Neural Networks, Convolutions and TensorFlow

Jose Gallego
University of Amsterdam
jose.gallegoposada@student.uva.nl

Abstract

One of the core problems in Computer Vision is that of Image Classification, which consists of assigning a label to an image from a fixed set of categories. In this work we explore Neural Network-based approaches towards solving this problem using the CIFAR-10 dataset. We perform extensive experimentation on hyperparameter effect for the MLP architecture. According to our experimental results, the convolutional architectures achieved significantly higher performance.

1 Introduction

One of the core problems in Computer Vision is that of Image Classification, which consists of assigning a label to an image from a fixed set of categories. In spite of the apparent simplicity of the task (for a human observer in general) the complexity arises from the need to deal with challenges as changes in the illumination conditions, deformation of the objects to be classified, and scale and viewpoint variations. A more thorough description of the problem and its relevance in Computer Vision is provided in [1].

In this work we use the CIFAR-10 dataset [2], which contains 60000 32x32 colour images in 10 mutually exclusive classes, with 6000 images per class. There are 50000 training images and 10000 test images. A sample from the dataset is displayed in Figure 1.

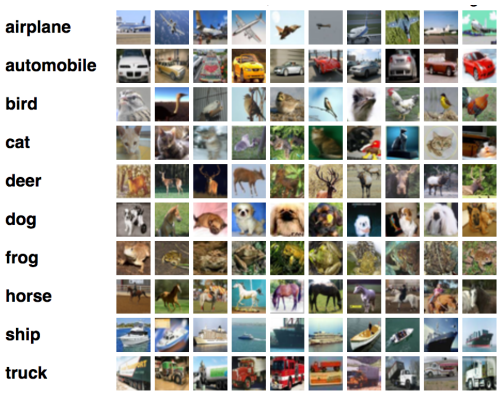


Figure 1: Samples from each of the classes in the CIFAR-10 dataset.

2 Task 1

The goal of this task was to implement a standard Multi-Layer Perceptron (MLP) using purely NumPy routines. Note that a NumPy implementation requires the computation of the gradients explicitly (see

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

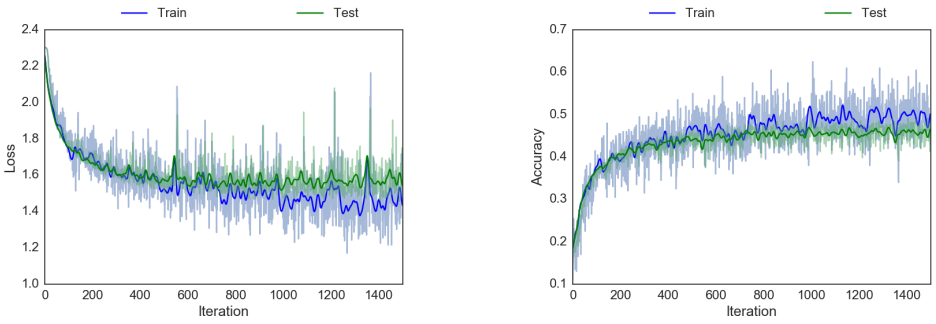
pen and paper exercises), instead of using the automatic differentiation paradigm of Tensorflow, to be explored in coming sections.

Along with the implementation itself, several features were included to improve the numerical stability and computational efficiency: all forward and gradient computations are performed with matrix operations; internal states of the layers as (pre)activations are cached to boost backward propagation efficiency; softmax computation resorts on the log-sum-exp trick [3] for greater improved stability.

This MLP was trained with the following parameters:

- Architecture: $[32 \times 32 \times 3, 100, 10]$
- Learning rate: $2 \cdot 10^{-3}$
- Weight initialization: $\mathcal{N}(0, 10^{-4})$
- Bias initialization: 0
- Optimization Method: Mini-batch Stochastic Gradient Descent
- Mini-batch size: 200
- Training steps: 1500

The training and testing performance of our MLP implementation is illustrated in Figures 2a and 2b. For clarity, bold lines represent Hanning smoothing of the actual values shown in transparent. The aforementioned parameters achieved 46.5% training and 45.3% test accuracy. Note that the loss starts from a value close to $2.3 = -\ln(\frac{1}{10})$, which correctly corresponds to the initial guesses due to the random initialization of the network. Besides, due to the lack of regularization, the training performance is consistently better than the testing performance.



(a) Loss (b) Accuracy

Figure 2: Statistics for the NumPy MLP implementation training.

3 Task 2

As in Task 1, an MLP was used to solve the Image Classification problem. The experiments in this section involve a Tensorflow implementation.

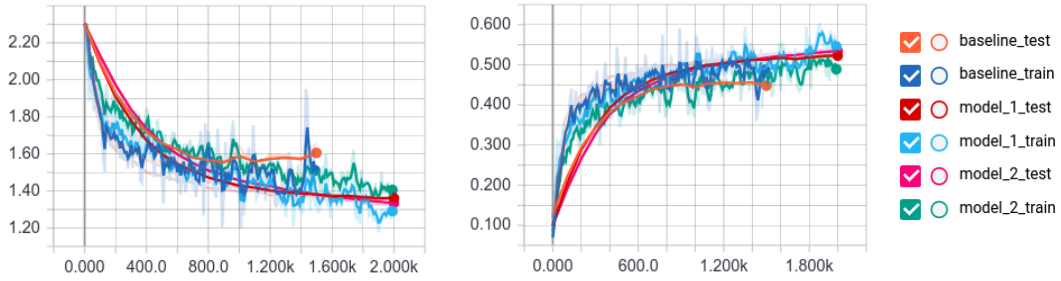
3.1 Manual Hyperparameter Tuning

Name	Architecture	Optimizer	Learning Rate	Steps	Dropout	Activation	Weight Initialization
Baseline	100	Adam	$2 \cdot 10^{-3}$	1500	0	ReLU	$\mathcal{N}(0, 10^{-4})$
Model 1	200, 200	Adam	$3 \cdot 10^{-4}$	2000	0.1	ReLU	$\mathcal{N}(0, 10^{-4})$
Model 2	500, 500	Adam	$7 \cdot 10^{-5}$	2000	0.2	ReLU	$\mathcal{N}(0, 10^{-4})$

Table 1: Parameter configurations for the manually explored models.

The *Baseline* model trained with the provided parameter settings achieved a performance of 47.5% training and 43.6% test accuracy. There is not a significant generalization gap, which would otherwise

be an indication of overfitting. For this reason, we have increased the model capability as well as the number of training steps and reduced the learning rate in hope to get a better performance. Additionally, we included 10% droput as a regularization to avoid overfitting in this larger model, denoted as *Model 2*. This model achieved a performance of 53% training and 52.5% test accuracy. Again, this model did not seem to overfit, and so we trained a more complex setting denoted by *Model 3*. In this case we have increased the number of neurons in the hidden layers, reduced again the learning rate and strengthened the dropout regularization to 20%. Finally, this model obtained 51% training and 53.8% test accuracy.



(a) Loss (b) Accuracy

Figure 3: Statistics for the manual hyperparameter tuning.

Figure 3 displays the training evolution of the aforementioned models. The test curves are smoother as the measurements are not made every training step. Figure 4 clearly shows how the two proposed models quickly diffuse the initially peaked predictions from just one class and finally achieve more uniform distributions for the predicted test labels than that of the baseline model.

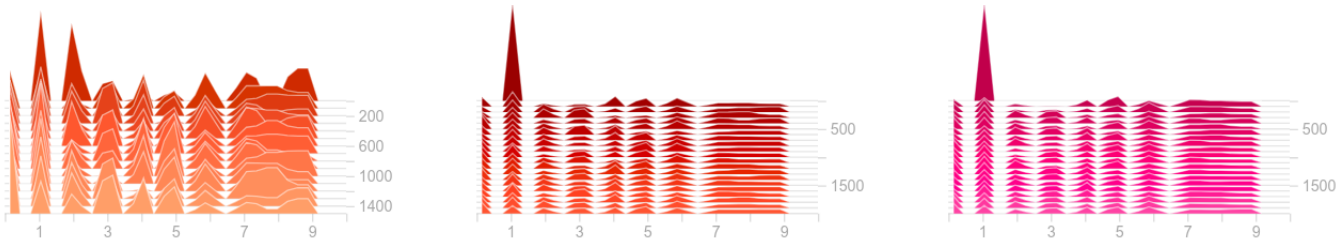


Figure 4: Predicted class histograms for test data. From left to right: baseline, model 1, model 2.

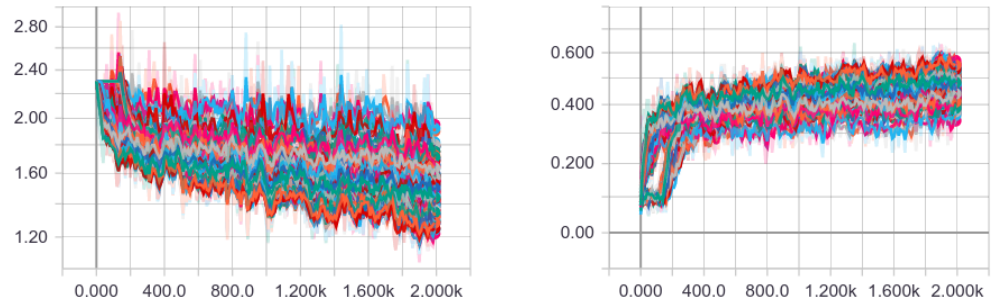
3.2 Grid Search

We conducted a grid search over 144 possible hyperparameter configurations to explore the effect of these choices and to test whether it was possible to improve the performance obtained in the previous experiments. The possible parameter values were as follows:

- Learning rate: $5 \cdot 10^{-4}$, $3 \cdot 10^{-4}$, $7 \cdot 10^{-5}$
- Weight initialization: Normal, Uniform
- Architecture: 100, $250-250$, $500-500$
- Activation: ReLU, ELU
- Optimizer: Adam, RMSPROP
- Dropout: 10%, 20%

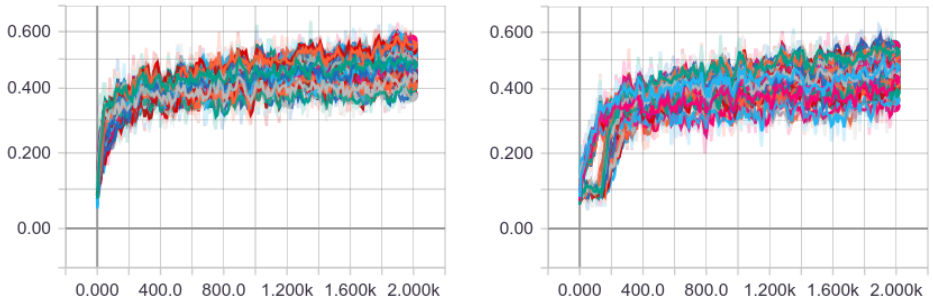
The general training statistics for all the experiments are shown in Figure 5. Clearly, there is a large diversity in the training behaviour: some configurations are very stable in learning, while some exhibit plateaus initially but after several iterations move towards regions with larger gradients; and of course, there is high variability in the accuracy of the trained models ranging from 35% to 60%.

Optimizer Initially we compare the performance of Adam and RMSPROP. Figure 6 shows the training and test accuracy for both optimizers. Note that the initial plateau behaviour mentioned earlier only occurred for RMSPROP and several of this configurations achieved poor performance,

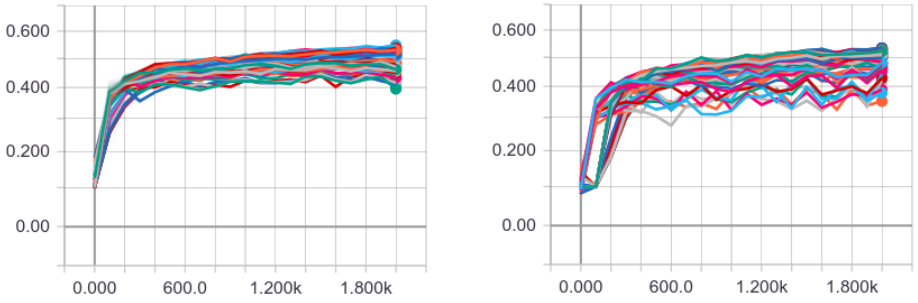


(a) Loss (b) Accuracy

Figure 5: Training statistics for all parameter configurations in grid search.



(a) Adam - Train (b) RMSPROP - Train



(c) Adam - Test (d) RMSPROP - Test

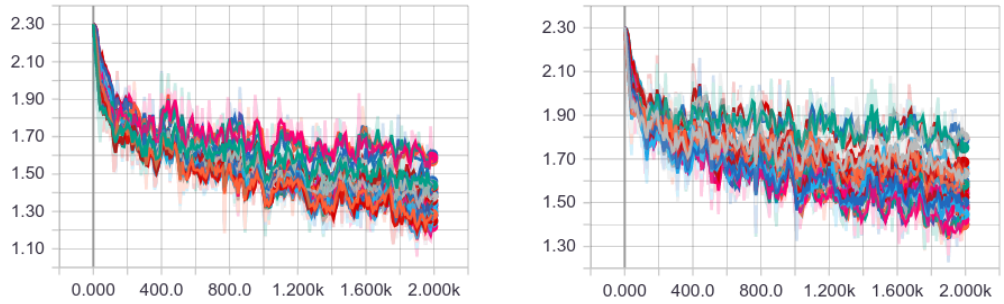
Figure 6: Comparison of optimization methods: Adam vs RMSPROP.

however this might be due to initialization conditions and a possible need for a higher learning rate. More importantly, we remark that Adam obtained higher lower bounds for training and testing accuracy with a clearly visible variance reduction with respect to RMSPROP.

In particular, Adam seemed to encourage small learning rates: for a learning rate of $5 \cdot 10^{-4}$ the test accuracy ranged between 39% and 51.5%, while for a learning rate of $7 \cdot 10^{-5}$ the test accuracy was always above 50% and reached 54.85%. The best RMSPROP model achieved 53.35% test accuracy.

Activation We did not discover significant difference between networks with ELU and ReLU activations. As a matter of fact, the top two models (in terms of test set accuracy) have the same parameter configuration except for the activation function.

Droupout We have introduced dropout in the network as a regularization mechanism to avoid overfitting. Figure 7 shows how the magnitude of the dropout affected the training loss. As expected, the model with the highest dropout rate incurred in larger losses due to the stochasticity in the neuron activations.



(a) Dropout 10%

(b) Dropout 20%

Figure 7: Effect of dropout rate in training loss. Note the difference in vertical scales.

Best Model After executing the grid search, the model with the best test set accuracy improved the results obtained in the manual hyperparameter tuning and reached 54.85%. This was achieved with a similar architecture than that in the previous section and is described in Table 2.

Architecture	Optimizer	Learning Rate	Steps	Dropout	Activation	Weight Initialization
500, 500	Adam	$7 \cdot 10^{-5}$	2000	0.1	ELU	$\mathcal{N}(0, 10^{-4})$

Table 2: Best grid search parameter configuration

Figure 8 displays the test set confusion matrix for the best model at the beginning and end of training. Note how the diffusion in the classifications gets removed over time and the model more accurately classifies the unseen test images.

(a) 100 Steps

(b) 2000 Steps

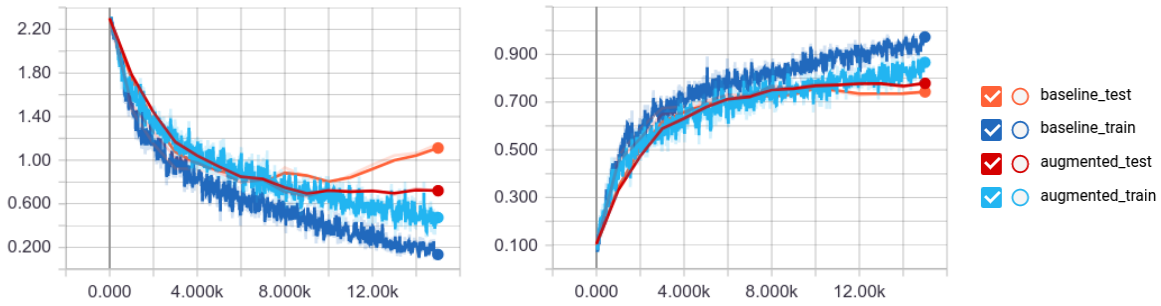
Figure 8: Test set confusion matrix for the best model.

4 Task 3

In this task we implemented a pre-defined CNN architecture in Tensorflow. We compared the baseline model with default hyperparameters against the addition of data augmentation and 10% dropout. The data augmentation was executed by zooming and/or rotating the training images, which makes sense due to the nature of the images: a picture of cat rotated some small amount is still a picture of a cat.

The inclusion of data augmentation and dropout is justified by the overfitting behaviour observed in the baseline model and displayed in Figure 9. Observe that the training loss is steadily decreasing, however around 10.000 steps the test loss starts to increase, which is a clear sign of overfitting. Note how the test loss in the case of enhanced model does not increase but rather stays flat.

The baseline model achieved 97.6% training and 74.6% test accuracy, while the model trained with augmented data and dropout reached 86.7% training and 78.5% test accuracy. There is clearly an advantage in the use of convolutional architectures for this kind of task with respect to MLP structures, which do not preserve spatial information.



(a) Loss (b) Accuracy

Figure 9: Statistics for the CNN models training.

5 Conclusion

In this work we explored the Image Classification task by implementing MLP and CNN architectures both in NumPy and Tensorflow and training such models using the CIFAR-10 dataset. We have performed extensive experimentation on hyperparameter effect for the MLP architecture. Finally, the spatial information retained by convolutional models provides a significant advantage against standard MLP models. However, the large number of parameters in the trained CNN makes the inclusion of regularization (either in the form of weight decay or dropout) a necessary component to avoid overfitting.

References

[1] A. Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/classification`, 2016. [Online; accessed 23-November-2017].

[2] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[3] Wikipedia. LogSumExp — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/LogSumExp`, 2017. [Online; accessed 23-November-2017].