

|  |  |
| --- | --- |
| Simon Chaisouang  shchaiso@uark.edu  010878218  Late Days: 0 | Venkata Kumari  vg021@uark.edu 011088021  Late Days: 0 |

CSCE 50103: Full Stack Deep Learning, Spring 2025

Homework #1

## Experimentation and Optimization of Character Network for MNIST Dataset - 1

#### **Introduction**

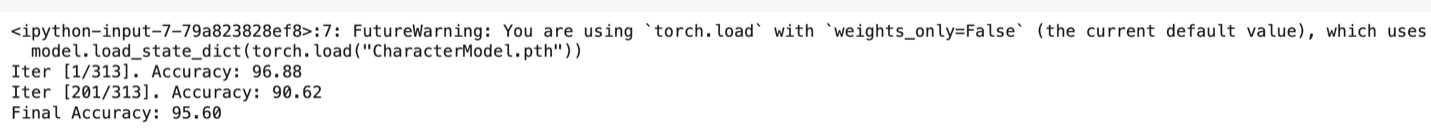
This report documents the process of experimenting with and optimizing the character network to achieve better performance on the MNIST dataset. The focus was on modifying the hyperparameters, particularly the batch size, learning rate, number of epochs, and activation function, while using the Adam optimizer for training.

**Default parameters:**

* **Batch Size**: 32
* **Learning Rate**: 0.1
* **Number of Epochs**: 1
* **Activation Function**: Sigmoid for hidden layer
* **Optimizer**: SGD

A screenshot of a graph

Description automatically generated



**Figure 1.1 Results of Default Network Architecture**

#### **Experiment Details**

The model was trained with the following configuration:

* **Batch Size**: 64
* **Learning Rate**: 0.001
* **Number of Epochs**: 10
* **Activation Function**: ReLU for hidden layers
* **Optimizer**: Adam

The dataset used was the MNIST dataset, containing 60,000 training samples and 10,000 test samples. The training data was divided into batches of 64 images, and the learning rate of 0.001 was applied to control the step size during optimization.

#### **Modifications Made**

The primary architectural change was the use of the ReLU activation function in hidden layers instead of Sigmoid. ReLU is known for faster convergence and avoiding vanishing gradient issues, making it a suitable choice for this task. The Adam optimizer, with its adaptive learning rate capabilities, was used to enhance training efficiency.

The number of epochs was set to 10 to limit the training time while ensuring sufficient exposure of the model to the dataset. The batch size of 64 was chosen as a balance between memory efficiency and gradient stability.

#### **Training and Evaluation**

The model was trained using the modified parameters and architecture. During training, the loss and accuracy were monitored. At the end of 10 epochs, the final performance on the test dataset was evaluated.

* **Training Results**:
  + The loss steadily decreased across the 10 epochs, indicating that the model effectively learned the patterns in the dataset.
  + The accuracy on the training set reached over 98%, showing that the model performed well during training.
* **Testing Results**:
  + The final accuracy on the testing set was approximately **97%**.
  + This result demonstrates that the model successfully generalized to unseen data.

A screenshot of a computer

Description automatically generated

A graph of loss and training

Description automatically generated with medium confidence

**Figure 1.1 Results of Enhanced Network Architecture**

#### **Observations**

1. **Batch Size**: A batch size of 64 provided stable gradient updates without significant memory constraints, contributing to smooth training progress.
2. **Learning Rate**: The learning rate of 0.001 was small enough to ensure convergence without overshooting the optimal solution.
3. **ReLU Activation**: The switch to ReLU significantly improved the training speed compared to Sigmoid, as ReLU avoids saturation issues.
4. **Adam Optimizer**: Adam proved effective for this task, adjusting the learning rate adaptively during training.

#### **Conclusion**

The modifications in hyperparameters and architecture led to a test accuracy of **97%** after 10 epochs, showcasing the effectiveness of the changes. The use of ReLU and Adam optimizer, combined with an appropriate batch size and learning rate, contributed to efficient training and improved performance. This task highlights the importance of careful tuning of hyperparameters and network design in achieving optimal results for image classification tasks.

## Integrating Character Detection and Recognition - 2

The **detect\_characters** function detects and classifies characters from a given input image using the previously trained character recognition model. The function starts by first loading the image using OpenCV and resizes 333 x 75 pixels to ensure consistency in preprocessing. The image is then converted to a greyscale where a threshold and operations are applied to enhance the contrast between the characters and the background.

Bound boxes are generated, and the **find\_contours** function is called to filter out small and large contours that do not resemble characters. A list of character images is returned and their bounding boxes. Each detected character is then resized to 28x28 pixels and converted to a tensor so it can be used in the trained model where its digit is predicted. The coordinates for detected character are retrieved, and a bounding box along with its prediction as the label is output as shown below.

|  |
| --- |
| A red sign with white numbers and green rectangles  AI-generated content may be incorrect. |
| A close-up of a sign  AI-generated content may be incorrect. |
| A red sign with white numbers and green squares  AI-generated content may be incorrect. |
|  |
| **Figure 2.1 Results of Default Network** |
|  |

After training the neural network with the given default parameters specified in the last section and testing it on the license plate images, we achieve the above results. As seen in the Manila license plate, the model is able to near-perfectly capture the characters inside each bounding box and provide an accurate prediction for each digit. However, the model did very poorly in providing accurate predictions for the Rhode Island plate. The misclassifications could possibly be due to font variations. Since the model is only trained using the MNIST dataset, it may perform poorly on digits with varying fonts.

Noise and lighting conditions from the image could also have caused the model to make low-confidence predictions. Looking onto the N.J. license plate, the model was able to provide an accurate prediction for digits such as: 0, 7, and 4. As for the three 1’s, the model couldn’t make a correct prediction. Here is another case of font variation being a potential cause of misclassification. There are also noted some minor misalignments with the bounding boxes for some of the digits detected on the plate.

|  |
| --- |
|  |
|  |
|  |
| **Figure 2.2 Results of Modified Network** |

After training and testing the data on our enhanced model architecture, the results pose a very similar outcome to the default network architecture. The new model was able to make accurate predictions on the manila license plate but failed to make accurate predictions for the Rhode Island plate and some of the digits on the N.J. plate.

From both results we can potentially conclude that overfitting of the data may have taken place in both models. Even though both models showcased high accuracy of around 95% and 97%, both models failed to make accurate predictions for license plates with different font variations and noise. This is possible due to the limitations of training solely on the MNIST dataset. Both models may predict poorly on images that contain different fonts. Therefore, in the future we can consider expanding the training data to account for more real-world examples of license plate images that contain varying fonts and noises.