Shared Map Convolutional Neural Networks for Real-Time Mobile Image Recognition

William Raveane and María Angélica González Arrieta

Universidad de Salamanca, Salamanca, Spain

Abstract. We present a technique for improving the speed of a convolutional neural network applied to large input images through the optimization of the sliding window approach. Meaningful performance gains and memory bandwidth reduction can be obtained by processing images in this manner, factors which play a crucial role in the deployment of deep neural networks within mobile devices.

1 Introduction

The Convolutional Neural Network (CNN) [1] has become a general solution for image recognition with variable input data. CNNs consist of two stages – one for automated feature learning, and another for classification – both of which can be successfully trained in tandem through gradient descent of the error surface [9]. Its results have consistently outclassed other machine learning approaches in large scale image recognition tasks [6], outperforming even human inspection of extensive datasets [2].

Compared to other feature-based computer vision methods such as SIFT [8] or HOG [4], CNNs are much more robust and tolerant to shape and visual variations of the images or objects intended to be recognized. However, contrary to such methods, an execution of a CNN will only recognize features on a single image block of size equal to the input dimensions of the network. As CNNs are usually trained with small image patches, this recognition area is likewise small. As a result, to run image recognition over a larger image size, it is necessary to repeatedly apply the same network over multiple regions. This is a very common technique named sliding windows, albeit a time consuming one as the execution time naturally grows in proportion to the number of sampled blocks.

With the increasing use of embedded hardware, it has naturally become a priority to endow mobile devices with computer vision capabilities. The use of CNNs, when applied through a sliding window methodology, allows a large range of important image recognition tasks to be carried out, many of which would have a great impact on the everyday usage of mobile hardware by end users. Some examples of this are text recognition [10] for visual language translators, human action [5] and face [7] recognition for greater user interactivity with social applications, or even traffic sign recognition [3] for embedded automotive applications. The unique task of logo recognition is taken as a sample usage of mobile implemented CNNs in this work, something which would have large

S. Omatu et al. (eds.), *Distributed Computing and Artificial Intelligence, 11th International Conference,* Advances in Intelligent Systems and Computing 290,
DOI: 10.1007/978-3-319-07593-8_56,

opportunities for commercial applications to increase company brand loyalty, perception and awareness among consumers, depending on the context it is used in. However, the same methods and network architecture described here would be equally applicable to solving similar problems, such as those described above.

Due to the high computational requirements of a CNN, the need for mobile computer vision has traditionally been met by outsourcing image analysis to a remote server in communication with the device over an internet connection. This approach, while effective, introduces large delays and is hardly an appropriate solution when user interactivity and real-time responsiveness are paramount. As embedded hardware capacity continues to grow with each new generation of low energy processors, this trend has gradually shifted towards implementing image recognition algorithms on the device itself with all computations carried out locally. Regardless, these devices continue to display performance limitations, as well as having intrinsic architecture constraints which result in slow memory access. It is therefore important to find new possible optimizations, so as to better utilize the computational power of the device.

We introduce one such improvement applicable to CNNs in particular. Our process contrasts with the traditional sliding window method where overlapping patches of a large image are sequentially analyzed by the network over individual executions. Instead, we apply per-layer computations over the entire image space, thereby producing a continuous flow of information from input to output in a single execution of the CNN. This results in a substantial boost in the throughput of the algorithm, especially when executed in an embedded mobile environment. Although applicable to most other platforms, we discuss the deployment of this algorithm on a mobile device as its memory access architecture makes it particularly sensitive to the improvements of the proposed algorithm.

This paper addresses these issues in particular. In Section 2, some background work is reviewed detailing the functionality of CNNs and sliding windows in general. We then introduce in Section 3 an optimized approach for the techniques discussed herein, including the architecture constraints that must be made to implement the proposed system. Finally, Section 4 concludes with the results obtained from the optimized method when compared to the traditional sequential algorithm and visits possible future enhancements to the presented work.

2 Background

The network on which our system is based upon is a standard CNN composed of alternating convolutional and max-pooling layers for the feature extraction stage, and one or more linear layers for the final classification stage. Fig. 1 depicts the layer structure of such a network, and it is the reference architecture used here to describe the concepts of the framework presented.

The first layer in the network consists of one or more neurons containing the image data to be analyzed, usually composed of the three color channels of the incoming image.

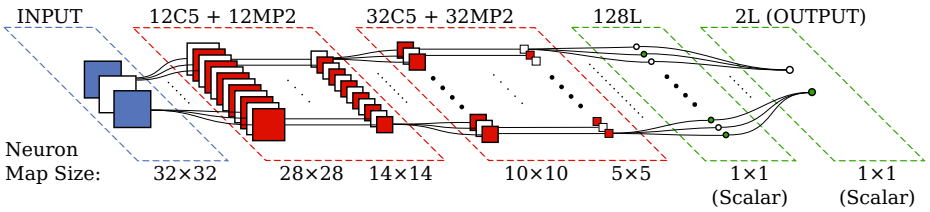

Fig. 1. A typical convolutional neural network architecture, with three input neurons for each color channel of an analyzed image patch, two feature extraction stages of convolutional and max-pooling layers, and two linear layers to produce a final one-vs-all classification output

The notation $N_j X K_j$ is used to describe the following layers, where N_j is the neuron map count of layer j, $X \in \{C, MP, L\}$ denotes the layer type, and K_j is the primary parameter value for the neurons in that layer.

The first part of every feature extractor stage is a convolutional layer. Each neuron linearly combines the convolution of one or more maps from the preceding layer. The map of a neuron in this layer will be slightly smaller than the incoming maps by an amount referred to as the kernel padding. This padding arises from the boundary conditions of the convolution algorithm and is defined as ${}^{K_j}/_2 - 1$, where K_j is the size of the convolutional kernels of layer j. Therefore, the layer's map size will be given by $M_j = M_{j-1} - {}^{K_j}/_2 - 1$, where M_{j-1} is the the preceding layer's map size.

Every convolutional layer is paired to a max-pooling layer which primarily reduces the dimensionality of the data. A neuron in this layer acts on a single map from a corresponding convolutional neuron in the previous layer. Its task is simply to pool as many adjacent values in the map as stated by the pool size parameter, to then determine the maximum value among them, and finally to pass this value as a subsample of the pooled region. The result is a map size that is inversely proportional to said parameter as given by $M_j = {}^{M_{j-1}}/_{P_j}$, where P_j is the pooling size factor of this layer.

The linear layers classify the final feature maps extracted on the previous layers by means of a linear combination operation identical to that of a regular multi layer perceptron, working with single pixel maps for both their input and output, such that $M_j = 1$ at every layer.

Ultimately, the output of the final classification layer decides the best matching class describing the input image, according to preselected training targets. Fig. 2 shows the information flow leading to this classification for a given image patch, where the CNN has been trained to identify a particular brand logo. The output of this execution is composed of two scalar values, each one representing the likelihood that the analyzed input image belongs to that neuron's corresponding class. In this case the logo has been successfully recognized as is dictated by the higher valued output neuron for the class "Logo".

Image recognition of images larger than the input size of a CNN is implemented by the sliding window approach, and its performance is intrinsically

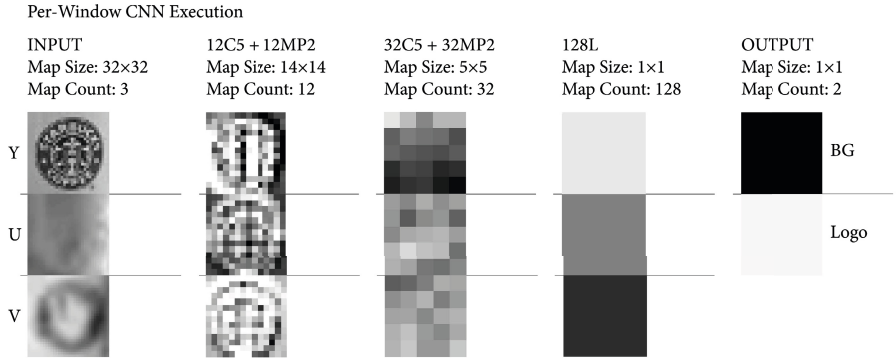

Fig. 2. A visualization of the data flow through the network, showing the first three neuron maps of each stage of the CNN. Note the data size reduction induced at each stage.

dependent on the details of this method. This algorithm is defined by two quantities, the window size S, usually fixed to match the network's designed input size; and the window stride T, which specifies the distance at which consecutive windows are spaced apart. The stride distance, therefore, inherently establishes the total number of windows analyzed for a given input. Therefore, it is important to choose the stride wisely, as this distance is inversely proportional to the resolution of the classifier, but also to the computing power invested in analyzing the number of resulting windows, W. For an input image of size $I_w \times I_h$, the total window count is given by:

$$W = \left(\frac{I_w - S}{T} + 1\right)\left(\frac{I_h - S}{T} + 1\right) \quad \Longrightarrow \quad W \propto \frac{I_w I_h}{T^2}$$

Figure 3 depicts the operation of this method applied on an input image downsampled to 144×92, extracting individual windows with $S = 32$ for the simplified case where $T = {}^{S}/_{2}$. A network analyzing this image would therefore require 40 executions to fully analyze the extracted window. The computational requirement is further compounded when a smaller stride is selected – an action necessary to improve the resolving power of the classifier. At $T = {}^{S}/_{8}$, for example, 464 separate CNN executions would be required.

Fig. 3. An overview of the sliding window method, where an input image is subdivided into smaller overlapping image patches, each being individually analyzed by a CNN

3 Optimized Network Execution

The method we propose introduces a framework wherein the chosen stride has no significant impact at all on the execution time of the feature extraction stages of a CNN – the most computationally demanding section of the network – as long as the selected stride is among a constrained set of possible values. This is made possible by two considerations. (i) Allowing each layer of the CNN to process the entire image at once as a single shared map instead of individually processing single windows. (ii) Guiding the image data through the network so as to perfectly align individual pixels over computational columns spanning the various network layers, by virtue of the imposed stride constraints.

By their nature, convolutional neural networks are designed with built-in positional tolerance. This is in part achieved by the reuse of the same convolutional kernel over the whole neuron map. Similarly, a max-pooling neuron performs the same singular algorithmic action at any point within its map. As a result of this behavior, the output of these layers is independent of the pixel offset within the map, such that overlapping windows will share the same convolved values. This is demonstrated in Fig. 4.

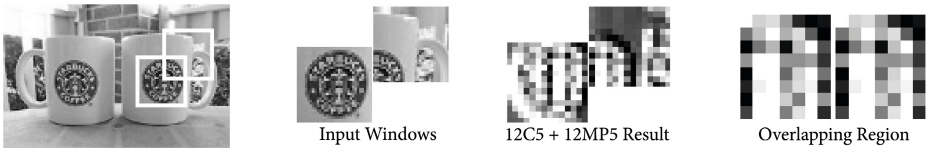

Fig. 4. Two adjacent windows extracted from an input image, passed through the 12C5 + 12MP5 feature extractor. A detailed view of the convolved maps in the overlapping top-right and bottom-left quarters of each window show that these areas match exactly.

This leads to the possibility of streamlining the feature extractors by running their algorithms over the full input image at once. Hence, each C + MP neuron will output a single map shared among all windows, where subdivisions of this map would normally match the outputs of the corresponding windows, had they been executed separately as in the traditional method. This greatly reduces the expense of re-calculating convolutions on overlapping regions of each window.

Figure 5 shows an overview of the shared map process, which passes the input image in its entirety through each stage of the network. By doing this, the output layer now produces a continuous and localized class distribution over the image space, a result which contrasts greatly to that of a single classification value as was previously seen in Fig. 2. The output of this execution consists of image maps where each pixel yields the relative position of all simultaneously classified windows. Similar to the per-window execution method, the intensity value of a pixel in the output map represents the classification likelihood of the corresponding window. Note how the relative position of both logos in the input image has been discovered after only one shared map execution of the network.

An account of the window size and stride is also displayed, illustrating how it evolves after each layer, while the total window count remains the same. Here, the correspondence of each 32×32 window in the input image can be traced to each 1×1 pixel in the output maps.

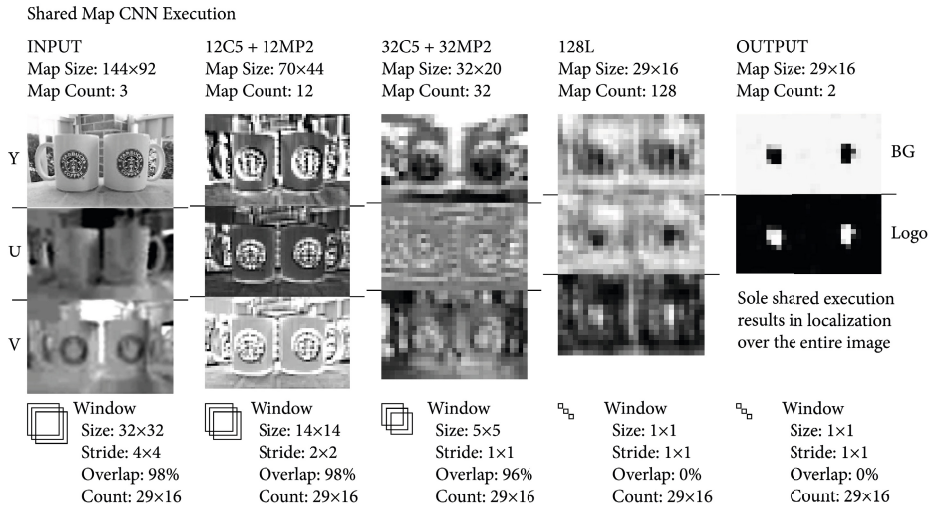

Fig. 5. The shared map execution method for a convolutional neural network, where each layer processes an entire image in a single pass, where each neuron is now able to process maps with dimensions that far exceed the layer's designed input size.

The operation of the shared map process relies heavily on the details of the dimensionality reduction occurring at each layer within the network. For this reason, it is necessary to lay certain constraints that must be enforced when choosing the optimum sliding window stride. The window size and stride at each layer is affected by the parameters of the layer in a well-defined manner:

$$S_j = \begin{cases} S_{j-1} - K_j - 1 & if\ j \in Convolutional\ Layers \\ S_{j-1}/P_j & if\ j \in Max\ Pooling\ Layers \\ S_{j-1} & ifj \in Linear\ Layers \end{cases}$$

$$T_j = \begin{cases} T_{j-1} & if\ j \in Convolutional\ Layers \\ T_{j-1}/P_j & if\ j \in Max\ Pooling\ Layers \\ T_{j-1} & if\ j \in Linear\ Layers \end{cases}$$

Where the size S_j and stride T_j of a window at layer j depends on the parameters of the layer and the size and stride values at the preceding $j-1$ layer.

This equation set can be applied over the total number of layers L of the network, while keeping as the target constraint that the final size and stride must remain whole integer values. By regressing these calculations back to the input layer $j = 0$, one can find that the single constraint at that layer is given by:

$$T_0 \quad | \quad \prod_{j=1}^{j=L} \begin{cases} P_j & if\ j \in Max\ Pooling\ Layers \\ 1 & otherwise \end{cases}$$

In other words, the input window stride must be divisible by the product of the pooling size of all max-pooling layers in the network.

Choosing the initial window stride in this manner, will ensure that every pixel in the final output map corresponds to exactly one input window.

4 Results and Conclusions

Table 1 gives a summary of the results of this technique. The tests were carried out on a mobile device equipped with a quad core 1.3 GHz Cortex-A9 CPU and a 12-core Tegra 3 GPU, running a parallel optimized GPU implementation of the CNN architecture in Fig. 1 over a large 512×512 input image.

Table 1. Results of tests with several input layer stride T_0 configurations, from the closest packed 4×4 to the non-overlapping 32×32 layouts. A window count W and the overlap coverage O is shown for each window stride selection. An average over 20 test runs for each of these configurations was taken as the execution time for each of the methods described herein – the traditional per-window execution method, and our shared map technique.

$\mathbf{T_0}$	**W**	**O**	**Execution Time (ms)**		**Speedup**
			Per-Window	Shared Map	
4×4	14,400	98.4%	115,212	4,087	28.1
8×8	3,600	93.8%	28,847	1,548	18.6
12×12	1,600	85.9%	12,789	1,025	12.5
16×16	900	75.0%	7,234	843	8.5
20×20	576	60.9%	4,608	757	6.1
24×24	400	43.8%	3,189	686	4.7
28×28	289	23.4%	2,312	621	3.7
32×32	225	0.00%	1,801	599	3.0

It is of great interest to note the final 32×32 configuration. Regardless of the fact that there is no overlap at this stride, a 3.0 speedup is observed over running the windows individually. This is due to our method introducing inherent memory bandwidth reductions through its pipelined execution approach, where

the entire image needs to be loaded once per execution. This contrasts the traditional approach where loading separate windows into memory at different times requires each to be individually sliced from the original memory block – a very expensive operation in the limited memory throughput of mobile devices.

Acknowledgements. This work has been carried out by the project Sociedades Humano-Agente: Inmersión, Adaptación y Simulación. TIN2012-36586-C03-03. Ministerio de Economa y Competitividad (Spain). Project co-financed with FEDER funds.

References

1. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J.: Flexible, High Performance Convolutional Neural Networks for Image Classication. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, pp. 1237–1242 (2011)
2. Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Convolutional Neural Network Committees For Handwritten Character Classification. In: 11th International Conference on Document Analysis and Recognition, ICDAR (2011)
3. Ciresan, D.C., Meier, U., Masci, J., Schmidhuber, J.: Multi-Column Deep Neural Network for Traffic Sign Classification. Neural Networks (32), 333–338 (2012)
4. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005, vol. 1, pp. 886–893 (June 2005)
5. Ji, S., Xu, W., Yang, M., Yu, K.: 3D Convolutional Neural Networks for Human Action Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 35, 221–231 (2013)
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks, vol. 25, pp. 1106–1114 (2012)
7. Lawrence, S., Giles, C., Tsoi, A.C., Back, A.: Face recognition: a convolutional neural-network approach. IEEE Transactions on Neural Networks 8(1), 98–113 (1997)
8. Lazebnik, S., Schmid, C., Ponce, J.: Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2006, vol. 2, pp. 2169–2178 (2006)
9. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE (November 1998)
10. Wang, T., Wu, D.J., Coates, A., Ng, A.Y.: End-to-end text recognition with convolutional neural networks. In: ICPR, pp. 3304–3308. IEEE