Image Classification: CIFAR-10
Neural Networks vs Support Vector Machines

by
Chahat Deep Singh

***Abstract*—This project aim towards the *CIFAR-10* image classification using Support Vector Machines (SVM) and Convolutional Neural Networks (CNN) and hence comparing the results between the two. Different types of architectures were trained and tested including LeNet5 and NIN. In these networks, few parameters were changed like the activation function, number of layers, weight decay, dropout *etc*. In SVM, Local Binary Patterns (LBP) were used as feature descriptor. We will see that the state-of-the-art (until 2011) technique (SVM using LBP) was out-classed by Neural Networks.**

I. Introduction

The intention behind the project is to compare the results between the image classification using NN and SVM. In order to do so, various kinds of Neural Networks architectures were used. The way Neural Networks can be applied to image processing is unfathomable. In each network, several parameters were changed in order to understand the changes in the results due to those parameters. Few of the architectures used during the process were LeNet-5 (Yann LeCun's Convolutional NN) and Network-In-Network (Min Lin *et al.*),

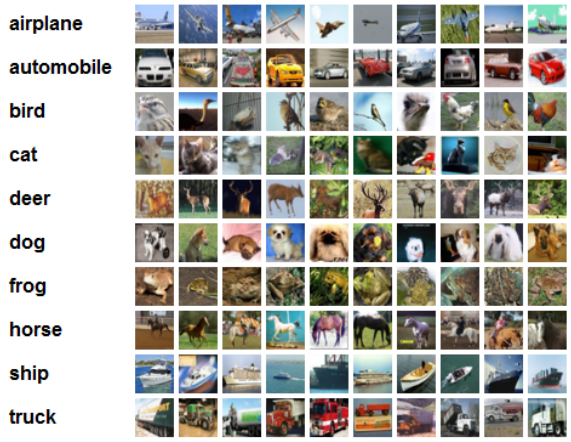

Fig. 1. CIFAR-10 Image Classification

Most of these networks were implemented on MATLAB with *MatConvNet* (a toolbox for computer vision applications) while few of the above networks were only studied and the comparisons will be stated in section IV of the report. *MatConvNet* is a simple, efficient toolbox and the state-of-the-art CNNs can easily be visualized. *MatConvNet* exposes simple MATLAB commands as the CNN building blocks like convolution, normalisation and pooling. Many of such blocks uses optimised CPU and GPU implementations written in C++ and CUDA. With *MatConvNet*, we get the performance close to as of *Caffe* and the visualization power of MATLAB. The implementation of these networks can be found in Section II.

In section III, CIFAR-10 images were classified using Local Binary Pattern and Support Vector Machines. In section IV, the results of CNN and SVM are compared.

II. Neural Networks

In this section, different kinds of neural network architectures will be discussed with their results and implications. These networks were implemented on MATLAB using *MatConvNet* toolbox.

In LeNet-5 architecture, several parameters such as pooling, weight decay, dropout, momentum, leaky and activation function were tweaked. Standard parameters of LeNet-5 are given in Table 1 down below.

Standard Parameters	Value
Pooling	Average
Weight Decay	0.0005
Dropout	0.5
Momentum	0.9
Function	ReLU
Leaky ReLU	Leaky=0
Learning Rate	0.1

Table 1: Standard parameters of LeNet-5 Regular Run

Now with each parameter change, the change in percentage error shall be discussed. Also, implications will be drawn from the results with each parameter change. Table 2 shows the top1err and top5err for training and testing. $\Delta\ x$ denotes the difference between the top1 error of training and testing data.

Network	1err train	1err test	Δx	5err train	5err test
Regular Run	12.9%	22.7%	9.8%	0.4%	1.8%
MaxPooling	32%	35%	3%	3.5%	4.1%
W. Decay=0.01	13%	17%	4%	0.5%	1%
Dropout=0	7%	18%	11%	0.2%	1.3%
Momentum=0.3	27%	29%	2%	2.5%	2.6%
Leaky=0.25	10%	18%	8%	0.4%	1%
Leaky=0.5	NaN	NaN	-	NaN	NaN
Sigmoid(10 ep)	86.5%	86.9%	0.4%	48.2%	48%

Table 2: Results with different change in parameters after epoch 30

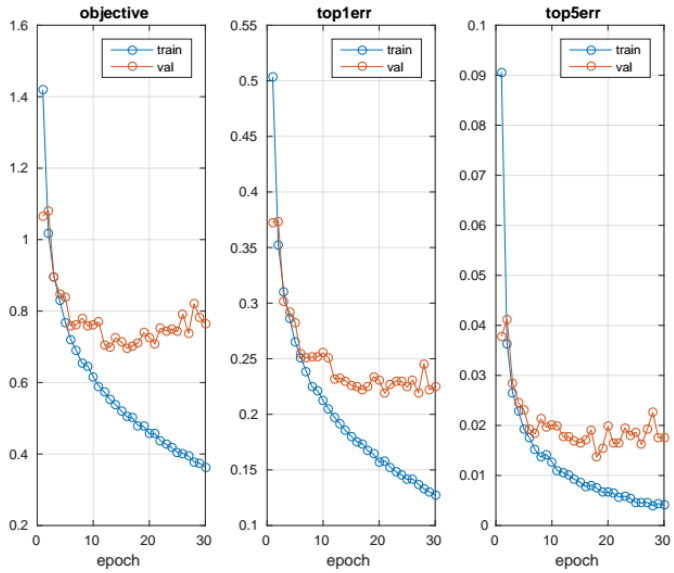

Fig. 2. Standard LeNet-5 error vs epoch

First, the learning rate was increased to 0.5 (ran for 30 epochs). This is like coarse-tuning and thus increased the training speed of the network. For learning rate 0.05, training speed decreased which acts like fine-tuning and is much slower. That is why in a neural network, learning should be in two phases: 1) relatively large initial alpha values (α= 0.3, 0.4,...0.9) or learning rate are used and then 2) small alpha values (α= 0.01, 0.02,..0.1) are used in order to achieve faster training and a fine-tuned network. Hence, α should be a function of time. For linear case:

$$\alpha(t) = \alpha(0) \cdot (1 - t/rlen)$$

where $rlen$ is the running length of the training

Another important parameter in a CNN is pooling. It is a form of non-linear down-sampling. The pooling was changed from average to maximum and the results are shown in figure 3. Comparing it with standard run (from table 2), we notice a considerable amount of increase in both training and testing error. But unlike the standard run, max pooling has Δx (difference between training and testing error) very small. The intuition this is that max pooling, one of the most common non-linear functions to implement pooling, partitions the input image into a set of non-overlapping rectangles and for each sub-region, maximum output is attained. It means that once the feature is found, its exact location is irrelevant. Its rough location is estimated relative to other feature. This reduces the amount of parameters and the computation in a network. In max pooling (with 2x2 filter, stride =2), 75% of the pixels are discarded in an image, *i.e.* a 4x4 image becomes a 2x2 image.

Another parameter to tweak is *Weight decay*. It is a regularization method that basically adds an additional error to each node which is proportional to the sum of weights or squared magnitude (though L_2 norm is not a good method) of weight vector. This prevents the weights from growing too

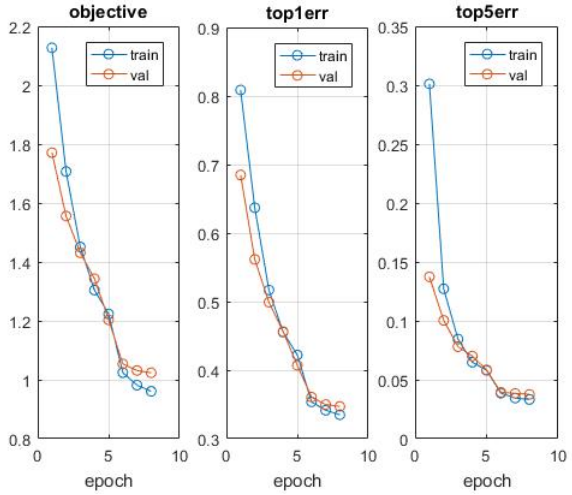

Fig. 3. Pooling = Maximum

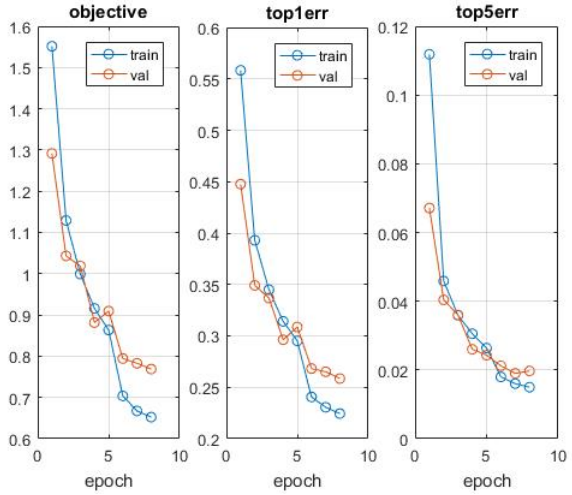

Fig. 4. Weight Decay = 0.01

large and thus it can be seen as gradient descent on a quadratic regularization term. Now, compare the weight vectors in table 1 and 2. The weight vector is doubled from 0.0005 to 0.001. There is no significant change in training error but the testing error decreased from 22.7% to 17%. Lesser the weight decay is, faster it will reach minimum error. The graph can be seen in figure 4.

In any NN, over-fitting is an issue. Since, most are the parameters are present in a fully connected layer, it becomes prone to over-fitting. Dropout is a simple method to prevent neural networks from over-fitting. With zero dropout, every node is retained *i.e.* no information is lost. We can see in the *table 2* and *fig. 5*, both the training and testing error decreases. While in the regular run, dropout is 0.5 and thus we can see that the over-fitting decreases in the network. Though, increasing the dropout improves the speed of training. Also, in case dropout=0, one can visualize that after epoch 20, the

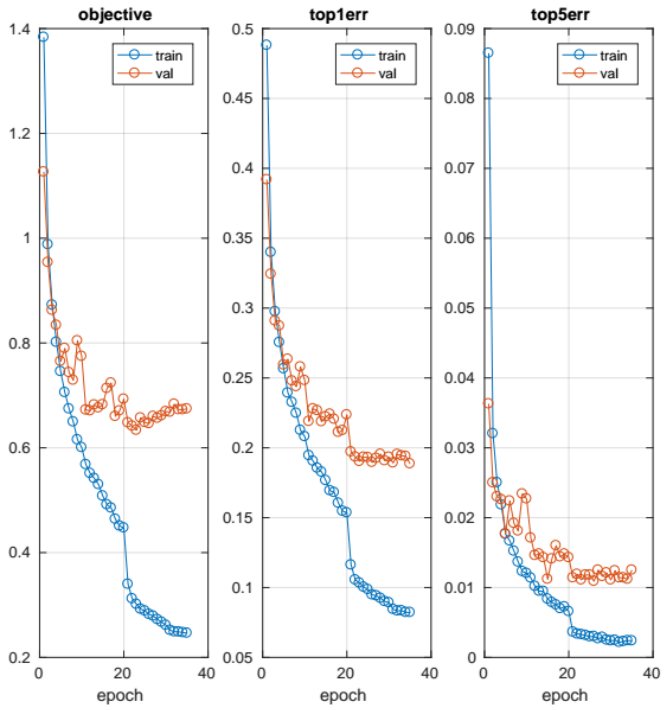

Fig. 5. No Dropout

objective testing value increases which leads to the over-fitting in the network. To summarize, dropout generally improves the performance and reduce the complexity of a neural network.

Momentum is one of the most popular extension of the back-propagation algorithm. Momentum adds a fraction, let's say m of the previous weight to the current weight. This keeps the gradient point in the direction of minima. In learning rate section, we visualized that higher the learning rate, better will be the speed but the system won't be much accurate. If we combine high learning rate with high momentum (close to 1), one might rush past the minimum with huge steps. This way, local minimum may never be achieved. In *fig. 6*, the momentum m was changed from 0.9 (standard LeNet5 value) to 0.3. So, with the decrease in momentum, the network training time increases.

Leaky ReLUs can allow a small but non-zero gradient (negative slope) when the unit node is inactive. This function $f(x)$ is defined as:

$$f(x) = 1 \ (x < 0)$$

$$f(x) = (\alpha x) + 1 \ (x >= 0)$$

where α is a small constant. Leaky ReLU was run a number of times but the result was not always consistent. With $\alpha! = 0$, initially the error rate decreased but with the increase in considerable amount of α, the network shows NaN (Not-a-Number). *Fig.7* and *table 2* compares the conditions: $\alpha = 0$, $\alpha = 0.25$ (error rate decreases significantly) and $\alpha = 0.5$ (NaN).

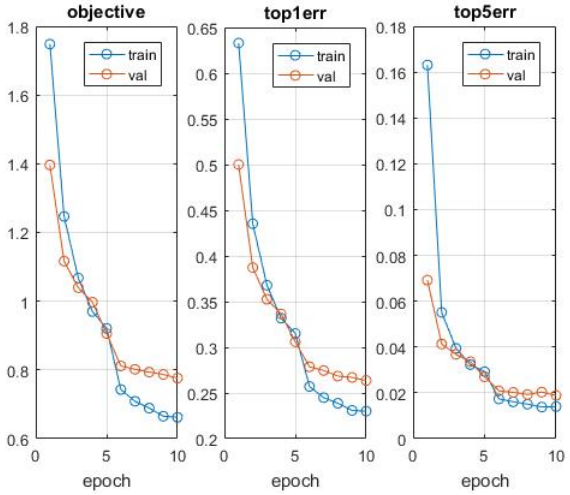

Fig. 6. Momentum = 0.3

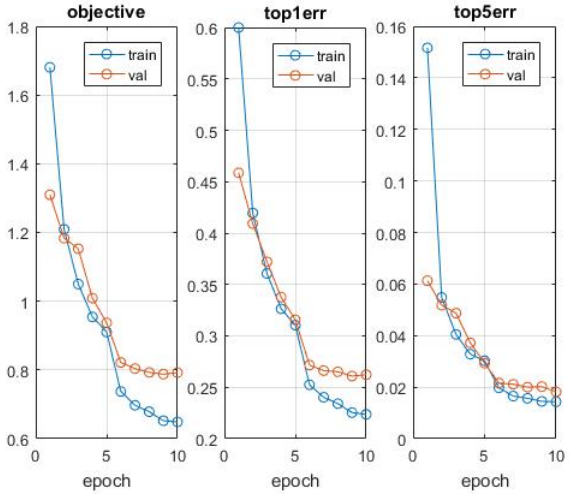

Fig. 7. Leaky = 0.25

ReLU is non-saturating activation function which increases the nonlinear properties of the decision function. Adding a non-linear activation function clearly results in neural network training several times slower as compared to a linear activation function like ReLU. Function such as sigmoid:

$$f(x) = (1 + e^{-x})^{-1}$$

results in much slower training without making any significant difference in accuracy. Clearly, in fig 8, we can see that after 10 epoch, error is more than 86%. It will probably take at least 100-200 epochs to get to 90% accuracy mark. Due to lack of computational resources, sigmoid was able to run only for about 10 epochs.

CIFAR-10 images were also classified using "Network-In-Network" (NIN). In this model, each network has a micro neural network built-in with a multilayer perceptron. This increases the accuracy of the image classification but it is ex-

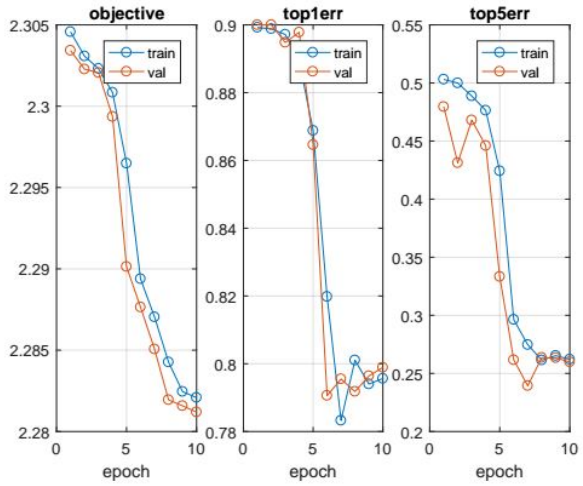

Fig. 8. Sigmoid Activation Function instead of ReLU

tremely slow as compared to conventional NN. With dropout, an accuracy of 89.64% (after 45 ep.) was achieved using NIN. which is better than the state-of-the-art(until NIN) CNN (with dropout and maxout). Without dropout=0.5, the test error rate (after 45 ep.) was 15.11%.

III. Support Vector Machines

Support vector machines combined with Local Binary Patterns (LBP) was the state-of-the-art until 2012. LBP is a good feature descriptor and the it's implementation with a multi-class SVM is easy. For CIFAR-10, one-vs-all SVM was implemented. Classifying the CIFAR-10 dataset using LBP-SVM, an error of 42.37% was observed. This was implemented in MATLAB (using *libsvm* and lbp library.)

Conclusion

Clearly, one can say that Neural network outclassed the LBP-SVM technique or any other classification algorithm in the field of image processing and computer vision. Though, in order to classify one single image, huge data set is required to train a network. LBP-SVM is fast but lacks accuracy by miles whereas CNNs requires considerable amount of computational value and processing power.

Acknowledgment

I would like to thank Nitin J. Sanket, Kiran Y.D.V and Bhargavi Patel for sharing their computer resources. In order to tweak several parameters in the given amount of time, a large number of networks were trained which exceeded the computational power of my system.

References

[1] Srivastava, Nitish; C. Geoffrey Hinton; Alex Krizhevsky; Ilya Sutskever; Ruslan Salakhutdinov (2014). "Dropout: A Simple Way to Prevent Neural Networks from overfitting" (PDF). Journal of Machine Learning Research. 15 (1): 19291958.

[2] Min Lin, Qiang Chen, Shuicheng Yan. "Network in Network" (2014). International Journal of Machine Learning.

[3] Alex Krizhevsky; Convolutional Deep Belief Networks on CIFAR-10 (2009); Master's Thesis, Department of Computer Science, University of Toronto.

[4] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton; ImageNet Classification with Deep Convolutional Neural Networks (2012)