads to the following strategy: The data transfer between the host system and the main memory has a low bandwidth. Thus, the communication here has to be minimized. At the beginning the net parameters are loaded once from the host system into the main memory. The same is done for every new input image. At the end of the detection process the results are written back to the host. For the threads the access time to the memory banks is much faster than to the main memory. Therefore most of the calculation should be done with the data stored in the local memory banks. Because for a convolution the pixel values have to be read several times this brings a significant speed gain. Normally an input image is to large to be stored in one memory bank, thus it is divided into equidistant rectangles. Each rectangles is loaded from the main memory into another memory bank and the partial images are treated by the threads of the according block. For the example given above this approach is shown in figure 4. In this cutout one of the four fields of Layer S_2 is calculated.

At first the filter for the convolution is loaded from the main memory in every block (1). Next, the rectangles are copied (2). Note, that the subimages are overlapping according to the size of the filter. Next the filter and the subimage are convolved parallely in each bank and the result is stored in local memory (3). If the result field has more than one input edge (as in L_3) step one to three can be repeated with other input images and filters. Than the results can be summed directly on the spot. In spite of writing the results of Layer S_1 back to the main memory it is kept in local memory and the subsampling from layer L_1 to L_2 is done. After that the pieces are assembled in the main memory (4). How many steps can be accomplished without writing data back to the main memory or sharing data between blocks depends on the net structure and can be optimized for a particular net. Not mentioned so far is the addition of a bias and the use of a sigmoid function after each layer. For every thread the bias value is kept in a register (not shown in fig. 3). The sigmoid function is applied before writing the result value from a register back to local memory. Therefore no additional reading operation on the local memory is required. Nvidia GPUs support a complex instruction set with any kind of trigonometric or hyperbolic functions respectively. Nevertheless, if you use a hardware device without or with slow support for a particular function required for the sigmoid function, we recommend the use of a taylor approximation, e.g. $tanh(x/2) \approx (d-6)/(d+6)$ with $d = 6+x\cdot(6+x\cdot(3+x))$. Another important aspect not mentioned yet is how a convolution between a subimage and a filter is handled in detail. Note, that especially for convolutions concurrencies can form the bottleneck of the feasible

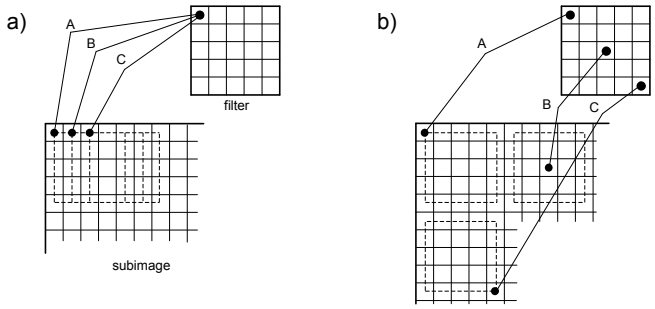

Fig. 5. Reading access for a convolution with three threads A,B and C. a) lexicographic access. b) spatially distributed access.

speed gain. According to Fig. 3 each subimage is treated by several threads with reading access to the same memory bank. Thus, each thread computes a subset of result values. Hence to the parallel execution concurrency between reading accesses occur. Each thread needs to read the whole filter and threads who compute neighboring values need access to overlapping image areas. To reduce latency hence to a limited fan in, reading accesses should be spread. This proceeding is demonstrated in figure 5 for an example with three threads (A, B and C). Part a) shows the proceeding when the computation is done in lexicographic order, as it is commonly done for a single process. When a thread has finished one result it continues at the next position in line of the subimage and restarts walking through the filter from the upper left corner. This causes a strong likelihood for latencies during the whole process. A better approach is shown in part b). Here the computation is spatially distributed over the subimage as well as over the filter.

5 Experiments

In order to test our accelerated detector under real-time-conditions, we implemented a recognition system for faces with variation of pose based on the researches of [3]. According to Fig. 6 a) the CNN we used consists of 42.750 parameters and it was trained with 6.000 non-faces and 6.000 faces with annotated poses (6 b). The system was applied in a smart conference room with 4 active ceiling-mounted cameras (6 c,d).

Although the main focus of this paper resides within an efficient implementation, we still want to briefly report on the detection rates here. Given a frontal/side view of a persons face the system is able to detect multiple persons with an average accuracy of 80%-90% percent and occasional occuring false positives (evaluated on a per frame basis for four longer sequences containing multiple persons). For a better detection we added additional training material for this particular environment, resulting in a well functional and usable system. For sake of completeness we also evaluated the proposed system on three stan-

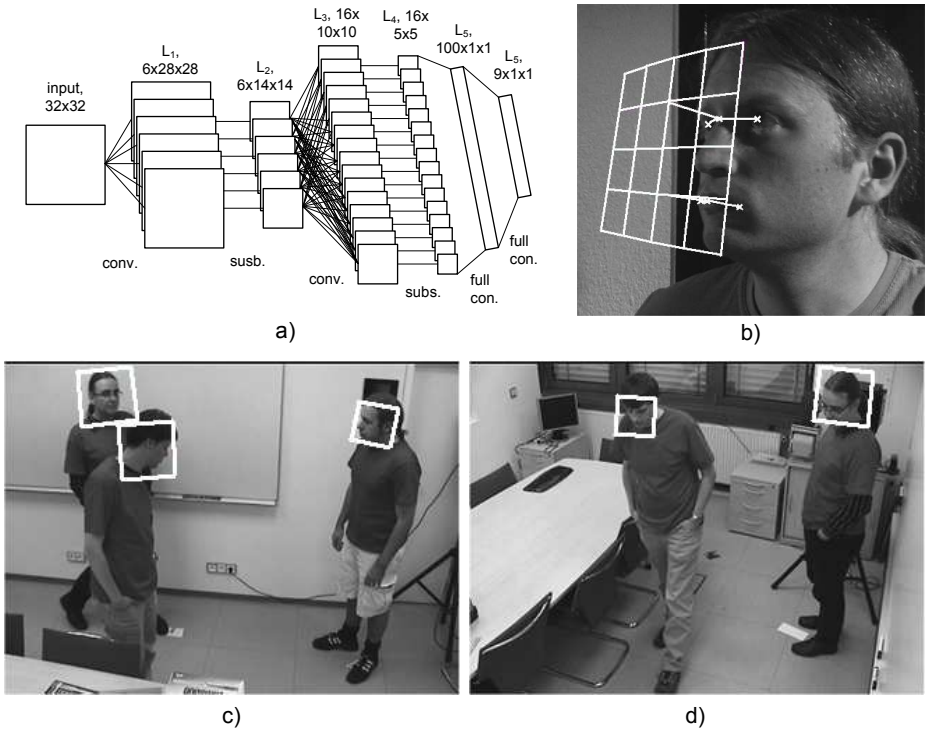

Fig. 6. Face detection system. a) CNN structure. b) annotation example. c) and d) indoor-environment for testing

dard data-sets [7, 5, 6]. We get the following average detection rates : 81% [7], 75% [5], and 83% [6] (with an average of 8 false positives). Note that we did not try to maximize detection rates for these data-sets since the applicability of convolutional neural networks for face detection was already sufficiently shown in [3].

To get an insight of the feasible acceleration by using a graphics card, we compared the parallel method with a corresponding single-CPU implementation (without a specific processor optimization). The GPU (Nvidia GeForce 8800 GT) comes with 14 Multiprocessors each composed of 8 processors with a clock rate about 600 MHz, while the CPU (Intel Pentium 4) comes with a clock rate about 3,4 GHz. Hence, the expected speed gain by full parallelization is about a factor of 19,76. In practice the gain is smaller, because of above mentioned latencies and additional overhead (e.g. loading data into the memory banks).

Table 1 shows the runtimes for our implementation. We tested three different image sizes and according to Fig. 2(a) we measured the runtime after each scaling step (up to eight). The given values specify the averagely elapsed milliseconds per frame. As we can see the actual speed gain is depending on the image size and number of scaling steps about a factor of ca. 11 up to 13.

		0	1	2	3	4	5	6	7	8
800x600	GPU	318	488	581	619	647	672	695	715	733
	CPU	4123	6165	7047	7554	7761	7858	7878	7885	7888
640x480	GPU	209	312	373	407	434	456	477	497	-
	CPU	2637	3933	4490	4811	4915	4959	4996	5044	-
378x278	GPU	73	109	137	162	184	203	222	-	-
	CPU	851	1276	1470	1541	1557	1575	1594	-	-

Table 1. runtime measurements for CPU and GPU (average milliseconds per frame).

6 Conclusions

We presented an parallelized implementation of convolutional neural networks for the task of face detection and pose estimation. The proposed high performance implementation showed a dramatic speedup compared to a conventional CPU based implementation. Given reasonable image sizes and image scaling steps we can expect speed gains about a factor of 11-13. Note that these speed gains are very likely to increase with the next generations of GPUs (since we effectively used an older generation of graphics cards, we expect further speedup using the currently available Nvidia GTX 280 cards).

References

1. Chellapilla, K., Pur, S., Simard, P.: High performance convolutional neural networks for document processing. In: Tenth International Workshop on Frontiers in Handwriting Recognition (2006)
2. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
3. Osadchy, M., LeCun, Y., Miller, M.: Synergistic face detection and pose estimation with energy-based models. Journal of Machine Learning Research 8(2007) **8**, 1197–1215 (2007)
4. Osadchy, R., Miller, M., LeCun, Y.: Synergistic face detection and pose estimation with energy-based models. In: Advances in Neural Information Processing Systems (NIPS 2004), pp. 1197–1215. MIT Press (2005)
5. Rowley, H., Baluja, S., Kanade, T.: Rotation invariant neural network-based face detection. Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on pp. 963–963 (1998)
6. Schneiderman, H., Kanade, T.: A statistical method for 3d object detection applied to faces and cars. IEEE Conference on Computer Vision and Pattern Recognition, 2000. Proceedings. **1**, 746–751 (2000)
7. Sung, K.K., Poggio, T.: Example-based learning for view-based human face detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on **20**(1), 39–51 (1998)
8. Vaillant, R., Monrocq, C., LeCun, Y.: An original approach for the localisation of objects in images. In: International Conference on Artificial Neural Networks, pp. 26–30 (1993)