Miniproject 3: Classification of Image Data

Nadia Blostein 260804876 **Vivian Eberle 260745995** **Christopher Scarvelis 260689983**

Abstract

In this project, we compared the performance of multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) in an image classification task. We trained these models to classify images from the CIFAR-10 dataset. We found that the MLP achieved its best validation performance with 3000 ReLU hidden units, a learning rate of 0.1, and an L2 regularization parameter value of 0.5; the corresponding test accuracy was 50.52%. We found that the convolutional neural network (CNN) with 3 convolutional layers, size 3 filters and the implementation of the Adamax optimization algorithm performed best, achieving a test accuracy of 68.75%. On the basis of these tests, we concluded that CNNs are likely to outperform MLPs on image recognition tasks.

1 Introduction

The purpose of this project was to implement deep learning algorithms in order to perform image recognition. More specifically, we sought to find the model that would recognize images from the CIFAR-10 dataset with the highest accuracy. Multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) with various hyperparameters were trained on 80% of the CIFAR-10 dataset and compared to each other based on their ability to classify the remaining 20% of the training set (our validation set). After hyperparameter tuning, the best MLP was compared to the best CNN and their performances on the CIFAR-10 test set were recorded as a function of the number of training epochs. As expected, the CNN significantly outperformed the MLP on the test set.

2 Related work

The literature generally indicates that CNNs perform better than MLPs for image classification tasks. Zhang et al. compare MLPs and CNNs for the purpose of building a hybrid model for image classification and finds that CNNs (with accuracies of 85.39% and 86.56%) perform better than MLPs (with a accuracies of 81.62% and 80.73%) on the two respective datasets employed in the paper [8]. With regards to the CIFAR-10 dataset, Krizhevsky (2010) achieved a 78.9% accuracy with a convolutional deep belief network (DBN) with two layers [5]. When studying CNNs, Krizhevsky et al. (2012) used two larger scale datasets with millions of images and tens of thousands of categories. When implementing a CNN with "five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax", the top-1 error rate achieved in one dataset was 37.5% (62.5% accuracy), and 67.4% in the other dataset. Krizhevsky et al. find that adding layers to the network decreases the error rate when comparing one layer versus five layers [6].

3 Dataset: CIFAR-10

3.1 Overview

The CIFAR-10 dataset was retrieved using 'torchvision.datasets.CIFAR10'. This dataset is comprised of 60,000 datapoints of single-labelled images classified as one of the following 10 categories: 'airplane', 'automobile', 'bird', 'cat',' deer', 'dog', 'frog', 'horse', 'ship', 'truck'. There is an equal number of labelled images in each category, at 6,000 images per category. The dataset is split into a train set and test set, each subset containing respectively 50,000 and 10,000 images – also with an equal number of images of each category in each subset. Each instance is a tuple comprised of the image data and the category's label (0 to 9).

3.2 Preprocessing

The dataset initially consisted of RGB color images (PIL.Images) of size 3x32x32 which we converted to PyTorch tensors and normalized according to the 'Training a Classifier' methodology [7]. We randomly split the trainset into a subset of 40,000 and 10,000 images, using the latter as a validation set. We then loaded the data from each subset (train set, validation set, test set) with the use of a PyTorch DataLoader object. We separated the image data (features) and labels in all subsets. While training the MLP, we also one-hot encoded the labels to enable the computation of the categorical cross-entropy loss.

4 Proposed approach

In this section, we will briefly outline the models tested in this project and describe our model selection and testing procedures.

4.1 Multilayer perceptron (MLP)

The MLP is an adaptive basis function model that uses a cascade of non-linear *hidden layers* connected by linear maps to learn representations of the input features. These representations are then used as inputs for a classification or regression task that is solved in the final layer of the MLP. MLPs are trained using gradient descent (typically a variant of SGD to accommodate the large size of modern datasets), and the relevant gradients are typically computed via the backpropagation algorithm popularized by Rumelhart et al. [3].

In this assignment, we implemented an MLP consisting of a single hidden layer. As we sought to perform a multiclass classification task, a softmax output layer was employed in conjunction with the categorical cross-entropy loss. We tested two activation functions in the hidden layer of our MLP: the *sigmoid* function and the *ReLU* function. The latter function has recently become standard in deep neural networks as it is less susceptible to the vanishing gradient problem that often affects neural networks with logistic activation functions. We also implemented L^2 regularization (i.e. weight decay) to control the capacity of our model and avoid overfitting. Gradients were computed as in the course slides and then implemented directly in PyTorch; we did not use automatic differentiation to compute gradients as the low complexity of our model rendered this unnecessary.

Model selection and hyperparameter tuning were carried out using a random subset of 40,000 training examples (the remaining 10,000 were set aside to serve as the validation set). An initial round of experiments were performed on the sigmoid and ReLU MLPs individually. For each activation function, a range of widths (number of hidden units), L^2 regularization parameters, and learning rates were tested and the resulting validation accuracies recorded. The batch size for SGD was 64 in all cases.

Once the preliminary data was collected, we first sought to determine the optimal learning rate. The validation accuracies were binned into three categories based on the width of the MLP that was tested. In particular, we computed the average validation accuracy for each tested learning rate on small MLPs (250,500, and 750 hidden units), medium MLPs (1000, 1500, and 2000 hidden units) and large MLPs (2500 and 3000 hidden units). After plotting these statistics and determining the optimal learning rate (0.1 in our case), we then computed the average validation accuracies over the same bins for the optimal learning rate and for each tested value of the L^2 regularization parameter. We used these statistics to select the optimal parameters for the sigmoid and ReLU MLPs.

We finally trained the sigmoid and ReLU MLPs with their best hyperparameters on the full training set and computed the training, validation, and test accuracies at the end of each epoch of training (we trained the model for 20 epochs before termination).

4.2 Convolutional neural network (CNN)

A CNN is another form of deep learning that performs very well on image classification tasks. Its architecture is made of convolutional, pooling and fully connected layers. For each convolutional layer, one needs to determine a filter size as well as the number of filters (i.e. number of channels output by that layer) to use. One can also adjust additional hyperparameters such as padding and stride size. Moving from one convolutional later to the next, one sequentially computes the dot product between the filter and all combinations of neighbouring input pixels that are the same size as the filter. The output dot products make up the input for the next layer. Typically, each convolutional layer is followed either by a max pooling or average pooling layer. Finally, the last layers in a CNN tend to be fully-connected layers that function in the same way as the layers in a MLP (see above). There are many design choices to be made in order to find the optimal CNN for image classification.

In order to classify the CIFAR-10 handwritten digits, we compared models with different combinations of hyperparameters by examining their accuracy on a validation set. 10,000 instances (20% of the training set) were split from the CIFAR-10 training data to be used for validation. The validation set remained the same across models and epochs. After the training-validation split, the final training data was always shuffled after each epoch iteration for each model. Furthermore, we manually implemented early stopping in order to make our runs more computationally efficient.

We hypothesized that our CNN would perform better on the handwritten digit classification task than our MLP. Hyperparameter selection began with the baseline CNN from the PyTorch tutorial [7]. This baseline CNN uses two convolutional layers of size 5 filters, each followed by max pooling (size 2 filter). The last convolution outputs 18 channels. This CNN ends in 3 fully-connected layers and uses stochastic gradient descent (SGD) with momentum of 0.9 and learning rate of 0.001. Starting with this baseline CNN and a mini-batch size of 20, we compared 10 optimization algorithms across 4 filter sizes. We decided to select the Adamax algorithm, a variant of the Adam algorithm that uses an infinity norm and leads to quicker convergence by acting both on the gradient and on the learning rate [4]. Adamax consistently yielded some of the highest accuracies for the smallest number of epochs and had relatively smooth curves (Appendix 2).

Keeping our two-convolutional-layer models and implementing the Adamax algorithm, we then looked at the accuracy of various batch sizes for size 3 and size 5 filters. We decided to keep our mini-batch size at 20 since it yielded high accuracies without needing to run through many epochs (Appendix 3).

In order to determine the actual architecture of our model, we examined several combinations of convolutional layers and filter sizes (all of which had their last two convolutional layers pooled). The CNN with 3 convolutional layers of size 3 filters and whose last layer output 18 channels performed the best on our validation set (Appendix 4). Finally, we compared several different pooling methods and decided to keep the pooling method used in the PyTorch tutorial [7] (max pool the last two convolutional layers) (Appendix 5). Since average pooling is analogous to implementing a smoothing kernel (and therefore reduces the noise in one's data) [1], we were not surprised that applying average pooling to our last convolutional layer yielded the highest accuracy. However, the accuracy of max pooling the last two layers was only slightly lower and it ran through fewer epochs during training. We therefore decided to max pool the last two convolutional layers of our model, as is done in the PyTorch tutorial.

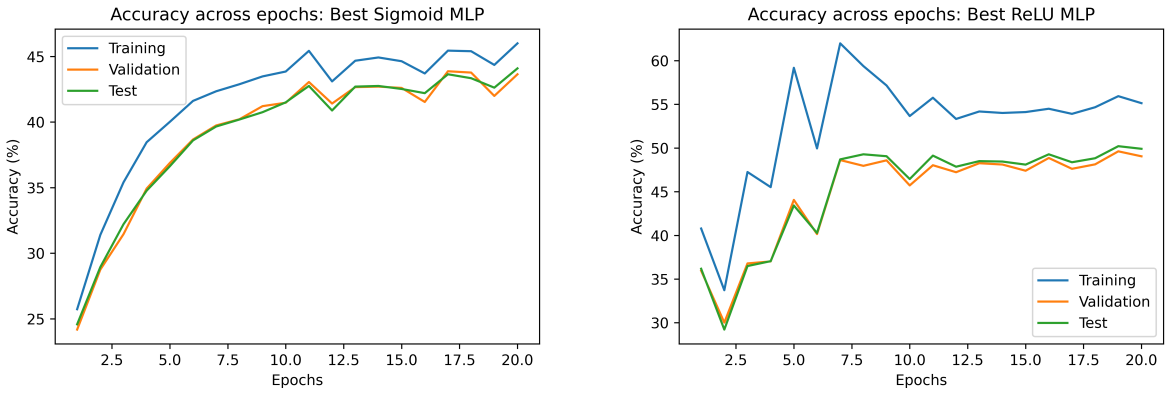
Finally, we examined the performance of different models ending in 3, 4 or 5 fully connected layers (Appendix 6). As expected, adding 1 or 2 fully connected layers to the original model [2] improved our validation accuracy slightly. However, the improvement was not significant enough to justify the computational expense of adding 1 or 2 fully connected layers. We therefore kept our model at 3 fully connected layers.

5 Results

Certain relevant figures for this section are found in the appendices. Our goal here is to outline the results of our model selection procedure and to describe our findings regarding the accuracy of each model as a function of the number of training epochs.

5.1 Multilayer perceptron (MLP)

The results of our model selection procedure are summarized in the figures included in Appendix 1. We found that a learning rate of 0.1 provided the best performance in most cases, though the difference in validation performance across learning rates was minor for the widest sigmoid MLPs. We then found that a small but non-zero level of L^2 regularization led to the best validation performance for both activation functions. In particular, a regularization parameter value of 0.25 provided the best results for the sigmoid MLP, whereas a parameter value of 0.5 provided the best results for the ReLU MLP. For the sigmoid activation function, small networks (250 hidden units) performed best on the validation set, whereas large networks (3000 hidden units) had a slight advantage for the ReLU MLP.



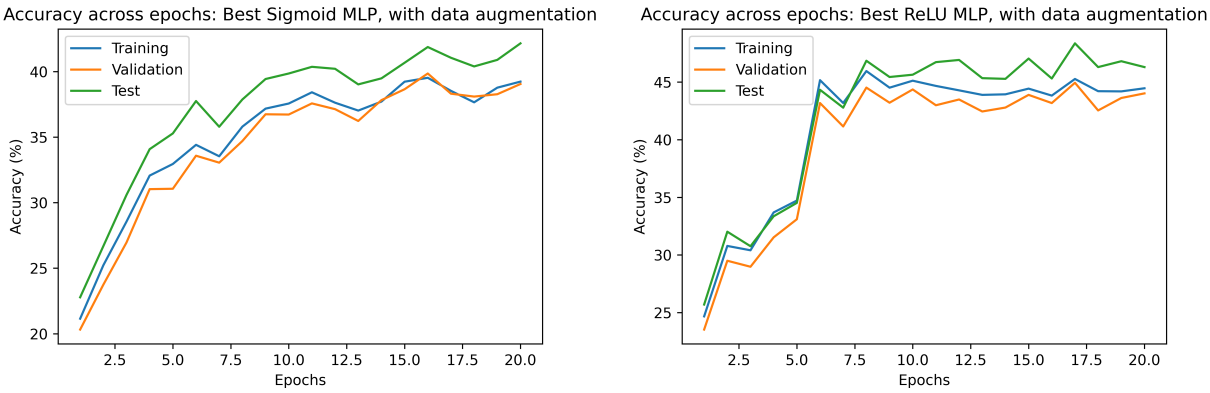
(a) Sigmoid MLP.

(b) ReLU MLP.

Figure 1: Accuracy by epochs for sigmoid and ReLU MLP with best hyperparameters.

The train, validation, and test performance of the sigmoid and ReLU MLP with their best hyperparameters are outlined in Figure 1 above. We observe that the test accuracy of the ReLU MLP plateaus after roughly 7 epochs of training, whereas the sigmoid MLP's accuracy begins to plateau later, at around 12 epochs. The ReLU MLP achieves somewhat better test performance (50.52%) at its plateau than the sigmoid MLP, which peaks at 43.92%. Interestingly, the ReLU MLP's accuracy on the training set is substantially greater than its validation and test accuracy, whereas the sigmoid MLP's training accuracy is only slighly better. This indicates that the ReLU MLP is overfitting to a greater extent than the sigmoid MLP, even as it obtains better test performance.

To help alleviate overfitting, we also carried out the same tests on our best models but augmented the training set using random transforms (including rotations and flips) from the torch.transforms library. The results are outlined in Figure 2 below.



(a) Sigmoid MLP.

(b) ReLU MLP.

Figure 2: Accuracy by epochs for sigmoid and ReLU MLP with best hyperparameters and augmented training set.

We find that data augmentation effectively mitigates overfitting - both models' test accuracies are higher than their training and validation accuracies in this regime. However, this comes at the cost of accuracy: the MLPs trained on the augmented dataset achieve somewhat lower test accuracy. This may be a consequence of underfitting the augmented dataset; further experiments would be required to confirm this hypothesis.

5.2 Convolutional neural network (CNN)

As can be seen in Figure 3, the model with the best combination of hyperparameters (3 convolutional layers, size 3 filters, 18-channel output from last convolution, Adamax optimization algorithm, max pool last 2 layers) performs better on the validation set than two slightly modified versions of that model (16-channel output from last convolution;

SDG optimization algorithm). It also performs better than the baseline model from the PyTorch tutorial (2 convolutional layers, size 5 filters, 16-channel output from last convolution, SDG optimization algorithm, max pool last 2 layers) as well as a slightly modified version (Adamax optimization algorithm) of the latter.

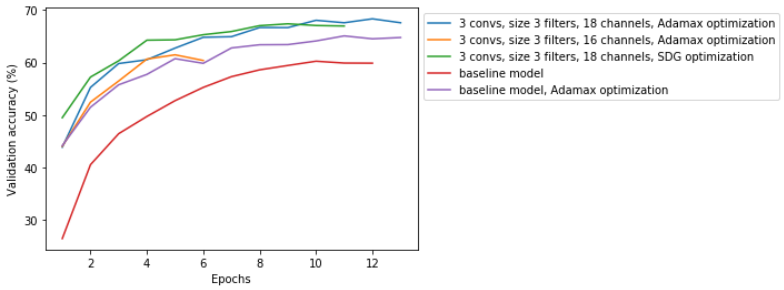
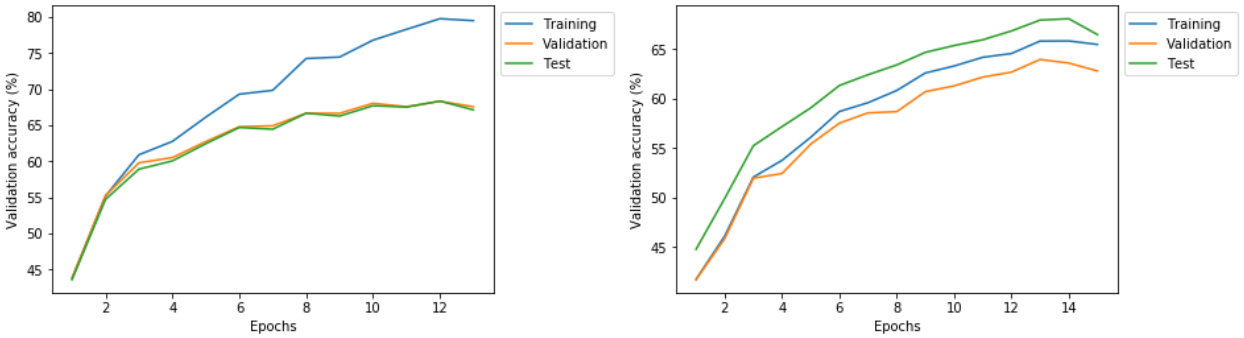


Figure 3: Validation accuracy by epoch across models

As expected, our CNN performed substantially better on the test set than our two MLP models (Figure 2, Figure 4), with a test accuracy of 68%. When trained on an augmented dataset, our CNN also achieved a test accuracy of 68%. Finally, when trained on both the training and validation sets (without data augmentation), the CNN achieved a test accuracy of 68.75%. Just as is the case with the MLP models, the augmented dataset did reduce overfitting, leading to a higher test accuracy than validation or training accuracy (Figure 4b).



(a) Integral training set.

(b) Augmented training set.

Figure 4: Accuracy by epochs for model with best hyperparameters with and without augmented training set.

6 Discussion and conclusion

As expected, the CNN performed better than the MLP on the image classification task. Within the context of the MLP, use of the ReLU activation function led to better test accuracy after fewer epochs of training. A small amount of L^2 regularization improved validation accuracy, likely by mitigating the effects of overfitting. Data augmentation was highly effective at reducing overfitting but did not lead to gains in test accuracy.

Designing an effective neural network classifier involves a large number of design choices, and it is computationally challenging to thoroughly search the space of hyperparameters while tuning these models. As such, future extensions to this work may focus on further exploring the hyperparameter space. For the MLP in particular, testing deeper neural nets would also be an interesting contribution, as most neural network models in computer vision now incorporate many hidden layers. It was however interesting to observe that adding layers to the CNN did not seem to significantly improve its performance.

7 Statement of contributions

Vivian prepared the datasets, Chris coded and tested the MLP and Nadia coded and tested the CNN. Each member did their respective part of the write-up.

References

[1] Jason Brownlee. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. 2019. URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/?fbclid=IwAR14jqf1OM64LphImxTOZ-0UVsOGDNfSmCc0l6Ti66Of1b7iKO77WDOOeXc.

[2] Pierre Ecarlat. *CNN — Do we need to go deeper?* 2017. URL: https://medium.com/finc-engineering/cnn-do-we-need-to-go-deeper-afe1041e263e.

[3] David E. Hinton, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* (1986), pp. 533–536. DOI: 10.1038/323533a0.

[4] Raimi Karim. *10 Stochastic Gradient Descent Optimisation Algorithms + Cheat Sheet*. 2018. URL: https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9.

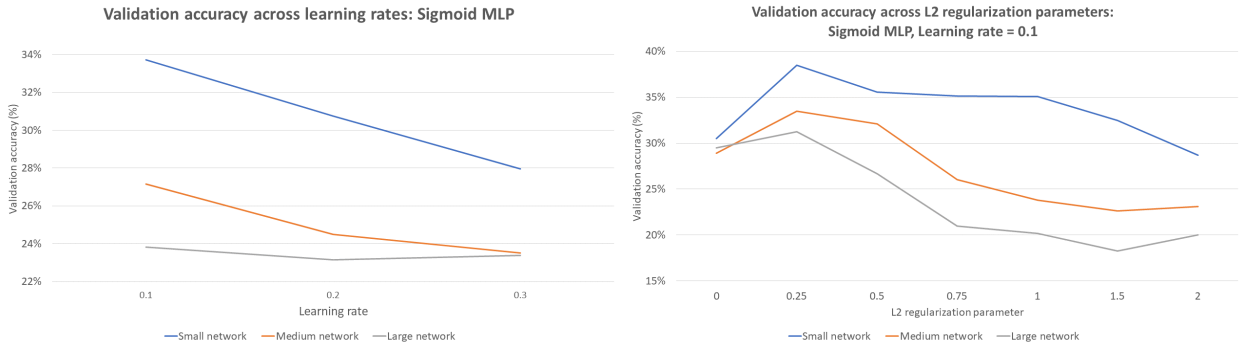
[5] Alex Krizhevsky. *Convolutional deep belief networks on CIFAR-10*. 2010.

[6] Alex Krizhevsky, Ilys Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Comm. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.

[7] *Training a classifier*. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html?fbclid=IwAR0jPUKonqJeWAl6y64e-wLraZ7brfMNaAHcc9U1s5bQFpY9s8IqOlljpLQ. Accessed: 2020-04-18.

[8] Ce Zhang et al. "A hybrid MLP-CNN classifier for very fine resolution remotely sensed image classification". In: *ISPRS Journal of Photogrammetry and Remote Sensing* (2018), pp. 133–144. DOI: 10.1016/j.isprsjprs.2017.07.014.

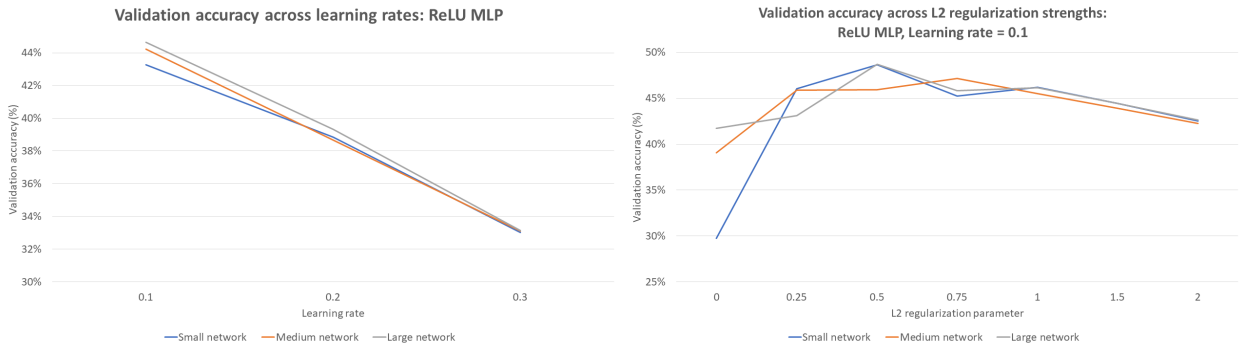
8 Appendix 1: MLP Model Selection



(a) Validation accuracy across learning rates.

(b) Validation accuracy across L2 parameters.

Figure 5: Model selection results for sigmoid MLP.

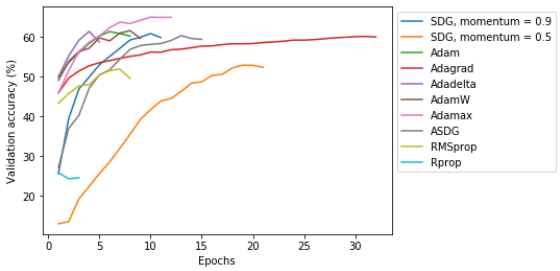


(a) Validation accuracy across learning rates.

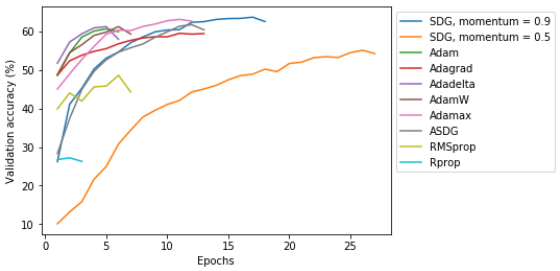
(b) Validation accuracy across L2 parameters.

Figure 6: Model selection results for ReLU MLP.

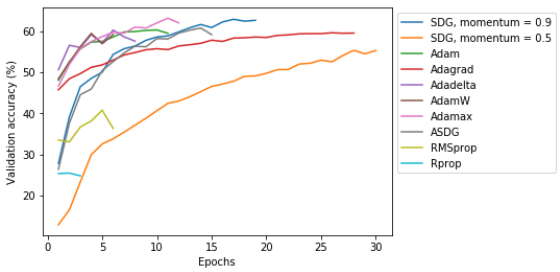
9 Appendix 2: Selection of CNN Optimization Algorithm



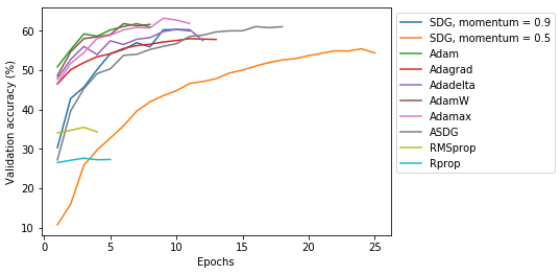
(a) 2 convolutions, size 2 filters



(b) 2 convolutions, size 3 filters



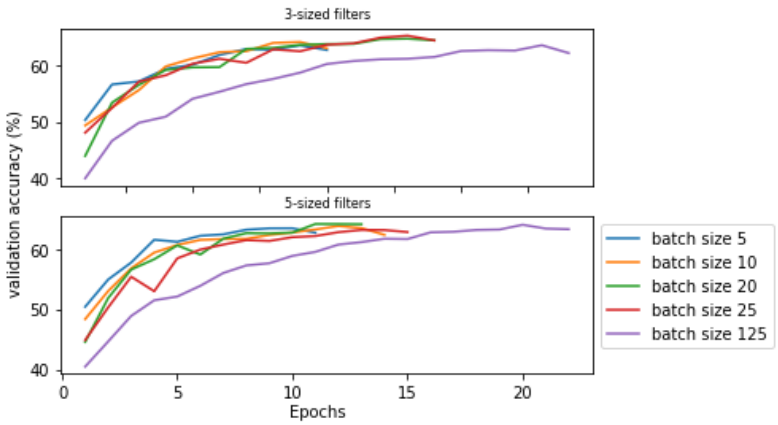
(c) 2 convolutions, size 4 filters



(d) 2 convolutions, size 5 filters

Figure 7: Validation accuracy by epoch across optimizers (batch size 20)

10 Appendix 3: Selection of CNN Minibatch Size



(a) 2 convolutions

Figure 8: Validation accuracy by epoch across batch sizes

11 Appendix 4: Selection of CNN Filter Size and Number of Convolutional Layers

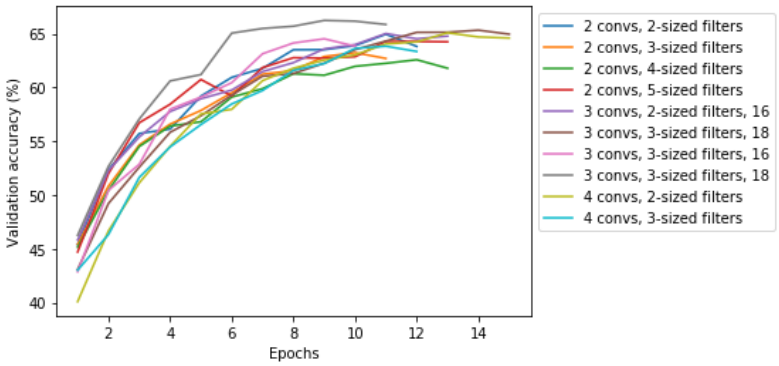


Figure 9: Validation accuracy by epoch across amount of convolutional layers and filter size

12 Appendix 5: Selection of Pooling Method

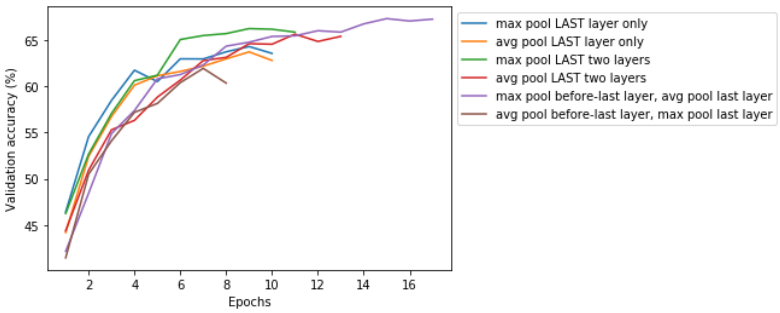


Figure 10: Validation accuracy by epoch across different pooling methods

13 Appendix 6: Selection of Number of Fully-Connected Layers

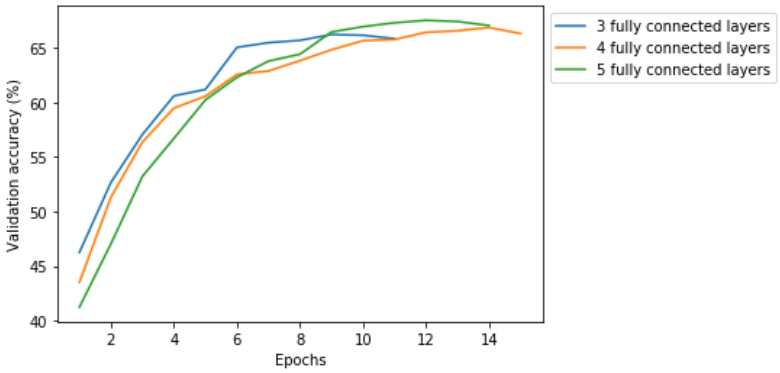


Figure 11: Validation accuracy by epoch across amount of fully connected layers