# Assignment 1: Neural Networks, Convolutions and TensorFlow

**Dana Kianfar**
11391014
University of Amsterdam
dana.kianfar@student.uva.nl

## Abstract

In this report we explore image classification on the CIFAR-10 dataset using Multi-layer Perceptrons (MLP) and Convolutional Neural Networks (CNN). We first present an implementation of an MLP written from scratch using `NumPy` and present performance results using a pre-specified architecture. Next we present results from a variety of MLP architectures with different parameter settings that were implemented in `Tensorflow`. Finally, we present a CNN model with optimizations such as dropout, batch normalization and data augmentation. We study the characteristics of our models and compare different architectures and parameterizations.

## 1 Introduction

Image classification is a common benchmarking task in machine learning and computer vision. In this assignment we used the CIFAR-10 dataset, which contains 60000 colour images in 10 classes. The dataset is split to 50000 training images and 10000 test images. The 10 classes are well-balanced and each hold 6000 images in total. The size of each image is $32 \times 32 \times 3$. The classes are displayed in Figure 1.

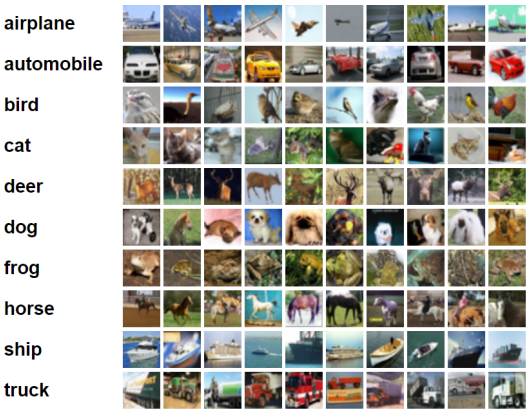


Figure 1: CIFAR-10 dataset classes

## 2 Task 1: Multi-Layer Perceptron in `NumPy`

MLPs are the standard feed-forward neural network (NN) architecture, where a sequence of fully-connected layers act as compositional non-linear functions that transform input data to output labels.

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

Each fully connected layer is a simple regressor that takes as input the activations of the previous layer (or the data at the first hidden layer), and performs regression (commonly called pre-activations). The activations of a layer is often a non-linear map of its pre-activations. Despite the simplicity of each individual layer, the predictive power of neural networks arise from the non-linear composition of several such simple layers.

| Hidden | Steps | Learning Rate | Weight Init. Scale | Train Loss | Test Loss | Train Accuracy | Test Accuracy |
|---|---|---|---|---|---|---|---|
| 100 | 1500 | 2e-3 | 1e-4 | 1.614 | 1.622 | 0.43 | 0.447 |

Table 1: Train and test performance for a simple neural network structure. A batch size of 200 was used to train the model.

We train a simple neural network with the provided parameters; it has a single hidden layer of 100 units with relu activations, the weights are initialized by a zero-mean normal distribution with $\sigma^2 = 1e-4$. We used the cross-entropy objective and trained with stochastic gradient descent (SGD). The training/testing cross-entropy loss and accuracy are presented in Table 1 and Figures 3 and 2. We observe that despite the typical noisy SGD behaviour, the network converges after 1500 steps.

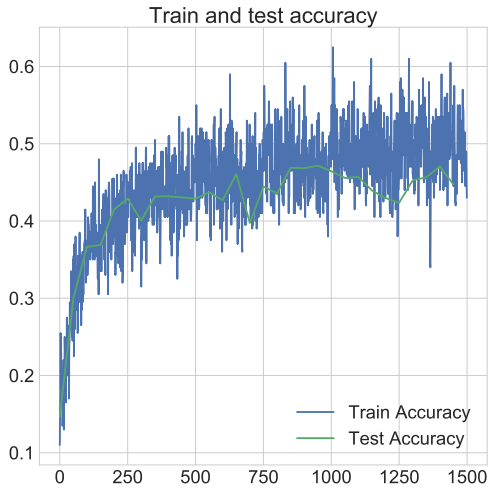


Figure 2: Train and test accuracy

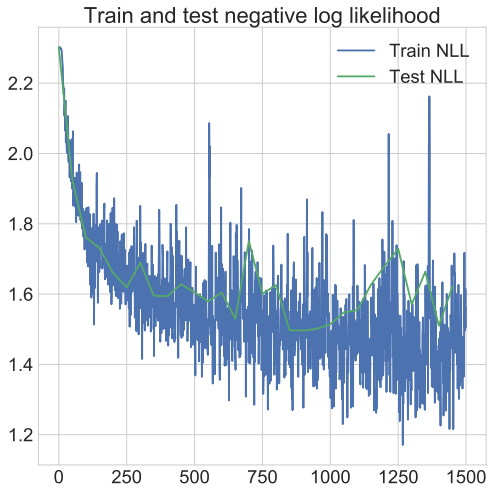


Figure 3: Train and test cross-entropy error (NLL)

To ensure the validity of our model, a number of optimizations are applied. During forward propagation, we cache pre-activation and activation values so that they can be used during backpropagation. To avoid singularities, when computing the softmax function we subtract the maximum value from the logits of each training sample. This ensure that the logits are close to zero, avoiding numerical underflow or overflow in the exponential terms. This is equivalent to the log-sum-exp trick, if one were to take the log of the softmax values.

Furthermore, we collect debugging statistics during training and raise warnings whenever necessary. One common case is exploding gradients. To this end, I monitor the norm of the logits, error signals (deltas), and the gradients themselves. The progression of these norms is presented in Figures 20, 5, and 6. An important sanity check is for these values to stabilize after some training rounds. We also observe in Figure 5 that the lower layer (in red) consistently has a smaller gradient norm than the higher layer (in blue). This is indeed desirable behavior as the error signal naturally vanishes as it reaches lower layers.

## 3 Task 2: Multi-layer Perceptron in `TensorFlow`

Discovering good (hyper-) parameter settings is difficult because parameters interact and have complex dynamics that are hard to understand. To this end, we performed several grid search experiments where various parameters and architectures were used. Among many experiments, we

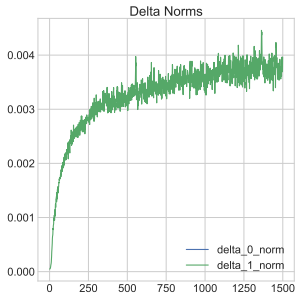


Figure 4: Norm of $\delta$

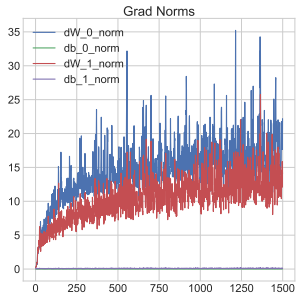


Figure 5: Norm of gradients

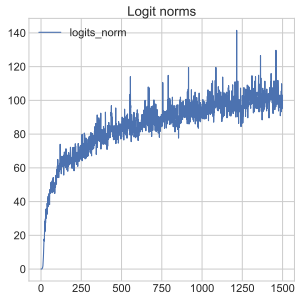


Figure 6: The norm of logits

only kept the most promising which are presented in this section. Our search was performed over values that were empirically shown to yield good performance, and can be summarized as follows.

- **Architecture**: $100$, $300 \times 300$, $500 \times 500$, $200 \times 200 \times 200$, $600$
- **Regularization**: L1 and L2 with $\lambda \in \{3e-5, 5e-3\}$. Dropout with rate 0.6
- **Activation**: relu and elu
- **Learning rate**: $\eta \in \{3e-2, 3e-4\}$
- **Initialization**: normal and uniform, with scale $s \in \{1e-5, 1e-3\}$
- **Optimizer**: Adam, SGD

All experiments were allowed to run for 6000 steps, however to prevent overfitting and to stop failed experiments, we introduced early stopping. After 500 training steps, the network continuously compares it's test accuracy against a moving-window average of the past 10 test accuracies. If the different between these two values falls below $1e-4$, then the training routine halts.

While not all the experiments were successful, we review our observations in each category below and compare all models with respect to their test accuracy. If a trained model has a test accuracy of 50% or below, we discard it from our results.

### 3.1 Architecture

We display three architectures with the highest test accuracy in Figures 7, 8, and 9. Our experiments with shallow (but wide) networks were largely unsuccessful. We attribute this to the idea that the predictive power of neural network strongly relies on its depth and composition of non-linear maps between its layers. We can observe that the $500 \times 500$ architecture performs better compared to the other two architectures. It should be noted that the models which are plotted in each figure may or may not have regularization, which plays a key role in generalization of deep and large neural networks.

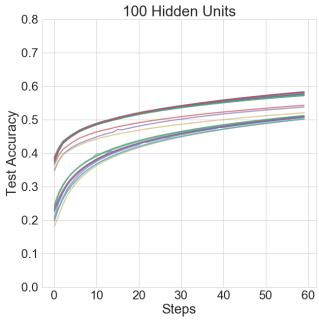


Figure 7: 100 hidden units

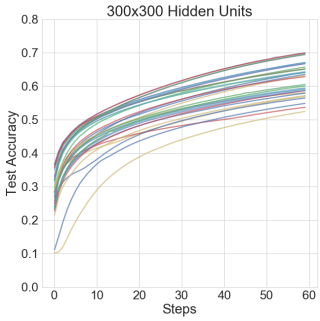


Figure 8: Two layers of 300 units each

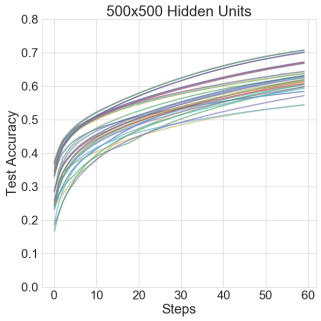


Figure 9: Two layers of 500 units each

### 3.2 Weight Initialization

Our experiments with uniform and normal weight initialization are presented in Figures 10 and 11. We experienced a strong sensitivity to this parameter where we observed a dynamic between certain weight initialization and the learning rate, where bad combinations would prevent the model from achieving better than random test accuracy. We found large initialization scales such as $[0.1, 1]$ to be highly problematic in that they would lead to numerical underflow. Choosing a small enough weight initialization scale helps to maintain stationary and additionally helps us avoid mean activation imbalance across the layers with relu activations.

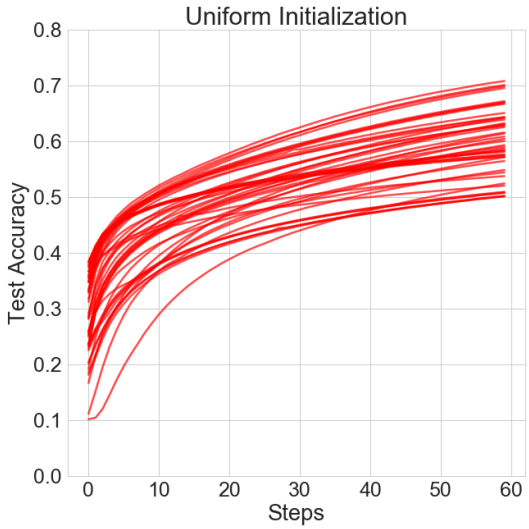


Figure 10: Uniform Initialization

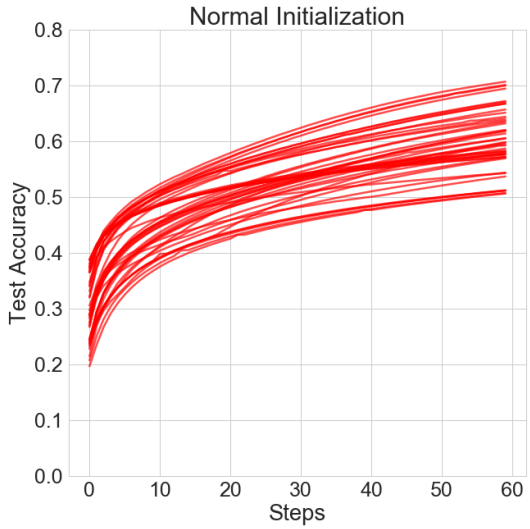


Figure 11: Normal Initialization

### 3.3 Weight Regularization

Overfitting is a very common obstacle to training neural networks with good generalization capabilities. One common way to combat overfitting is regularization. Using a scheme such as weight decay, where the norm of the network parameters are added to the objective function, allows us to regularize our models without significant computational overhead. Furthermore, weight decay is simple to understand and has a probabilistic interpretation. The most common method to perform weight decay is by placing a standard normal prior over each individual weight scalar in our network. The negative log prior corresponds to the deviation of the weights from the standard normal distribution. This forces the weights to stay close to zero, and can easily be cast as an optimization problem. Weight decay corresponds to performing maximum a posteriori inference instead of maximum likelihood.

In addition to L2 regularization, we also experimented with L1 and dropout regularization. L1 regularization can be understood as weight decay with a Laplace prior on the weights instead of a standard normal. This encourages sparsity among the weights. Dropout is a technique where randomly selected activations are set to zero. This introduces noise into the network and does not allow the weights to co-adapt too far and overfit on the data.

We found regularization to be a strong contributor to good performance. We visualize our results with respect to regularization in Figures 12, 13, and 14. Dropout with a rate of 0.6 was found to be very useful for achieving good test performance. In our weight decay experiments, we set the regularization strength to $\lambda \in \{3e-5, 5e-3\}$. Figures 12 and 13 display $\lambda = 3e-5$ by red, and $5e-3$ by blue. We observe that L1 weight decay is more sensitive to these parameters. This behavior can be attributed to the sparsity-inducing property of L1 weight decay, which in the case of $\lambda = 5e-3$ we can argue that it is too large and causing networks to underfit.

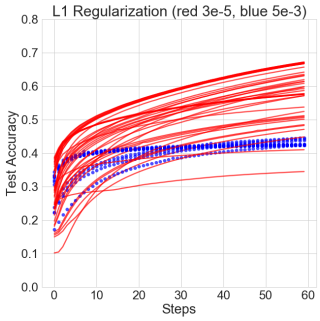


Figure 12: Uniform Initialization

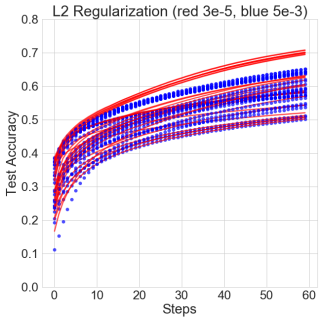


Figure 13: Normal Initialization

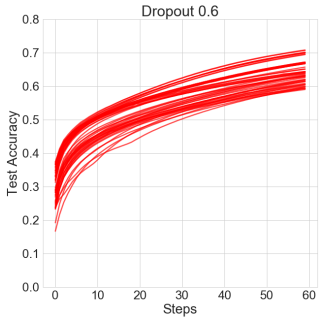


Figure 14: Dropout with rate 0.6

### 3.4 Activation Functions

Activation functions play a central role in neural networks in that their non-linearity allows the network to learn complex functions. In addition to lending predictive power to the network, a good activation function should have two properties. Firstly, it should ideally produce zero-centered values to avoid covariance shift, i.e. difference activation distributions between the layers. Secondly, a good activation function should have strong gradients. This is especially important in very deep neural networks where the gradients can shrink rapidly when being multiplied across layers. In our experiments, visualized in Figures **??** and 16, we show that the ELU activations performs better than ReLU. ELU units tend to maintain zero-centered activations better than the ReLU.

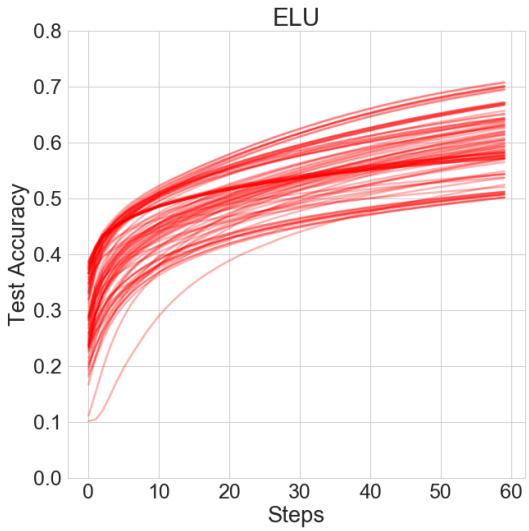


Figure 15: ELU Activations

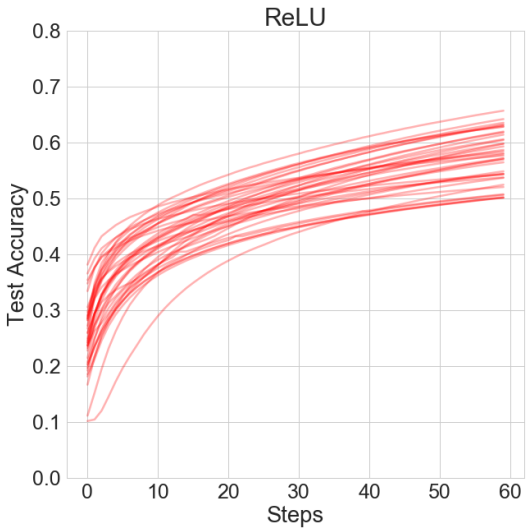


Figure 16: ReLU Activations

### 3.5 Optimizer

In our experiments with the optimizer, shown in Figures 17 and 18, we found Adam to be a better optimizer than SGD. This is not a surprising result, as Adam uses momentum and advanced optimization to make a better estimate of the gradient. Not only was Adam able to learn faster, it also discovered better local optima.

We found SGD with a learning rate $\eta = 3e - 4$ to be a slow learner, while the same learning rate was the best in our grid search with ADAM. Conversely, Adam overshot with $\eta = 3e - 2$. Since SGD does not adjust it's learning rate, it is better to start with a larger $\eta$ and decrease it according to a decay schedule. One the other hand, Adam adjusts its own learning rate and thus a smaller learning rate often suffices.

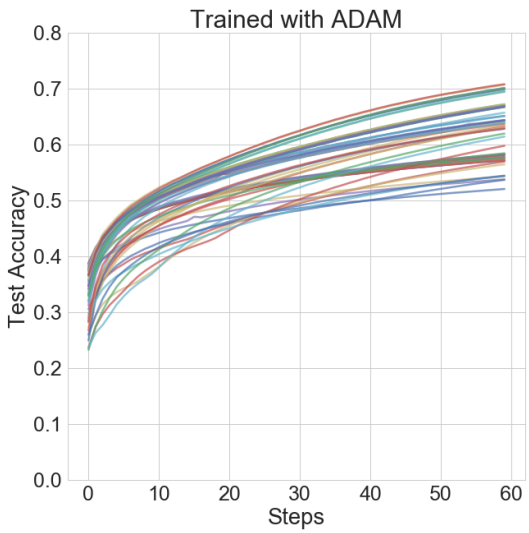


Figure 17: Adam Optimizer

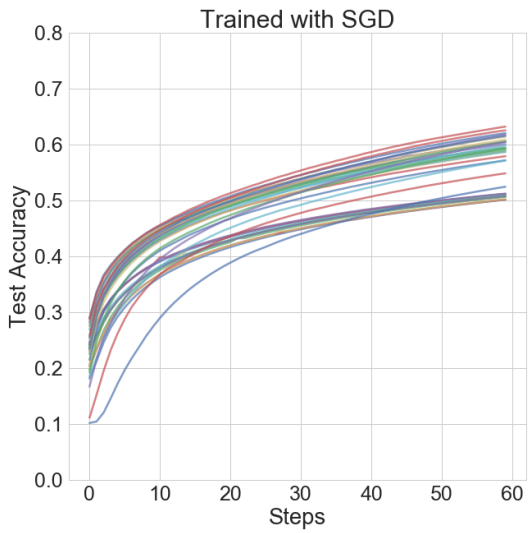


Figure 18: SGD Optimizer

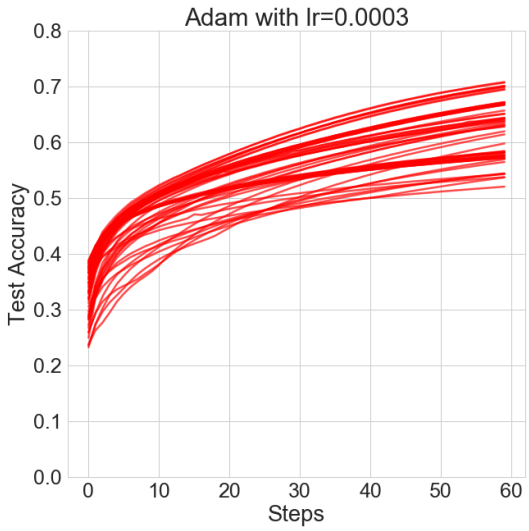


Figure 19: Adam with $\eta = 3e-4$

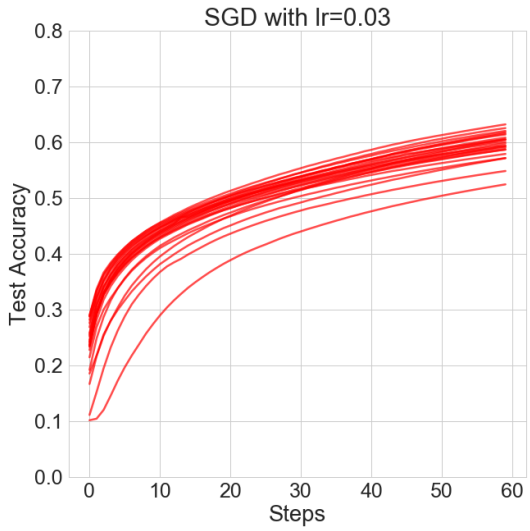


Figure 20: SGD with $\eta = 3e-2$

### 3.6 Best Model

Our best model in terms of test accuracy is shown in table 2. The large discrepancy between the train and test loss however could suggest that the model is overfitting.

| Hidden | Learn. Rate | Weight Scale | L2 Weight | Activation | Train/Test Loss | Train/Test Accuracy |
|---|---|---|---|---|---|---|
| $500 \times 500$ | 3e-4 | 1e-3 | 3e-5 | ELU | 1.42 / 2.59 | 70.71 / 70.69 |

Table 2: Train and test performance for our best MLP. A batch size of 256 was used to train the model.

### 3.7 Confusion Matrix

We plot the confusion matrix of out best model before (Figure 21) and after (Figure 22). The brightness of the diagonal reflects the ration of true positives to false positives. The model beings with random guesses and converges to a more stable regime.

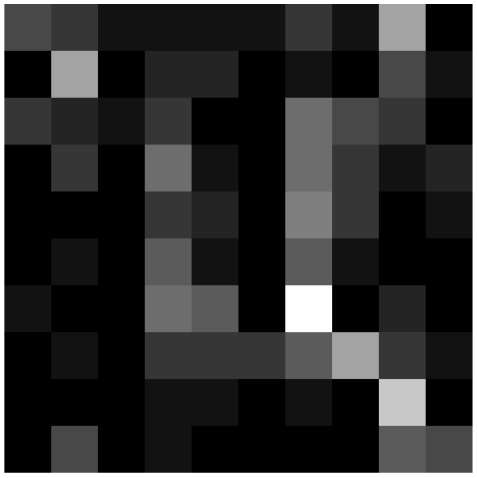

Figure 21: Confusion matrix of best model before training

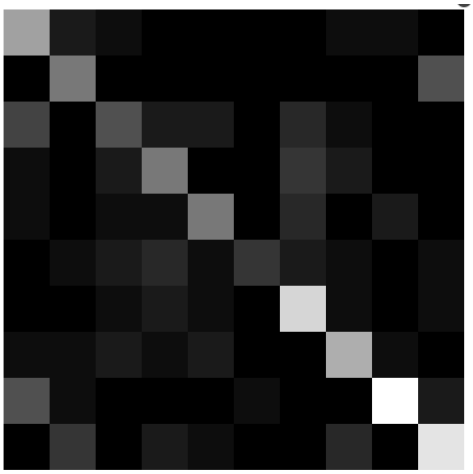

Figure 22: Confusion matrix of best model after training

### 3.8 Misclassification Examples

Figure 23 presents a few examples of over-confident misclassifications performed by our best model. In these cases, there are a lot of similarity between the false and the true labels. For example, a bird and an airplane can look alike from a distance. Similarly, a ship in a blue see resembles an airplane in a blue sky.

Figure 23: Misclassification examples. From left to right, they were tagged as bird (airplane), airplane (ship), airplane (bird), truck (car), dog (cat).

## 4 Task 3: Convolutional Neural Networks (CNN)

A better way to approach image classification is with CNNs. In our previous experiments with MLPs, we flattened each image into a vector and essentially lost all spatial information between pixels. This is an important aspect of image data, as a single pixel does not carry much information but a local neighborhood of pixels carry a lot of information. CNNs are precisely suited for this aspect. Moreover, the convolutional filters learned are shared across the surface of the image. This allows CNNs to hierarchically learn low to high level features about images.

Due to time constraints and lack of capable hardware, we were not able to perform a grid search over possible parameters. We present our results in Table 3 and display its learning curves in Figure 24.

| Hidden | Learn. Rate | Weight Scale | Grad Clip. | Dropout | Train/Test Loss | Train/Test Accuracy |
|---|---|---|---|---|---|---|
| $500 \times 500$ | 1e-4 | 1e-4 | [-1., 1.] | 0.6 | 0.2379 / 0.763 | 74.46 / 76.31 |

Table 3: Train and test performance for our best CNN. A batch size of 256 was used to train the model.

### 4.1 Data Augmentation and Batch Normalization

We experimented with augmenting the data by rotation, shift and zoom transforms. While in principle data augmentation can improve generalization, our best model does not use data augmentation. On average, we were able to obtain comparable performance with data augmentation. Due to lack of

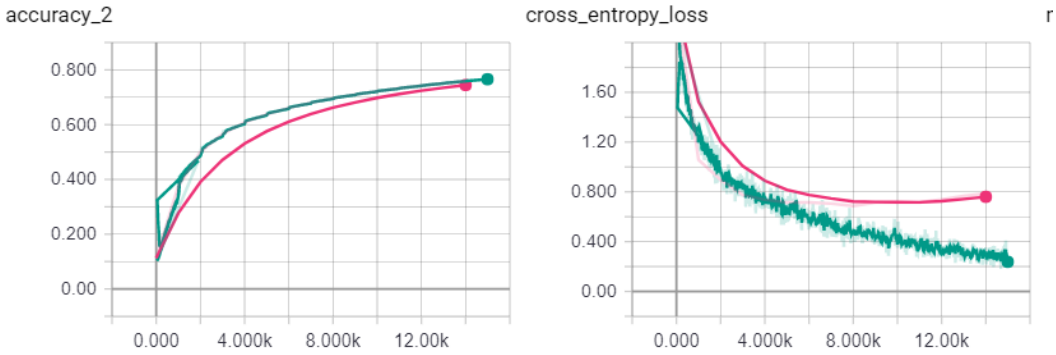


Figure 24: Learning curves for our best CNN model

time, we were not able to investigate further and train our networks for more steps to compensate for the larger amount of variance in the augmented data.

We also applied batch normalization to our network. However, we believe that this module requires data normalization and more fine-tuning that given the time constraints we was not able to do.

## 5 Conclusion

We explored two popular neural network architectures and applied them to an image classification task. We obtained decent performance using a CNN, and comparable performance with a MLP. Due to the fact that MLPs cannot learn spatial information as easily as CNNs can, we attribute the high accuracy of the MLP to overfitting.