COMP 551 Project 3 Report

Harsh Banthia, Dragos Cristian Manta, Brandon Smith

April 19, 2020

Abstract

Our task for this assignment was to compare the performances of Convolutional Neural Networks (CNN) and Multilayered Perceptrons (MLP) for classification of images from CIFAR-10 image dataset. For CNN, we achieved an accuracy of 68.04% on the test set by using Leaky ReLU as the activation function, with batch size of 8, 13 epochs, and by using Stochastic gradient descent with the learning rate of 0.001. For MLP, we achieved an accuracy of 38.81% on the test set by using an MLP shape of [3072 1024 120 10], with Sigmoid activation function, a batch size of 8, using backpropagation and L1 cost-function, and a learning rate of 0.1.

1 Introduction

Our task for this assignment was to develop models to classify image data. The models to be implemented were Multilayered Perceptron (MLP) and Convolutional Neural Network (CNN). We used the CIFAR-10 image dataset to train, test and compare the performance of the models as a function of training epochs.

2 Related Work

We looked at some studies conducted that compared the effectiveness and performance of CNN and MLP on different image sets with low resolutions [see 2]. The studies reviewed heavily favored CNN over MLP. Some deeming MLPs as insufficient for modern advanced computer vision tasks. MLPs have a high growing number of total parameters as they increase in size and layers. Making it rely on heavy computational powers. Another major drawback noticed was the use of flattened vectors and fully connected layers often disregarding spatial information, making it even more ineffective. This effect, however, was not so prominent with low-resolution images.

CNNs, on the other hand, are considered more efficient and easier to train than MLPs. They require lower computation powers since the weights are smaller and shared - they are less wasteful in terms of the resources consumed. Since it provides great flexibility with filter panning with specific sizes and strides, it is effective with local connectivity- which makes it effective in dealing with spatial information and local pattern recognition.

3 Dataset and Setup

The CIFAR-10 dataset consists of 60000 32x32 colour images and is divided into 50000 training images and 10000 test images. The train set is divided into five equal-sized batches. The test batch contains exactly 10000 randomly-selected images from each class. The classes are considered to be mutually exclusive with no overlapping between them, i.e. no image in the set can possibly be attributed to two or more classes. All data was loaded and normalized using Pytorch's DataLoader class.

4 Proposed Approach

4.1 Convolutional Neural Network

The complete list of all hyperparameters and their comments (batch size, convolutional layer parameters, activation function, loss function, early stopping tolerance, "grace" period for decreasing the loss, the maximal number of epochs to iterate over) has been placed at the top of the source code in `cnn.py` for the reader's convenience (except the optimizer, which had to be placed after the definition of `class Net(nn.Module)`.

4.1.1 Methodology

We decided to randomly split the original train data (which contains 50000 instances) into a train set (40000 instances) and a validation set (10000 instances). Then, during training, after each epoch, we evaluated the model on the validation set and recorded the average loss (per instance) as well as the accuracy for it. We also recorded the accuracy on the test set at the end of each training epoch (to satisfy project instructions). However, we didn't use this information for hyperparameter tuning. We then compiled the results in the form of plots (see Figures 1, and 2 for an example of our best runs).

4.1.2 Early Stopping

We implemented an early stopping tolerance (see Figure 2) indicating what is the minimum amount by which the loss has to decrease over a certain number of consecutive epochs (another hyperparameter) to continue training. In the example of Figure 2, if there is a period of 3 consecutive epochs at the end of which the loss doesn't decrease by at least 0.001, then the training halts.

4.1.3 Accuracy Report and Hyperparameter Tuning

During the training phase, we kept track of the history of all the accuracies on the **validation set** between the epochs (what is shown in Figure 1) and, once the training halted (due to early stopping), we looked to see after how many training epochs was this accuracy maximal. Then, we looked in the list of all accuracies on the **test set** (remember: we kept track of that as well) and found the one obtained after the same amount of training epochs and reported that as the official and final accuracy of our model on the test set. Thus a part of the Hyperparameter tuning (that is, the best number of training epochs) is already automatically done for us given the tolerance and `number_of_epochs_for_loss_improvement`. For the remaining part of the hyperparameter tuning process, we manually tuned them and judged their quality by the accuracy of the model on the **validation set** (see Table 1).

4.1.4 Network Architecture

We decided to use in total 3 convolutional layers, each followed by a max-pooling layer, and then have 4 fully connected linear layers. We used the same activation function for every (convolutional and linear) layer and, as a regularization strategy and to make the network more robust, we decided to apply a dropout with probability $p = 0.5$ to the second linear layer. We were tempted to apply dropout to each layer (and we tried that), but it led to worse performance overall, so we decided to use that sparingly. All the max-pooling kernels have the same parameters: `kernel_size` $= 2$, `stride` $= 2$ and `padding` $= 1$. The same is true with the other convolution kernels: `kernel_size` $= 4$, `stride` $= 1$ and `padding` $= 3$. The only difference between the different layers' parameters is the number of input and output channels, which can be read off from the `convolutional_parameters_array` parameter at the beginning of the source code in `cnn.py`.

4.2 Multilayered Perceptron

We implemented the MLP from scratch following a fully connected architecture for arbitrary numbers of layers and arbitrary numbers of nodes per each layer. As required, we integrated backpropagation combined with a minibatch gradient descent algorithm as an optimization algorithm. For limiting the degrees of

freedom we chose to use the same activation function across all layers, but implemented Sigmoid, ReLu, and Tanh as possible activation functions. Finally, the cost function used for computing the output layer and backpropagation was L1.

4.2.1 Methodology

As the above-mentioned methodology for CNN, for MLP we split the 50'000 training instances into 40'000 train and 10'000 validation, running the backpropagation- minibatch optimization on all 40'000 train instances for a single epoch, and then evaluating the accuracy on the 10'000 validation instances. Different from the CNN however, for MLP we did not evaluate the test accuracy for each epoch as traditionally that is considered to be bad practice and we had run all the tests before the TA's made it clear that we should also plot the test accuracy per epoch. (See Figures 3 and 4)

4.2.2 Early Stopping

Again, as above we implemented early stopping in order to avoid over-fitting problems, but for MLP we used a cutoff of 0.05 because the model seemed to overfit if we didn't see drastic enough changes between each epoch. (See Figures 3 and 4)

4.2.3 Accuracy Report and Hyperparameter Tuning

Most of the process was identical to the above described for CNN, however, for MLP we evaluated the test accuracy only at the end of the final epoch (See Table 2).

4.2.4 Network Architecture

As previously mentioned, the network architecture for our MLP design is abstracted such that the number of layers and the number of nodes per layer are parameters to be tuned alongside all the other parameters. The connectivity of the network is "Fully Connected" such that every layer's nodes are connected to every node of the following layer. (See Table 2)

5 Results

5.1 Convolutional Neural Network

In Appendix A, we present all the significant trials that we made on the CNN model with the list of hyperparameter values tried as well as the resulting final accuracies on the test and validation sets, with the number of epochs required to achieve those results (see Table 1). All the hyperparameters not mentioned in the table were kept fixed (see the complete list of all hyperparameters). Note that, because we used stochastic-based optimization methods, the results can vary by a small amount if one reproduces the experiments, but this shouldn't change the overall conclusion. In Table 1 again, we define N as being the number of consecutive epochs at the end of which the loss must decrease by the early stopping tolerance (otherwise early stopping terminates the training). It is the hyperparameter encoded in the variable called `number_of_epochs_for_loss_improvement` in `cnn.py`. We stress that the last column of that table is an **output**, not a hyperparameter. It indicates after how many epochs was the best validation accuracy attained (the role of this was detailed in section 4.1.3).

Figures 1 and 2 contain data about the run with the best combination of hyperparameters. If we observe them, we can see how useful the early stopping technique was. Indeed, towards the last epochs, we see that the loss in Figure 2 is about to stall or, even worse, increase.

5.2 Multilayered Perceptron

In Appendix B, Table 2 conveys a set of tests and their hyper-parameters that demonstrate significant findings. The table shows the list of hyper-parameters as well as the validation, and test accuracy's achieved

in N epochs. Only one result was shown for ReLu and Tanh each as both activation functions did not seem to yield good results, whereas Sigmoid resulted in fairly acceptable results. Figures 3 and 4 demonstrate the logarithmic increase expected from one epoch to the next for the best run and the run with largest number of hidden layers respectively. This pattern was present in most of the trials run. If we were to strictly follow the good practices, we would need to set the test data aside during the whole hyperparameter tuning process, report the combination of hyperparameters yielding the highest accuracy on the **validation** set, and then test the model with those parameters on the test set once and report **that** as the final test accuracy. Following this logic, we need to report an accuracy of 38.81 % as opposed to 39.23 % (we pretend that we don't know that the 39.23 % exists for the sake of good practice).

6 Discussion and Conclusion

Based on the results we obtained, it can be determined that the Convolutional Neural Network proved to be a better classifier than the Multilayered Perceptron. The results achieved in this paper appear to hold true to the findings in researching related works (as mentioned above) in that MLP's appear to struggle with more complex tasks such as image recognition and CNN's appear to significantly improve classification accuracies on said tasks. Both models demonstrated expected behaviours in the accuracy increase per epoch, both increasing logarithmically as the number of epochs increased linearly. A major drawback that impacted our experiments is the lack of computational power. Serious Machine Learning researchers have dedicated servers and run their model on several GPU's in parallel for a very long time. Moreover, the implementation of the MLP from scratch made it very difficult to optimize for speed, so a single experiment has taken us more than one night in one extreme case. The CNN in this project benefited from ready-to-use already optimized libraries, which helped explore more possibilities without fear of running out of time. An improvement that we would like to bring, if we had to redo all of this again with more time on our hands, is to add many more layers (of all types) in the CNN to exploit its expressive power.

7 Statement of Contributions

Brandon: MultiLayered Perceptron Code, Figures, and description.
Cristian: Convolutional Neural Network Code, Figures, and description.
Harsh: Abstract, Related Work Research, Dataset description.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC.

[2] S Ben Driss et al. "A comparison study between MLP and Convolutional Neural Network models for character recognition". In: *Real-Time Image and Video Processing 2017*. Vol. 10223. International Society for Optics and Photonics. 2017, p. 1022306. URL: https://hal-upec-mlv.archives-ouvertes.fr/hal-01525504/document?fbclid=IwAR0q2lRInYOAI6jlsvacI0zeUlZHei0PHrAg8VlLC9N-6sIKNi6MseHoEe0.

[3] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *CIFAR10*. University of Toronto, Apr. 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[4] Popescu Marius et al. "Multilayer perceptron and neural networks". In: *WSEAS Transactions on Circuits and Systems* 8 (July 2009).

[5] Kevin P. Murphy. *Machine Learning : A Probabilistic Perspective*. MIT Press.

[6] Various. *Towards Data Science*. URL: https://towardsdatascience.com/.

[7] M Xin and Y Wang. "Research on image classification model based on deep convolution neural network". In: *J Image Video Proc* 8 (2019). URL: https://doi.org/10.1186/s13640-019-0417-8.

Appendices

A Results for the CNN

batch size	N	activation	optimizer	best validation accuracy (%)	accuracy on the test set (%)	number of epochs
4	7	leaky ReLU	SGD (lr = 0.001)	67.32	66.62	13
8	3	leaky ReLU	SGD (lr = 0.001)	68.02	68.04	13
16	3	sigmoid	SGD (lr = 0.01)	10.55	9.94	3
16	7	ReLU	SGD (lr = 0.001)	67.08	66.76	15
16	5	ReLU	SGD (lr = 0.001)	67.02	66.68	17
8	3	tanh	SGD (lr = 0.001)	65.74	65.52	12
8	3	leaky ReLU	Adam (lr = 0.001)	63.65	63.84	9

Table 1: Summary of the experiments with the CNN model. The row colored in green indicates the combination of hyperparameters with the best accuracy on the test set. The left half of the table represents hyperparameter values, while the right half represents outputs.

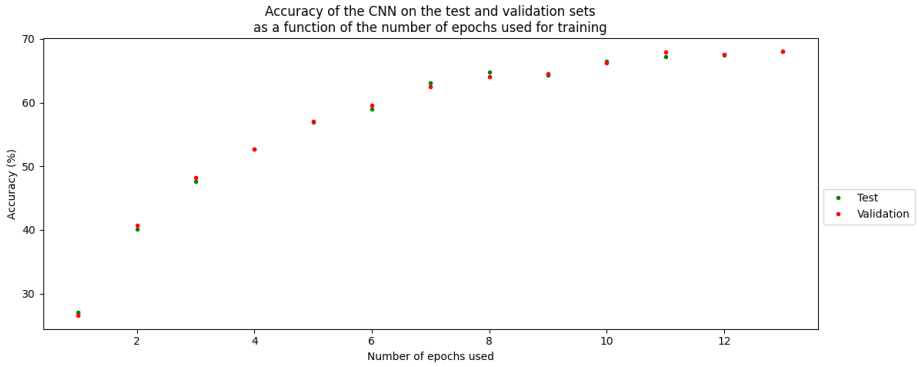

Figure 1: CNN run with the best combination of hyperparameters.

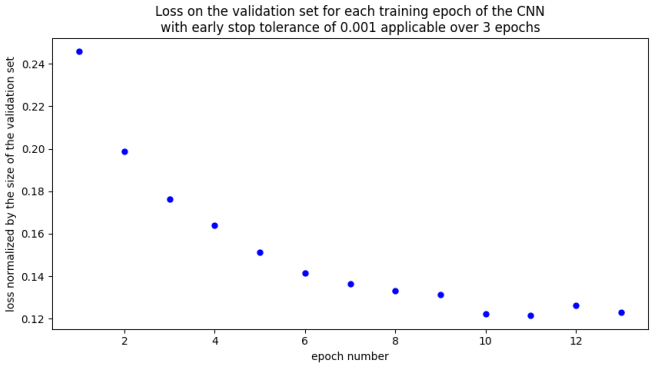

Figure 2: CNN run with the best combination of hyperparameters.

B Results for the MLP

Run Identifier	MLP Shape	Acti-vation	Cost	Batch Size	Learning Rate	Best Validation Accuracy (%)	Accuracy on Test Set (%)	Number of Epochs
Largest Hidden	[3072 6144 1024 120 10]	Sigmoid	L1	8	0.1	31.00	31.72	5
Best	[3072 1024 120 10]	Sigmoid	L1	8	0.1	39.44	38.81	4
ReLu with Best	[3072 3072 120 10]	ReLu	L1	8	0.1	9.73	10.00	4
Tanh with Second Best	[3072 1024 120 10]	Tanh	L1	8	0.1	14.48	13.73	3
Second Best	[3072 3072 120 10]	Sigmoid	L1	8	0.1	39.22	39.23	4

Table 2: Summary of the experiments with the MLP model. The row colored in green indicates the combination of hyperparameters with the best accuracy on the **validation** set

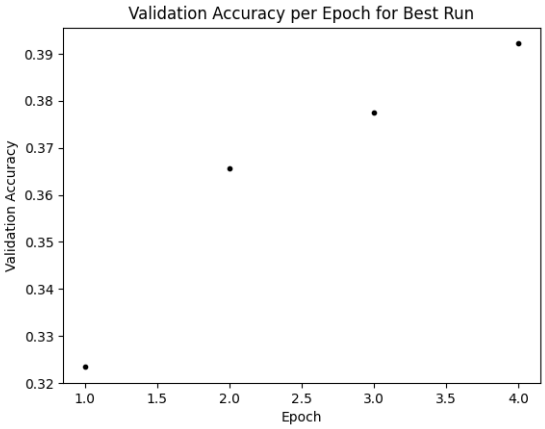

Figure 3: MLP run with the best combination of hyperparameters.

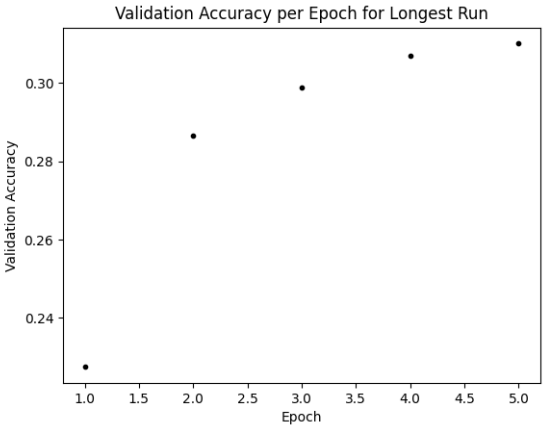

Figure 4: MLP run with the largest set of hidden nodes.