COMP 6321 Machine Learning

Assignment 4 Answers

Name : Parsa Kamalipour , *StudentID* : 40310734

1 Exercise 1, Question 1: Introduction

In this exercise, I had to implement the VGG11 convolutional neural network and train it on the MNIST dataset. The VGG11 model is a simpler version of the well-known VGG16 model. It has 11 layers with convolutional, batch normalization, and fully connected parts. The goal was to train the model to classify the digits from the MNIST dataset.

2 Exercise 1, Question 1: Methodology

2.1 Environment Setup

I implemented the model in Python using PyTorch. I made sure that the code works for different hardware setups:

- CUDA (for systems with an NVIDIA GPU)
- MPS (for Apple Silicon devices like M1, M2)
- CPU (if no GPU is available)

The following dependencies were used:

- `torch`
- `torchvision`
- `numpy`

2.2 MNIST Dataset

The MNIST dataset was loaded using the `torchvision.datasets` module. The images were resized from 28×28 to 32×32 to match the VGG input size. The dataset was split into training and testing sets, and normalization was applied.

2.3 Model Architecture

The VGG11 model that I implemented for this experiment consists of these layers:

- Convolutional layers with batch normalization and ReLU activation
- Max pooling layers after some convolutional layers
- Fully connected layers with ReLU activation and Dropout
- Output layer with 10 units for the classes (digits 0-9)

The model uses the following configuration for the convolutional and fully connected layers:

```
- Conv(1, 64, 3, 1, 1) - BatchNorm(64) - ReLU - MaxPool(2, 2)
- Conv(64, 128, 3, 1, 1) - BatchNorm(128) - ReLU - MaxPool(2, 2)
- Conv(128, 256, 3, 1, 1) - BatchNorm(256) - ReLU
- Conv(256, 256, 3, 1, 1) - BatchNorm(256) - ReLU - MaxPool(2, 2)
- Conv(256, 512, 3, 1, 1) - BatchNorm(512) - ReLU
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)
- Linear(512, 4096) - ReLU - Dropout(0.5)
- Linear(4096, 4096) - ReLU - Dropout(0.5)
- Linear(4096, 10)
```

3 Exercise 1, Question 1: Training Process

The model was trained with the following settings:

- Loss function: Cross-entropy loss (`torch.nn.CrossEntropyLoss`)
- Optimizer: Stochastic Gradient Descent (SGD) with learning rate 0.01 and momentum 0.9
- Number of epochs: 20
- Batch size: 64

The model was trained for 20 epochs, and I printed the training loss periodically to track the progress. The training loop involved zeroing gradients, forward propagation, computing the loss, backward propagation, and updating the parameters.

4 Exercise 1, Question 1: Results

The model was successfully trained for 20 epochs. The training process showed that the loss decreased steadily, which means the model was learning to classify the digits correctly. I saved the trained model for future evaluation and testing.

5 Exercise 1, Question 2: Introduction

In this exercise, I needed to evaluate the training process of the VGG11 model, which I implemented in Question 1, on the MNIST dataset. The goal was to analyze the model's performance by plotting both the training and testing accuracies, as well as the training and testing losses for each epoch. This helps me understand how well the model is learning and how it generalizes.

6 Exercise 1, Question 2: Methodology

I used Python and PyTorch to perform the evaluation. The model that I trained earlier was loaded, and I evaluated it using both the training and test datasets. For the evaluation, I stored the training loss, test loss, training accuracy, and test accuracy for each epoch.

6.1 Environment Setup

I ran the evaluation using the same environment setup as before, where the hardware setup included:

- CUDA (for systems with NVIDIA GPU)
- MPS (for Apple Silicon devices like M1, M2)
- CPU (if no GPU is available)

6.2 Evaluation Procedure

The model was evaluated for 20 epochs. In each epoch, I:

- Trained the model and computed the training loss.
- Evaluated the training accuracy by checking how well the model predicted labels on the training data.
- Evaluated the test accuracy and test loss by using the model on the test data without updating the weights.

The metrics were calculated using the following functions:

- `calculate_accuracy`: This function calculates the percentage of correct predictions out of the total samples.
- `calculate_loss`: This function computes the average loss across all samples.

7 Exercise 1, Question 2: Results

The evaluation process showed how the model's accuracy and loss evolved over time.

- The training accuracy and test accuracy were recorded at the end of each epoch.
- The training and test loss were also recorded to see if the model was converging well.

The graphs for the training and test accuracy and loss are shown in Figures 1 and 2.

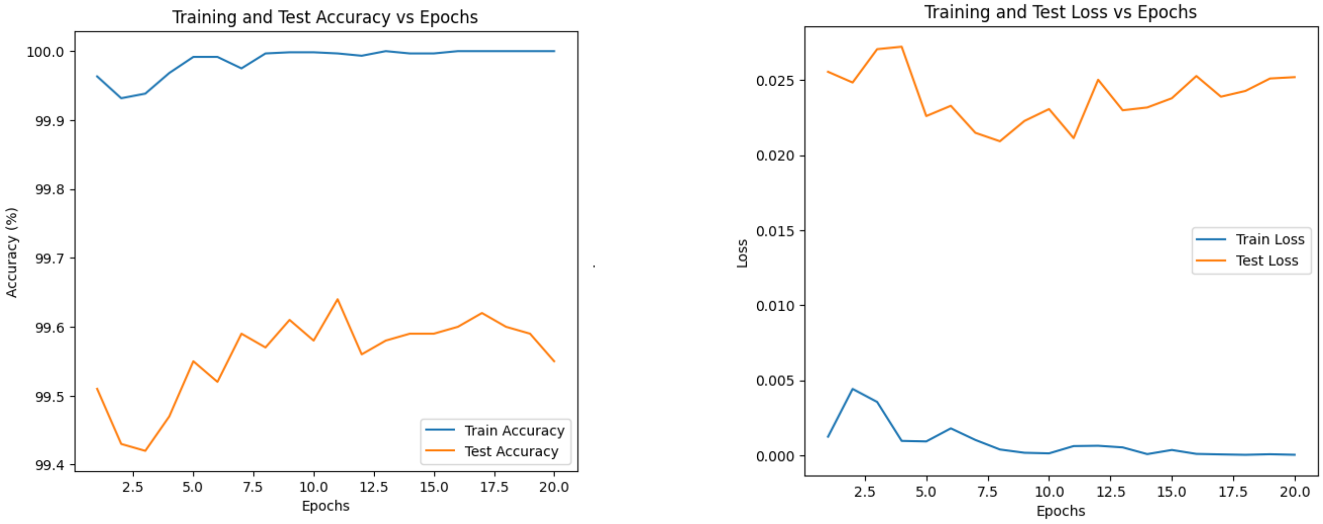

Figure 1: Training and Test Accuracy vs Epochs

Figure 2: Training and Test Loss vs Epochs

Figure 3: Training and Test Metrics for VGG11 Model on MNIST

The results show that the training accuracy was very high, almost reaching 100% throughout the epochs, while the test accuracy was slightly lower and fluctuated between 99.4% and 99.7%. This indicates that the model performed well on the training data but had some difficulty generalizing to new data.

The training loss was very low and approached zero by the end of the training, which means the model was fitting the training data very well. However, the test loss was higher and fluctuated, suggesting that the model had some challenges generalizing perfectly to the test data. The fluctuations in test accuracy and loss indicate that while the model was effective, there was still some room for improvement in terms of generalization.

The graphs also helped me check if the model was overfitting. Although the training accuracy was higher than the test accuracy, the difference was not too large, suggesting that overfitting was not severe, but there was some overfitting present.

8 Exercise 1, Question 2: Conclusion

From the evaluation, I observed that the VGG11 model was able to learn the MNIST dataset quite well over the course of 20 epochs. The training accuracy was consistently high, and the training loss decreased significantly. However, the test accuracy was lower than the training accuracy, and the test loss fluctuated, indicating that the model's ability to generalize was not perfect, and some overfitting might be present.

9 Exercise 1, Question 3: Introduction

In this part of the exercise, I evaluated the generalization capabilities of the VGG11 model on the MNIST dataset under different transformations. The goal was to understand how well the model performs when test images are flipped, blurred, or have Gaussian noise added to them. By doing this, I wanted to see how robust the model is to changes in the input data that it did not encounter during training.

10 Exercise 1, Question 3: Methodology

The model was evaluated on the original MNIST test set as well as on versions of the test set that were transformed in several ways:

- **Horizontal Flip**: Images were flipped from left to right.
- **Vertical Flip**: Images were flipped from top to bottom.
- **Combined Horizontal and Vertical Flip**: Images were flipped both horizontally and vertically.
- **Gaussian Blur**: Images were blurred with a Gaussian filter.
- **Gaussian Noise**: Gaussian noise was added to the images with variances of 0.01, 0.1, and 1.

The VGG11 model was evaluated on each of these transformed datasets, and the test accuracy was calculated for each transformation.

11 Exercise 1, Question 3: Results

The results of the evaluation are summarized in Table 1.

Transformation	Accuracy (%)
Original Test Set	99.09
Horizontally Flipped Test Set	38.49
Vertically Flipped Test Set	41.79
Horizontally and Vertically Flipped Test Set	42.06
Blurred Test Set	97.65
Gaussian Noise (Variance 0.01)	99.09
Gaussian Noise (Variance 0.1)	98.84
Gaussian Noise (Variance 1)	10.55

Table 1: Test accuracy of VGG11 on different transformations of the MNIST dataset.

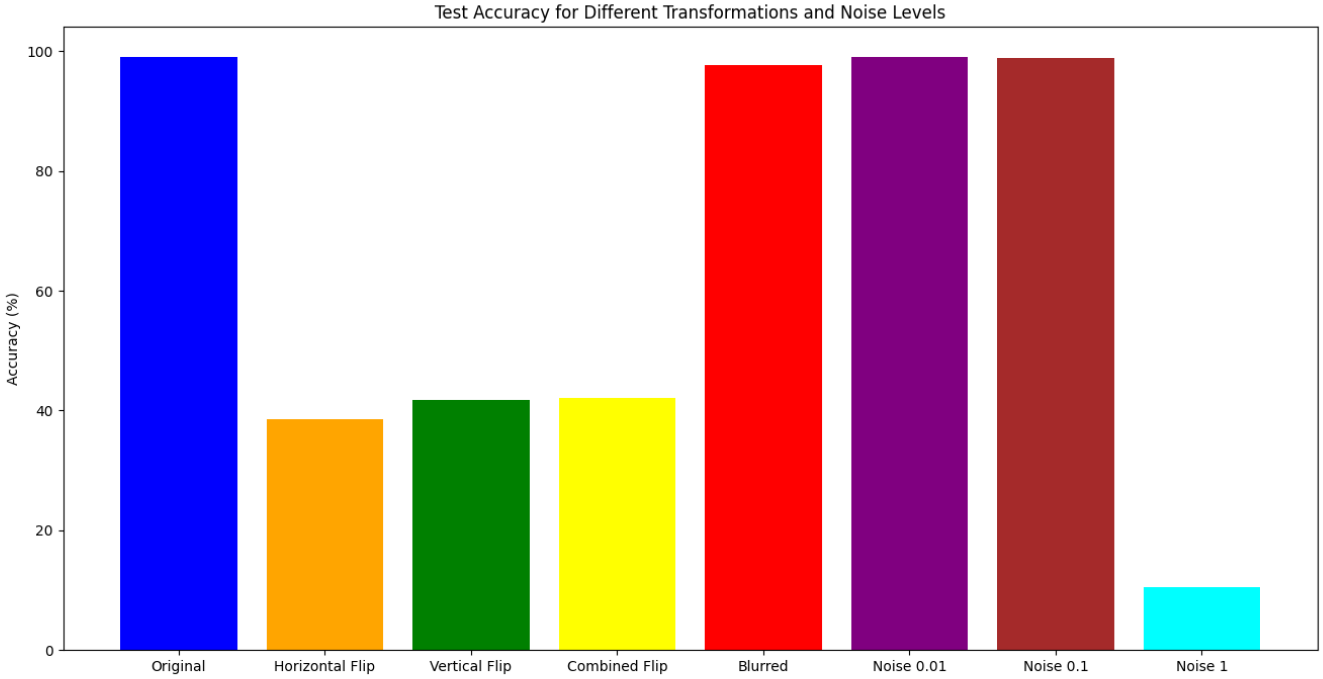

Figure 4: Plot accuracies for better visualization

12 Exercise 1, Question 3: Analysis

The results show that the VGG11 model performed very well on the original MNIST test set, achieving an accuracy of 99.09%. The model also handled small levels of Gaussian noise (variance 0.01 and 0.1) quite well, with accuracies of 99.09% and 98.84%, respectively. However, when the noise level was increased to a variance of 1, the model's accuracy dropped drastically to 10.55%. This suggests that the model struggles with high levels of noise.
The model's performance on horizontally and vertically flipped images was significantly lower, with accuracies of 38.49%, 41.79%, and 42.06% for horizontal, vertical, and combined flips, respectively. This indicates that the model is not invariant to image flips, likely because it was not trained on flipped versions of the images. The accuracy on blurred images was 97.65%, showing that the model is relatively robust to minor blurring.

13 Exercise 1, Question 3: Conclusion

The evaluation shows that the VGG11 model is highly accurate on the original MNIST test set and is also fairly robust to small levels of Gaussian noise and blurring. However, the model struggles with flipped images and high levels of noise. This suggests that the model's training did not include sufficient variations of the data to handle these transformations effectively.

14 Exercise 1, Question 4: Introduction

In this part of the exercise, I retrained the VGG11 model on the MNIST dataset, but this time I applied different data augmentation techniques. The goal was to observe how data augmentation helps the model generalize better and how it affects the overall performance. I used transformations such as random horizontal flips, rotations, affine transformations, and color jitter to make the training data more diverse and challenging.

15 Exercise 1, Question 4: Methodology

For the retraining, I used the following data augmentation techniques:

- **Random Horizontal Flip**: Randomly flipped images horizontally with a 50% probability.
- **Random Rotation**: Applied random rotations up to 10 degrees.
- **Random Affine Transformation**: Applied affine transformations with translations of up to 10% of the image.
- **Color Jitter**: Adjusted brightness, contrast, saturation, and hue with small variations to make the images appear different.

The training was done for 20 epochs, and the loss was calculated at each mini-batch. I also evaluated the model on the test set after each epoch to observe the changes in accuracy.

16 Exercise 1, Question 4: Results

The final test accuracy after retraining the model with data augmentation was 99.03%, which is slightly lower compared to the original training without data augmentation. The training loss fluctuated throughout the epochs, indicating that the model faced challenges in some mini-batches, but overall it was able to learn effectively. Below are the training and test metrics observed during the retraining process.

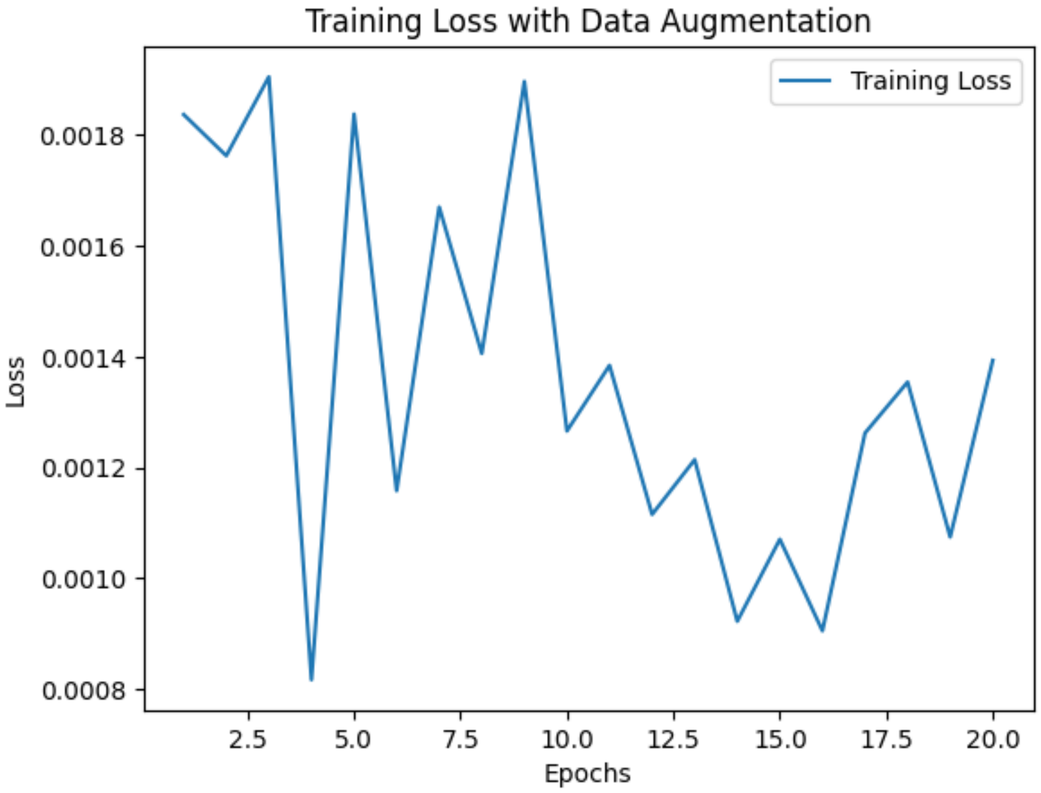

Figure 5: Training Loss over 20 Epochs with Data Augmentation

Figure 5 shows the fluctuation in training loss over the epochs. The model managed to achieve a good level of learning despite the variations in the loss. The test accuracy was consistently high, around 99%, throughout the retraining process, which shows that the data augmentation did not hurt the model's performance.

17 Exercise 1, Question 4: Analysis

The results show that the VGG11 model, even with augmented data, maintained a high level of accuracy on the MNIST test set. Although the final test accuracy was slightly lower compared to the training without augmentation, I believe this slight drop is because the augmented data makes learning more challenging, which prevents overfitting. The diverse augmented images allowed the model to learn more general features, which is beneficial for better generalization.

18 Exercise 1, Question 4: Conclusion

Retraining the VGG11 model with data augmentation showed that the model could still achieve a high accuracy of 99.03% on the test set. The training loss fluctuated, and the test accuracy remained high, which suggests that data augmentation helped the model learn more robust features and avoid overfitting. This confirms that data augmentation is an effective way to improve the generalization capabilities of deep learning models.

19 Exercise 1, Question 5: Introduction

In this part of Exercise 1, I compared the performance of three different optimizers (AdaDelta, Adam, and SGD) while training the VGG11 model on the MNIST dataset. The objective was to see how each optimizer affected the accuracy and convergence during training and on the test set.

20 Exercise 1, Question 5: Methodology

For training, I used the following three optimizers:

- **AdaDelta**: Used with a learning rate of 1.0.
- **Adam**: Used with a learning rate of 0.001.
- **SGD (Stochastic Gradient Descent)**: Used with a learning rate of 0.01 and momentum of 0.9.

I trained each model for 20 epochs, tracking both the training and test accuracies after each epoch. The results were plotted to compare the performances visually.

21 Exercise 1, Question 5: Results

The training accuracy and test accuracy results for all three optimizers over 20 epochs are shown in the figures below.

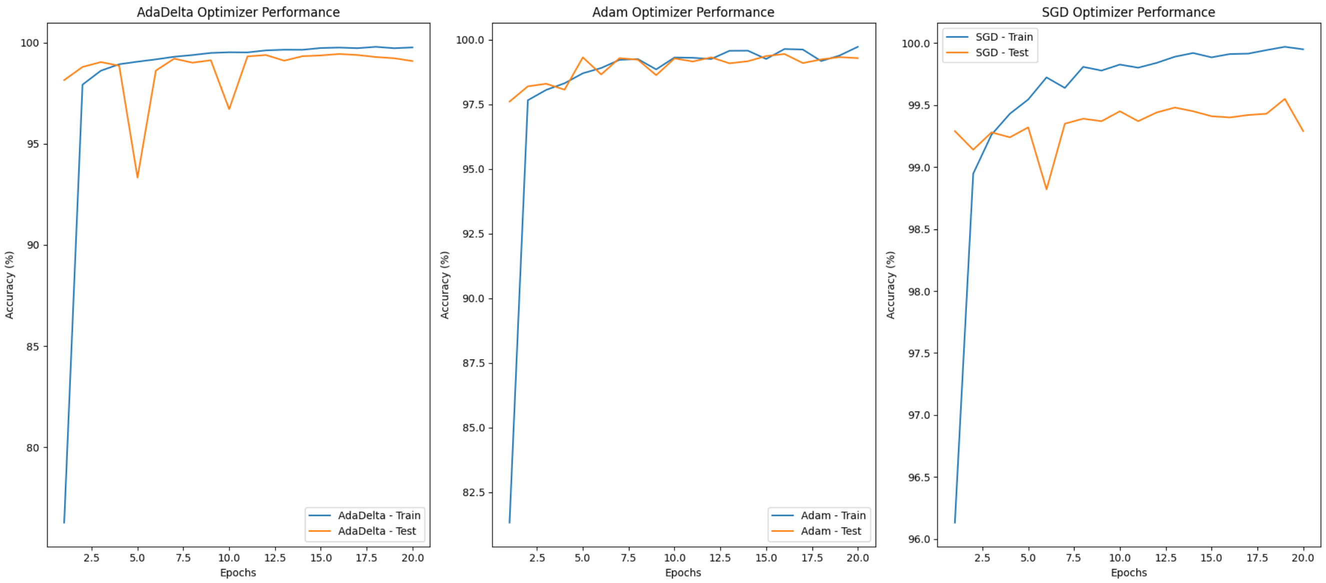

Figure 6: Training and Test Accuracies for AdaDelta, Adam, and SGD Optimizers

The performance of each optimizer was also compared in a combined plot, which is shown in Figure 7.

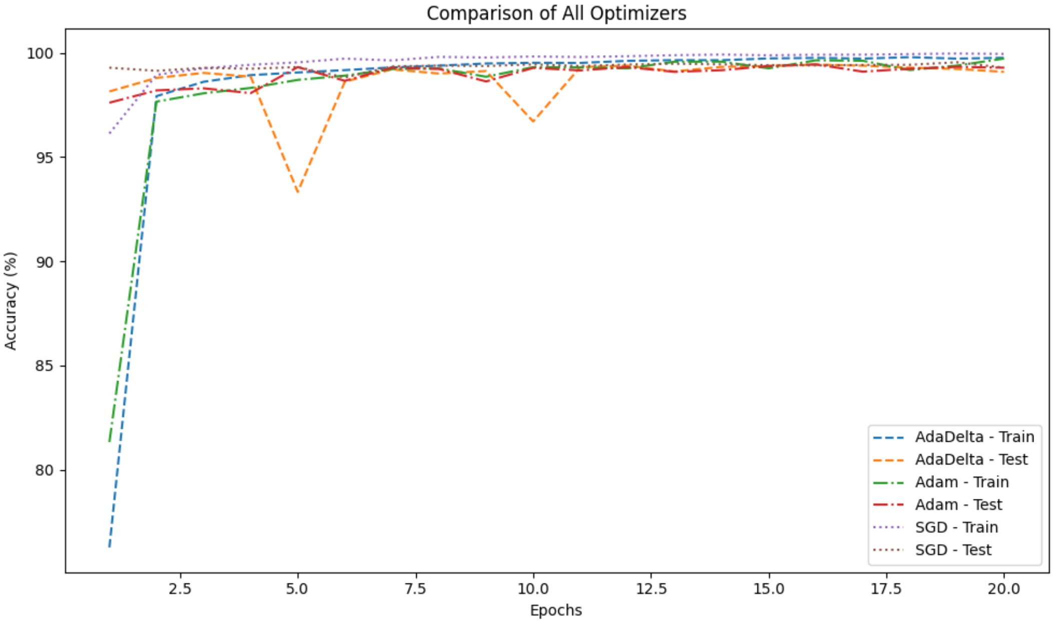

Figure 7: Comparison of All Optimizers (AdaDelta, Adam, SGD)

22 Exercise 1, Question 5: Analysis

Based on the training results, the final test accuracy for each optimizer was as follows:

- **AdaDelta**: Achieved a maximum test accuracy of 99.44%. Test accuracy fluctuated during training, with a notable dip in epoch 5 (93.33%).
- **Adam**: Achieved a maximum test accuracy of 99.45%. The accuracy was mostly consistent, with a slight dip in epoch 9 (98.63%).
- **SGD**: Achieved a maximum test accuracy of 99.55%. It showed a steady improvement and had the highest overall accuracy.

The SGD optimizer showed the highest final test accuracy, while both AdaDelta and Adam also performed quite well. SGD seemed to converge to a high accuracy more steadily compared to AdaDelta, which had fluctuations in the test accuracy during training. Adam also showed a stable performance similar to SGD, but with a slightly slower convergence.

From the results:

- **AdaDelta** started with a lower training accuracy (76.27%) but quickly improved, reaching a high training accuracy of 99.77%. However, it had some instability in the test accuracy during training.
- **Adam** had a more stable training process, starting with a training accuracy of 81.33% and achieving 99.73% by the end of training. The test accuracy remained mostly consistent.
- **SGD** started with a high training accuracy (96.13%) and reached 99.95% by the end of training. It achieved the best final test accuracy (99.55%) and showed consistent improvement throughout the training process.

In my opinion, the SGD optimizer is better for this dataset because it reached the highest accuracy and had a more consistent performance across the epochs. However, both Adam and AdaDelta are good choices as they also achieved high accuracy, and they might be preferred in other scenarios due to their adaptive learning rate capabilities.

23 Exercise 1, Question 6: Introduction

In this part of Exercise 1, I replaced the ReLU activations in the VGG11 model with Sigmoid activations. The objective was to understand the impact of using Sigmoid activations compared to ReLU on the training and test accuracies of the model.

24 Exercise 1, Question 6: Methodology

I trained the VGG11 model using Sigmoid activation functions in place of ReLU. The optimizer used for training was Stochastic Gradient Descent (SGD) with a learning rate of 0.01 and momentum of 0.9, similar to what was used in Exercise 1.2. The model was trained for 20 epochs, and the results for both training and test accuracies were plotted for analysis.

25 Exercise 1, Question 6: Results

The training and test accuracies of the model with Sigmoid activations are shown in Figure 8.

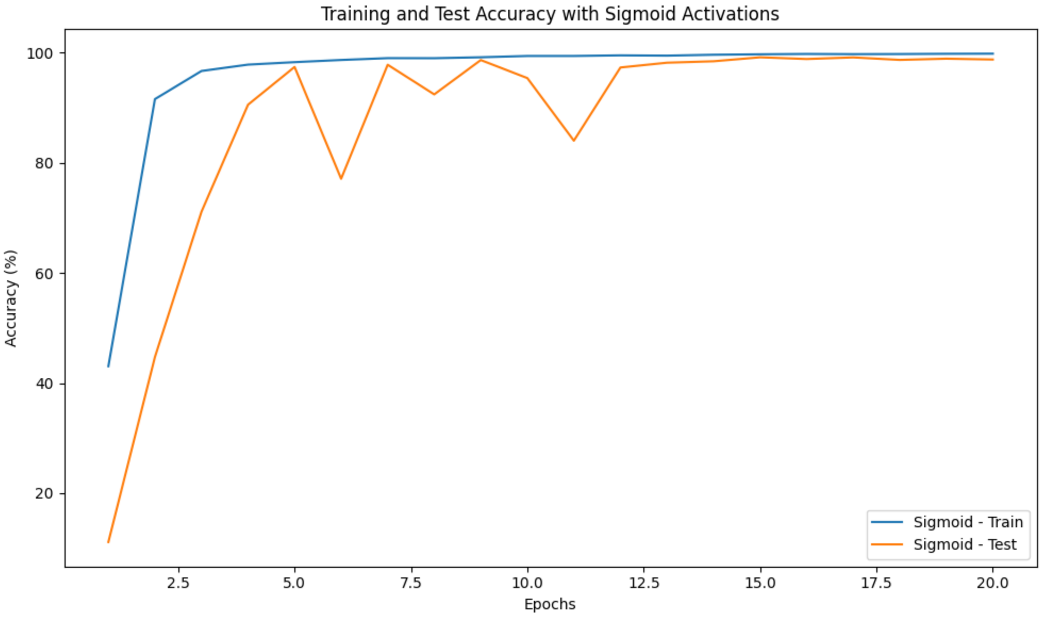

Figure 8: Training and Test Accuracy with Sigmoid Activations

The comparison between the Sigmoid and ReLU activation functions is shown in Figure 9.

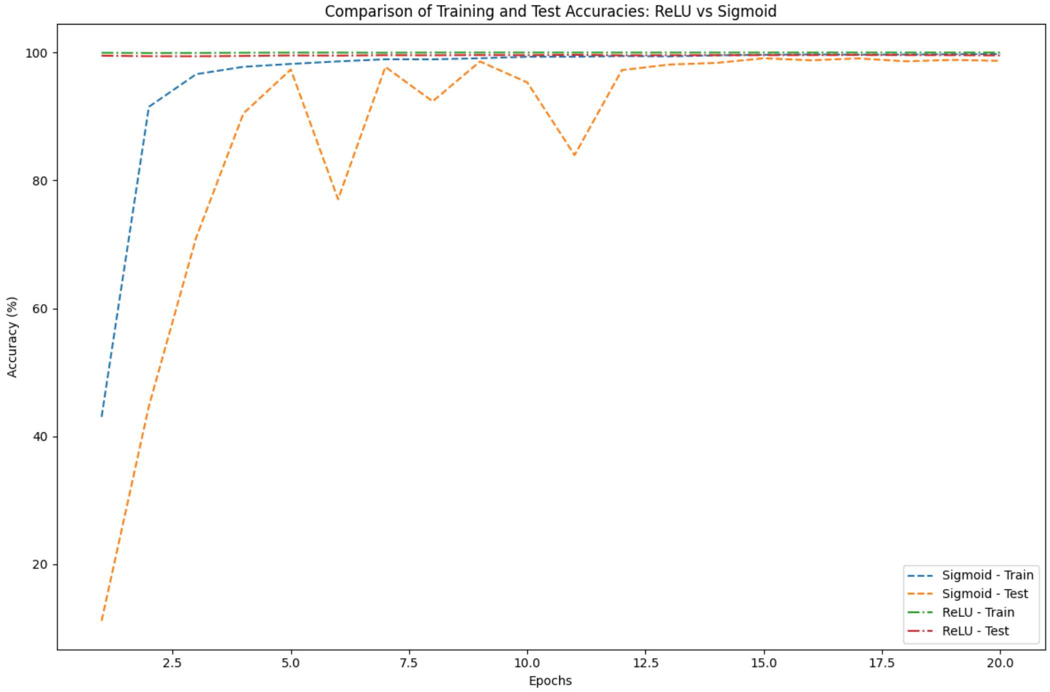

Figure 9: Comparison of Training and Test Accuracies: ReLU vs Sigmoid

26 Exercise 1, Question 6: Analysis

Based on the training results, I observed significant differences between the performance of Sigmoid and ReLU activations:

- **Training Accuracy**: The Sigmoid model eventually reached a high training accuracy (99.76%), similar to the ReLU model. However, it started with a lower initial accuracy and took more epochs to converge compared to the ReLU model.
- **Test Accuracy**: The test accuracy for the Sigmoid model fluctuated more compared to the ReLU model. The Sigmoid activation caused the model to struggle with generalization, especially in the earlier epochs, as evident by the dips in accuracy (e.g., in Epochs 6, 11).
- **Comparison**: ReLU activations performed better overall, with more stable and higher test accuracy. The Sigmoid model had issues like vanishing gradients, which made convergence slower and less consistent.

The ReLU model maintained a more consistent and higher test accuracy throughout training. Sigmoid activations tend to suffer from the vanishing gradient problem, which affects the learning capability of deep networks like VGG11. This explains the slower convergence and fluctuating test accuracy observed in the Sigmoid model.

27 Exercise 1, Question 6: Conclusion

In conclusion, the comparison showed that ReLU is a more suitable activation function for the VGG11 model on the MNIST dataset. ReLU provided better convergence and higher stability in both training and testing phases, whereas Sigmoid activations led to slower convergence and fluctuating test accuracy due to the vanishing gradient issue.

28 References

- PyTorch documentation: `https://pytorch.org/`
- Torchvision datasets: `https://pytorch.org/vision/stable/datasets.html#mnist`