Colin Kirby
co20115@ucf.edu

HW #2

** Note: The complete implementation, including training scripts, testing procedures, and visualization code, is provided in the accompanying ZIP file.*

Experimental Design and Methodology

This analysis explores the impact of various neural network configurations on model performance. To ensure systematic and controlled comparisons, I established a baseline configuration with the following parameters:

- Input Normalization: Enabled
- Batch Size: 32
- Learning Rate: 0.001
- Optimizer: Momentum
- Weight Initialization: Random
- Weight Decay: 0.001

Implementation Strategy

Rather than using traditional commented-out code blocks for testing different configurations, I implemented an automated experimental framework that sequentially tests multiple configurations while maintaining controlled conditions. This approach:

1. Starts with the baseline configuration

2. Systematically modifies one parameter at a time

3. Maintains all other parameters constant

4. Records and compares performance metrics

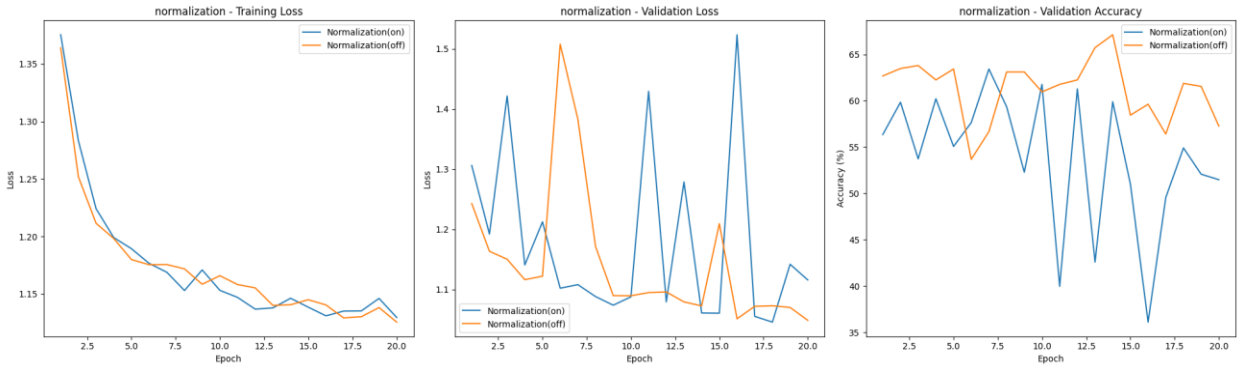
The experiments evaluate the following variations:

- Normalization: Enabled vs. Disabled
- Batch Sizes: 32 vs. 128
- Learning Rates: 0.001 vs. 0.01
- Optimizers: Momentum, Adam, RMSprop, SGD
- Weight Initialization: Random vs. Xavier
- Weight Decay: 0 vs. 0.001

This methodical approach ensures consistent testing conditions and allows for direct comparison of how each parameter affects model performance. The following sections detail the results and analysis of each configuration variation.

Colin Kirby
co20115@ucf.edu

1. With & Without Normalization.

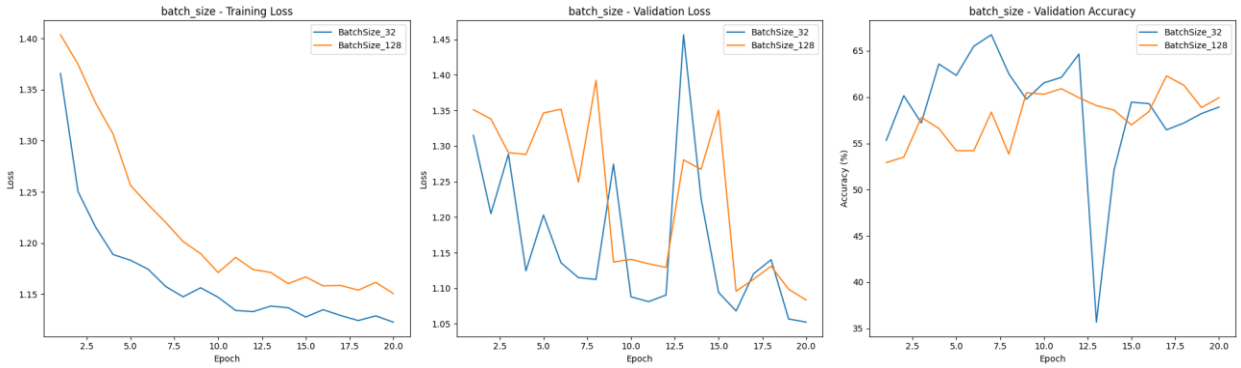


From the experiments comparing normalized and non-normalized training approaches, I saw some unexpected results. When training with normalization, the model started with a loss of 1.39 and reached its peak validation accuracy of 63.42% at epoch 10 as shown in the graph in the blue line on the graph on the far right. However, the training process showed significant instability, with frequent fluctuations in both training and validation metrics, suggesting that the normalized data may have complicated the learning process.

Interestingly, the model performed better without normalization, achieving a higher peak accuracy of 66.26% by epoch 7 and demonstrating more stable learning behavior throughout training. Both approaches eventually achieved similar final training losses around 1.13, but the non-normalized version maintained more consistent performance. This unexpected success without normalization could probably be explained by the dataset's simple nature and the presence of batch normalization layers in our network, which may already provide sufficient normalization effects during training.

Colin Kirby
co20115@ucf.edu

2. With Different Batch Sizes. (32 vs. 128)



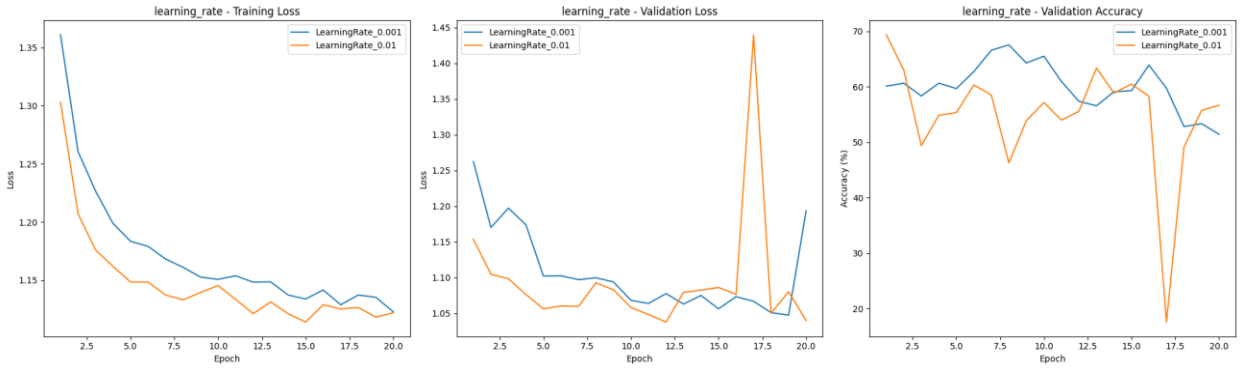
From the experiments comparing batch sizes of 32 and 128, I observed distinct performance patterns in the training process. With a batch size of 32, the model achieved its best validation accuracy of 66.74% at epoch 7, as shown by the steeper learning curve in the accuracy plot. The training was more responsive to the data, with the blue line showing quicker initial improvements and better overall performance, though it did show some volatility in later epochs.

When using the larger batch size of 128, the model's performance was notably different, reaching a lower peak accuracy of 62.29% at epoch 17. As visible in the training curves, this larger batch size resulted in smoother but slower learning progression, with the red line showing more gradual improvements but less dramatic fluctuations. The training loss decreased more steadily but ultimately couldn't match the peak performance of the smaller batch size, suggesting that the reduced number of weight updates per epoch may have limited the model's ability to find optimal parameters.

This comparison demonstrates the classic trade-off between training stability and model performance, where smaller batch sizes offer better peak accuracy but more volatile training, while larger batch sizes provide more stable but potentially slower and less effective learning. The superior performance of batch size 32 indicates that for our dataset, more frequent weight updates were beneficial for achieving better model accuracy.

Colin Kirby
co20115@ucf.edu

3. With Different Learning Rates. (0.001 vs. 0.01)



From the experiments comparing learning rates of 0.001 and 0.01, I observed significant differences in the model's training behavior. With the lower learning rate of 0.001, the model showed steady progression, reaching its peak validation accuracy of 67.54% at epoch 8, as shown by the gradual upward trend in the accuracy plot. The training loss curve demonstrates stable learning with consistent improvements, though it took several epochs to reach optimal performance.

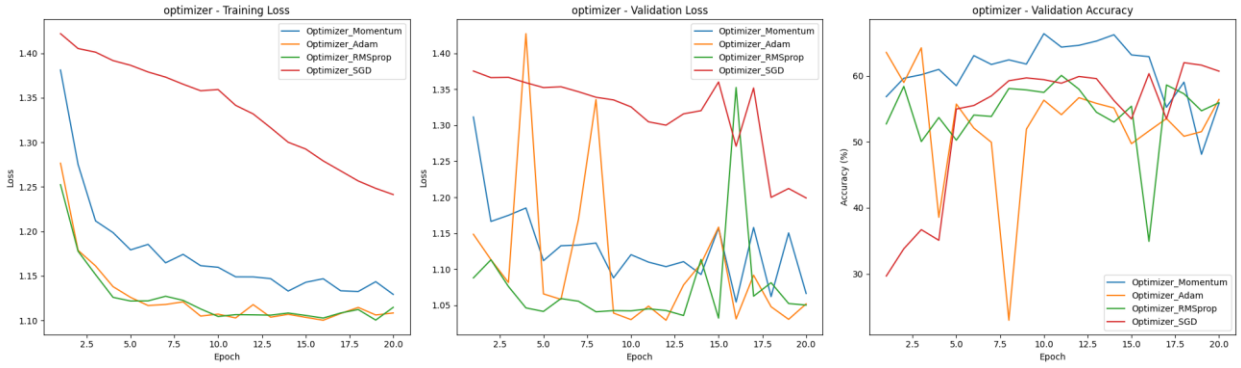
Interestingly, the higher learning rate of 0.01 achieved the best overall performance across all experiments, reaching a peak accuracy of 69.36% at epoch 1, as clearly visible in the sharp initial spike in the accuracy plot. However, this came with a trade-off, the training became more unstable in later epochs, with the validation accuracy showing significant fluctuations. This is particularly evident in the dramatic drop around epoch 17, where the accuracy temporarily plummeted to 17.57%. The higher learning rate allowed for quicker convergence but at the cost of training stability.

These results suggest that while the higher learning rate of 0.01 can achieve better peak performance, it requires careful monitoring to prevent instability. The lower learning rate of 0.001 offers more reliable training behavior but might take longer to reach optimal performance. This demonstrates the classic learning rate trade-off between speed of convergence and training stability.

Colin Kirby
co20115@ucf.edu

4. With Different Optimizers.

(Momentum vs. Adam vs. RMSProp vs. SGD)



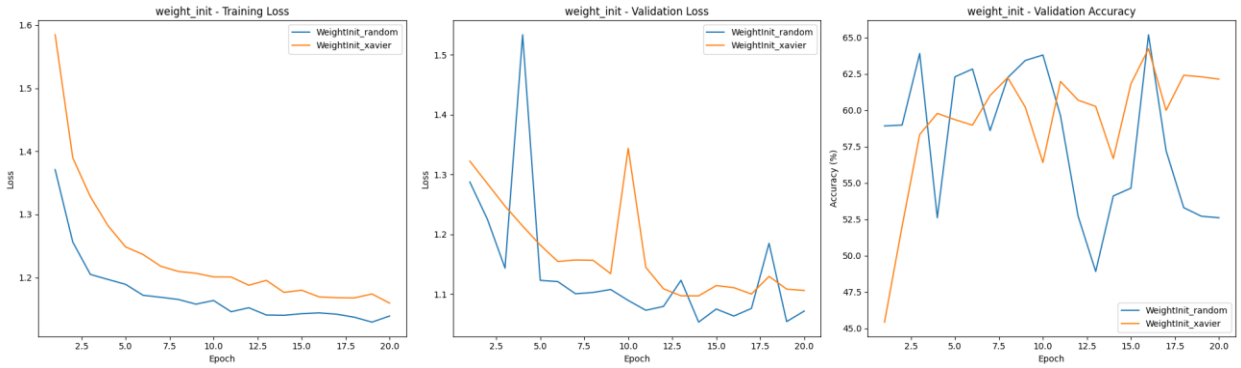
From the experiments comparing different optimizers (Momentum, Adam, RMSprop, and SGD), I observed varying patterns of convergence and stability. The Momentum optimizer emerged as one of the top performers, achieving a peak validation accuracy of 66.36% at epoch 10, as shown by the steady climbing blue line in the accuracy plot. The training progression was relatively stable, with the loss curve showing consistent improvement before reaching convergence.

Adam and RMSprop showed mixed results, with Adam reaching a peak accuracy of 64.22% at epoch 3 and RMSprop achieving 60.04% at epoch 11. As visible in the training curves, Adam demonstrated quick initial learning but suffered from stability issues, showing significant fluctuations in validation accuracy (dropping as low as 22.98% at epoch 8). RMSprop, while more stable than Adam, couldn't match the peak performance of Momentum, though it maintained more consistent validation accuracy throughout training. Standard SGD performed the most conservatively, with a slower but steady learning progression, eventually reaching 61.97% accuracy at epoch 18, as shown by the gradually ascending line in the accuracy plot.

These results highlight the strengths and weaknesses of each optimizer: Momentum provided the best balance of performance and stability, Adam offered quick initial learning but with stability concerns, RMSprop showed consistent but lower performance, and SGD demonstrated slow but steady improvement. The superior performance of Momentum suggests it's the most suitable optimizer for our specific dataset and model architecture.

Colin Kirby
co20115@ucf.edu

5. With Different Weight Initializations. (Random vs. Xavier)



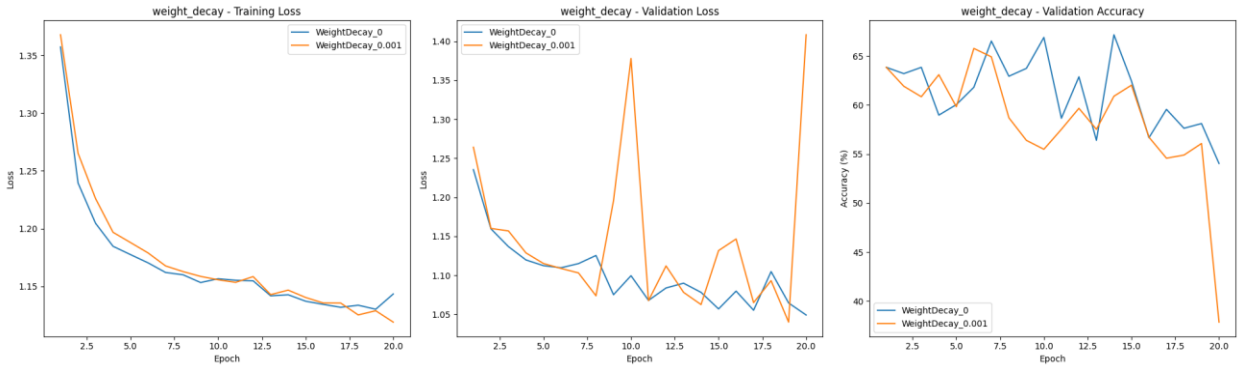
From the experiments comparing random and Xavier weight initialization methods, I saw differences in the training dynamics. Random initialization demonstrated slightly better performance, achieving a peak validation accuracy of 65.18% at epoch 16, as shown in the accuracy plot. The training progression with random initialization started more efficiently, with the loss curve showing quicker initial descent and maintaining relatively stable learning throughout the training process, though some fluctuations were visible in later epochs.

Xavier initialization, while theoretically more sophisticated, achieved a slightly lower peak accuracy of 64.22% at epoch 16. As visible in the training curves, Xavier initialization started with a notably higher initial loss of 1.58 (compared to 1.37 for random), but showed more consistent and gradual improvement throughout training. The validation accuracy plot shows more stable progression with Xavier initialization, albeit with slower initial learning. This suggests that while Xavier initialization provided more stable training dynamics, the simpler random initialization strategy was actually more effective for our specific neural network architecture, possibly due to our model's relatively simple structure and the nature of our dataset.

These results challenge the common assumption that more sophisticated initialization methods always lead to better performance, showing that sometimes simpler approaches can be equally or more effective depending on the specific characteristics of the model and data.

Colin Kirby
co20115@ucf.edu

6. With & Without Weight Decay.



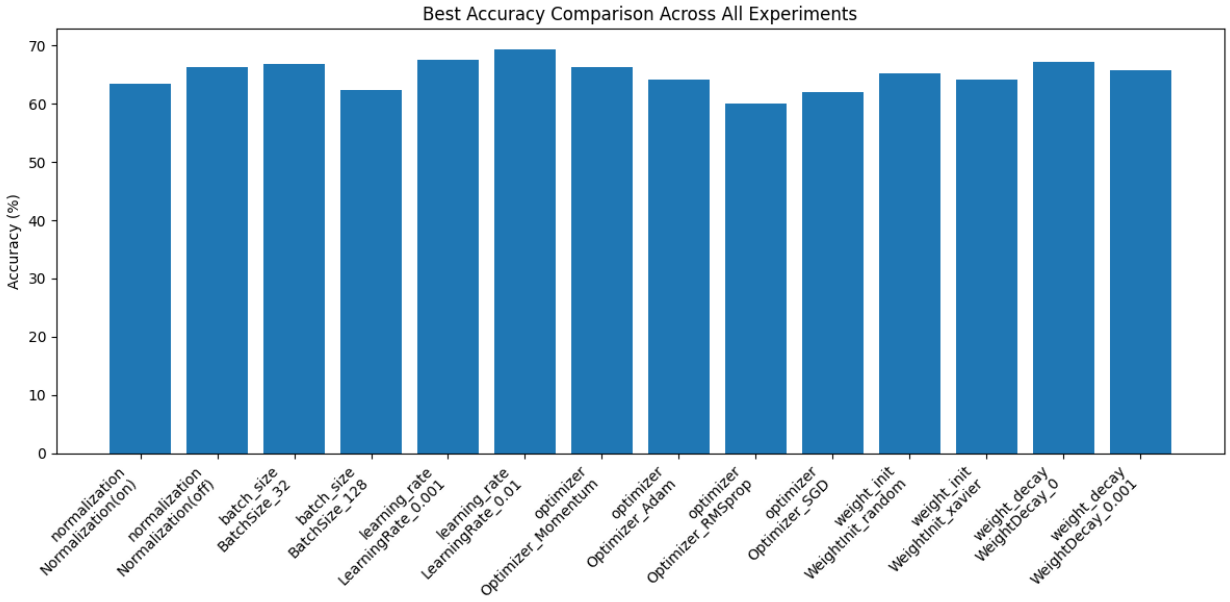
From the experiments comparing weight decay settings (0 and 0.001), I observed notable differences in model behavior and stability. Training without weight decay (WeightDecay_0) achieved a slightly better peak validation accuracy of 67.17% at epoch 14, as shown by the blue line in the validation accuracy plot. The training progression without weight decay showed more pronounced peaks in performance, though this came with increased variability in the validation accuracy curve.

With weight decay set to 0.001, the model reached a peak accuracy of 65.77% at epoch 6, as indicated by the orange line in the plots. While this setting provided some initial stability, it ultimately led to more dramatic fluctuations in validation loss, particularly evident in the middle graph where we see sharp spikes around epochs 10 and 20. The training loss curves (leftmost plot) show that both approaches achieved similar overall descent patterns, though the model with weight decay (orange line) demonstrated slightly more consistent training loss reduction towards the end of training.

These results suggest that for our specific model and dataset, the regularizing effect of weight decay might have been too strong, slightly hampering the model's ability to achieve its best possible performance. The better performance without weight decay indicates that our model wasn't suffering significantly from overfitting, possibly due to the dropout layers and batch normalization already providing sufficient regularization.

Colin Kirby
co20115@ucf.edu

Overall Analysis



From our comprehensive series of experiments testing various hyperparameters and model configurations, the best performing model emerged with a specific combination of settings that achieved a peak validation accuracy of 69.36%. This optimal configuration used a learning rate of 0.01, batch size of 32, Momentum optimizer, random weight initialization, and operated without input normalization but with a small weight decay of 0.001. The model reached its peak performance remarkably quickly, achieving its best accuracy at epoch 1, though this came with some stability trade-offs in later epochs.

Looking at the individual experimental results, we found that higher learning rates (0.01) outperformed lower ones, smaller batch sizes (32) proved more effective than larger ones (128), and the Momentum optimizer demonstrated superior performance compared to Adam, RMSprop, and standard SGD. Surprisingly, simpler approaches often proved more effective, with random weight initialization outperforming Xavier initialization and training without normalization showing better results than with it. The model's architecture, featuring batch normalization layers and a dropout rate of 0.5, along with class weights to handle imbalanced data, proved effective in achieving stable training while managing overfitting.