An Improved Convolutional Neural Network Architecture for Image Classification

A. Ferreyra-Ramirez, C. Aviles-Cruz, E. Rodriguez-Martinez(✉), J. Villegas-Cortez, and A. Zuñiga-Lopez

Departamento de Electrónica, Universidad Autónoma Metropolitana, Unidad Azcapotzalco, Av. San Pablo 180, Col. Reynosa-Tamaulipas, 02200 Mexico City, Mexico
{fra,caviles,erm,jvc,azl}@azc.uam.mx

Abstract. This manuscript presents the design and implementation of an improved convolutional neural network (CNN) for image classification which was carefully crafted to avoid overfitting. Contrary to most CNNs which apply normalization before pooling, our proposed architecture reverse the order of such tasks. The performance of the proposed architecture, named ACEnet, was evaluated using a hold-out method over five selected databases: Olivia, Paris, Oxford Buildings, Caltech-101, and Caltech-256. We present three main results: processing time, training performance and testing performance for each database. Also, we present a comparison versus the well-known Alexnet architecture, where our CNN proposal improves 5.11% the mean testing performance over the selected databases.

Keywords: Convolutional neural network · Image classification · Mini-batch size · Epochs number · Overfitting

1 Introduction

Nowdays, convolutional neural networks are a central topic in computer vision, specifically in applications such as video and image processing, where deep learning architectures have proven to outperform traditional image classification approaches. A CNN is a variation of a multilayer perceptron, but its performance makes it much more effective in machine vision tasks because each stage models one part of the visual cortex. Theoretical foundations of CNNs are based on the Neocognitron introduced by Fukushima in 1980 [13], improved by LeCun in 1998 [19] and fine-tuned by Ciresan in 2011 [11].

As classifiers, CNNs have an incredible generalization capability, which increases with the size of the training set, because the greater the amount of information available, the better the millions of parameters contained in its structure will be trained [3,12]. For databases with limited number of labeled images, it has been suggested that fine-tuning a trained architecture with a robust database is better than training the same architecture from scratch [10,14,21,27].

J. A. Carrasco-Ochoa et al. (Eds.): MCPR 2019, LNCS 11524, pp. 89–101, 2019.
https://doi.org/10.1007/978-3-030-21077-9_9

One of the key issues for a successful CNN is choosing the structure and parameters that represent image information accurately and uniquely. These structure is comprised of stages, where each stage is conformed by *convolution, rectification, pooling and normalization* tasks. When the CNN is used in classification problems, there is also a classification stage comprised of fully connected layers followed by an output layer, where the number of neurons is equal to the number of classes.

The configuration parameters in a CNN are divided into two categories: (a) those concerning the architecture –such as kernel size, stride and padding– and (b) those concerning the training algorithm –such as mini-batch size, regularization type, number of training epochs, learning-rate drop period, learning-rate drop factor, learning-rate schedule, and initial learning rate.

In this paper, we present the design and implementation of an improved convolutional neural network for image classification which was carefully crafted to avoid overfitting. The performance of the proposed architecture, which we named *ACEnet*, was compared against that of *Alexnet* [18], achieving a 5.11% increase in classification accuracy.

ACEnet consists of six feature extraction (FE) stages, instead of five as proposed in [17,18]. The additional stage is comprised of convolution, rectification and pooling tasks. Another important difference between Alexnet and our proposal is on the second FE stage, where in the sequence of tasks: *convolution, rectification, normalization and pooling*, proposed by *Alexnet*, the order of the last two task is inverted.

To compare the performance of both architectures, each was trained from scratch using five databases: Oliva and Torralba (Oliva) [4], Paris [6], Oxford [5], Caltech-101 [1] and Caltech-256 [2]. For each selected database we present processing time and classification accuracy for both training set and testing set.

The rest of the paper is organized as follows. Section 2 describes the state of the art. Section 3 presents a general convolutional neural network architecture. The proposed architecture is presented in Sect. 4. Section 5 presents the employed methodology. Results are described in Sect. 6. Finally, Sect. 7 summarizes the main conclusions and briefly talks about future directions.

2 State of the Art

The classical techniques to avoid over-fitting in CNNs have been dropout and data augmentation, first proposed in [18]. Recently, several works have shifted the attention into other regularization forms that attempt to modify the loss function used by the gradient descent algorithm or its variants. An intermediate layer between pooling and convolution layers is proposed in [23], such layer is a *micro network* which can be modelled as a small classifier, such as SVM or softmax classifiers. Each micro network adds a regularization term to the CNN's loss function, which simultaneously minimize final classification error while enforcing hidden layers to learn more discriminative features.

The term *micro network* was introduced in [20] and refers to a modification in the architecture of a CNN where a multi-layer perceptron (MLP), or other

small network, is used instead of a linear filter at each convolutional layer. Each patch of the input volume is fed to the MLP, and the MLP output forms a voxel of the output matrix. The feature maps are build by sliding the MLP over the whole input volume. The architecture using micro networks is known as *network-in-network* (NiN).

An extension of [20] was proposed in [26], where the output of a NiN is routed into multiple fully-connected layers, each concatenated to a different loss function. The authors of [26] claim that using multiple loss functions drag the training algorithm away from overfitting to one particular single-loss function. The intuition behind using multiple loss functions is that each will model a different aspect of the same task. For instance, the pairwise ranking loss [24] prioritizes learning a given order of labelled pairs, while the LambdaRank loss [9] optimizes the top-k classification accuracy.

A redundancy regularizer was proposed in [25] to reduce the number of correlated kernels at each convolutional layer. Such regularizer showed a slight improvement over dropout, but when combined with it and early stopping the improvement was significant.

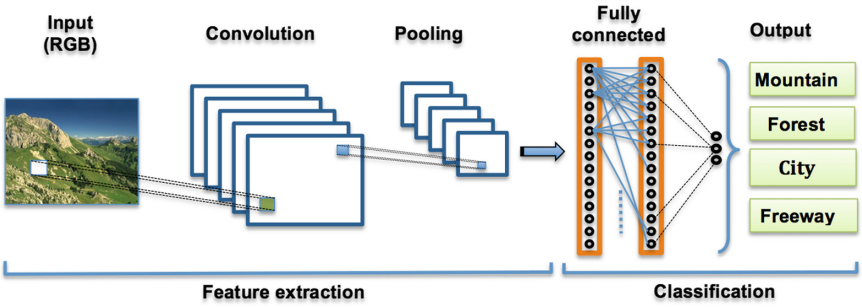


Fig. 1. Hierarchical representation of a convolutional neural network.

3 Convolutional Neural Network

Figure 1 displays a generic CNN architecture. It is a hierarchical structure composed of five principal layers:

Input layer. It is considered as a pre-processing layer, where the input images can be resized, rotated or subsampled.

Convolution layer. It is considered as the basic building block in a CNN, and performs most of the computational work. This layer receives an input volume with H^c_{in} pixels height, W^c_{in} pixels width, D^c_{in} channels deep, and P pixels for padding. The input volume is processed with a set of k filters, which are encoded as weights and connections between neurons. Such filters are based on convolutional masks, called *kernels*, and are defined with the following parameters: spatial extension $(E, [W_f, H_f])$ and stride (S_x, S_y), where

E is the kernel depth along D^c_{in}, $[W_f, H_f]$ is the area size where convolution will be computed, and (S_x, S_y) define the number of pixels that will be skipped between consecutive convolutions. The output volume size is defined as $(H^c_{out}, W^c_{out}, D^c_{out})$, where each dimension is defined in terms of the input volume as $W^c_{out} = (W^c_{in} - E + 2P)/S_x + 1$, $H^c_{out} = (H^c_{in} - E + 2P)/S_y + 1$, and $D^c_{out} = k$.

Pooling layer. Its purpose is to shrink the convolution layer output so that the dimensionality of the extracted features can also be reduced, while keeping the information they encode. The pooling layer receives an input volume of size $(W^p_{in}, H^p_{in}, D^p_{in})$, which is processed with a set of filters with the following parameters: spatial extension $(F, [W_f, H_f])$ and stride (S_x, S_y). The output volume size is defined as $(H^p_{out}, W^p_{out}, D^p_{out})$, where each dimension is defined in terns of the input volume as $W^p_{out} = (W^p_{in} - F)/S_x + 1$, $H^p_{out} = (H^p_{in} - F)/S_y + 1$, and $D^p_{out} = D^p_{in}$.

Fully connected layer. This layer computes the weighted sum of the pooling layer output, or of the convolution layer when no pooling layer is present.

Output layer. This layer holds one neuron per category in the classification task.

The first three layers comprise the FE stage while the last two layers encode the classification stage. In a deep convolutional neural network, several FE stages are present, while only one classification stage remains at the end of the data flow.

4 Proposed Architecture

The proposed architecture, named *ACEnet*, is shown in Table 1. It is comprised of 29 layers, from which 6 are convolution layers and 4 max-pooling layers, all of them using different kernels and strides. Some of the convolutional layers use padding to make their output conform with the input size needed for the next feature-extraction stage. *ACEnet* uses six FE stages. Each of the first three FE stages comprises of four layers: a convolution layer, a rectification layer, a pooling layer, and a normalization layer. The next two FE stages only have two layers: a convolution layer followed by a rectification layer, while the last FE stage also have a pooling layer at the top of the previously mentioned layers. Layers 20–25 form two generalization stages, intended to avoid overfitting in the network. Each generalization stage calculates the weighted sum of its inputs and rectifies it, to subsequently apply a regularization technique known as drop-out. In the following we describe some characteristics of our proposed architecture.

Convolution layers: They were designed using small kernels (i.e. $E \in \{7, 5, 3\}$) to get highly representative features and decrease the number of training parameters. The first convolution layer gets low-level features (i.e. edges, lines and curves), while subsequent convolution layers get high-level features.

Rectification layers: These layers have neurons with the non-linear, non-saturating activation function $f(x) = \max(0, x)$. They are known as Rectified Linear Units (ReLU) because their behavior is similar to the half-wave rectifier in electrical engineering. They provide a better model of biological neurons, with similar or better performance than the logistic-sigmoid function and the hyperbolic-tangent function. It has been proved that ReLU units reach good performance without resorting to unsupervised pre-training and, although their training requires large amounts of labeled data, there is no negative effect to their performance [15]. The CNNs with ReLU layers are trained several times faster then their equivalent with hyperbolic-tangent units [18].
Pooling layers with overlap: Pooling in CNNs is highly disruptive. If the filters do not overlap, the pooling layers loose information about objects localization on the image, which is needed to detect the precise relation among them. The most popular pooling layers use 2×2 filters with a stride of 2, shrinking the input image size by half, discarding about 75% of the activations generated by the previous layer. To avoid information loss and overfitting, the pooling layers used in the proposed architecture consist of 3×3 filters with a stride of 2, also called pooling with overlap; if pooling windows overlap enough, location information will be preserved.
Normalization layers: Their propose is to add generalization ability to the network. Normalization of a neuron's output implements a kind of lateral inhibition much more like the found in real neurons, promoting competition among several neurons output at times of great neural activity [18].
Drop-out layer: These layers were included to force fully connected layers to learn more robust features. Drop-out is a highly recommended technique to cope with overfitting [22], in which the output of some randomly-selected neurons are set to zero so they can't contribute with the backpropagation of the error at training, reducing complex coadaptation of neurons; in this way, every time a new example is fed for training, the neural network shows a different architecture, however all the different architectures share their weights.

5 Methodology

In this section we provide a detailed description of three important experiments that allow us to justify the proposed architecture, as well as the training parameters. In the first subsection, we describe the model selection method used to set the mini-batch size and epochs number in the training algorithm for *ACEnet*. In the last subsection, we provide the settings for the numerical comparison of the classification performance between *ACEnet* and *Alexnet*.

We used the algorithm Stochastic Gradient Descent with Momentum (SGDM) to train the compared architectures. Let θ_i be the vector with all the weights of the network at the i-th iteration of the SGDM algorithm, which are updated by the rule $\theta_{i+1} = \theta_i - \eta \nabla E(\theta_i) + \mu(\theta_i - \theta_{i-1})$, where η is the initial

Table 1. Architecture of the proposed convolutional neural network

No.	Layer	Output volume size			Kernel	Stride	Padding
		Width	Hight	Deep			
0	Input	227	227	3	-	-	-
1	Convolution1	111	111	50	7	2	-
2	ReLu1	111	111	50	-	-	-
3	Pooling1	55	55	50	3	2	-
4	Normilization1	55	55	50	-	-	-
5	Convolution2	55	55	100	5	1	2
6	ReLu2	55	55	100	-	-	-
7	Pooling2	27	27	100	3	2	-
8	Normalization2	27	27	100	-	-	-
9	Convolution3	27	27	256	3	1	2
10	ReLu3	27	27	256	-	-	-
11	Pooling3	13	13	256	3	2	-
12	Normalization3	13	13	256	-	-	-
13	Convolution4	13	13	400	3	1	1
14	ReLu4	13	13	400	-	-	-
15	Convolution5	13	13	400	3	1	1
16	ReLu5	13	13	400	-	-	-
17	Convolution6	13	13	256	3	1	1
18	ReLu6	13	13	256	-	-	-
19	Pooling6	6	6	256	3	2	-
20	Fully connected	1	1	4800	-	-	-
21	ReLu	1	1	4800	-	-	-
22	Drop-out	1	1	4800	-	-	-
23	Fully Connected	1	1	2400	-	-	-
24	ReLu	1	1	2400	-	-	-
25	Drop-out	1	1	2400	-	-	-
26	Fully connected	1	1	NC	-	-	-
27	Softmax	1	1	NC	-	-	-
28	Classification	1	1	1000	-	-	-

learning rate, μ is the momentum, and $E(\theta_i)$ is the loss function described as $E(\theta_i) = \frac{1}{\beta}\sum_{k=1}^{\beta} E_k(\theta_i) - \frac{\lambda}{2}\theta_i^T\theta_i$, where $E_k(\theta_i)$ is the loss function value at the k-th training example in the mini-batch of size β, and λ is the regularization coefficient. The SGDM parameters used to train Alexnet were set as recommended in [18]. Some other parameters were kept in common for both architectures and were set as follows: the initial weights at every layer were randomly drawn

from the gaussian distribution $N(0, 0.01)$; the activation thresholds at each layer were initialized to zero; finally, the number of neurons in the output layer was adjusted to match the number of classes in each database. Both architectures were trained on a GPU NVIDIA GeForce GTX TITAN X with 3072 cores and 12 GB of memory, using MatLab and the Deep Learning Toolbox.

Table 2. SGDM parameters used for training of ACEnet and Alexnet

Parameter	Value	
	ACEnet	*Alexnet*
Initial learn rate (η)	0.001	0.01
Learn rate schedule	Piecewise	Piecewise
Learn rate drop factor	0.1	0.1
Learn rate drop period	30	30
Max epochs	100	100
Momentum (μ)	0.9	0.9
L2 regularization (λ)	0.0005	0.0005
Mini batch size (β)	25	128

In case of *ACEnet*, we decided to set the learning rate at 0.001 to get a finer control over the steps given by the SGDM algorithm, and by setting $\lambda = 0.0005$, we reduced the probability of overfitting and the complexity of the CNN [16]. We performed a sequential search to find the optimal value for the mini-batch size and epoch number. In each step of the search the classification error was computed using the hold-out method and the optimal was found in the minimum value of the classification error.

To test the ACEnet generalization performance, we selected five complex databases with real-world images in JPEG format.Relevant statistics for each database are displayed on Table 3. A sixth database was build by combining all selected databases, and used only for performance comparison against

Table 3. Summary of the selected databases for image classification. "Min p. class" and"Max p. class" refer to the minimum and maximum images per class, respectively.

Database	Categories	Dimensions			No. images		
		Width	Hight	Depth	Total	Min p. class	Max p. class
Oliva [7]	8	256	256	3	2,668	260	410
Paris [6]	12	1024	768	3	6,412	147	1,497
Oxford [5]	17	1024	768	3	5,063	59	1,502
Caltech-101 [1]	101	300	200	3	8,677	31	800
Caltech-256 [2]	256	300	200	3	29,780	80	798

Alexnet. Both, model selection and performance comparison was completed for each database. In every experiment, we estimated the classification performance by means of hold-out, where each database was split into training and testing sets. We used 70% of the total number of images as training set and the rest as testing set.

5.1 Training Parameters

Mini-batch Size. The mini-batch size is one of the parameters that directly impacts CNNs performance. It indicates the number of images from the training set that are considered at each iteration of the SGDM algorithm. The optimal mini-batch size was found by means of sequential search in the interval [10, 120] with increments of 10 units. At each point in the search, we recorded the classification accuracy and training time, and selected the optimal mini-batch size as the point in the search with minimum training time and maximum accuracy. In this experiment, every time we run the SGDM algorithm we fixed the epochs number to 100.

Epochs Number and Overfitting. Besides drop-out layers, controlling the epochs number in the training algorithm is an efficient way to avoid overfitting in a CNN [18]. We trained ACEnet with different values for the epochs number, ranging from 10 to 100 with increments of 10. We recorded the classification error for the training set and testing set at the end of the training algorithm. It is well known that the training and testing errors steadily decrease before overfitting, and they diverge when the CNN has been overfitted [8]. Thus, the optimal epochs number is the point before the classification errors diverge.

5.2 Performance Comparison

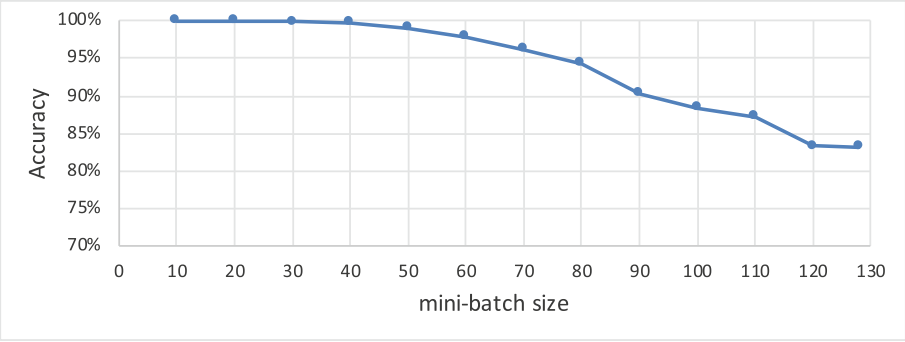
We compared the classification performance of ACEnet against that of the well-known Alexnet architecture. The training parameters for Alexnet and for ACEnet were the same except for the mini-batch size, which was set using the results of the model selection method for our proposed architecture. The mini-batch size as well as the rest of the training parameters for Alexnet were taken from [18] and are listed in Table 2. For each of the five selected databases, we trained both architectures using 70% of the total number of images until the SGDM converged. After reaching convergence, we recorded training time and classification accuracy on the training set. Next, we computed and recorded the classification accuracy on the testing set.

6 Results

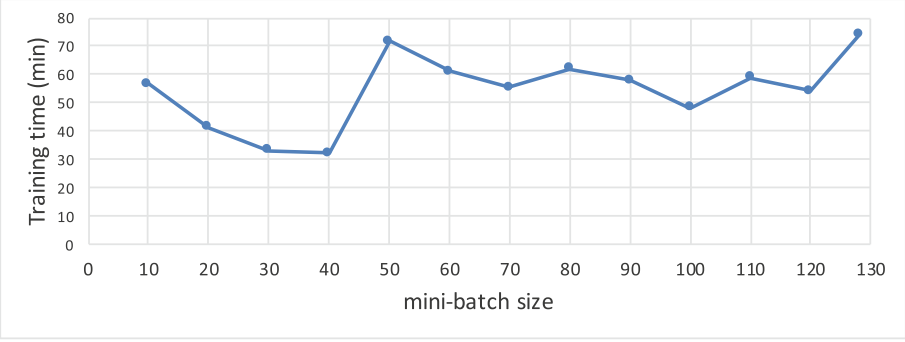
In this section we start by describing the results of the parameter selection experiment and then we will proceed to the comparison with the well-known Alexnet architecture.

6.1 Training Parameters

Mini-batch Size. Figure 2 shows the recorded training times and classification accuracies for every mini-batch size tested in the model selection method described in Sect. 5.1 for Oliva database. It can be seen in Fig. 2 that the best accuracy is located between 1 and 50 images per batch, while the minimum training time is located between 20 and 40 images per batch. So, having a trade-off between accuracy and training time, we propose to use a mini-batch size of 25 for Oliva database. The same results were found for other image datasets.



a) Classification performance on the training set.

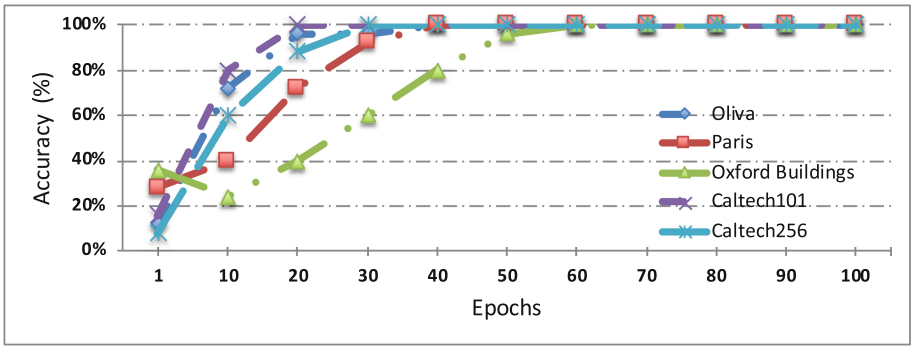


b) Training time as a function of mini-batch size.

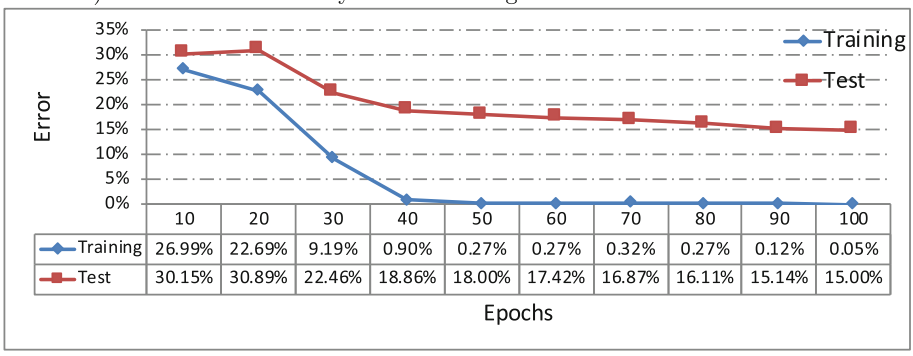
Fig. 2. Mini-batch size selection for olivaba database.

Epoch Number and Overfitting. We can observe in Fig. 3a the classification accuracy reached when ACEnet is trained using different number of epochs for each of the selected databases. Clearly, our proposed architecture correctly learns the training set in about 60 epochs, regardless of the database used. Thus, setting the number of epochs to 100 give a good training margin for future databases, however it must be tested if ACEnet does not overfit to the training set with such large number of epochs. Figure 3b displays training and testing errors recorded

for different values of the epochs number using the Oliva database. It is worth noting that although the testing error has a light increment when setting the epochs number to 20, both training and testing errors keep decreasing steadily until the 50 epochs point. Such increment could be considered as an overfitting indicator, however in the next point along the grid we see both errors decrease again, which points towards the ability of our architecture to recover from overfitting. The training error can be considered stable after the 50 epochs point, however the testing error keeps decreasing until the 100 epochs point. Therefore, we can safely train ACEnet for 100 epochs without incurring in overfitting.



a) Classification accuracy on the training set for all selected databases.



	10	20	30	40	50	60	70	80	90	100
Training	26.99%	22.69%	9.19%	0.90%	0.27%	0.27%	0.32%	0.27%	0.12%	0.05%
Test	30.15%	30.89%	22.46%	18.86%	18.00%	17.42%	16.87%	16.11%	15.14%	15.00%

b) Overfitting test for Oliva-Torralba database.

Fig. 3. Epoch number selection based on overfitting analysis.

6.2 Performance Comparison

Table 4 shows an overview of the experimental results from the performance comparison against Alexnet. In the following we elaborate further on the three aspects shown in Table 4, namely training time T_{tr}, and classification accuracy for the training A_{tr} and testing A_{te} set. ACEnet required consistently less time to reach a 100% of classification accuracy on the training set than Alexnet.

The achieved speed up for Oliva, Paris, Oxford, Caltech-101, and Caltech-256 databases was 1.4x, 1.2x, 1.3x, 1.9x, 1.8x and 1.4x, respectively. Although ACEnet has one more stage than Alexnet, such increase in model complexity did not directly impacted on the time needed for training. Regarding classification accuracy on the test set, ACEnet performed better than Alexnet for all selected databases. The performance improvement for Oliva, Paris, Oxford, Caltech-101, and Caltech-256 databases was 9.58%, 8.60%, 4.35%, 12.05%, and 17.60%, respectively. On average, our proposal improves 5.11% the mean testing performance over the five selected databases. For the artificially build database, our proposal outperformed Alexnet by 6.65%.

Table 4 also shows other measures of classification performance for both networks, namely F1-score, G-measure, and Matthews correlation coefficient. While F1-score provides the harmonic mean of precision and recall, G-measures is their geometric mean. On the other hand, Matthews correlation coefficient indicates the disagreement between predicted and true labels, and it is robust to the imbalanced-class problem. As can be seen, ACEnet generally outperforms Alexnet in all the mentioned measures. Additionally, we performed the nonparametric Friedman's test to prove the statistical difference in the results obtained by the two architectures. The resulting p-values for all the selected performance measures are given in the bottom of Table 4. From such tests, we can say that the results obtained with ACEnet are statistically different from those obtained with Alexnet with 97% of confidence.

Table 4. Time and performance comparison between ACEnet and Alexnet. Training time T_{tr} is given in hours, while classification accuracy for the training A_{tr} and testing A_{te} set are given in percentages. F1 is the harmonic mean of precision and recall, while G is their geometric mean. MCC displays Matthew correlation coefficient.

Database	ACEnet						Alexnet					
	T_{tr}	A_{tr}	A_{te}	F1	G	MCC	T_{tr}	A_{tr}	A_{te}	F1	G	MCC
Oliva	0.58	100	85.11	0.86	0.86	0.84	0.83	98.83	77.67	0.81	0.81	0.79
Paris	4.36	98.93	46.09	0.43	0.46	0.41	5.27	97.61	42.44	0.41	0.43	0.37
Oxford	3.36	99.49	31.42	0.25	0.30	0.20	4.35	99.97	30.11	0.20	0.26	0.15
Caltech-101	1.82	99.95	71.51	0.59	0.69	0.71	3.45	99.51	63.83	0.01	0.50	0.64
Caltech-256	7.68	99.99	36.99	0.01	0.14	0.37	13.57	99.60	31.20	0.01	0.08	0.30
All	18.63	96.81	35.51	0.01	0.23	0.35	26.88	96.64	28.95	0.01	0.14	0.28
p-value			0.0143	0.030	0.030	0.030						

7 Conclusions and Future Works

We have presented an improved convolutional neural network architecture for image classification. Designing a CNN for image classification involves setting properly the feature extraction layers, the classification stage, and the parameters of the learning algorithm. The architecture of a CNN can be empirically

designed if the behavior and purpose of each stage are clearly understood, however, the parameters of the learning algorithm must be fine-tuned by a model selection method. The proposed architecture can be trained from scratch with any database, which avoids reusing pre-trained networks.

The results presented show that the proposed architecture do not suffer from overfitting, allowing us to increase the number of epochs in the training algorithm. We showed that while the training error reaches stability, the testing error keeps decreasing as the number of epochs increases, however the training time will also increase. In order to keep a low training time, we must sacrifice generalization performance, however we obtained testing errors comparable to those in the state-of-the-art.

Comparing our proposal with the well-known Alexnet architecture. Our proposal improves 5.11% the mean testing performance over five real-world databases. Regarding mean processing time, our proposal required 17.8 h to process all five databases, while Alexnet needed 27.47 h. Our proposal is faster due to the use of a smaller mini-batch and kernel size, and due to the reduction in the number of epoch. However we empirically demonstrated that such size reduction do not affect the classification accuracy. While Alexnet was specifically designed to get a low classification error on the ILSVRC-2012 database, our proposed architecture does not depend on a specific database and has a better generalization performance due to the extra processing stage. As future work, we would like to use more complex image databases, beside we would like to use the weights of intermediate layers in pretrained CNN as a set of features for image classification/identification, where the final aim is to conform a content-based image retrieval (CBIR) system.

References

1. Caltech-101 dataset. http://www.vision.caltech.edu/Image_Datasets/Caltech101
2. Caltech-256 dataset. http://www.vision.caltech.edu/Image_Datasets/Caltech256
3. Imagenet dataset. http://www.image-net.org/
4. Labelme dataset. http://cvcl.mit.edu/database.html
5. The Oxford Buildings dataset. http://www.robots.ox.ac.uk/~vgg/data/oxbuildings
6. The Paris dataset. http://www.robots.ox.ac.uk/~vgg/data/parisbuildings
7. Oliva, A., Torralba, A.: Modeling the shape of the scene: a holistic representation of the spatial envelop. Int. J. Comput. Vis. **42**(3), 145–175 (2001)
8. Bishop, C.M.: Neural Networks for Pattern Recognition. Clarendon Press, Gloucestershire (1996)
9. Burges, C., et al.: Learning to rank using gradient descent. In: 22nd International conference on Machine Learning. ACM Press (2005)
10. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: delving deep into convolutional nets. In: proceedings of ECCV (2014)
11. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L., Schmidhuber, J.: Flexible high performance convolutional neural networks for image classification. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, vol. 2, pp. 1237–1242 (2011)

12. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Li, F.F.: ImageNet: a large-scale hierarchical image database. In: IEEE Computer Vision and Pattern Recognition, pp. 248–255 (2009)
13. Fukushima, K.: Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybern. **36**(4), 193–202 (1980)
14. Girshick, R.B., Danahue, J.: Rich features hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
15. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the International Conference on Artificial Intelligence and Statistics, vol. 15, pp. 315–323 (2011)
16. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT Press, Cambridge (2016)
17. Hertel, L., Barth, E., Kaster, T., Martinetz, T.: Deep convolutional neural networks as generic feature extractor. In: Proceedings of the IEEE International Joint Conference on Neural Networks, pp. 1–4, July 2015
18. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 25, pp. 1097–1105 (2012)
19. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**, 2278–2324 (1998)
20. Lin, M., Chen, Q., Yan, S.: Network in network. In: Proceedings of the 2nd International Conference on Learning Representation (2014)
21. Razavjan, A.S., Azizpour, H., Sullivan, J., Carlsson, S.: CNN features off-the-shelf: an astounding baseline for recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 512–519 (2014)
22. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to precent neural network from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
23. Sun, W., Su, F.: A novel companion objective function for regularization of deep convolutional neural networks. Image Vis. Comput. **60**, 58–63 (2017)
24. Usunier, N., Buffoni, D., Gallinari, P.: Ranking with ordered weighted pairwise classification. In: Proceedings of the 26th Annual International Conference on Machine Learning, ACM Press (2009)
25. Wu, B., Liu, Z., Yuan, Z., Sun, G., Wu, C.: Reducing overfitting in deep convolutional neural networks using redundancy regularizer. In: Lintas, A., Rovetta, S., Verschure, P.F.M.J., Villa, A.E.P. (eds.) ICANN 2017. LNCS, vol. 10614, pp. 49–55. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68612-7_6
26. Xu, C., et al.: Multi-loss regularized deep neural network. IEEE Trans. Circ. Syst. Video Technol. **26**(12), 2273–2283 (2016)
27. Yosinski, J., Clune, J., Bengio, Y.: How transferable are features in deep neural networks? In: Advances in Neural Information Processing Systems, vol. 27, pp. 3320–3328 (2014)