Hybrid System for Mobile Image Recognition through Convolutional Neural Networks and Discrete Graphical Models

William Raveane and María Angélica González Arrieta

Universidad de Salamanca, Salamanca, Spain

Abstract. A system is presented which combines deep neural networks with discrete inference techniques for the successful recognition of an image. The system presented builds upon the classical sliding window method but applied in parallel over an entire input image. The result is discretized by treating each classified window as a node in a markov random field and applying a minimization of its associated energy levels. Two important benefits are observed with this system: a gain in performance by virtue of the system's parallel nature, and an improvement in the localization precision due to the inherent connectivity between classified windows.

Keywords: computer vision, deep neural networks, graphical models.

1 Introduction

Hybrid intelligent systems have consistently shown benefits that outperform those of their individual components in many tasks, especially when used along neural computing [1]. In recent years, two main areas of computer vision have gained considerable strength and support: On one side, soft computing techniques based on non-exact but very accurate machine learning models like neural networks, which have been successful for high level image classification [7]. Contrasting these systems, computer vision techniques modeled by graphical models have enjoyed great reception when performing low level image processing tasks such as image completion [6]. In this paper, we combine both of these techniques to successfully classify and localize a region of interest within an input image.

We use Convolutional Neural Networks (CNN) [3] for the classification of image content. CNNs have become a general solution for image recognition with variable input data, as their results have outclassed other machine learning approaches in large scale image recognition tasks [4]. Paired to this CNN classifier, we use energy minimization of a Markov Random Field (MRF) [8] for inference and localization of the target within the image space. Graphical models such as this have been implemented in areas of computer vision where the relationship between neighboring regions plays a crucial role [2].

We review the implementation of this system specifically within a mobile device. With the increasing use of mobile hardware, it has become a priority to provide these devices with computer vision capabilities. Due to the high computational requirements, this need has mostly been met by outsourcing the analysis to a remote server over the internet. This approach introduces large delays and is hardly appropriate when interactivity and responsiveness are paramount. Embedded environments have intrinsic architecture constraints which require algorithms to make the best use of the available computing capacity. The proposed system exploits this specific platform by reducing the overall required memory throughput via a parallel execution approach. This is achieved by applying layer computations over the entire image space, as opposed to running smaller patches individually, as is common with the sliding window approach normally used in this type of image classification.

2 Background

The network on which our system is based upon is a standard CNN. Figure 1 depicts the layer structure of such a network, and it is the reference architecture used throughout this paper to describe the concepts of the framework presented.

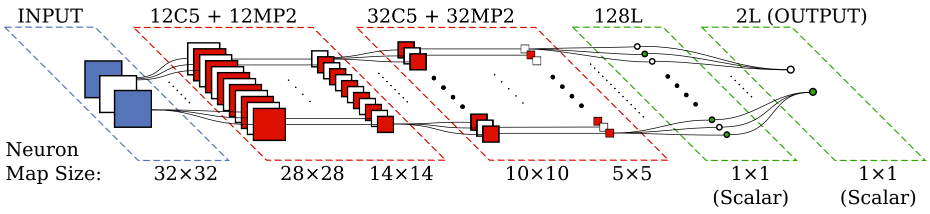


Fig. 1. A typical convolutional neural network architecture, with three input neurons for each color channel of an analyzed image patch, two feature extraction stages of convolutional and max-pooling layers, and two linear layers to produce a final one-vs-all classification output

In the initial stages of the CNN, a neuron consists of a two-dimensional grid of independent computing units, each producing an output value. As a result, every neuron will itself output a grid of numerical values, a data structure in $\mathbb{R}^2$ referred to as a map. When applying CNNs to image analysis, these maps represent an internal state of the image after being processed through a connective path leading to that particular neuron. Consequently, maps will usually bear a direct positonal and feature-wise relationship to the input image space. As data progresses through the network, however, this represenation turns more abstract as the dimentionality is reduced. Eventually, these maps are passed through one or more linear classifiers, layers consisting of traditional single unit neurons which output a single value each. For consistency, the outputs of these neurons are treated as 1×1 single pixel image maps, although they are nothing more than scalar values in $\mathbb{R}^0$.

2.1 CNN Layer Types

The first layer in the network consists of the image data to be analyzed, usually composed as the three color channels. The notation $N_j X K_j$ is used to describe all subsequent layers, where N_j is the neuron map count of layer j, $X \in \{\mathcal{C}, \mathcal{MP}, \mathcal{L}\}$ denotes the layer type group (Convolutional, Max-Pooling, and Linear), and K_j is the parameter value for that layer.

The first part of every $\mathcal{C} \rightarrow \mathcal{MP}$ feature extraction stage is a convolutional layer. Here, each neuron linearly combines the convolution of one or more preceding maps. The result is a map slightly smaller than the input size by an amount known as the kernel padding, which arises from the boundary conditions of the valid convolution algorithm. It is defined as ${}^{K_j}/_2 - 1$, where K_j is convolutional kernels size 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.

A max-pooling neuron acts on a single map from a preceding convolutional neuron, and its task is to subsample a pooled region of size K_j. The result is a map size that is inversely proportional to said parameter by $M_j = {}^{M_{j-1}}/_{K_j}$.

Linear layers classify feature maps extracted on preceding layers through a linear combination as in a perceptron – always working with scalar values – such that $M_j = 1$ at every layer of this type.

Finally, the output of the final classification layer decides the best matching label describing the input image. 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 company logo.

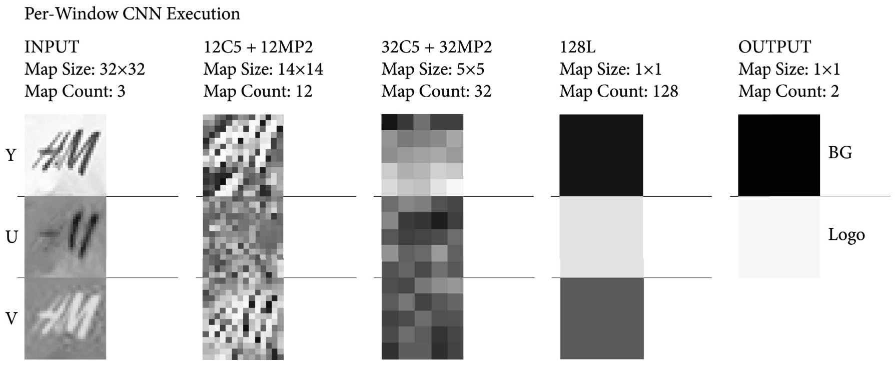


Fig. 2. Visualization of the first three neuron maps at each stage of the CNN. Note the data size reduction induced at each stage. The output of this execution consists of two scalar values, each one representing the likelihood that the analyzed input image belongs to that neuron's corresponding class. In this case the logo has been successfully recognized by the higher valued output neuron for class "Logo".

2.2 The Sliding Window Method

Recognition of images larger than the CNN input size is achieved by the sliding window approach. This algorithm is defined by two quantities, the window size S, usually fixed to match the CNN's designed input size; and the window stride T, which specifies the distance at which consecutive windows are spaced apart. This stride distance establishes the total number of windows analyzed W for a given input image. For an image of size $I_w \times I_h$, the window count is given by:

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

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

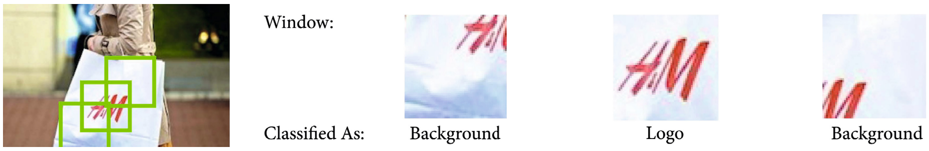


Fig. 3. An overview of the sliding window method, where an input image is subdivided into smaller overlapping image patches, each being individually analyzed by a CNN. A classification result is then obtained for each individual window.

3 Optimized Network Execution

The method proposed introduces a framework where the stride has no significant impact on the execution time of the $\mathcal{C} \rightarrow \mathcal{MP}$ stages, as long as the selected stride is among a constrained set of possible values. This is achieved by allowing layers to process the full image as a single shared map instead of individual windows. Constraints in the possible stride values will result in pixel calculations to be correctly aligned throughout the layers.

3.1 Shared Window Maps

CNNs have a built-in positional tolerance due to the reuse of the same convolutional kernels over the entire neuron map. As a result of this behavior, their output is independent of any pixel offset within the map, such that overlapping windows will share convolved values. This is demonstrated in Fig. 4.

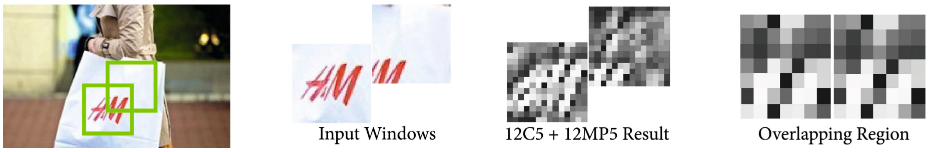


Fig. 4. Two adjacent windows extracted from an input image, passed through the 12C5 + 12MP5 feature extractor. A detailed view of the convolved maps in the overlapping top-right and bottom-left quarters of each window shows that these areas fully match.

This leads to the possibility of streamlining the feature extractors by running their algorithms over the full input image at once. Hence, each $\mathcal{C} \rightarrow \mathcal{MP}$ neuron will output a single map shared among all windows. This greatly reduces the expense of calculating again 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.

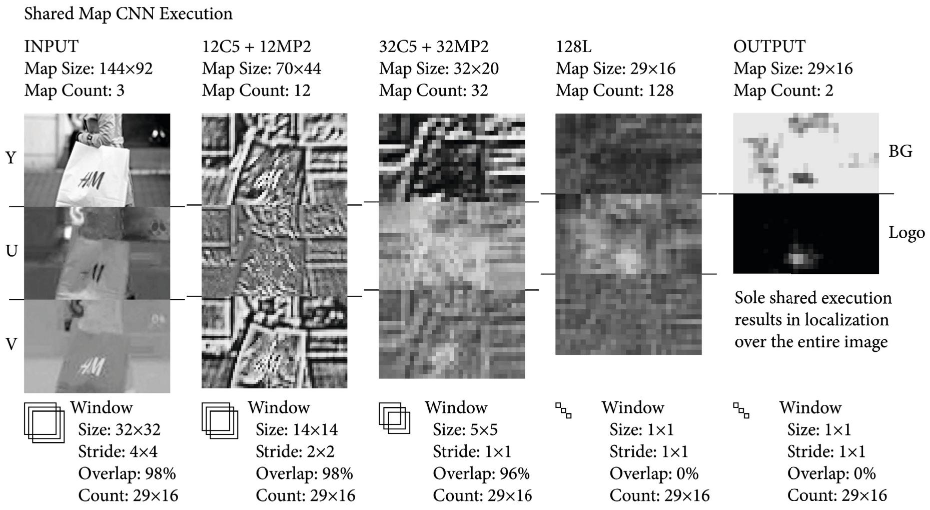

Shared Map CNN Execution
INPUT
Map Size: 144×92
Map Count: 3
12C5 + 12MP2
Map Size: 70×44
Map Count: 12
32C5 + 32MP2
Map Size: 32×20
Map Count: 32
128L
Map Size: 29×16
Map Count: 128
OUTPUT
Map Size: 29×16
Map Count: 2
Y
U
V
BG
Logo
Sole shared execution results in localization over the entire image
Window Size: 32×32 Stride: 4×4 Overlap: 98% Count: 29×16
Window Size: 14×14 Stride: 2×2 Overlap: 98% Count: 29×16
Window Size: 5×5 Stride: 1×1 Overlap: 96% Count: 29×16
Window Size: 1×1 Stride: 1×1 Overlap: 0% Count: 29×16
Window Size: 1×1 Stride: 1×1 Overlap: 0% Count: 29×16

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

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 the logo 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 one of the pixels in the output maps.

3.2 Window Configuration

The operation of the shared map process relies greatly 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.

At each layer, the window size and stride are reduced until they eventually become single pixel values at the final linear layers. The amount of reduction at each stage varies according to the type of the layer and its parameters. All of these quantities can be found in a well defined manner as given by:

$$S_j = \begin{cases} S_{j-1} - K_j - 1 & if \;\; j \in \mathcal{C} \\ S_{j-1}/K_j & if \;\; j \in \mathcal{MP} \\ S_{j-1} & if \;\; j \in \mathcal{L} \end{cases} \tag{2}$$

$$T_j = \begin{cases} T_{j-1} & if \;\; j \in \mathcal{C} \cup \mathcal{L} \\ T_{j-1}/K_j & if \;\; j \in \mathcal{MP} \end{cases} \tag{3}$$

Where the window size S_j and its stride T_j at layer j depends on the various parameters K_j of the layer and the window size and stride values at the preceding $j-1$ layer. This equation set can be applied over the total number of layers 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 remaining constraint at that layer is given by:

$$T_0 \equiv 0 \bmod \prod_{j \,\in\, \mathcal{MP}} K_j \tag{4}$$

In other words, the input window stride must be perfectly 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 is correctly aligned thoughout all shared maps and corresponds to exactly one input window. Fig. 6 follows the evolution of the window image data along the various layers of the sample network architecture, showing this pixel alignment throughout the CNN.

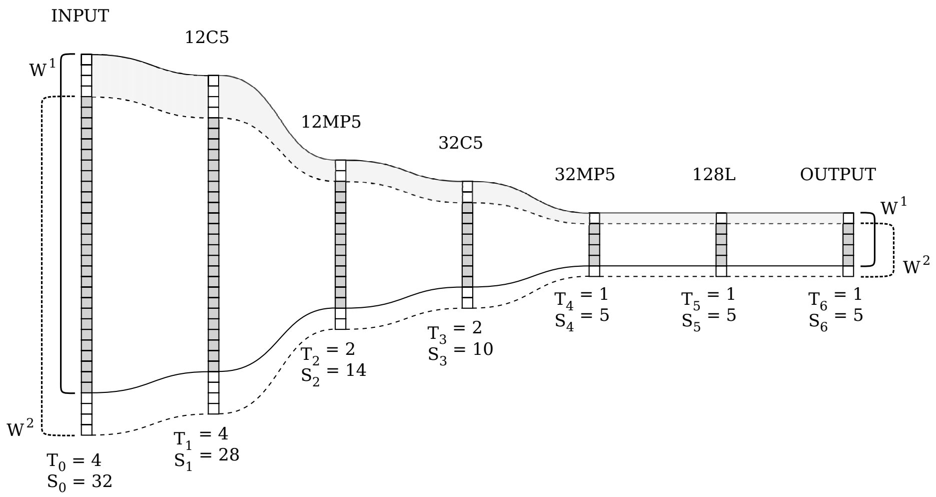


Fig. 6. The CNN layers and their effect on the window pixel space, illustrated in one dimension for simplicity. Two successive 32×32 windows W^1 and W^2 are shown. Overlapping pixels at each layer are shaded. Starting with an input layer window stride $T_0 = 4$, the final output layer results in a packed $T_6 = 1$ window stride, so that each output map pixel corresponds to a positional shift of 4 pixels in the input windows, a relationship depicted by the darkened column path traversing all layers.

4 Discrete Inference of CNN Output

The common practice to obtain a final classification from an output value set as seen in Fig. 5 is to identify which class has a higher output value from the CNN at each each window (here, each pixel in the output map). While efficient, results from this procedure are not always ideal because they only take into account each window separately.

Furthermore, maximum value inference is prone to false positives over the full image area. Due to their non-exact nature, neural network accuracy can decrease by finding patterns in random stimuli which eventually trigger neurons in the final classification layer. However, such occurrences tend to appear in isolation around other successfully classified image regions. It is therefore possible to improve the performance of the classifier by taking into account nearby windows.

There exist many statistical approaches in which this can be implemented, such as (i) influencing the value of each window by a weighted average of neighboring windows, or (ii) boosting output values by the presence of similarly classified windows in the surrounding area. However, we propose discrete energy minimization through belief propagation as a more general method to determine the final classification within a set of CNN output maps. The main reason being that graphical models are more flexible in adapting to image conditions and can usually converge on a globally optimal solution.

4.1 Pairwise Markov Random Field Model

Images can be treated as an undirected cyclical graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where nodes $n_i \in \mathcal{N}$ represent an entity such as a pixel in the image, and graph edges $e_{ij} \in \mathcal{E}$ represent the relationship between these nodes. If, for simplicity, 4-connectivity is used to represent the relationship between successive nodes in a graph; then each node will be connected to four others corresponding to its neighbors above, below, and to each side of the current element.

The output space of the convolutional neural network can therefore be represented in this manner through a graph. However, instead of describing pixel intensity values, each node in the graph represents the classification state of the corresponding window. This state takes on a discrete value among a set of class labels $c \in \mathcal{C} \equiv \{BG, Logo\}$ corresponding to the classification targets of the CNN. Thus, each node in the graph can take on one of several discrete values, expressing the predicted class of the window that the node represents. Fig. 7 (Left) displays the structure of such a graph.

It can be seen that if nodes represent classification outcomes, there is a strong relationship between them. The reason is that continuity throughout a map tends to be preserved over neighboring regions due to strong local correlation in in input images. This inflicts a Markovian property in the graph nodes where there is a dependency between successive nodes. Therefore, this graph follows the same structure as an MRF, and any operations available to this kind of structure will be likewise applicable to the output map.

4.2 Energy Allocation

To implement energy minimization on an MRF, it is necessary to assign energy potentials to each node and edge. These energies are usually adapted from observed variables, and in this case, they correspond to the values of the output maps and combinations thereof. Therefore, MRF optimization over a graph $\mathcal{G}$ can be carried out by minimizing its Markov random energy E, given by:

$$E(\mathcal{G}) = E(\mathcal{N}, \mathcal{E}) = \sum_{n_i \in \mathcal{N}} \Theta_i(n_i) + \sum_{e_{ij} \in \mathcal{E}} \Theta_{ij}(e_{ij}) \tag{5}$$

Here, $\Theta_i(\cdot)$ corresponds to the singleton energy potential of node n_i, and $\Theta_{ij}(\cdot)$ is a pairwise potential between nodes n_i and n_j. Starting from the CNN output map observations, the singleton potentials can be assigned as:

$$\Theta_i = \begin{bmatrix} \omega_i^0 \\ \omega_i^1 \\ \vdots \\ \omega_i^C \end{bmatrix} \tag{6}$$

$$\omega_i^a = \sum_{c \in \mathcal{C}} \begin{cases} 1 - (O_i^c)^2 & if\ a = c \\ (O_i^c)^2 & otherwise \end{cases} \tag{7}$$

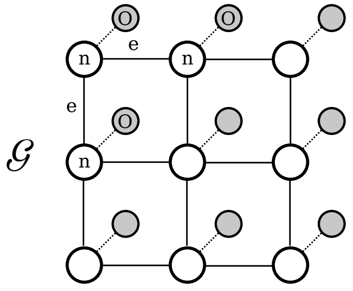
Where C is the total number of classes in set $\mathcal{C}$ (2 in the sample CNN architecture), and $O\,{}_{i}^{c}$ is the observed CNN value for window $n_i \in \mathcal{N}$ and class $c \in \mathcal{C}$. In this manner, each ω_i^a value is an MSE-like metric that measures how far off from ideal training target values did the CNN classify window n_i as. Thus, a lower potential value will be assigned to the most likely class, while a higher potential value will be given to other possible classes at this node.

Pairwise potentials can be defined as:

$$\Theta_{ij} = \begin{bmatrix} \delta\,{}_{ij}^{00} & \delta\,{}_{ij}^{01} \dots & \delta\,{}_{ij}^{0C} \\ \delta\,{}_{ij}^{10} & \delta\,{}_{ij}^{11} & \delta\,{}_{ij}^{1C} \\ \vdots & \ddots & \\ \delta\,{}_{ij}^{C0} & \delta\,{}_{ij}^{C1} & \delta\,{}_{ij}^{CC} \end{bmatrix} \tag{8}$$

$$\delta\,{}_{ij}^{ab} = \mid O\,{}_{i}^{a} - O\,{}_{j}^{b} \mid \tag{9}$$

Where each value δ_{ij}^{ab} is a straightforward distance metric that measures the *jump* in CNN output values when switching from class a to class b between windows n_i and n_j. Thus, these potentials will be small if the same class is assigned to both nodes, and large otherwise. Fig. 7 (Right) shows all energy assignments per node pair.



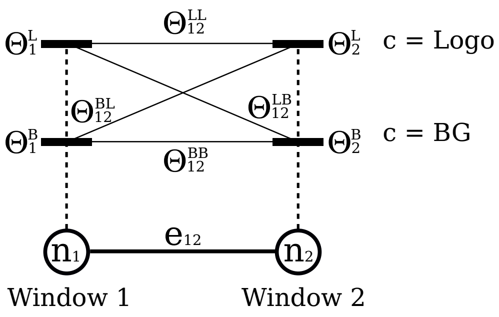


Fig. 7. Left: A subset of the MRF graph $\mathcal{G}$ formed by the CNN output space, where each node n_i represents the classification state of a corresponding window analyzed with the network, whose outputs are implemented into this system as the observed hidden variables O. Nodes have a 4-connectivity relationship with each other represented by the edges e_{ij} thus forming a grid-like cyclical graph. **Right:** A detail of the potential energies assigned to each of two nodes $\{n_1, n_2\}$ connected by edge e_{12}. The singleton potentials $\Theta\,{}_{i}^{a}$ correspond to the energy associated with node i if assigned to class a, and the pairwise potentials $\Theta\ ij^{ab}$ are the changes in energy that occur by assigning class a to node n_i and class b to node n_j.

Applying Belief Propagation [5] to find the lowest possible energy state of the graph will now yield an equilibrium of class assignments throughout the image output space.

5 Results

The test application is developed for the Android mobile OS as an OpenGL ES shader which makes use of the available computing capabilities of the device GPU. The main logic of the system is placed within a fragment shader running the CNN per-pixel over a Surface Texture memory object. The test device is equipped with a quad core 1.3 GHz Cortex-A9 CPU with a 12-core 520 MHz Tegra 3 GPU. This SoC architecture embeds 1 Gb of DDR2 RAM shared by both the CPU and GPU.

The test system executes the same CNN architecture described in Fig. 1, except for the classification layer having 32 output neurons corresponding to one background label and 31 different logo labels. This network is exectued over 8 simultaneous 144×92 images forming a multi-scale image pyramid. The energy minimization technique as described in Section 4 is then applied, but over a 3D graph formed with 6-connectivity between nodes such that each window is also aware of window classifications at the corresponding larger and smaller scale steps. Table 1 gives a summary of the results obtained from this setup.

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 total window count W at each pyramid level and over the full 8 level pyramid, as well as the window overlap coverage OC per input map is given for each of the stride selections. 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. A speedup factor is calculated showing the performance improvement of our method over the other.

$\mathbf{T_0}$	**W**		**OC**	**Execution Time (ms)**		**Speedup**
	Level	Pyramid		Per-Window	Shared Map	
4×4	464	3,712	98.4%	29,730	1,047	28.4x
8×8	112	896	93.8%	7,211	387	18.6x
12×12	60	480	85.9%	3,798	311	12.2x
16×16	32	256	75.0%	2,051	240	8.5x
20×20	24	192	60.9%	1,536	252	6.1x
24×24	15	120	43.8%	945	203	4.7x
28×28	15	120	23.4%	949	200	4.7x
32×32	8	64	0.00%	514	171	3.0x

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 still observed over running the windows individually. This is due to the inherent reduction in memory bandwidth through the system's pipelined execution approach, where the entire image needs to be loaded only 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.

The results of the inference system are more of a qualitative nature, as it is difficult to objectively establish a ground truth basis for such experiments. This system aims to localize classified windows, therefore it is subject to an interpretation of which windows cover enough of the recognition target to be counted as a true positive. Regardless, Table 2 gives an indicative comparison of the system against the competing techniques previousy described. Fig. 8 shows a visual comparison.

Table 2. Results of various inference algorithms for the final classification, describing the Accuracy ($^{TP+TN}/_{ALL}$), PPV ($^{TP}/_{TP+FP}$), and F_1 ($^{2TP}/_{2TP+FP+FN}$) metrics

Algorithm	**Accuracy**	**PPV**	**F$_1$**
Maximum Value	0.942	0.341	0.498
Weighted Average	0.964	0.391	0.430
Neighbor Boosting	0.972	0.489	0.591
Energy Minimization	**0.981**	**0.747**	**0.694**

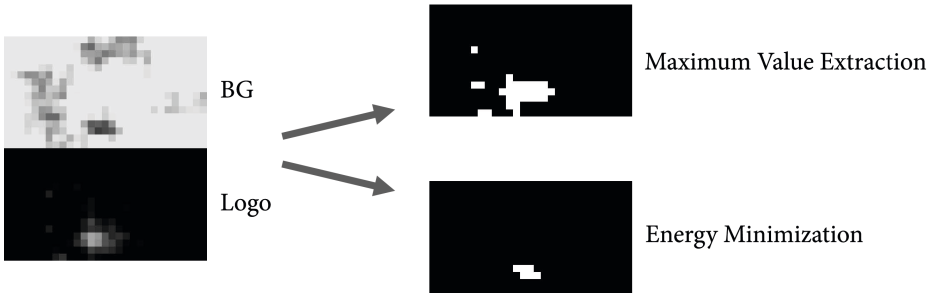


Fig. 8. Comparison of the final "Logo" classification and localization, applying the classical maximum value per class extraction vs. our proposed energy minimization inference method on the two CNN output maps introduced in Figure 5

6 Conclusions

A system for the optimization of convolutional neural networks has been presented for the particular application of mobile image recognition. Although a simple logo classification task was used here as a sample application, CNNs allow for many other image recognition tasks to be carried out. Most of these processes would have great impact on end users if implemented as real time mobile applications. Some examples where CNNs have been successfully used and their possible mobile implementations would be (i) text recognition for visually interactive language translators, (ii) human action recognition for increased user interactivity in social applications, or even (iii) traffic sign recognition for embedded automotive applications. Any of these applications could be similarly optimized and discretized by the system presented here.

In addition to the CNN classifier, the MRF model is very flexible as well and its implementation can be adjusted to domain-specific requirements as needed by each application. For example, a visual text recognizer might implement pair-wise energy potentials which are modeled with the probabilistic distribution of character bigrams or n-grams over a corpus of text, thereby increasing the overall text recognition accuracy.

Therefore, we believe this to be a general purpose mobile computer vision framework which can be deployed for many different uses within the restrictions imposed by embedded hardware, but also encouraging the limitless possibilities of mobile applications.

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. Borrajo, M.L., Baruque, B., Corchado, E., Bajo, J., Corchado, J.M.: Hybrid neural intelligent system to predict business failure in small-to-medium-size enterprises. International Journal of Neural Systems 21(04), 277–296 (2011)
2. Boykov, Y., Veksler, O.: Graph Cuts in Vision and Graphics: Theories and Applications. In: Paragios, N., Chen, Y., Faugeras, O. (eds.) Handbook of Mathematical Models in Computer Vision, pp. 79–96. Springer, US (2006)
3. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J.: Flexible, High Performance Convolutional Neural Networks for Image Classification. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, pp. 1237–1242 (2011)
4. 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)
5. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient Belief Propagation for Early Vision. International Journal of Computer Vision 70(1), 41–54 (2006)
6. Komodakis, N., Tziritas, G.: Image completion using efficient belief propagation via priority scheduling and dynamic pruning. IEEE Transactions onImage Processing 16(11), 2649–2661 (2007)
7. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Advances in Neural Information Processing Systems, vol. (25), pp. 1106–1114 (2012)
8. Wang, C., Paragios, N.: Markov Random Fields in Vision Perception: A Survey. Rapport de recherche RR-7945, INRIA (September 2012)