AI6121 Computer Vision Course Project

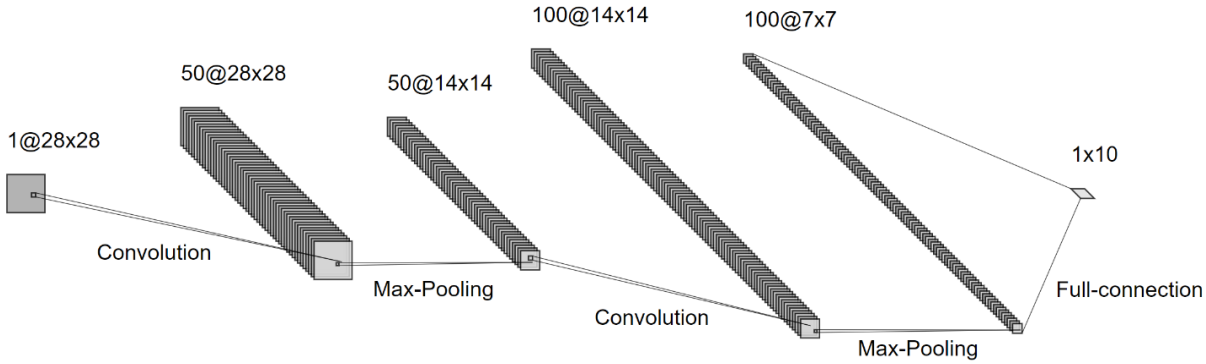
Chen Yongquan (G2002341D)
Nanyang Technological University

MNIST Handwritten Digits Recognition

Contents

Simple Convolutional Neural Network Design

The topic chosen for this course project is MNIST handwritten digits recognition. The MNIST dataset consists of 60000 training samples and 10000 testing samples. Samples are 28x28 images and consists of 10 classes for each digit.



The basic design of our convolutional neural network consists of 2 convolutional layers, interweaved with ReLU activation and max pooling layers, followed by a fully connected layer that output 10 probabilities for each digit. There are 50 filters in the first convolutional layer and 100 filters in the second convolutional layer. When feedforwarding to the fully connected layers, feature maps are flattened from a dimension of 100x7x7 to 4900. This is multiplied by the batch size since training is done in mini-batches of 128 samples. The loss function used is cross-entropy loss which is often used for multiclass classification. Cross-entropy loss is the negative log likelihood of the softmax of the probabilities outputted and is given by the formula:

$$loss(x, class) = -log\left(\frac{exp(x[class])}{\sum_j exp(x[j])}\right)$$

Weights are updated by stochastic gradient descent. Figure 1 shows the training accuracy per iteration for 1 epoch. Because accuracy saturates after 1 epoch on the simple MNIST dataset, we show accuracy by iteration instead of epoch, which would result in more unstable spikes as the loss optimized per iteration is an estimate done on each mini-batch instead of the loss across the entire dataset. Validation accuracy on the test set is 97.220%.

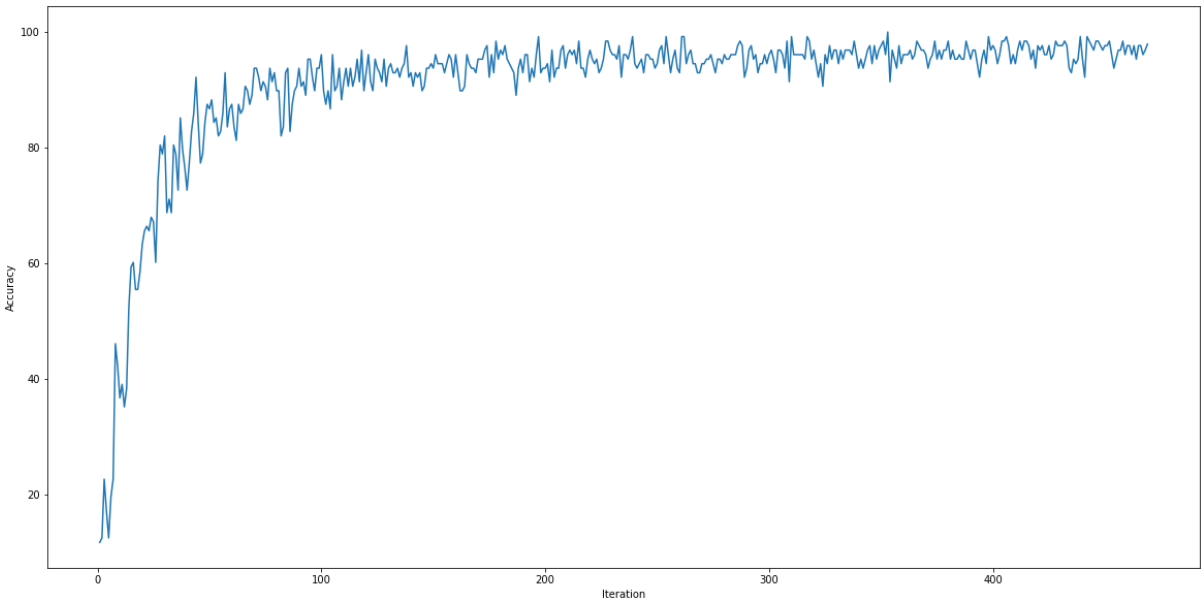


Figure 1

Learning Rate

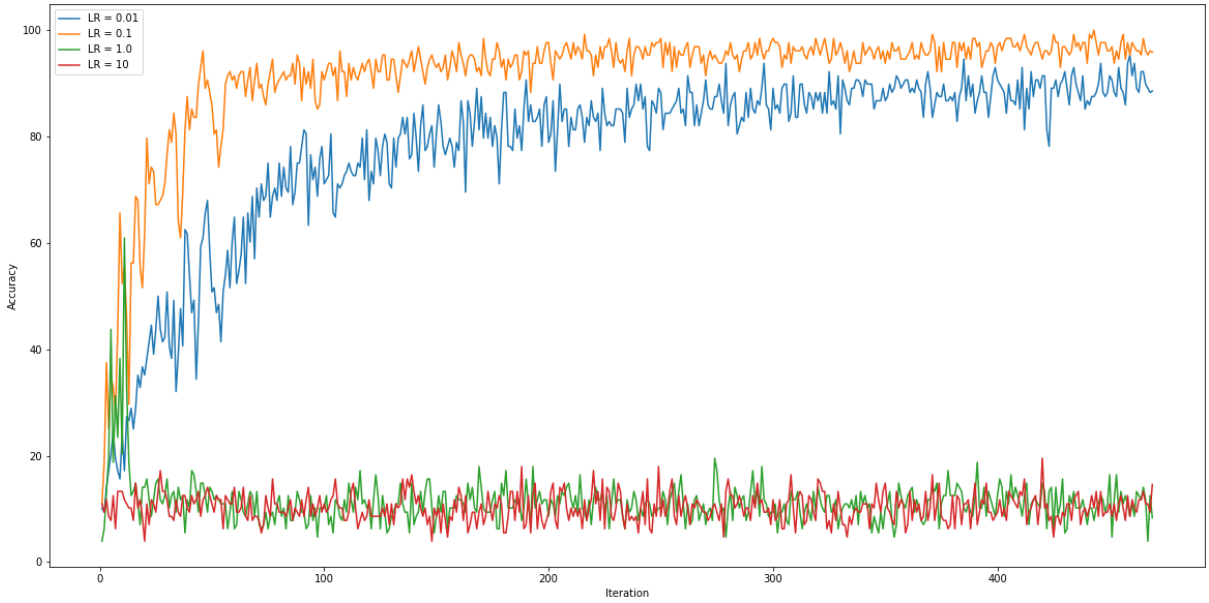


Figure 2

We then experiment with different learning rates as shown in Figure 2. We can see that a smaller learning rate of 0.01 results in slower learning though training still progresses smoothly and is expected to reach the same accuracy with more training epochs. A learning rate of 0.1 seems to give the best accuracy in the least number of iterations and epochs. Using a learning rate of 1.0, training starts smoothly then abruptly drops and becomes stuck at a bad local minimum as the network rubber-band about the local minimum or saddle point. Using too high a learning rate thus inhibits proper learning of optimal weights.

Batch Size

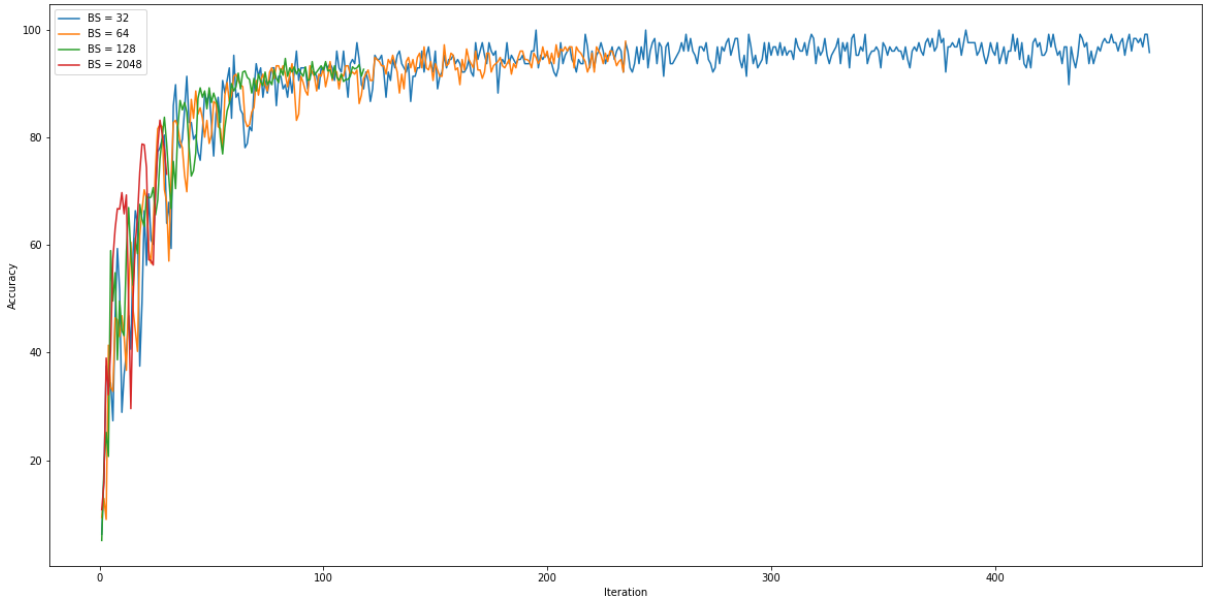


Figure 3

Figure 3 shows a comparison of different batch sizes. Using a larger batch size would result in fewer iteration per epoch. This would mean less overhead involved in transferring data from DRAM to GPU memory but having fewer iterations also mean weights are updated less times per epoch and training is slower as a result.

We can see that even though a batch size of 2048 seems to give a more accurate estimation of the loss and a relatively higher accuracy per iteration, by the end of 1 epoch, the accuracy of smaller batch sizes far exceed that of it as they had more training iterations performed. Thus, too small a batch size would give more unstable training due to poor estimation of losses while too large a batch size would slow training too much and is also limited by the GPU memory.

Momentum

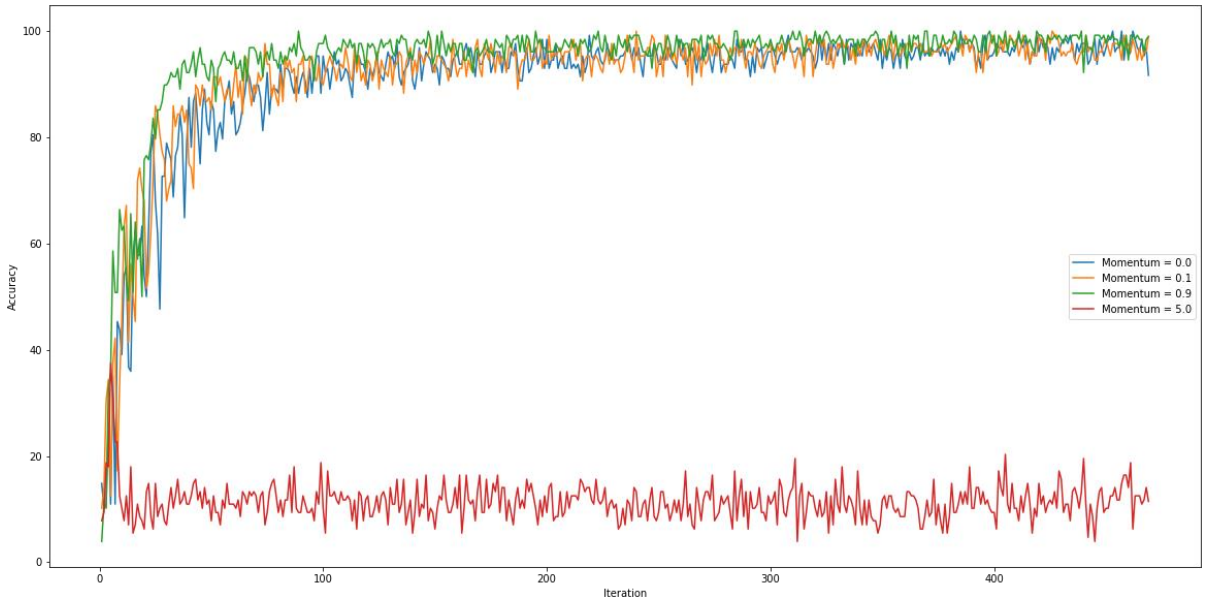


Figure 4

We next experiment with a momentum factor, where instead of minimizing the loss multiplied by the learning rate directly, we minimize by a velocity term, which consists of a moving average of previous velocities and the loss multiplied by the learning rate. We can see that a momentum of 0.9 shown by the green line allows training to progress faster in the initial iterations while too high a momentum prevents training from progressing smoothly, akin to using too high a learning rate.

L2 Regularization

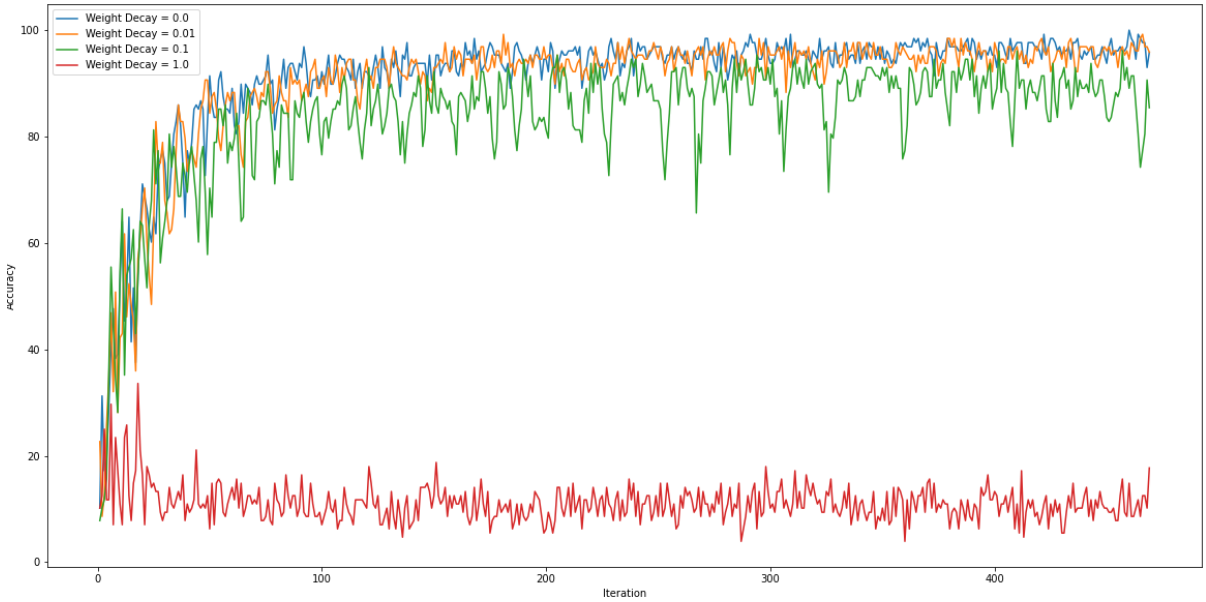


Figure 5

L2 regularization is a way of preventing a model from overfitting on the training set by adding a penalty term to the loss function. This is also equivalent to weight decay here. Figure 5 shows a comparison of different weight decay factors. Higher weight decay factors seem to progressively result in lower accuracies on both the training set and test set with a respective accuracy of 96.670, 95.380, 90.570 and 11.350 on the latter. Regularization prevents overfitting but may also lower accuracy on the training set consequently and may require more training epochs to achieve high accuracy on both training and test sets.

Optimizers

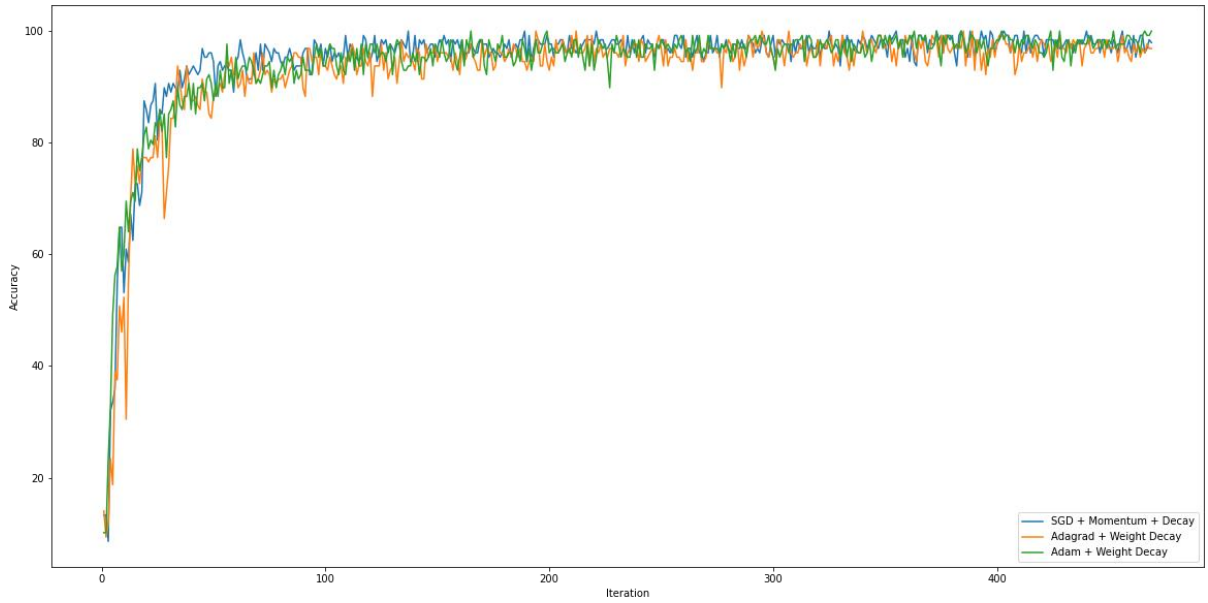


Figure 6

Figure 6 shows a comparison of different optimizers, mainly stochastic gradient descent with momentum, Adagrad and Adam. SGD seems to give the best performance with an accuracy of 98.310 on the test set followed by Adam, 97.950 and then Adagrad, 97.340.

Loss Functions

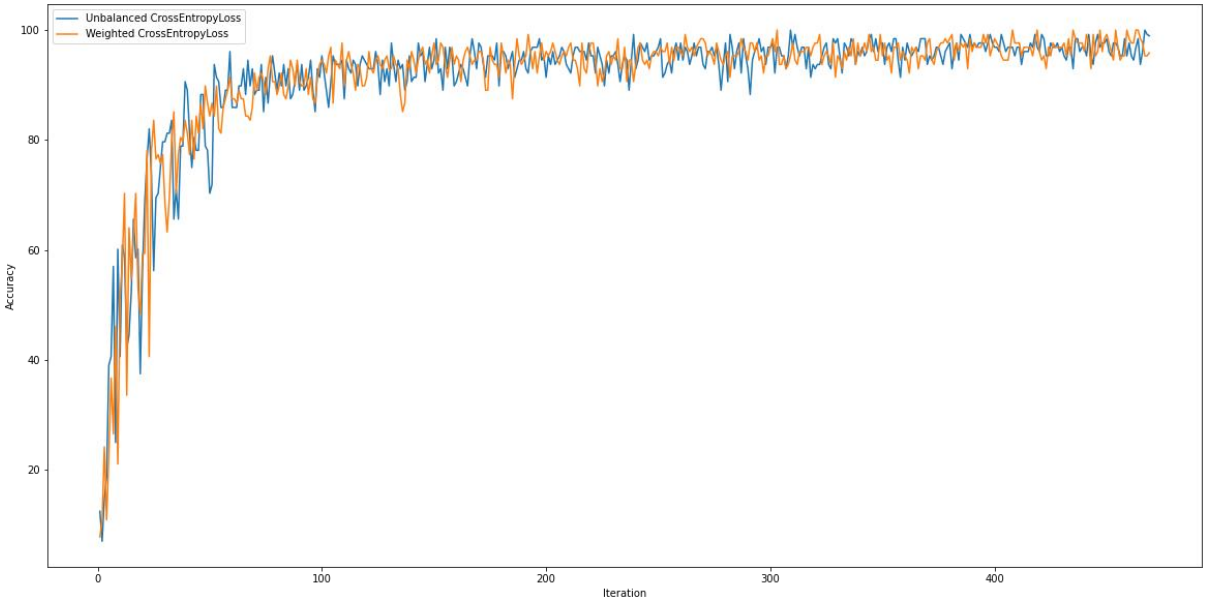


Figure 7

Balanced cross-entropy loss is a method for dealing with unbalanced datasets where one or more classes far outnumber those of the other classes. If undealt with, the trained model will overfit on the classes with the majority of samples in the training set. By weighing the loss based on a balancing factor, usually proportional to the inverse of the number of samples per class over the total dataset size, it is possible to tune down the loss for the classes with ample samples so that their losses do not drown out the loss from the rare classes and allows for training to progress evenly for all classes. Because MNIST is a relatively balanced dataset, we do not see much difference in network performance between the unbalanced cross-entropy loss function and its weighted version. We weigh the classes by the inverse of their sample count divided by 60000, which is then divided by 10 each to be around the range of 1.0. Accuracy between both can be considered to be within the margin of error of 1% or less and results vary between multiple runs. One may be better than the other on some runs, and vice-versa.

Network Depth: Convolutional Layers

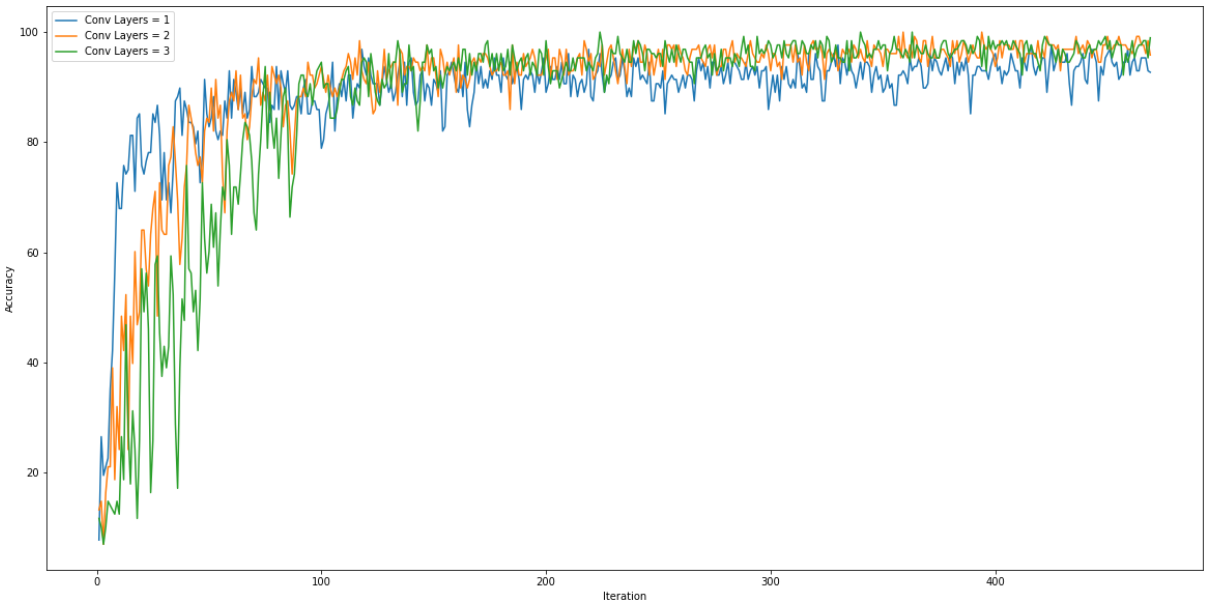


Figure 8

Figure 8 shows our experiment by varying the number of convolutional layers. We can see that having more convolutional layers result in relatively slower learning initially, but accuracy is higher as a result. Test accuracies are 93.820, 96.560 and 97.560 respectively.

Network Depth: Fully Connected Layers

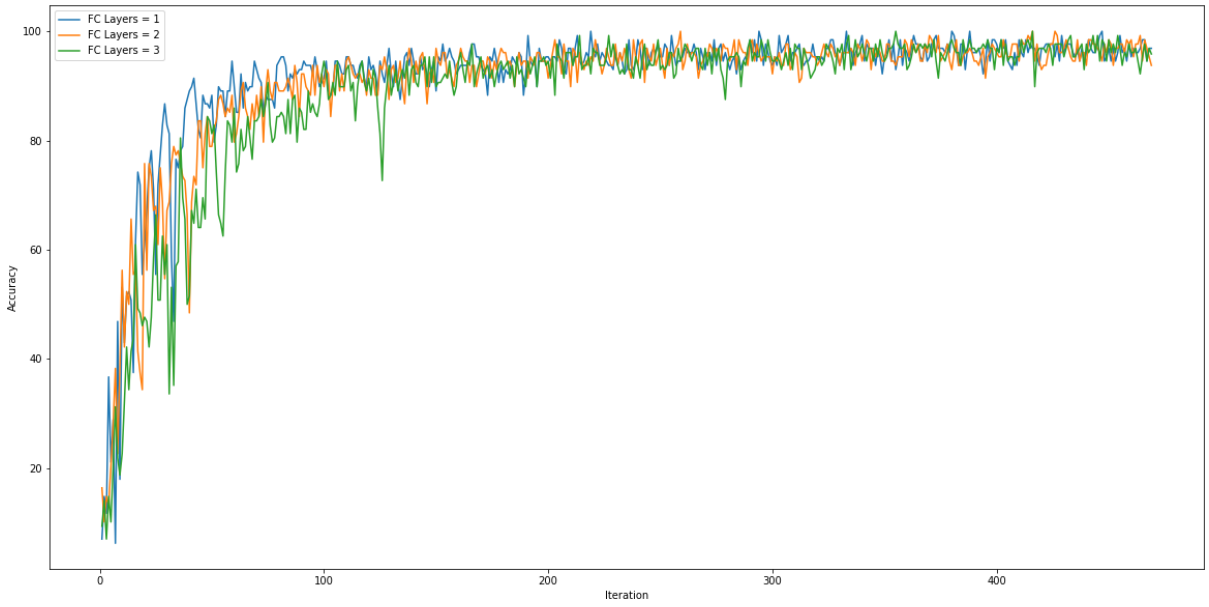


Figure 9

Figure 9 is our experiment with different number of fully connected layers. The number of fully connected layers present a similar pattern with respect to training speed and training accuracy. However, having more fully connected layers may result in overfitting to the training set as test accuracies drops as more fully connected layers are added. Accuracies are 97.280, 97.090 and 96.440 respectively.

Network Width

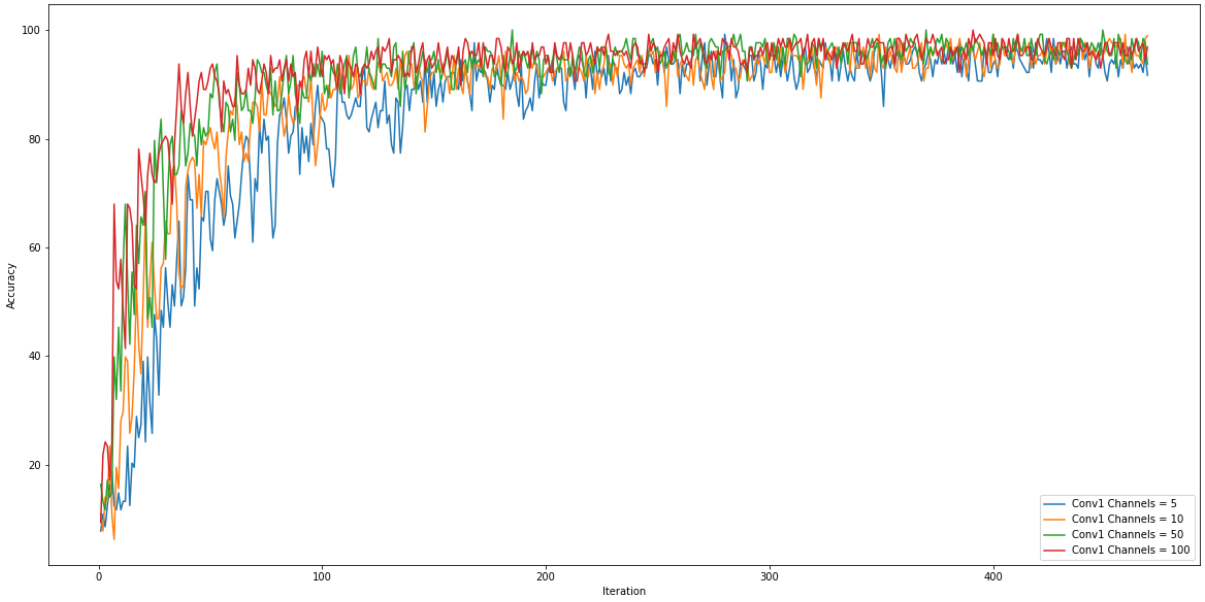


Figure 10

We then experiment with the number of channels in the network. The legend shows the number of channels in the first convolutional layer. The channels are doubled in the second convolutional layer. We can see that having more channels definitely have a positive impact on the accuracy. Test accuracies are 95.170, 96.280, 96.870 and 97.430 respectively.

State of the Art: ResNet

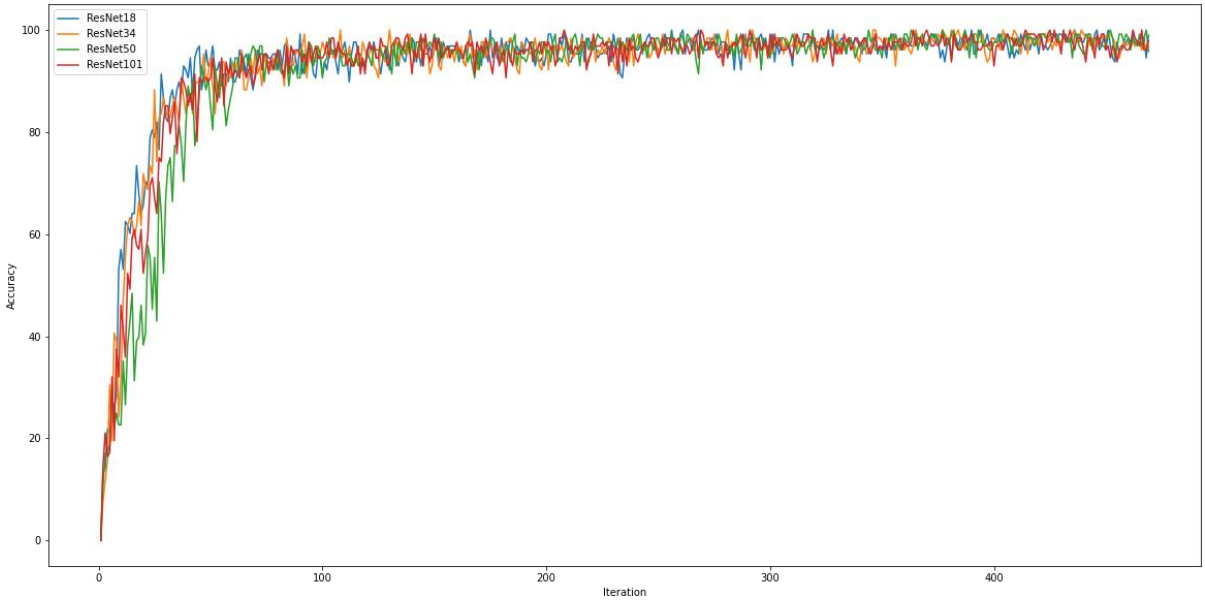
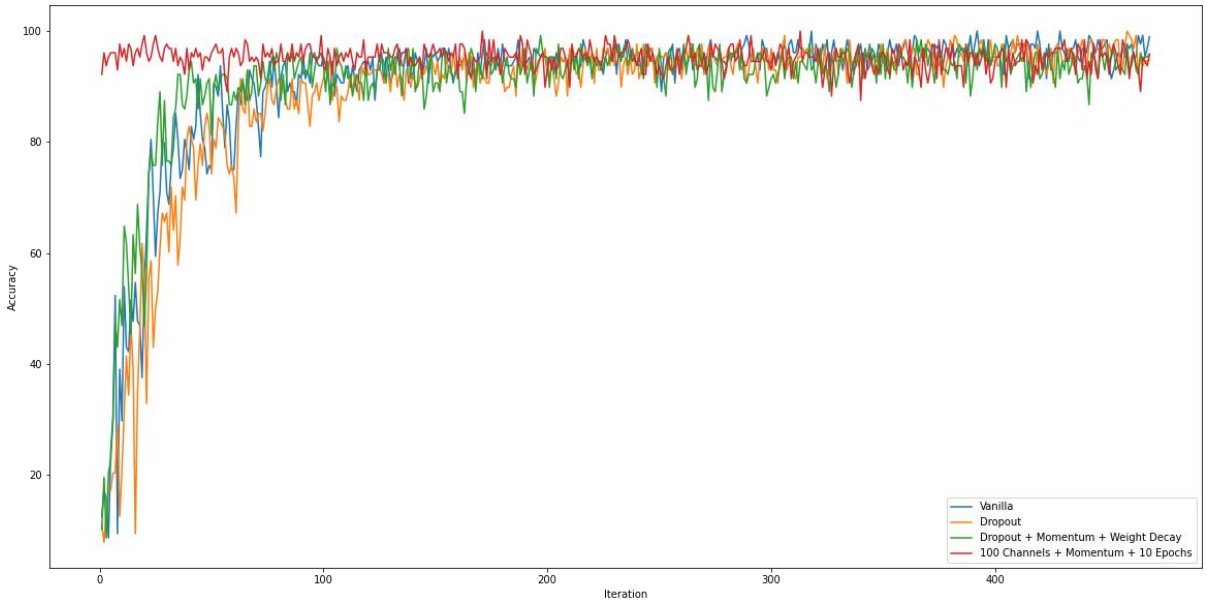


Figure 11

Finally, we use benchmark the accuracy using the ResNet architecture which is a relatively state-of-the-art model that many networks use as a backbone today. Newer architectures such as ResNeXt and ResNeSt are also based on the ResNet architecture. The main idea behind ResNet is to add skip connections from the input of each block to the output of each block, summing the input with the output and thus force the weights in each block to learn the residual between the input and the original mapping instead. This allows the network to have an easier time learning the identity mapping or optimal mappings that are close to the identity

mapping, which in turn allows training to progress smoothly even with very deep layers. Another novelty introduced by the ResNet paper is bottleneck blocks which downsamples the input at the start of each block before the main convolution operation using 3x3 filters is performed. Then, feature maps are upsampled back at the end of each block. By performing this trade-off, the network can have more layers at the same computational cost. Figure 11 thus shows the results of our training using different depths of ResNet. Test accuracies are 97.870, 96.810, 97.950 and 97.630 respectively. A curious thing to note is that ResNet-101 seems to achieve faster training than ResNet-50 initially. This may be a random occurrence due to having initial weights closer to a local optimum or it might mean that training can be faster once network becomes sufficiently deep.

Improvements



Before we end the report, we present a possible improvement that can be made on our simple CNN architecture which is to add dropout layers to the fully connected layers. Dropout layers randomly deactivates neural connections and forces the network to learn redundant weights and thus allows for better generalization. We also present our best possible model depicted by the red line, which is the vanilla network modified to 100 channels in the first layer and 200 channels in the second layer. Because MNIST is a relatively simple dataset, it is possible to reach 100% accuracy on the training and test set with sufficient epochs using a simple model. We thus train this model with 10 epochs with a momentum of 0.9. The chart shows the training accuracy on the 10^{th} epoch so as not to overwhelm the other line plots which are done for only 1 epoch. The vanilla network has a test accuracy of 96.960 while adding dropout layers between 2 fully connected layers resulted in a test accuracy of 97.200. We can see that even though training accuracy is lower, test accuracy is higher. Finally, adding momentum and weight decay to the dropout model further bumps test accuracy to 97.460. The best model, consisting of 2 fully connected layers with dropout and a base of 100 hidden channels in the convolutional layers, trained till the 10^{th} epoch gave a test accuracy of 97.890 with a top-5 test accuracy of 100%. Best test accuracy obtained during the 10 epochs was 98.320.