Handwritten Digit Classification using a Self-Designed Artificial Neural Network (January 2024)

Bishwambhar Dahal[1] **and Sirjana Bhatta**[1]

[1]Department of Electronics and Computer Engineering, IOE, Thapathali Campus, Kathmandu 44600, Nepal
Corresponding author: Bishwambhar Dahal(dahalbishwambhar@gmail.com) and Sirjana Bhatta(sirjanabhatta6@gmail.com)

Abstract This project presents the development and application of a self-designed Artificial Neural Network (ANN) for the classification of handwritten digits. The rise of deep learning and neural networks has sparked significant interest in solving complex pattern recognition problems. In this study, we propose a novel approach to address the task of handwritten digit classification using a custom-built ANN. Furthermore, the project explores the effects of various hyperparameters, activation functions, and training strategies on the network's performance. By implementing various initialization techniques like Kaimig initialization and Xavier initialization, dropout and multiple hidden layers, we analyzed the behaviour of various neural network models. This analysis contributes to a deeper understanding of neural network behavior and aids in the fine-tuning of the ANN architecture. The successful implementation of the self-designed ANN for handwritten digit classification showcases the potential of custom neural network architectures in solving real-world pattern recognition tasks.

Keywords *Artificial Neural Network, Handwritten Digit Classification, Pattern Recognition, Multi-Layer Perceptron, Deep Learning.*

I Introduction

In the realm of artificial intelligence and machine learning, the resurgence of neural networks has redefined the landscape of pattern recognition and classification. Handwritten digit classification, a quintessential challenge within this domain, has experienced a paradigm shift with the advent of deep learning. This study introduces an innovative approach by employing a self-designed Artificial Neural Network (ANN) [1] enriched with a spectrum of activation functions, including Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh), Sigmoid, and Softmax activations, for the precise and efficient classification of handwritten digits.

Traditional methodologies for digit recognition often relied on manual feature engineering and rule-based algorithms, struggling to adapt to the intricacies of varying writing styles and complex digit structures. The resurgence of neural networks, particularly deep architectures, has empowered systems to autonomously extract hierarchical representations directly from raw data. In this project, we leverage the potency of neural networks and, crucially, explore the impact of diverse activation functions on their performance.

The fundamental aim of this study is to underscore the versatility and effectiveness of a self-designed ANN, distinguished by its incorporation of a range of activation functions. Through the construction of a neural network architecture from scratch and the integration of multiple activation functions, we unravel the intricate behaviors of these functions and their roles in the handwritten digit classification process. This approach not only addresses challenges inherent to handwritten digit recognition, such as varying writing styles and intricate patterns, but also provides insights into the interplay between activation functions and network performance.

We offer a comprehensive exposition of our custom-designed ANN, elucidating its architectural components, activation functions, and training modalities. The study encompasses meticulous data preparation, preprocessing, and augmentation steps to curate a well-tailored dataset, forming the bedrock for training and evaluating the neural network's performance. Furthermore, an exhaustive assessment of the ANN's capabilities is conducted, focusing on key performance metrics like accuracy, precision, recall, and F1-score.

In parallel with the activation functions exploration, this research probes the influence of distinct hyperparameters and training techniques on the neural network's efficacy. Implementation of various initialization techniques to draw insights on effect of initialization on neural networks is done. By systematically unraveling the impact of different activation functions on handwritten digit classification, we provide valuable insights into the nuanced interactions between activation functions and neural network architectures. By implementing various models, we analyzed the ceffect of hypermeters on convergence time and accuracy.

In summary, this project stands as a testament to the prowess of a self-designed Artificial Neural Network in the realm of handwritten digit classification, enriched by the integration of diverse activation functions. Through a fusion of theoretical exploration and practical implementation, this study contributes to the broader understanding of custom neural network architectures and their pivotal role in surmounting real-world pattern recognition challenges.

II Methodology

A Brief Theory

Artificial Neural Networks are computational models inspired by the structure and function of biological neural networks in the human brain. ANNs consist of interconnected nodes, also known as neurons, organized into layers. Each neuron performs a weighted sum of its inputs, passes the result through an activation function, and produces an output. ANNs are capable of learning patterns and relationships in data through a process called training.

ANN consists of a input layer, various hidden layers and a output layer. The input layer receives raw data or features from the external world. In the context of handwritten digit classification, each input neuron corresponds to a pixel in a digit image. Hidden layers are the intermediate layers between the input and output layers. These layers process and transform the information from the input layer through a series of weighted connections and activation functions. The number of hidden layers and the number of neurons in each hidden layer are hyperparameters that can impact the network's capacity to learn complex patterns. The output layer produces the final predictions or classifications based on the computations performed in the hidden layers. For handwritten digit classification, each neuron in the output layer corresponds to a possible digit class (0-9).

The architecture of the presented project is underpinned by a cohesive framework encompassing several essential components, each contributing uniquely to the functionality and efficacy of the self-designed Artificial Neural Network (ANN) for handwritten digit classification.

Neuron computation constitutes a fundamental cornerstone, encompassing the dynamic interplay of weights, biases, and activation functions. Weights, serving as adjustable parameters, intricately modulate the strength of connections between neurons, thereby regulating the transmission of information throughout the network. Concurrently, biases inject adaptability by introducing an adjustable bias term to the neurons' weighted sum, effectively steering the outcome of activation functions. The ANN's training phase facilitates the adaptation of these weights and biases to minimize the dissonance between predicted and actual outputs, thereby enhancing classification accuracy.

Activation functions, a pivotal facet of neural network operations, inject the requisite non-linearity essential for processing complex patterns. This suite of functions, including Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh), Sigmoid, and Softmax, collectively contributes to shaping the transformed output of neurons [2]. By imbuing outputs with non-linear properties, activation functions enhance the network's capacity to capture intricate relationships within the input data. The ReLU actovation function returns the input "z" if it is greater than or equal to zero, and returns zero otherwise. The tanh function scales the input values to a range between -1 and 1, providing a smooth curve with S-shaped characteristics. The sigmoid activation function transforms the input using the exponential function to produce an output value between 0 and 1, making it suitable for generating probabilities in classification tasks.

The propagation mechanisms, comprising forward propagation and backpropagation, orchestrate the flow of information within the network. Forward propagation propels input data through successive layers, with neurons calculating weighted sums followed by activation function applications. This cascade of operations culminates in the production of predictions. Conversely, the iterative process of backpropagation, an indispensable facet of training, underpins weight and bias adjustments by computing prediction errors and propagating these errors backward through the network. Optimization techniques, exemplified by gradient descent, intricately recalibrate weights and biases to fine-tune the network's predictive capabilities.

The training phase embodies the heart of the project's neural network architecture. Iterative exposure to labeled data guides weight and bias adaptation, enabling the network to learn intricate patterns inherent in handwritten digit images. Subsequently, the model's performance is meticulously evaluated utilizing a distinct dataset. Metrics such as accuracy, precision, recall, and F1-score furnish a comprehensive overview of the network's proficiency in classifying digits, thus underscoring the culmination of the project's architecture in achieving accurate and efficient handwritten digit recognition.

Gradient descent is an optimization algorithm commonly used in machine learning and deep learning to minimize a function, typically a loss function, by iteratively adjusting the parameters of a model. It's a fundamental technique for training models and finding the optimal set of parameters that result in the best performance on a given task.

Dropout and regularization are techniques used in Artificial Neural Networks (ANNs) to prevent overfitting, which occurs when a model learns to perform exceptionally well on the training data but fails to generalize to new, unseen data. Both dropout and regularization help to improve the model's ability to generalize by reducing the impact of complex patterns that might be specific to the training data. Dropout is a regularization technique where randomly selected neurons are ignored during training. During each training iteration, individual neurons are ”dropped out” with a certain probability, meaning they do not contribute to the forward pass and backpropagation for that iteration. This prevents the network from relying too heavily on any particular neuron, ensuring that different subsets of neurons learn meaningful features. Dropout essentially acts as an ensemble technique, as it trains multiple subnetworks with shared weights.

Kaiming Initialization (also known as He Initialization) and Xavier Initialization (also known as Glorot Initialization) are techniques used to initialize the weights of the neurons in neural networks. Proper weight initialization can significantly impact the convergence speed and overall training performance of deep learning models. The only difference is that the Kaiming takes into account the activation function, whereas Xavier does not (or rather, Xavier approximates the derivative at 0 of the activation function by 1).

B Mathematical Formulae

1 Activation Functions

The sigmoid activation function, often denoted as σ(z), is a key element of neuron computation in Artificial Neural Networks (ANNs). It transforms the weighted sum of inputs (z) into an output value within the range of 0 to 1. The formula for the sigmoid activation function is:

$$Sigmoid(z) = \frac{1}{1+e^{-z}} \tag{1}$$

Here, ”e” represents the base of the natural logarithm, and ”z” is the weighted sum of inputs and biases.
The derivative of the sigmoid function, also known as the logistic function is given as:

$$\frac{d\sigma(x)}{dx} = \sigma(x)\cdot(1-\sigma(x)) \tag{2}$$

The formula for the Rectified Linear Unit (ReLU) activation function is:

$$f(z) = \max(0, z) \tag{3}$$

Here, ”f(z)” represents the output of the ReLU activation function for a given input ”z”.

The derivative of the Rectified Linear Unit (ReLU) activation function is a piecewise function given as:

$$\frac{d}{dx}(\text{ReLU}(x)) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{4}$$

The formula for the Hyperbolic Tangent (tanh) activation function is as:

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{5}$$

The derivative of Hyperbolic Tangent (tanh) activation function is given as:

$$\frac{d}{dx}(\tanh(x)) = \text{sech}^2(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}} \cdot \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6}$$

Mathematically, if we have an input vector $z = [z_1, z_2, \ldots, z_n]$, the Softmax function computes the probabilities p_i for each class i as follows:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{7}$$

2 Forward Propagation Equations

The forward propagation equations used in building the neural network are as follows:

$$Z_1 = W_1 \cdot X + b_1 \tag{8}$$

where,

- Z_1 represents the output from the input layer.

Let A_1 be the output after Z_1 is fed through the activation function. Then,

$$A_1 = ReLU(Z_1) \tag{9}$$

Now, the final output of the hidden layer will act as the input for the output layer, i.e. A_1 will be the input to the output layer. Let W_2 be the weight matrix of the output layer. Let Z_2 be the output of the hidden layer, then its value will be given by:

$$Z_2 = W_2 \cdot A_1 + b_2 \tag{10}$$

where,

- b_2 stands for the bias matrix applied to the hidden layer.

Let A_2 be the output after Z_2 is fed through the activation function. Then,

$$A_2 = SoftMAX(Z_2) \tag{11}$$

where,

- the size of A_2 will be the same as Z_2.

For the added hidden layers, the output of the layer will still follow the same equation format. Let Z_i be the output after the respective hidden layer with weight matrix W_i and bias matrix b_i.

$$Z_i = W_i \cdot input + b_i \tag{12}$$

where,

- $input$ is the input vector to the respective layer.

Likewise, the output for the hidden layer after the respective activation will be:
For Sigmoid activated hidden layer,

$$A_i = Sigmoid(Z_i) \tag{13}$$

and for hyberbolic tangent activated hidden layer,

$$A_i = tanh(Z_i) \tag{14}$$

3 Backward Propagation

The ANN we created, uses categorical cross entropy loss as its primary cost function. The formula for categorical cross-entropy loss is given by:

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{15}$$

Where:

$L(y, \hat{y})$	: Categorical cross-entropy loss
N	: Number of classes
y_i	: Actual target label (ground truth) for class i
$\hat{y}_i$	: Predicted probability for class i

Based on the gradient descent optimization method, the back propagation equations can be represented as follows:

$$\Delta Z_2 = A_2 - \mathrm{Y} \tag{16}$$

$$\Delta W_2 = \frac{1}{m} \cdot \Delta Z_2 \cdot A_1^T \tag{17}$$

$$\Delta b_2 = \frac{1}{m} \cdot \sum_i \Delta Z_2 \tag{18}$$

$$\Delta Z_1 = W_2^T \cdot \Delta Z_2 \cdot \ ReLU_derivative(Z_1) \tag{19}$$

$$\Delta W_1 = \frac{1}{m} \cdot \Delta Z_1 \cdot X^T \tag{20}$$

$$\Delta b_1 = \frac{1}{m} \cdot \sum_i \Delta Z_1 \tag{21}$$

where,

- Y is the ground truth vector.
- m is the total number of instances i.e. $41,000$.

When the additional layers are added, the values of ΔZ_i will modify as:

$$\Delta Z_4 = A_4 - Y \tag{22}$$

$$\Delta Z_3 = W_4^T \cdot \Delta Z_4 \cdot \tanh\_derivative(Z_3) \tag{23}$$

$$\Delta Z_2 = W_3^T \cdot \Delta Z_3 \cdot \ sigmoid\_derivative(Z_2) \tag{24}$$

$$\Delta Z_1 = W_2^T \cdot \Delta Z_2 \cdot \ ReLU\_derivative(Z_1) \tag{25}$$

The subsequent weights and biases are then updated as:

$$W_i = W_i - \alpha \cdot \Delta W_i \tag{26}$$

$$b_i = b_i - \alpha \cdot \Delta b_i \tag{27}$$

where,

- α is the learning rate.

The formula for accuracy is used to evaluate the performance of a classification model. The formula for accuracy is as follows:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{28}$$

Mathematically, if TP represents the number of true positives (correctly predicted positive instances), TN represents the number of true negatives (correctly predicted negative instances), FP represents the number of false positives (incorrectly predicted positive instances), and FN represents the number of false negatives (incorrectly predicted negative instances), then the accuracy formula can be expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{29}$$

C System Block Diagram

The Block diagram of the system is shown in figure 1.

D Working Principle

At its core, training an Artificial Neural Network (ANN) is a multi-step process aimed at enabling the network to learn from data and make accurate predictions. The journey begins with the division of a dataset into two key subsets: the training set and the test set. The training set serves as the educational tool, while the test set acts as the benchmark to assess the model's performance on new, unseen data.

With the data organized, the next step involves initializing the parameters of the neural network – the weights and biases. These initial values, often random and small, set the stage for learning. They determine the impact of each neuron's input on its output.

Activation functions play a pivotal role in introducing non-linearity to the network. Functions like ReLU, sigmoid, and tanh are employed to ensure the network can capture complex relationships within the data. Additionally, in scenarios involving multiple classes, the softmax function transforms the raw outputs of the network into probability distributions, aiding in class prediction.

During training, the derivative of the activation function is essential. It enables the quantification of the change in the output concerning changes in the input. This information is crucial for determining how each neuron's output contributes to the overall network error, a key component in weight adjustment.

Optimization functions, like the well-known Gradient Descent, come into play to reduce the difference between predicted and actual values. These functions iteratively adjust the weights and biases to minimize the loss, which quantifies the prediction error. The learning rate acts as a guide, determining the size of each step taken in the direction that reduces the error.

Forward propagation involves sending input data through the network layer by layer, producing predictions. Subsequently, during backpropagation, the error is propagated backward through the network, revealing how each weight influenced the prediction error. This insight guides the weight and bias updates, which are performed in alignment with the chosen optimization function.

Architectural design decisions encompass the arrangement of layers, the number of neurons in each layer, and the connections between them. A well-structured architecture aligns with the problem's complexity and enhances the network's capacity to learn.

Training takes place across multiple epochs, with each epoch involving a complete iteration through the training data. The learning rate, a hyperparameter, governs the magnitude of parameter updates. Striking the right balance in learning rate selection is crucial for both stability and convergence speed.

As training progresses, the model is continually assessed using the test set. The accuracy, calculated by comparing model predictions to true labels, quantifies the model's performance in terms of correct classifications.

In conclusion, the working principle of training an ANN involves a synergistic blend of data manipulation, mathematical computations, and iterative optimization. The objective is to iteratively adjust the network's parameters, thereby enhancing its ability to understand complex patterns within data and make accurate predictions on unseen instances.

E Instrumentation Details

This project's primary objective is to design and train a neural network capable of accurately recognizing hand-written digits. The code encompasses a series of logical steps, each contributing to the creation and training of the neural network.

The project begins by importing essential libraries such as `pandas`, `numpy`, and `matplotlib`. These libraries are pivotal for data manipulation, numerical computations, and data visualization, forming the backbone of the project's technical implementation.

With the libraries in place, the project proceeds to the data processing phase. The digit dataset is sourced from a CSV file named `digit_data.csv`. This dataset likely contains a collection of images, where each image represents a hand-written digit. Upon loading the dataset using the `pandas` library, it undergoes preprocessing. The dataset is shuffled to introduce randomness, followed by a division into training and development (`dev`) sets. This step is vital to ensure the model's performance is assessed on unseen data.

The core of the project revolves around the implementation of a neural network. The neural network's structure and functionality are encapsulated within a series of functions. The initial parameters of the network, including weights and biases, are set using the `init_params()` function. Moreover, the project introduces the utilization of different weight initialization techniques: Kaiming and Xavier initializers. The `kaiming_initializer()` and `Xavier_initializer()` functions allow for the selection of these advanced techniques, enhancing the network's convergence and training efficiency.

Furthermore, the code leverages dropout as a regularization technique to prevent overfitting. The `dropout_function()` is defined to implement dropout during training, where a certain proportion of randomly selected neurons are "dropped out" or ignored during forward and backward propagation.

Activation functions like ReLU and softmax are defined using dedicated functions (`ReLU()` and `softmax()`), contributing to the model's ability to capture complex relationships within the data.

The forward propagation mechanism, responsible for computing the network's predictions, is realized through the `forward_prop()` function. Conversely, the `backward_prop()` function handles the crucial process of backpropagation. Backpropagation computes gradients that denote the sensitivity of the network's output with respect to changes in parameters. These gradients guide the subsequent parameter updates.

The project incorporates gradient descent as the optimization algorithm. The `update_params()` function employs the computed gradients to iteratively adjust the parameters, steering the model toward making better predictions. The `get_accuracy()` function quantifies the model's performance by comparing its predictions against the actual labels.

The project's main training loop resides in the `gradient_descent()` function. This loop iterates over a predetermined number of epochs, continually enhancing the model's performance. The code periodically prints the accuracy achieved during training and plots an accuracy-versus-iterations graph, facilitating the assessment of the model's learning progress.

Following training, the project focuses on evaluating the model's accuracy using the `dev` set. The model's predictions are compared against the ground truth labels to gauge its effectiveness in digit classification. Additionally, the code enables the visualization of the model's predictions on select training examples using the `test_prediction()` function.

In summary, this project's journey commences with data loading and preprocessing, leading to the construction and training of a neural network for digit classification. The code provides a clear illustration of fundamental concepts in neural networks, including forward and backward propagation, activation functions, gradient descent, and regularization techniques like dropout. The addition of Kaiming and Xavier initializers and dropout underscores the project's commitment to enhancing training efficiency and generalization. The project underscores the critical stages involved in training a neural network and evaluating its performance on new data.

III EXPERIMENTAL RESULTS

Firstly, we built an artificial neural network using random weight initialization. The Accuracy vs. Iteration plot in this case is depicted in figure 2. The predictions on some images in this case is shown in figure 3, 4, 5, and 6. In this model, we achieved a training accuracy of 86.55% and testing accuracy of 87.20% which can be seen in classification report shown in figure 7.

Similarly, the model with multiple hidden layer is also implemented whose Accuracy vs. Iteration plot is depicted in figure 8. The predictions on some sample images using this model is shown in figure 9, 10, 11, and 12. In this model, we achieved a training accuracy of 76.5% and testing accuracy of 77% which can be seen in classification report shown in figure 13.

Again, the model with Xavier initialization is also implemented whose Accuracy vs. Iteration plot is depicted in figure 14. The predictions on the same sample images using this model is shown in figure 15, 16, 17, and 18. In this model, we achieved a training accuracy of 11.11% and testing accuracy of 11.40% which can be seen in classification report shown in figure 19.

Similarly, the model with Kaiming initialization is also implemented whose Accuracy vs. Iteration plot is depicted in figure 20. The predictions on the same sample images using this model is shown in figure 21, 22, 23, and 24. In this model, we achieved training accuracy of 90.00% and testing accuracy of 90.30% which can be seen in classification report shown in figure 25.

Furthermore we implemented a model by applying dropout. The Accuracy vs. Iteration plot is depicted in figure 26. The predictions on the same sample images using this model is shown in figure 27, 28, 29, and 30. In this model, we achieved training accuracy of 79.10% and testing accuracy of 87.10% which can be seen in classification report shown in figure 31.

IV Discussion and Analysis

In the conducted experiments, we examined the performance of various artificial neural network models using distinct weight initialization techniques. The accuracy outcomes revealed intriguing insights into the impact of these techniques on model effectiveness. Notably, the model initialized with random weights achieved an accuracy of 87.20%, showcasing its potential in delivering solid results. Conversely, the Xavier initialization technique yielded a significantly lower accuracy of 11.40%, possibly due to its incompatibility with the problem and MNIST dataset at hand. Another possible reason maybe that kaiming initialization doesn't work better for our chosen architecture and activation functions. On the other hand, the Kaiming initialization stood out by producing an impressive accuracy of 90.30%, underscoring the vital role of appropriate weight initialization in model success.

Furthermore, the architectural complexity of the models played a pivotal role in accuracy outcomes. The model with multiple hidden layers attained a comparatively modest accuracy of 78.60%, indicating that increased complexity didn't necessarily translate to enhanced performance. Additionally, the introduction of dropout regularization led to an accuracy of 87.10%, highlighting its positive effect on preventing overfitting while maintaining competitive accuracy levels.

he study emphasizes the need for thoughtful decisions in model design, including choosing activation functions, weight initialization methods, and regularization techniques. The interplay of these factors can significantly impact the model's performance. The MNIST dataset's characteristics, including similarity among instances may also have influenced the performance variations observed. This interaction between dataset attributes and model architecture highlights the importance of adapting models to specific datasets.

From training accuracy versus iterations graphs of different models, we can see that the model with Xavier initialization and Kaiming initialization are converged faster than other models. This reflects the impact of weight initialization methods on the training process.
We also analyzed that Xavier initialization has tends to lead to fastest convergence among all but have least accuracy. This phenomenon is often seen in neural network training, and it can be explained by considering the way Xavier initialization affects the scale of weights in the network. Xavier initialization sets the initial weights in a way that aims to balance the scale of activations and gradients throughout the network layers. This balance facilitates smoother and faster convergence during training. However, this technique might not be ideal for all activation functions and architectures, especially when dealing with certain non-linearities. The reason for the accuracy discrepancy could be that the scale of weights introduced by Xavier initialization might not be optimal for your specific architecture and task. When the weights are initialized too small or too large, the network might struggle to learn the intricate patterns present in the data, resulting in sub-optimal accuracy.

Similarly, we see that the model with multiple hidden layers take highest time to converge. Models that incorporate multiple hidden layers become more intricate due to the substantial increase in parameters, encompassing weights and biases, in comparison to shallower counterparts. This heightened parameter space introduces a heightened level of complexity to the optimization landscape, thereby decelerating the convergence process. The optimization algorithm is required to navigate a more extensive solution space in its quest to pinpoint optimal parameter values.

The discrepancy between training and testing accuracy provides valuable insights into each model's ability to generalize. Models with smaller differences tend to generalize well, suggesting that they have captured meaningful patterns and relationships in the data without being overly influenced by noise. On the other hand, larger differences indicate potential overfitting or sub-optimal initialization, both of which require careful consideration and adjustments to achieve better model performance. Here, the model with dropout has more difference between training and testing accuracy. This signifies an improvement in generalization compared to other models, emphasizing the regularization's role in mitigating overfitting tendencies. Other model's relatively small difference between training accuracy and testing accuracy suggests that they have successfully learned to generalize well to unseen data. This indicates that the model has avoided overfitting and is likely to perform consistently on new instances.

Analyzing the learning curves and convergence patterns from the ”Accuracy vs. Iteration” plots provided valuable insights into the learning dynamics of each model. These observations shed light on their convergence speeds and stability throughout the training process. Meanwhile, the sample predictions unveiled consistent misclassifications, offering potential insights into the challenges faced by the models when classifying specific types of images.

However, it's important to acknowledge certain limitations of the study. The dataset's potential skewness, limited hyperparameter tuning, and unexplored architectural configurations could have influenced the results. Despite these limitations, the findings underscore the significance of meticulous weight initialization and architecture selection in achieving higher model accuracy. Moving forward, there's ample room for exploration, such as investigating alternative architectures, refining hyperparameters, and exploring additional regularization techniques. In conclusion, this study underscores the critical role that initialization techniques and architecture intricacies play in shaping the performance of neural network models, offering valuable guidance for future research and application.

V Conclusion

In conclusion, our investigation into various artificial neural network models, each employing distinct weight initialization methods and architectural complexities, has yielded valuable insights into their performance and generalization abilities. The disparities between training and testing accuracy have served as critical indicators of each model's effectiveness in tackling the task at hand.

The disparities in training and testing accuracy across the diverse models highlight the significance of thoughtful model design, weight initialization techniques, and regularization strategies. Smaller discrepancies reflect well-generalized models capable of robust predictions, while larger gaps point towards potential issues such as overfitting or architectural mismatches. These findings underscore the complex interplay between model complexity, initialization, and regularization, offering essential guidance for creating neural network models with optimal performance and adaptability in real-world scenarios.

Our comprehensive exploration of artificial neural network (ANN) models for the classification task on the MNIST dataset has provided deep insights into the intricate interplay between architecture, weight initialization, activation functions, and regularization. The MNIST dataset, with its digit images and corresponding labels, has served as an ideal canvas to investigate the effectiveness of different model variations.

REFERENCES

[1] J. Zupan, “Introduction to artificial neural network (ann) methods: what they are and how to use them,” *Acta Chimica Slovenica*, vol. 41, pp. 327–327, 1994.

[2] A. D. Rasamoelina, F. Adjailia, and P. Sinčák, “A review of activation function for artificial neural network,” in *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 2020, pp. 281–286.

Bishwambhar Dahal is a fourth-year student of Electronics, Communication, and Information Engineering. With a deep fascination for Artificial Intelligence (AI), He is driven by the potential of AI to transform industries and tackle complex challenges. His academic journey has equipped him with a strong foundation in AI concepts, including machine learning and data analysis. He possess a relentless curiosity and is always eager to explore the latest advancements in AI. His goal is to apply his knowledge and skills in AI to make meaningful contributions to research and development, pushing the boundaries of what is possible with intelligent algorithms.(THA076BEI009)

Sirjana Bhatta is a fourth-year student of Electronics, Communication, and Information Engineering with a keen interest in the field of Artificial Intelligence (AI). She possess a strong academic foundation and practical skills in machine learning, deep learning, and data analysis. Her passion lies in leveraging AI to revolutionize industries and solve complex problems. She is a motivated learner, constantly staying updated with the latest advancements in AI. She is seeking opportunities to contribute to AI research and development and make a positive impact on society. (THA076BEI036)

Appendix

A Figures

1 Block Diagram

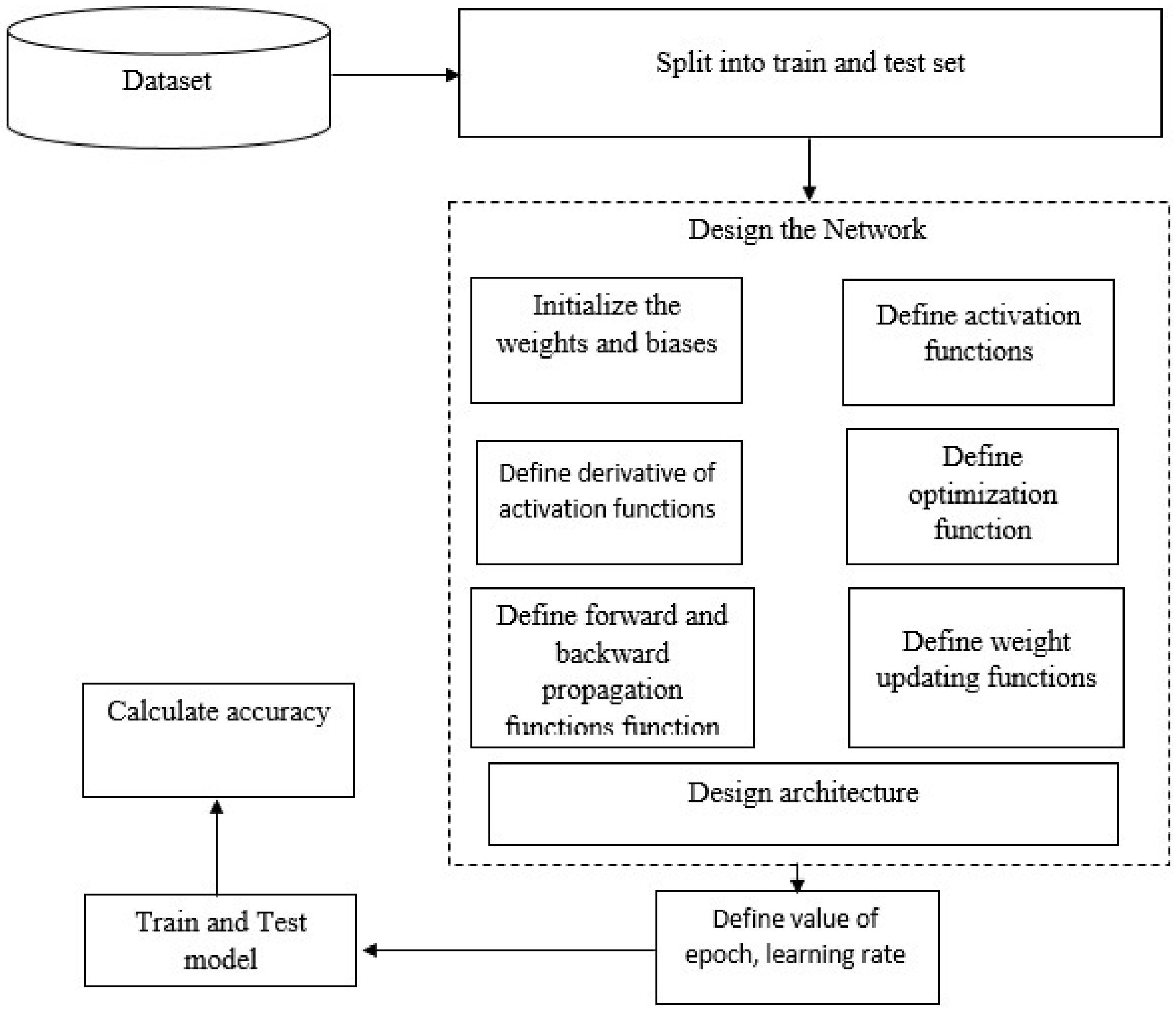

Dataset
Split into train and test set
Design the Network
Initialize the weights and biases
Define activation functions
Define derivative of activation functions
Define optimization function
Define forward and backward propagation functions function
Define weight updating functions
Design architecture
Calculate accuracy
Train and Test model
Define value of epoch, learning rate

Figure 1: Block diagram

.

2 Result

Base Solution

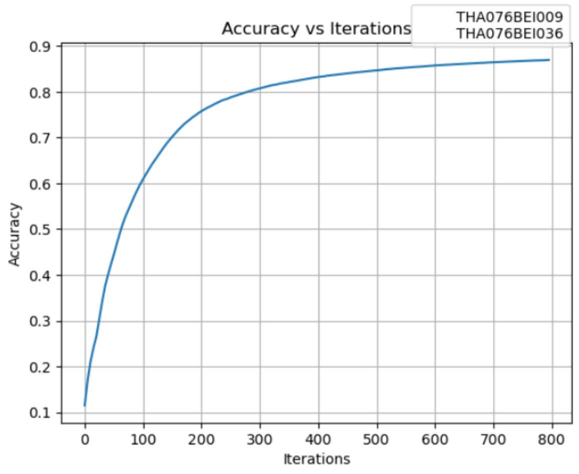


Figure 2: Training Accuracy vs. Iteration plot for randomly initialized model

```
Prediction:  [2]
Label:  2
```

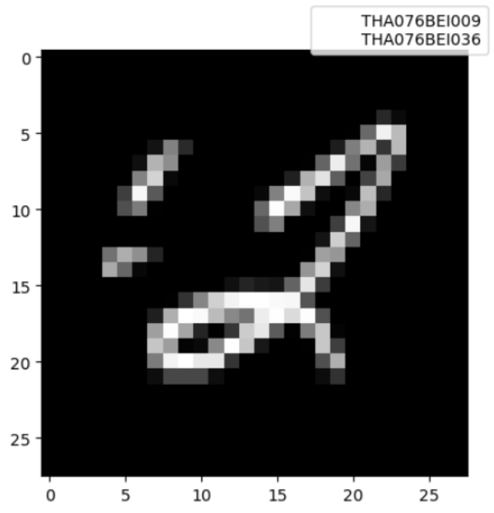


Figure 3: Prediction on image containing digit ’2’

```
Prediction:  [3]
Label:  3
```

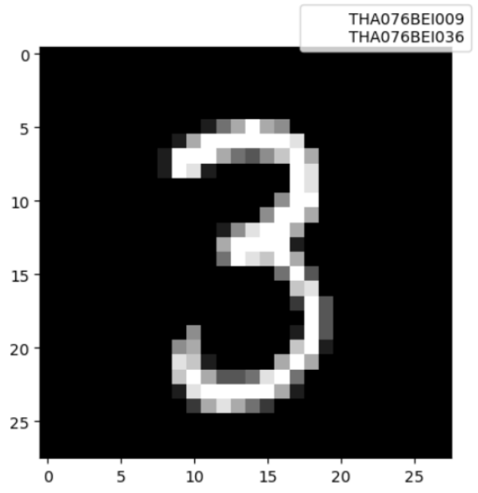


Figure 4: Prediction on image containing digit ’3’

```
Prediction:  [6]
Label:  4
```

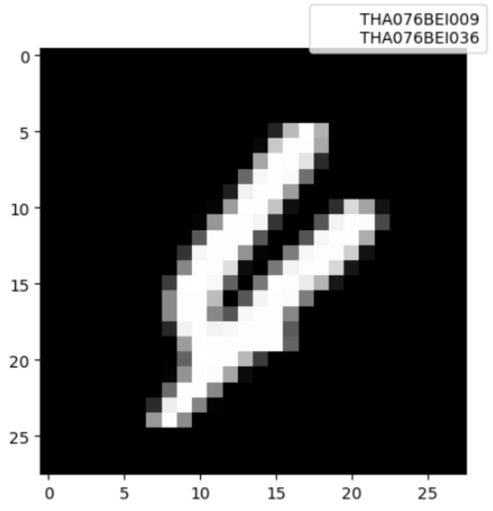


Figure 5: Prediction on image containing digit ’4’

```
Prediction:  [9]
Label:  4
```

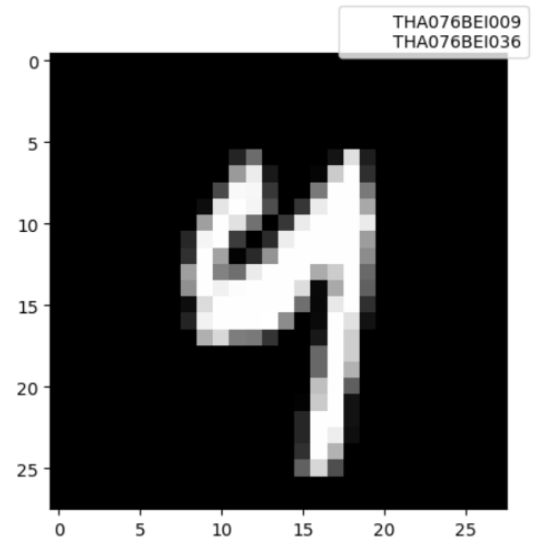
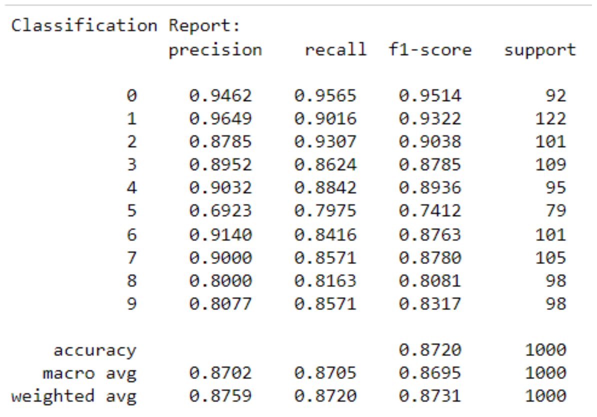


Figure 6: Prediction on image containing digit ’4’



```
Classification Report:
              precision    recall  f1-score   support

           0     0.9462    0.9565    0.9514        92
           1     0.9649    0.9016    0.9322       122
           2     0.8785    0.9307    0.9038       101
           3     0.8952    0.8624    0.8785       109
           4     0.9032    0.8842    0.8936        95
           5     0.6923    0.7975    0.7412        79
           6     0.9140    0.8416    0.8763       101
           7     0.9000    0.8571    0.8780       105
           8     0.8000    0.8163    0.8081        98
           9     0.8077    0.8571    0.8317        98

    accuracy                         0.8720      1000
   macro avg     0.8702    0.8705    0.8695      1000
weighted avg     0.8759    0.8720    0.8731      1000
```

Figure 7: Classification report for base solution

ANN with multiple hidden layers

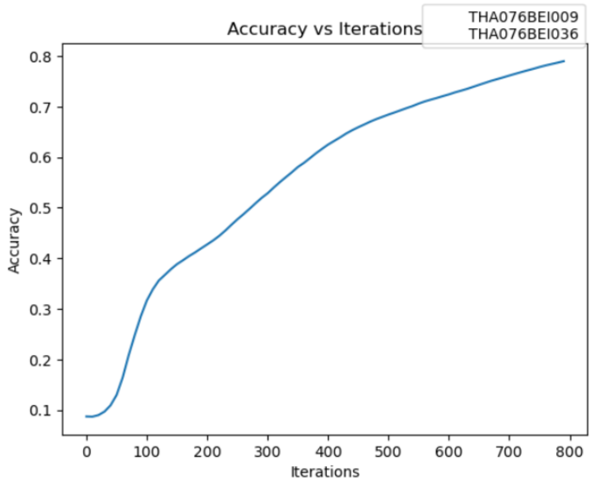


Figure 8: Training Accuracy vs. Iteration plot for randomly initialized model with multiple hidden layers

```
Prediction:  [2]
Label:  2
```

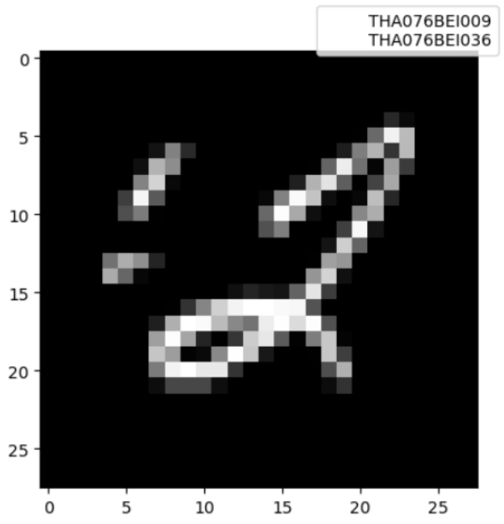


Figure 9: Prediction on image containing digit ’2’

```
Prediction:  [3]
Label:  3
```

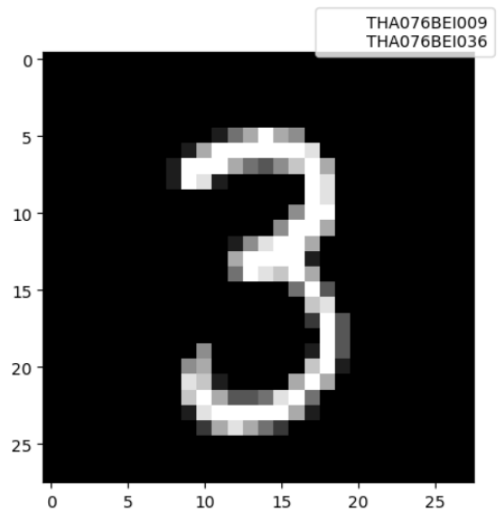


Figure 10: Prediction on image containing digit ’3’

```
Prediction:  [6]
Label:  4
```

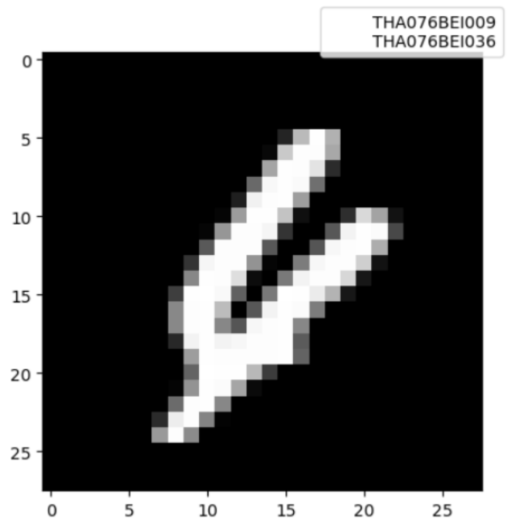


Figure 11: Prediction on image containing digit ’4’

```
Prediction:  [9]
Label:  4
```

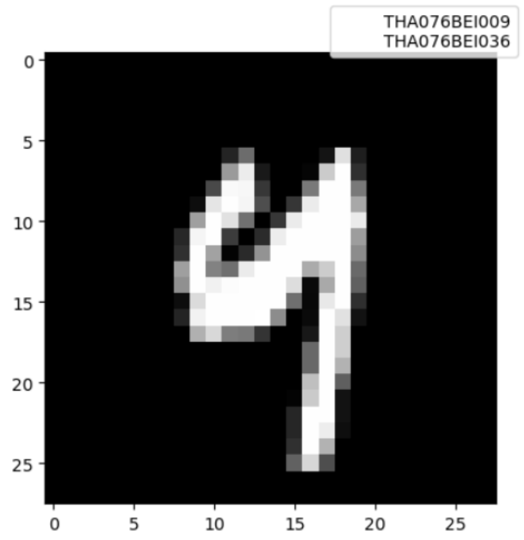


Figure 12: Prediction on image containing digit ’4’

```
Classification Report:
              precision    recall  f1-score   support

           0     0.9140    0.7944    0.8500       107
           1     0.9474    0.8640    0.9038       125
           2     0.8037    0.8113    0.8075       106
           3     0.7143    0.8065    0.7576        93
           4     0.8602    0.8333    0.8466        96
           5     0.4725    0.5811    0.5212        74
           6     0.8602    0.7921    0.8247       101
           7     0.8300    0.8137    0.8218       102
           8     0.7400    0.6916    0.7150       107
           9     0.6923    0.8090    0.7461        89

    accuracy                         0.7860      1000
   macro avg     0.7835    0.7797    0.7794      1000
weighted avg     0.7977    0.7860    0.7898      1000
```

Figure 13: Classification report of model with multiple hidden layers

ANN with Xavier initialization

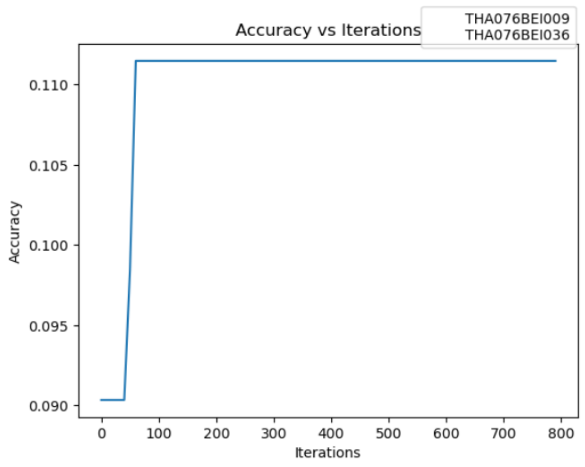


Figure 14: Training Accuracy vs. Iteration plot of model using Xavier initialization

```
Prediction:  [1]
Label:  2
```

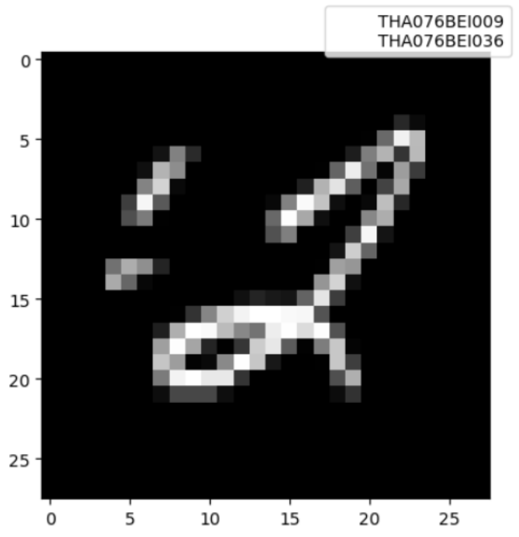


Figure 15: Prediction on image containing digit ’2’

```
Prediction:  [1]
Label:  3
```

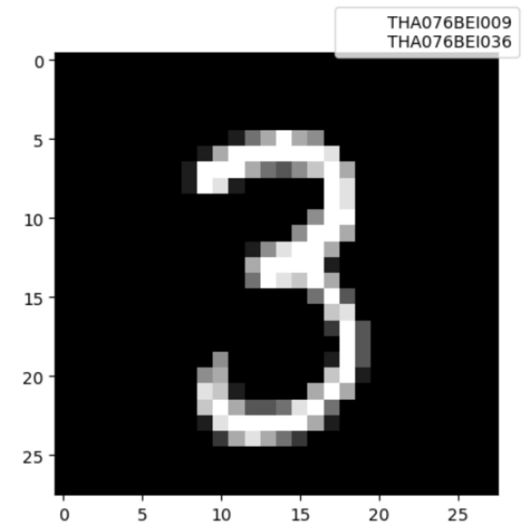


Figure 16: Prediction on image containing digit ’3’

```
Prediction:  [1]
Label:  4
```

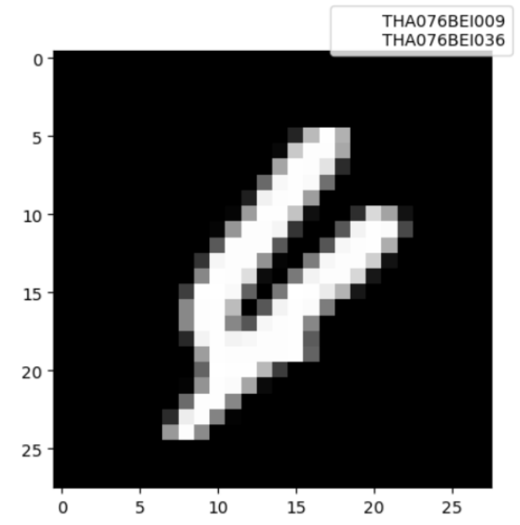


Figure 17: Prediction on image containing digit ’4’

```
Prediction:  [1]
Label:  4
```

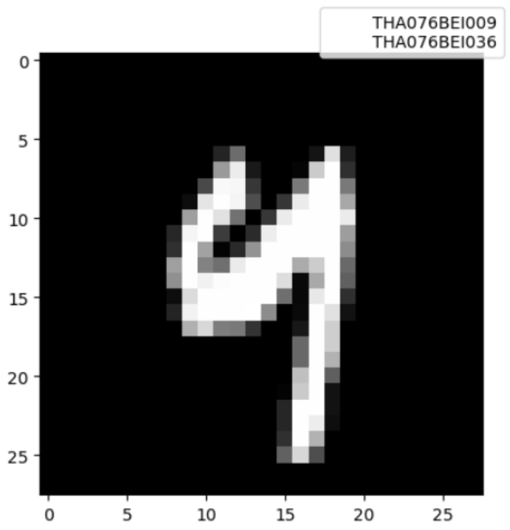
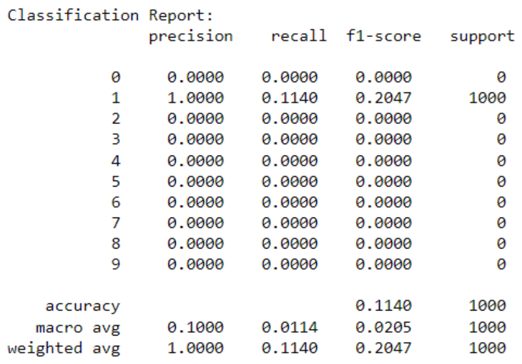


Figure 18: Prediction on image containing digit ’4’



```
Classification Report:
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000         0
           1     1.0000    0.1140    0.2047      1000
           2     0.0000    0.0000    0.0000         0
           3     0.0000    0.0000    0.0000         0
           4     0.0000    0.0000    0.0000         0
           5     0.0000    0.0000    0.0000         0
           6     0.0000    0.0000    0.0000         0
           7     0.0000    0.0000    0.0000         0
           8     0.0000    0.0000    0.0000         0
           9     0.0000    0.0000    0.0000         0

    accuracy                         0.1140      1000
   macro avg     0.1000    0.0114    0.0205      1000
weighted avg     1.0000    0.1140    0.2047      1000
```

Figure 19: Classification report for model with Xavier initialization

ANN with Kaiming initialization

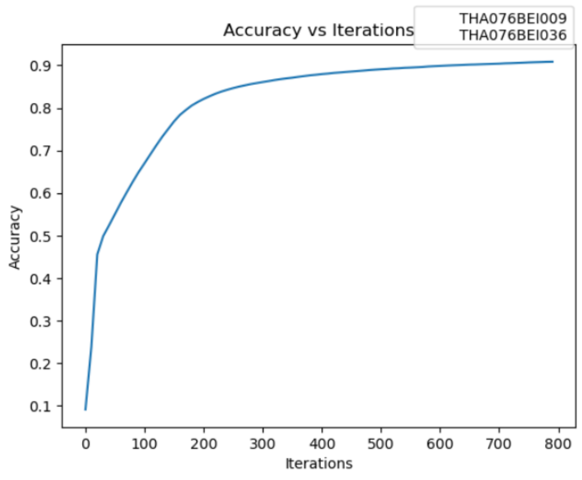


Figure 20: Training Accuracy vs. Iteration plot of model using Kaiming initialization

```
Prediction:  [1]
Label:  2
```

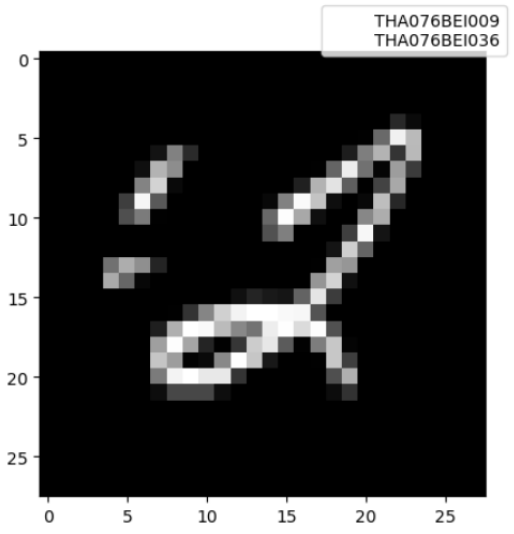
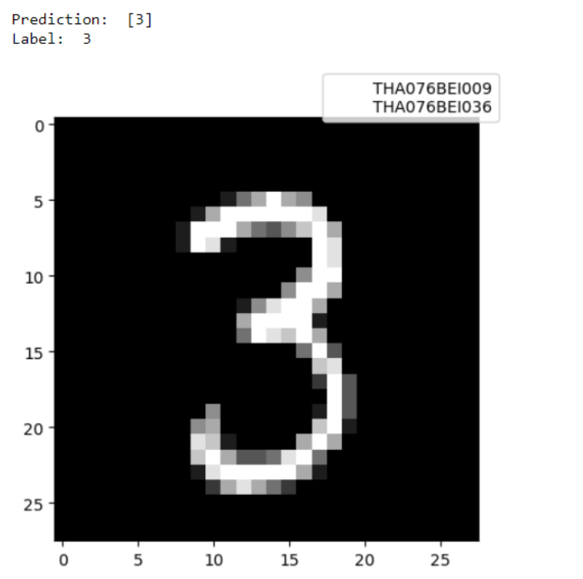
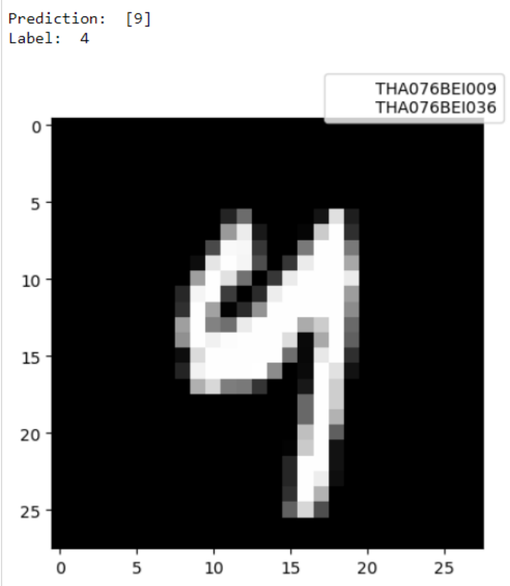


Figure 21: Prediction on image containing digit ’2’



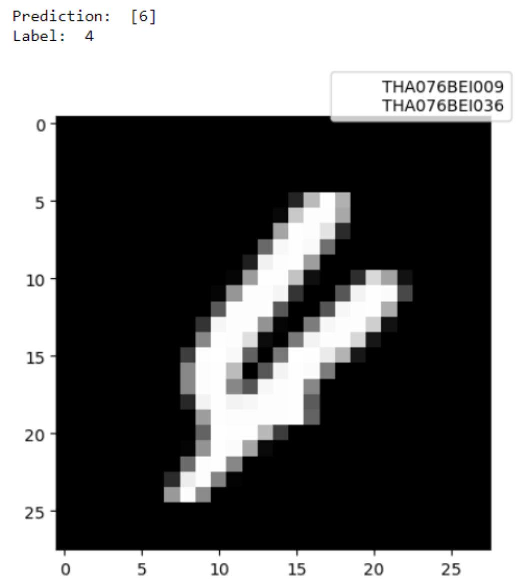
```
Prediction:  [3]
Label:  3
```

Figure 22: Prediction on image containing digit ’3’



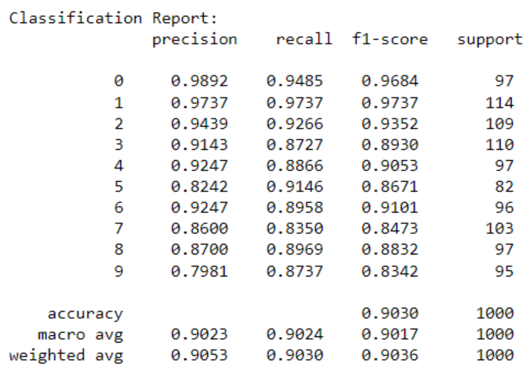
```
Prediction:  [9]
Label:  4
```

Figure 24: Prediction on image containing digit ’4’



```
Prediction:  [6]
Label:  4
```

Figure 23: Prediction on image containing digit ’4’



```
Classification Report:
              precision    recall  f1-score   support

           0     0.9892    0.9485    0.9684        97
           1     0.9737    0.9737    0.9737       114
           2     0.9439    0.9266    0.9352       109
           3     0.9143    0.8727    0.8930       110
           4     0.9247    0.8866    0.9053        97
           5     0.8242    0.9146    0.8671        82
           6     0.9247    0.8958    0.9101        96
           7     0.8600    0.8350    0.8473       103
           8     0.8700    0.8969    0.8832        97
           9     0.7981    0.8737    0.8342        95

    accuracy                         0.9030      1000
   macro avg     0.9023    0.9024    0.9017      1000
weighted avg     0.9053    0.9030    0.9036      1000
```

Figure 25: Classification report for model with Kaiming initialization

ANN with dropout

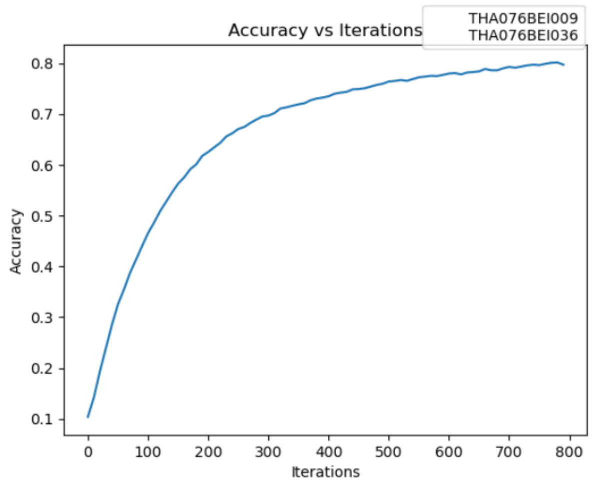


Figure 26: Training Accuracy vs. Iteration plot of model with dropout

```
Prediction:  [3]
Label:  3
```

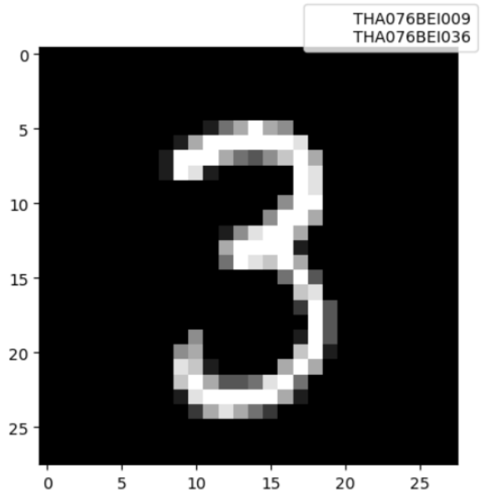


Figure 28: Prediction on image containing digit '3'

```
Prediction:  [2]
Label:  2
```

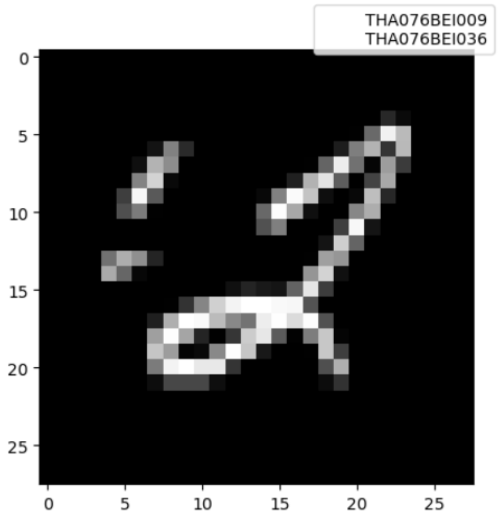


Figure 27: Prediction on image containing digit '2'

```
Prediction:  [6]
Label:  4
```

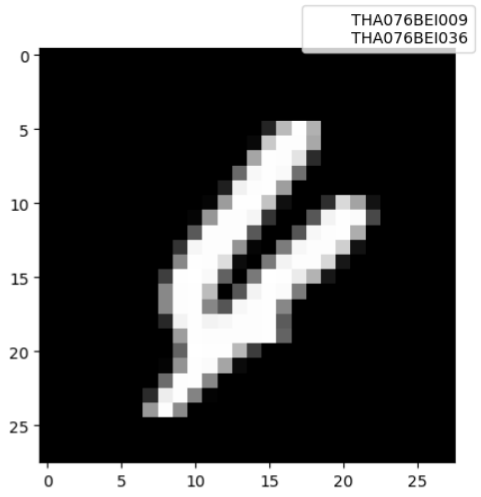


Figure 29: Prediction on image containing digit '4'

```
Prediction:  [9]
Label:  4
```

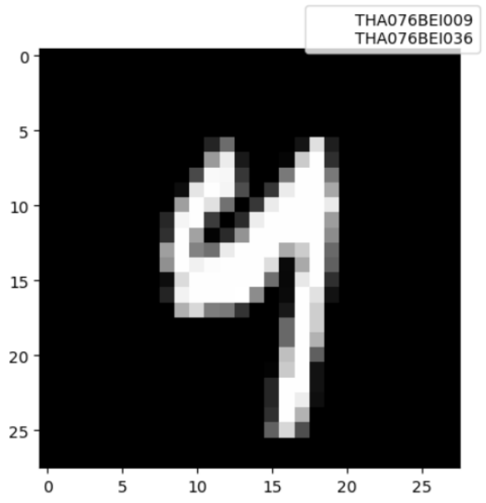
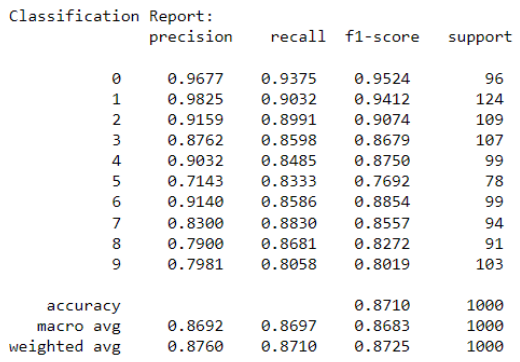


Figure 30: Prediction on image containing digit ’4’



```
Classification Report:
              precision    recall  f1-score   support

           0     0.9677    0.9375    0.9524        96
           1     0.9825    0.9032    0.9412       124
           2     0.9159    0.8991    0.9074       109
           3     0.8762    0.8598    0.8679       107
           4     0.9032    0.8485    0.8750        99
           5     0.7143    0.8333    0.7692        78
           6     0.9140    0.8586    0.8854        99
           7     0.8300    0.8830    0.8557        94
           8     0.7900    0.8681    0.8272        91
           9     0.7981    0.8058    0.8019       103

    accuracy                         0.8710      1000
   macro avg     0.8692    0.8697    0.8683      1000
weighted avg     0.8760    0.8710    0.8725      1000
```

Figure 31: Classification report for model with dropout layers

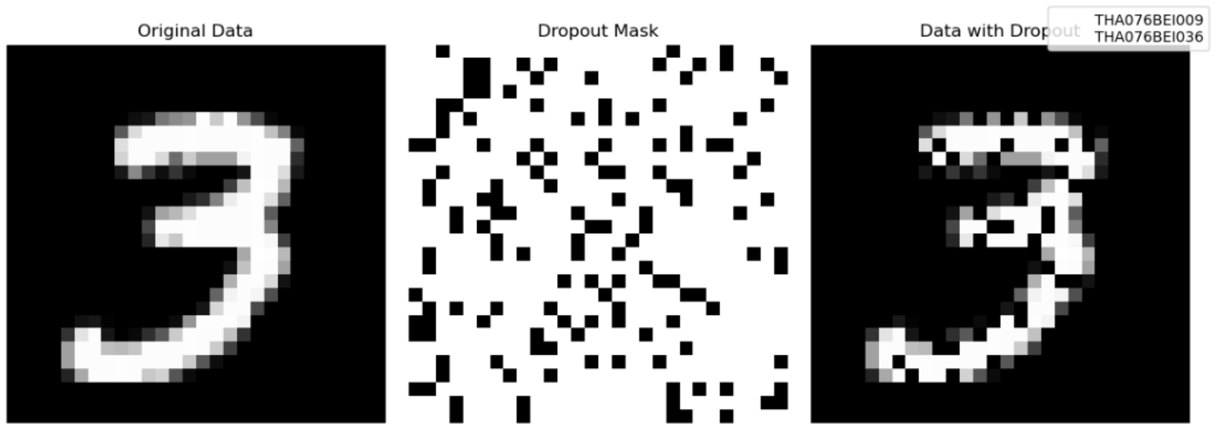


Figure 32: Dropout visualization

Backpropagation equations in Neural Network

In the context of neural networks, let's delve into the equations and notation associated with backpropagation.

Forward Pass Equations

The forward pass equations for a neural network are as follows:

$$\begin{aligned} z^{[1]} &= w^{[1]}A^{[0]} + b^{[1]} \\ A^{[1]} &= \text{ReLU}(z^{[1]}) \\ z^{[2]} &= w^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= \text{Softmax}(z^{[2]}) \end{aligned}$$

Cross Entropy Loss

The cross-entropy loss for a single instance is calculated as:

$$L_i = \sum_{k=0}^{9} y_{ik} \cdot \log(A_{ik}^{[2]})$$

Where:

$$\begin{aligned} y_{ik} &: \text{True probability value for class } k \text{ for instance } i \\ A_{ik}^{[2]} &: \text{Predicted probability value for class } k \text{ for instance } i \\ L_i &: \text{Loss value for instance } i \end{aligned}$$

The cost function is then calculated as the average over all instances:

$$C = -\frac{1}{m}\sum_{i=0}^{m}\sum_{k=0}^{9} y_{ik} \cdot \log(A_{ik}^{[2]})$$

Where:

$$m : \text{Total number of data points}$$

Derivatives of Cost Function

The derivative of the cost function with respect to $A_{ik}^{[2]}$ for a single point is calculated as:

$$\frac{\partial C}{\partial A_{ik}^{[2]}} = -\frac{y_{ik}}{A_{ik}^{[2]}}$$

Since $A_{ik}^{[2]}$ is the output of the Softmax function, its derivative with respect to $z_{ik}^{[2]}$ is:

$$\frac{\partial A_{ik}^{[2]}}{\partial z_{ik}^{[2]}} = A_{ik}^{[2]} \cdot (1 - A_{ik}^{[2]})$$

The derivative of the cost function with respect to $z_{ik}^{[2]}$ can be calculated as:

$$\frac{\partial C}{\partial z_{ik}^{[2]}} = \frac{\partial C}{\partial A_{ik}^{[2]}} \cdot \frac{\partial A_{ik}^{[2]}}{\partial z_{ik}^{[2]}} = -y_{ik} \cdot (1 - A_{ik}^{[2]})$$

Since y_ik is the true label and $A_{ik}^{[2]}$ is the predicted probability,

$$\frac{\partial C}{\partial z_{ik}^{[2]}} = A_{ik}^{[2]} - y_{ik}$$

If y is the matrix of true labels with m columns (where m is the total number of training data points) and $A^{[2]}$ is the matrix consisting of predicted probabilities for m training data points, then the change in $z^{[2]}$ can be written in matrix format as:

$$\Delta z^{[2]} = \frac{\partial C}{\partial z^{[2]}} = A^{[2]} - y$$

For the calculation of change in weight and bias, consider each upcoming value as a singular data point. The change in weight $\Delta w^{[2]}$ for a single data point is calculated as:

$$\Delta w^{[2]} = \frac{\partial C}{\partial w^{[2]}} = \frac{\partial C}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w^{[2]}} = \frac{1}{m}\Delta z^{[2]} \cdot A^{[1]}$$

The weights are updated when the entire dataset is passed through the network. Considering them as a matrix which consists of all the data points, the weight update is determined by taking the average change in weight for each data point:

$$\Delta w^{[2]} = \frac{1}{m}\Delta z^{[2]} \cdot [A^{[1]}]^T$$

Similar can be done for the bias term:

$$\Delta b^{[2]} = \frac{\partial C}{\partial b^{[2]}} = \frac{\partial C}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial b^{[2]}} = \frac{1}{m}\Delta z^{[2]}$$

The factor $\frac{1}{m}$ is introduced to calculate the average bias value since the bias value remains the same for a single epoch. Thus, we need to compute the average change in bias value across all individual data points.
For the dimension of $b^{[2]}$ as (10×1), the dimension of $\Delta b^{[2]}$ must also be (10×1). To ensure dimensional consistency, we sum all column values in each individual row, reducing $(10 \times m)$ to (10×1) dimension:

$$\Delta b^{[2]} = \frac{1}{m}\sum_{i=1}^{m} \Delta z_i^{[2]}$$

For the calculation of $\Delta z^{[1]}$, a similar chain rule can be applied. The dimension of $\Delta z^{[1]}$ must be $(10 \times m)$, so:

$$\Delta z^{[1]} = \frac{\partial C}{\partial z^{[1]}} = \frac{\partial C}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial z^{[1]}} = [w^{[2]}]^T \cdot \Delta z^{[2]} \circ \text{ReLU}'(z^{[1]})$$

The $\circ$ symbol denotes element-wise multiplication operation.
Weight and bias terms $w^{[1]}$ and $b^{[1]}$ are calculated as:

$$\Delta w^{[1]} = \frac{\partial C}{\partial w^{[1]}} = \frac{\partial C}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial w^{[1]}} = \frac{1}{m}\Delta z^{[1]} \cdot [A^{[0]}]^T$$

$$\Delta b^{[1]} = \frac{\partial C}{\partial b^{[1]}} = \frac{1}{m}\sum_{i=1}^{m} \Delta z_i^{[1]}$$