Contrasting Convolutional Neural Network (CNN) with Multi-Layer Perceptron (MLP) for Big Data Analysis

Conference Paper · August 2018

DOI: 10.1109/ICIAS.2018.8540626

4 authors, including:

Muhammad Moinuddin
King Abdulaziz University
114 PUBLICATIONS **522** CITATIONS
SEE PROFILE

Ubaid M Al-Saggaf
King Abdulaziz University
98 PUBLICATIONS **1,029** CITATIONS
SEE PROFILE

Syed Saad Azhar Ali
Universiti Teknologi PETRONAS
96 PUBLICATIONS **496** CITATIONS
SEE PROFILE

Contrasting Convolutional Neural Network (CNN) with Multi-Layer Perceptron (MLP) for Big Data Analysis

Abdelaziz Botalb
Electrical and Computer Engineering
King Abdulaziz University
Jeddah, Saudi Arabia
azboutalb@gmail.com

2nd M. Moinuddin
Center of Excellence in Intelligent Engineering System
King Abdulaziz University
Jeddah, Saudi Arabia
mmsansari@kau.edu.sa

3rd U. M. Al-Saggaf
Center of Excellence in Intelligent Engineering System
King Abdulaziz University
Jeddah, Saudi Arabia
usaggaf@kau.edu.sa

4th Syed S. A. Ali
Center for Intelligent Signal and Image Processing
Universiti Teknologi PETRONAS
Malaysia
saad.azhar@utp.edu.my

***Abstract*—Recently, CNNs have become very popular in the machine learning field, due to their high predictive power in classification problems that involve very high dimensional data with tens of hundreds of different classes. CNN is a natural extension to MLP with few modifications which resulted in a breakthrough. Mainly, the MLP algebraic dot product as a similarity function was replaced with 2-d convolution; in addition to a pooling layer which reduces parameter dimensions making the model equi-variant to translations, distortions, and transformations. The sparse connectivity nature of CNN is also a variation to the MLP. The two models were implemented on the EMNIST dataset which was used as 50% and 100% of its capacity. The models were trained with fixed and flexible number of epochs in two runs. Using 100% of EMNIST; for the fixed run CNN achieved test accuracy of 92% and MLP 31.43%, where in the flexible run the CNN achieved 92% and MLP 89.47%. Using 50% of EMNIST; for the fixed run CNN achieved test accuracy of 92.9% and MLP 33.75%, where in the flexible run of 92.9% and MLP 88.20%. The CNN demonstrated a good maintenance of high accuracy for image like inputs and also proved to be a better candidate for big data applications.**

***Index Terms*—CNN, MLP, Convolution, Pooling, Hyper-parameters, Normalization**

I. Introduction

Convolutional Neural Network (CNN) is a class of Deep Learning which proved to be very promising in pattern recognition applications. It has been used successfully in different application fields lowering the need for manual feature extraction leading to state of-the-art performance in fields such as speech recognition and computer vision [1] [2] .

CNN had attracted attention partially when the large-scale CNN for image classification successfully outperformed all other techniques in the ImageNet 2012 competition [3]. After that, a few CNN models were developed by different researchers that competed with humans in terms of image recognition [2] [4] [5]. The stunning advantage of CNNs lies in their ability to reliably and automatically extract relevant features instead of manual extraction techniques from high dimensional dataset which is highly susceptible to human errors.

MLPs and CNNs are two similar models of Neural Networks; however they differ greatly in terms of performance. Unlike MLPs, CNN architectures are deep and require computationally expensive operations; hence two training down sides are tagged here; long time and high computation power resources. The contribution of this work is to investigate whether giving more training epochs and more data samples to MLP could hit the performance of CNN. In other words we will find out whether we still need those high computationally expensive operations of CNN to get optimal results, whereas we can have an MLP trained with more epochs and more data samples and get the same quality of results. This will give us an insight on how both models scale-up for big data applications. This work will first start with giving a neat informative insight into the mechanism of CNN, and its relation to MLP including how they differ and relate.

II. The Working Principle

The CNN comprises three main types of layers: Convolution, Pooling, and Fully connected. This latter is simply the very well-known Multi-layer Perceptron Network (MLPN).Convolution and pooling layers are two types of hidden layers, which are analogous to hidden layers in MLPN.

A. Convolution Layer

The input to the network here is in 2-D format unlike in MLPN (1-D format), in fact everything in the convolution and pooling layers is in 2-D formats or volumes. The input is

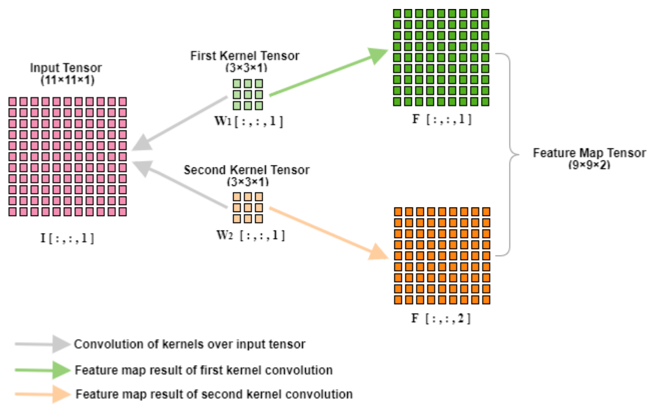


Fig. 1. Convolution Operation

normally a tensor of order 4 whose dimension is $H \times W \times C \times N$, where C is the number of channels and N is number of samples. The kernel (also called filter or feature detector) is also a tensor of order 4 whose dimensions are $H \times W \times C' \times N'$, where C' is the depth of the filter which is equal to the depth of the input volume (could be an image or data that is structured in 2-D format) $C = C'$, and N' is the number of kernels used. The feature map is a tensor of order 3 whose dimension is $H \times W \times D$ where D is its depth and it is equal to the number of kernels N' where $D = N'$. The volume depth in CNN indicates the number of matrices in a volume, for example, the feature map in figure 1 has a depth of 2, but kernels has only depth of 1 as $C = C'$ and $C = 1$ for the input. i.e. one single channel. Note that the number of filters reflects the number of how many features we are looking for inside the input volume. The depth in CNN architecture indicates how many hidden layers it comprises.

In Figure 1, the squares in feature map represents neurons in MLPNN hidden layer, and the squares in kernels represent the weight values in MLPN. The main difference between CNN and MLPN, is that in this latter every input element is connected to every neuron in the hidden layer as shown in Figure 2, and this is why MLPN is also given the name fully connected network, while in CNN only a predetermined neighborhood (limited number of input elements; sparse connectivity) called the receptive field is connected to ONLY part of the hidden layer. In the example reported in Figure 1, we are using a filter of size 3×3, this means that we are connecting an area of the input matrix of 3×3 which is a receptive field of hight and width of 3, and this is the idea of Sparse Connectivity of CNNs. Note that best practice confines that filters should be square.

The output of the feature map (also called activation map) is found by using a similarity function called Convolution (where in MLPNN we used algebraic dot product as a similarity function), as we convolve the kernel over the input matrix. Each convolution operation will produce the output of one neuron in the feature map. Figure 3 is a clear view of this operation for the first channel of the feature map of Figure 1. As clarified in the Figure 3 the convolution is moving to the right by one step, this is called Stride which is equal to 1 in this case. The first filter will be looking for a certain feature within the input volume and will generate its own feature map which shows where that feature exists within the original image. The second filter will also do the same generating its own feature map.

The convolution is a lossy operation. i.e. it does not maintain the spatial resolution. The bigger the stride and filter size, the more we lose spatial resolution. The spatial resolution of the original input can be preserved by introducing zero-padding. i.e. if we want the original size of the input volume to be equal the size of the feature map we wrap the input volume with zeros as shown belowin Figure 4.

Note that it is possible to use a convolution layer as a fully connected layer. Lets say we have conv1 as input to conv2, conv2 (sparse connectivity) can operate in the same way as MLPN (full connectivity) if the feature map size (width and height) of conv1 is equal to the size of the filters of conv2.

The mathematical computation that every neuron is doing in both network types is expressed as follows:

- The output of every neuron in the MLPN was found by using the algebraic dot product as similarity function:

$$< W^l, a^{l-1} >= a_j^l = \phi\Big(\sum_{i=1}^{N} w_{ji}^l a_i^{l-1} + b_j^l\Big) \qquad (1)$$

- As for the CNN, the convolution output of every neuron in the feature map is expressed as:

$$w_{x,y}^l * x_{x,y}^{l-1} = \sum_{x'}\sum_{y'} w_{x',y'}^l x_{x-x',y-y'}^{l-1} + b_j^l \qquad (2)$$

Where for both expressions :

- a^{l-1} is the input vector from last layer, in case of first hidden layer this input will be input data x as shown in (2) otherwise is the activation as we expressed in (1) .
- W^l is the weight matrix of the l^{th} layer
- a_j^l is the activated ouput of the j^{th} neuron at the l^{th} layer
- $\phi()$ is the nonlinear activation function
- N is the caridinality of input vector a^{l-1}
- a_i^{l-1} is the i^{th} entry of the input vetcor a^{l-1}at the previous layer $l-1$.
- w_{ji}^l is the weight connection from i^{th} element of vector input a_j^{l-1} to the j^{th} neuron at current layer l.
- b_j^l is the bias term of the j^{th} neuron at the l^{th} layer

B. Pooling Layer

The pooling layer (also called down-sampling) is dedicated to reduce the size of the spatial resolution of the feature maps, which leads to a drastic decrease in the number of parameters (and computations) which ultimately reduce the risk of model overfitting leading to a higher generalization. It reduces the dimensionality while retaining the most important information. Another important advantage of pooling is that it makes the CNN invariant to minor transformations, distortions

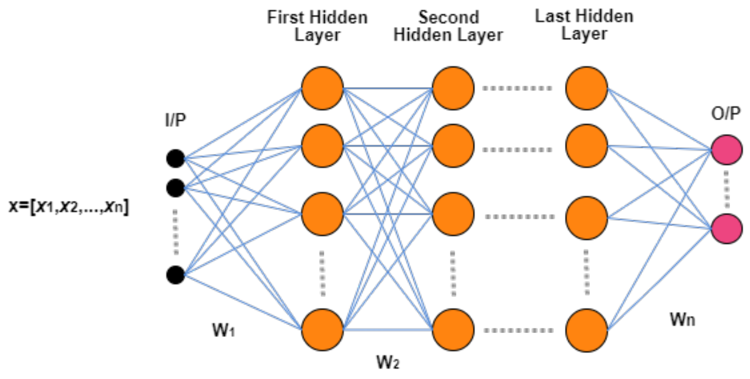


Fig. 2. General MLPN Architecture

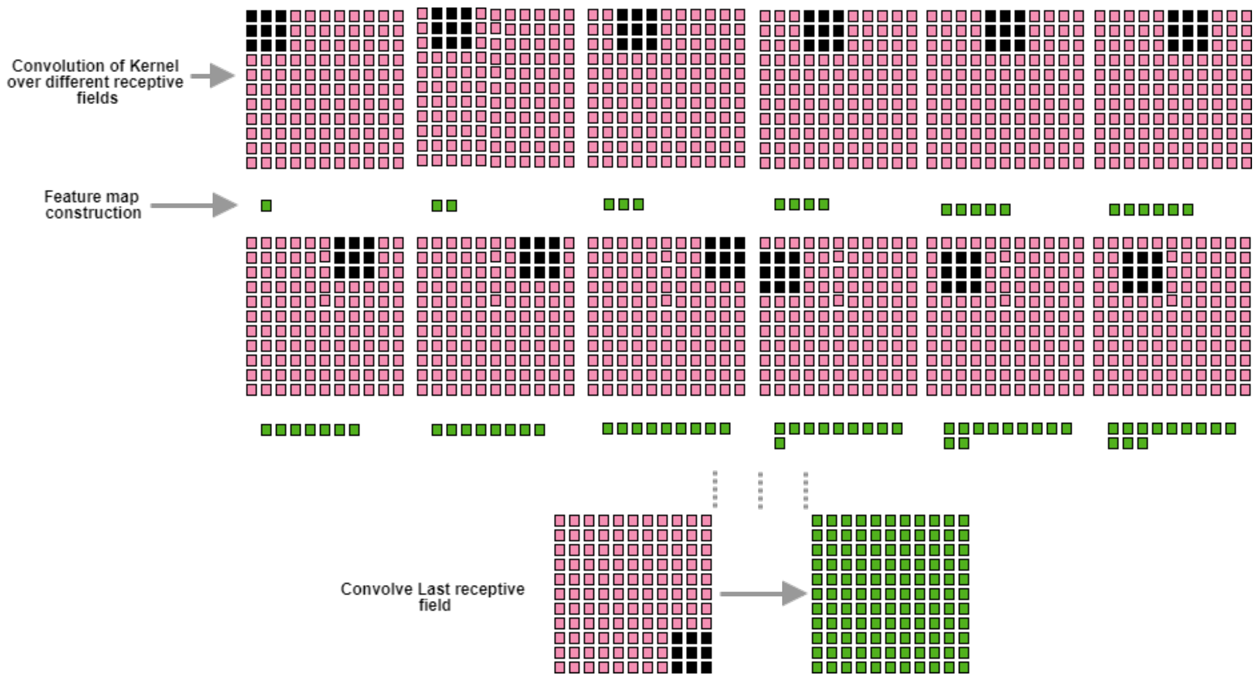

Convolution of Kernel over different receptive fields
Feature map construction
Convolve Last receptive field

Fig. 3. Feature Map Construction

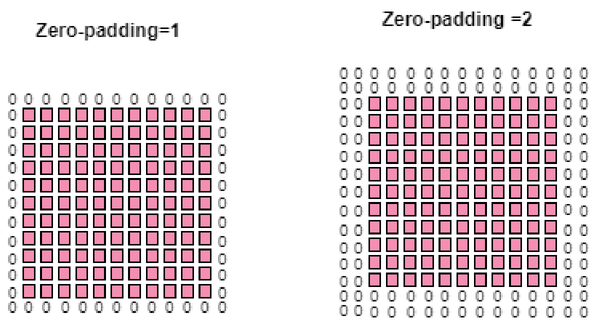


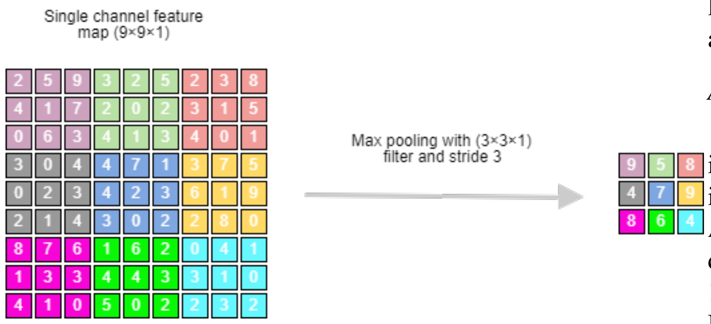
Fig. 4. Zero-Padding the Input Volume

and translations in the input volume. The most known types of pooling are max-pooling, average pooling and even L2-norm pooling. We choose a window size and a stride value, and then we perform the operation. Figure 5 shows the max pooling operation using window size of 3×3 and stride 3.

Note that for every receptive field region of 3×3 in the feature map only the maximum value taken. The colors in the resulted feature map correspond to the receptive fields in the input feature map on which the pooling was performed. It is clear that this feature map is reduced in spatial resolution from 9×9 to 3×3.

C. CNN Summary

We can add as many convolution and pooling layers as we want as long as it will achieve high performance. Pooling layers are not necessarily following every convolution layer. For example, we might have two, three, or four consecutive convolution layers then followed by one pooling layer. Note, in many works, they consider the activation function such as ReLU as a separate layer, but we personally consider it as



Single channel feature map (9×9×1)
Max pooling with (3×3×1) filter and stride 3

Fig. 5. Pooling Operation Over a Feature Map

a layer. Not every layer has to be followed by an activation, it could be applied after a number of consecutive layers. In general the sequence and size of layers and activations in CNN will entirely depend on the problem at hand, but there is a general sense that the more convolutional layers you add the more the CNN network will have the power to extract complicated features, which should be the case for CNNs, because they are categorized as deep learning methods, deep in the sense of the many layers they normally have, and they are used for complex classification problems with big and high dimensional datasets including large class space.

Debugging CNNs is not a trivial task, as the hyper-parameter space is very large including several dimensions such as number of convolution and pooling layers, number and size of filters, activation functions, cost functions, generalization techniques (such as regularization, drop out, momentum, etc), Learning rate, learning algorithm, and many others. In fact there is no strong theory behind neural networks unlike other models such as SVMs, hence there are no strict rules to follow about choosing the right hyper-parameters, everything is done heuristically and by a grid search, but knowledge about how these hyper-parameters interact between each other is a must so as to build a good model with less disappointment. Once the model is developed, its predictive power has to be evaluated by a number of performance metrics such us Confusion Matrix, Precision, Recall, Receiver Operating Characteristic (ROC) curve, etc. Relying on the single value of classification accuracy (number of right predictions over total number of instances) might be misleading especially when the dataset set is imbalanced. And one important note is that there is no one model works for all, it all depend on the type of dataset at hand.

III. Implementation

There are a number of good libraries to implement CNNs or any other Deep Learning model, this includes Caffe and Tensorflow which require knowledge of Python Language. In this work, Matlab software was used to implement both CNN and MLP. For this latter the built-in Neural Network toolbox found at [6] was used, and for CNN a third party Matlab library called MatConvNet found at [7] was used. MatConvNet is dedicated only for CNN design, it is all written in Matlab language except for the low level blocks such as convolution and max-pooling which are written in C_{++} and $CUDA$ code.

A. Dataset Preperation

The balanced EMNIST handwritten character dataset is used in this work; it contains 29,198 images. There are 26 classes in this dataset, and each image has a dimension of $32 \times 32 \times 1$. As a cross validation approach using the hold-out method, the dataset is split into 70% for training, 15% for validation, and 15% for testing. The same split is used for both CNN and MLP so as to make sure that performance comparison is not affected by dataset split. Both paradigms will be run twice first on 50% of dataset and second on full dataset.

B. Normalization of the Input Tensor

Data normalization is a very useful and important operation that has to be done not only for CNNs but for all other machine learning models. Normalization is about transforming the dataset into a specific range, so that each feature in the input volume will have the same range as other features. There are a number of normalization techniques out there; in our case we have subtracted the input volume mean, by this we ensure our input sticks to a specified range, this is useful because during training we are convolving the input volume with weights and activations which are then back-propagated with the gradients to train the model, and if our features has similar range (which is the case after we normalized our dataset), our gradients will not go out of control, hence the weights and biases will converge to the minima (local or global) leading to stable learning process.

C. Hyper-parameters

Setting the right hype-parameters for optimal predictive power is not a trivial task, and requires more time because the optimization space is quite large ; however this is what could be obtained for the time limit given to this work:

- *Learning algorithm* :Backpropagation algorithm using gradient descent
- *Cost function* : Log loss which is also called Logarithmic loss or Logistic loss
- *Activation Function* : Sigmoidal
- *Regularization (weight decay)*: L2 norm
- *Learning Rate*:0.001
- *Momentum*: 0.9

D. CNN Architecture

Table I highlights the architecture of the CNN. Note that final layer conv4 is actually used as a fully connected layer (the MLP classifier with one hidden layer) and this. Note again the output of this layer is fed to a Softmax Function that takes the vector input and outputs as a probability distribution summing to one, this helps a lot in interpreting the output of the net. Note also that there is no fixed rule about the choice of number and size of filters, its all heuristics which are rules of thumb, as still there is no universal robust theory behind hyper-parameter choice.

TABLE I
Network Macro-structural level Hyper-Parameters

Layer Type	Conv1	Pool1	Conv2	Pool2	Conv3	Sigmoid activation	Conv4
Depth of Filter	1	N/A	20	N/A	50	N/A	500
Number of Filters	20	N/A	50	N/A	500	N/A	26
Stride	1	2	1	2	1	N/A	N/A
Zero Padding	0	0	0	0	0	N/A	0

E. *MLP Architecture*

The MLP architecture was set to 1024 input neurons, one hidden layer with 100 neurons, and 26 output neurons; cross entropy was used as a cost function together with sigmoid as activation function, and as before backpropagation is used as a learning algorithm. MLP was presented with same dataset as CNN; this latter in the form of order 4 tensor, and the precedent in the form of order 2 tensor. Also note that the same cross-validation technique was used.

IV. Results

TABLE II
Classification Accuracy on Testing Set

Model	50% of dataset	100% of dataset	Epochs
CNN	92.9%	92%	15
MLP	30.75%	31.43%	15

TABLE III
MLP Classification Accuracy on Testing Set for More Epochs

Model	50% of dataset	100% of dataset
MLP	88.20% (155 epochs)	89.47% (202 epochs)

From table II, the number of epochs was restricted to 15 for both paradigms. CNN succeeded to converge getting a high accuracy during the given epochs, compared to MLP which did not converge (achieving low accuracy) during the same number of epochs. Different architectures were tried for the MLP to boost performance, including more hidden layers, and increased/decreased intra-layer number of neurons; however the best result (around 30% for both datasets) was achieved with 100 neurons in one hidden layer.

Table III shows that after allowing flexible epochs for MLP (fixed for CNN), it converges and achieved high accuracy in two different numbers of epochs for the two datasets. From the two tables, CNN maintained a higher accuracy than MLP, and if the number of epochs was also increased for CNN, the difference in accuracy would even increase.

The conclusion is that CNN requires less epochs to converge to local/global minima compared to MLP, in other words the MLP needs to see the whole dataset so many times in order to learn the weights compared to CNN, and this proves the efficiency of the CNN architecture explained before. This does not necessarily mean that CNN will consume less time in training in contrast with MLP, it actually all depends on the set of hyper-parameters and size of dataset. Doubling the dataset allowed MLP to have higher accuracy requiring even more epochs; however CNN accuracy was decreased by 0.9 because it was not allowed to train for more epochs to better learn the new added data, but even though it outperformed MLP, and if it was trained more it would definitely make the difference even higher. The most two important outcomes of this work:

1) CNN achieves higher accuracy than MLP no matter for how many epochs the MLP is trained.
2) More data allows for higher accuracies, so big data will make it possible to design machine learning algorithms that can achieve classification accuracies that are equal or better than that of humans.

V. Conclusion

During this work we have provided clear insights into how CNNs work , we have also proved that for the task being given CNN has outperformed MLP for the two different dataset sizes. We also concluded that CNN scales up very well compared to MLP for the given amount of data maintaining a high accuracy, which make them a better candidate for machine learning applications in big data.

VI. Acknowledgments

This project was funded by the Center of Excellence in Intelligent Engineering Systems (CEIES), King Abdulaziz University, under Grant No. (CEIES_18_04_01). The authors, therefore, acknowledge the technical and financial support of CEIES and Universiti Teknologi PETRONAS (UTP).

References

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, B. Kingsbury, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82-97, Nov. 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *in Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778, 2016.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *in Advances in neural information processing systems*, pp. 1097-1105, 2012.

[4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large scale image recognition,” *ArXiv preprint* arXiv:1409-1556, 2014.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *in Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 19, 2015.

[6] Mathworks, “Getting Started with Neural Network Toolbox,” https://www.mathworks.com/help/nnet/getting-started-with-neural-network-toolbox.html, 2014. [Version, R2014a].

[7] A. Vedaldi, “MatConvNet: CNNs for MATLAB,” http://www.vlfeat.org/matconvnet, 2014. [Version, 1.0-beta23].