Implementation of Artificial Neural Network (ANN) From Scratch (July 2024)

Kristina Ghimire (THA077BCT023)[1], Punam Shrestha (THA077BCT038)[1]

[1]Department of Electronics and Computer Engineering, IOE, Thapathali Campus, Kathmandu 44600, Nepal

Corresponding author: Kristina Ghimire(ghimirekristina10@gmail.com)

ABSTRACT This study explored various Artificial Neural Network (ANN) configurations on the MNIST and CIFAR-10 datasets to understand their performance. For MNIST, different setups like varying the number of hidden layers and using different weight initialization techniques were tested. The best results were achieved with a more complex model featuring two hidden layers, Kaiming initialization, and LeakyReLU activation, reaching an accuracy of 86.061%. On the more challenging CIFAR-10 dataset, a similar ANN setup achieved 26.326% accuracy, highlighting the need for more advanced models like convolutional neural networks (CNNs) for better performance.

INDEX TERMS Artificial Neural Network, Convolutional Neural Network, Kaiming initialization, LeakyReLU.

I INTRODUCTION

Artificial Neural Networks (ANNs) are computing systems inspired by the biological neural networks that constitute animal brains They consist of interconnected nodes, or neurons, which process and transmit information. Neurons in ANNs can be called ”units” or ”nodes.” These neurons use a network of weighted connections to process and transform input data. ANNs are organized into layers, typically including an input layer, one or more hidden layers, and an output layer. Each connection between neurons is associated with a weight that adjusts during the learning process, allowing the network to learn patterns and relationships from data.

ANNs are fundamental in machine learning, and used extensively in applications such as classification, regression, and more. They are powerful tools in AI due to their ability to adapt to changes and new data, making them suitable for dynamic environments. ANNs can generalize from training data to make predictions or classifications on unseen data, provided they are properly trained. They can automatically learn useful features from raw data, reducing the need for manual feature extraction. Additionally, ANNs exhibit graceful degradation and robustness to noise, making them suitable for handling noisy or incomplete datasets. Their flexibility enables them to solve complex problems across various domains, contributing significantly to advancements in AI and machine learning.

Regularization in ANNs is crucial to prevent overfitting, where a model learns not only underlying patterns in the training data but also noise and random fluctuations, resulting in poor performance on new data. L2 Regularization (Weight Decay) is a common technique where a regularization term, proportional to the sum of the squares of the weights, is added to the loss function. This encourages smaller weights, preventing any single weight from dominating the model’s output and thus helping control overfitting.

ReLU (Rectified Linear Unit) is a widely adopted activation function in modern deep learning architectures due to its simplicity, computational efficiency, and effectiveness in

mitigating training issues like the vanishing gradient problem. It introduces non-linearity by outputting the input directly if it is positive, and zero otherwise, which accelerates convergence during training and helps in learning complex patterns in data.

In summary, ANNs leverage interconnected neurons and weighted connections to process data, regularization techniques like L2 regularization prevent overfitting, and activation functions like ReLU enhance model performance and training efficiency in AI and machine learning applications.

II RELATED WORK

Zupan et al. [1] explored fundamental concepts of Artificial Neural Networks (ANNs), including error back-propagation, Kohonen networks, and counter-propagation. They detailed the structure of neural networks, illustrating how neurons are arranged into layers such as input, hidden, and output. To illustrate the application of these methods, they provided a case study involving the classification and prediction of the origin of various olive oil samples. Their work offers a solid introduction to ANNs, covering both theoretical aspects and practical uses, particularly in the field of chemistry.

Keiron O'Shea et al. [2] focused on Convolutional Neural Networks (CNNs), a sophisticated type of ANN specifically designed for image recognition tasks. CNNs are appreciated for their effective yet simple structure, which makes them a great entry point for working with ANNs. Unlike basic neural networks with a straightforward three-layer architecture (input, hidden, and output), CNNs are tailored to process image data more efficiently by concentrating on specific features. This specialized approach makes CNNs easier to construct and apply for image analysis tasks. Their paper explains the essential concepts of CNNs, including the necessary layers and the optimal setup for image analysis.

These studies provide a foundational understanding of neural networks, their implementation, and their applications, setting the stage for further exploration in the field.

III METHODOLOGY

A DATASET DESCRIPTION

The MNIST dataset is a collection of 42,000 images of handwritten digits, each image being 28x28 pixels in size. Initially, each image consists of 785 pixels, which is 28x28 pixels for the image and 1 for the actual label. The first column is used for the actual label of the image. For this experiment, the first 1000 data are used for testing and the remaining for training. This dataset is widely used for training and testing machine learning models, particularly in the field of image classification. Its simplicity and well-defined structure make it an excellent starting point for anyone learning about neural networks, as it allows easy experimentation and quick understanding of basic concepts in artificial neural networks (ANNs).

The CIFAR-10 dataset, on the other hand, consists of 60,000 color images divided into 10 different classes, such as airplanes, cars, birds, cats, deers, dogs, frogs, horses, ships, and trucks. Each image is 32x32 pixels in size. Each data contains column "data" for pixel value and column "labels" for the actual label of the image. The dataset is split into 50,000 training images and 10,000 test images. CIFAR-10 is more complex than MNIST because it involves color images and a broader variety of objects. This complexity makes it useful for developing and testing more advanced ANN architectures, helping to improve the models' ability to generalize and handle real-world image classification tasks.

B SYSTEM BLOCK DIAGRAM

The system block diagram is shown in figure 1.

C THEORETICAL FORMULATION

Artificial Neural Networks resemble greatly with the human brain. A Neural Network consists of a set of inputs, outputs, and several hidden layers where each connection has a certain weight associated with it. McCulloch and Pitts designed the first artificial neural network. In this experiment, we used multiple neurons.

1 Input Layer

The input layer is denoted as $A^{(0)}$ is usually two-dimensional data. For our experiment, the input layer has 784 neurons as there are 28x28 pixels of an input image. So, the dimension of $A^{(0)}$ is 784xm, where m is the number of training data. The size of one image is 1x784 which is flattened from (28x28). Similarly, based on the size of the input, the input layer is defined. The general dimension for the input layer is nXm, where n is the total input size and m is the number of training data.

2 Output Layer

The output layer is denoted as $A^{(2)}$ is usually two-dimensional data. For our experiment, the input layer has 10 neurons as there are 10 classes of the input image. So, the dimension of $A^{(2)}$ is 10xm, where m is the number of training data. The output of the model is the prediction of class of the image. The image is numbers and the model needs to predict the number from 0 to 9. Similarly, based on the size of the output, the output layer is defined. The general dimension for the output layer is nXm, where n is the total input size and m is the number of training data.

3 Hidden Layer

The hidden layer is denoted as $A^{(1)}$ is usually two-dimensional data. For our experiment, the hidden layer has 10 neurons. The dimension of $A^{(1)}$ is 10xm, where m is the number of training data. The hidden layer lies in between the input and output layers. The larger the hidden layer, the more complex the model. The size of the hidden layer is based on the input and output layer size.

4 Activation Function

There are different types of activation functions. The activation function regulates the outputs of each layer of a neural network. Also, the activation function can be used to perform non-linearity in the network. There are kinds of activation functions: Linear and non-linear. Linear activation functions produce linear decisions whereas non-linearity activation functions allow to give complex functions.

5 Neural Network Training

The training in neural networks involves two ways: Forward propagation and Backpropagation. The learning in Neural Networks involves adjusting the weights which can be used for predicting using unseen data. Neural networks require large datasets and a long time to train.

Feedforward Propagation

Feedforward Propagation involves the initialization of weights, calculation of outputs, and use of activation functions in different layers. First, the weights are initialized randomly. For the experiment, other weight initialization methods like Kaiming and Xavier initialization are used. The output is calculated in different branches. Then, using the true label, the cost function i.e. Cross Entropy loss is calculated on the predicted label. The equations involved in forward propagation are shown in appendix D part 1.

Back-Propagation

After calculating the cost function, the weights need to be reassigned. This process is called Back-Progation. Starting from the output layer, each weight and bias value gets updated. It is the reverse process flow rather than feedforward.

Back-Propagation Algorithm

Step-1: Initialize the weights and biases. Each unit has a bias associated with it.

Step-2: Feed the training samples to the neural network.

Step-3: Propagate the inputs forward by applying the activation functions.

Step-4: Back-propagate the error.

Step-5: Update weights and biases to reflect the propagated errors.

Step-6: Repear and apply terminating conditions.

The equations involved in backpropagation with their derivations are given in appendix D part 3.

6 Loss Optimization

The network weights and bias should achieve the lowest loss. First, calculate the derivative and cost function concerning each weight. For Loss Optimization, the Gradient Descent concept is used. If the current point of the scope (gradient) is positive, then choose the direction to the left. If the current slope is negative, then choose the direction to the right. The negative slope gives the direction of descent.

D MATHEMATICAL FORMULAE

1 Activation Function

Some of the activation functions used in our model are:

Sigmoid Function

The formula of the Sigmoid function is given in equation 1. The sigmoid function is non-linear. This function has a smooth gradient. This function output ranges from 0 to 1, so this gives the probability of any output. The derivative of this function is almost zero towards the tail of the curve. So this function gives rise to the "Vanishing gradients" problem and no further learning occurs. The output is not zero centered which makes it difficult for optimization.

Tanh Function

The equation for the Tanh function is given in the equation 4. The Tanh function is non-linear. Its output is zero-centered which makes it easy for optimization. The gradient is stronger for Tanh than for Sigmoid. The output ranges from -1 to 1. But, this function also suffers from the "Vanishing gradients" problem.

Rectified Linear Unit(ReLU)

The equation for the ReLU function is given in the equation 3. The ReLU function gives the value of between 0 and infinity. This ReLU function converges much faster and is less computationally expensive than Sigmoid and Tanh. This avoids the "Vanishing gradients" problem in the positive x-axis region. This function can cause the activation to be large and the output is not centered.

LeakyReLU

The equation for the LeakyReLU function is given in the equation 5. The LeakyReLU function possesses all the benefits of ReLU. This LeakyReLU function allows a small, nonzero, constant gradient (.$\alpha = 0.01$. This avoids the "Vanishing gradients" problem in the positive x-axis region. This function can cause the activation to be large and the output is not centered.

Softmax Function

The equation for the Softmax function is given in the equation 2. The output values of the Softmax function are between 0 and 1. The sum of all probabilities is 1, making it a valid probability distribution. Larger logits result in higher probabilities. This characteristic allows the model to favor classes with higher scores.

2 Accuracy

Accuracy measures correctly predicted instances relative to the total number of instances in a dataset. High accuracy indicates effective model performance suitable for real-world applications, accurately predicting both positive and negative instances. The formula of accuracy is shown in equation 6.

3 Cross Entropy Loss

Cross-entropy loss is widely used as a loss function for classification tasks. This loss function is always non-negative and can be applied to both binary and multiclass classification problems, with formula adjustments as needed. In multiclass classification, it measures how well the predicted probabilities align with the actual class labels. A high loss indicates that the predicted probability for the correct class is low, while a low loss means the prediction is closer to the true label. The formula of cross-entropy loss for multiclass classification is shown in appendix D part 2.

E DATA PREPROCESSING

Different steps for data preprocessing are done which are shown below:

1 Reshaping the input data

The given input data is in the form of two-dimensional and three-dimensional. The input for the ANN should be in a one-dimensional format. Initially, each row denotes the num-

ber of samples, and each column has its pixel values. So, the given input data first needs to be transposed as the number of samples should be in the y-axis and pixels in the x-axis. Then, the data is reshaped for easy implementation or flattened into one dimension.

2 Normalization of data

Normalization involves scaling numerical data into a standardized range. These techniques ensure that data from different scales can be compared directly and are essential for many machine learning algorithms to perform effectively and efficiently. Normalization is an important part of models like ANN because all the pixel value ranges from 0 to 255. Here, normalization is done by dividing each pixel value by 255 to scale them from 0 to 1.

3 One Hot Encoding

For any image to predict its class, we first hot encoded the input to determine its class. This process is done on a label or target dataset for both training and testing data i.e. y_train and y_test.

IV INSTRUMENTATION DETAILS

Python is widely recognized for its simplicity and readability as a popular interpreted programming language that executes code line by line, aiding in quick error detection and resolution without the need for variable declarations before assignment. JupyterLab serves as an interactive web-based environment supporting multiple programming languages, prominently Python, allowing users to develop and share code, equations, visualizations, and textual content, particularly favored in data science for its flexibility and intuitive interface. Matplotlib, a robust library, facilitates the creation of diverse plots such as lines, scatter plots, histograms, and more, offering extensive customization options suitable for producing high-quality visualizations in data analysis and presentations. Seaborn, built upon Matplotlib, specializes in crafting informative and visually appealing statistical graphics that seamlessly integrate with Pandas data structures, ideal for exploring complex datasets and generating elegant statistical plots. It simplifies tasks like visualizing variable relationships, analyzing data distributions, and comparing categories through its diverse functions and customizable settings. Open-CV library cv2 is used for different image-related tasks like reading an image, converting images from BGR to RGB, and displaying images.

V EXPERIMENTAL RESULTS

A PROBLEM 1: MNIST Dataset

The MNIST dataset consists of 28x28 pixel grayscale images of handwritten digits, ranging from 0 to 9. Figure 2 shows sample images from this dataset.

1 Simple ANN

The input data is transformed into a 2-dimensional format, transposed, and normalized. The simple Artificial Neural Network (ANN) consists of 784 neurons in the input layer, 10 neurons in the hidden layer, and 10 neurons in the output layer. The initial weights $W^{(1)}$, $W^{(2)}$, and biases $b^{(1)}$, $b^{(2)}$ are randomly initialized. The dimensions are $W^{(1)}$: 10×784, $W^{(2)}$: 10×10, $b^{(1)}$: 10×1, and $b^{(2)}$: 10×1. The ReLU activation function is used in the hidden layer and the Softmax activation function in the output layer. Figures 5 and 4 show the plots of the ReLU function and its derivative, and the Softmax function, respectively. The forward propagation computes the intermediate outputs $Z^{(1)}$, $Z^{(2)}$, $A^{(1)}$, and $A^{(2)}$. Backpropagation updates the weights and biases. For every 5 iterations, accuracy is calculated using one-hot encoded y_{train} and predictions. The model is trained with a learning rate of 0.001 for 1000 iterations, achieving the highest accuracy of 13.278% as shown in figure 8.

2 ANN with Kaiming Weight Initialization

This ANN employs Kaiming initialization for weights $W^{(1)}$ and $W^{(2)}$ while biases $b^{(1)}$ and $b^{(2)}$ are randomly initialized. The dimensions remain $W^{(1)}$: 10×784, $W^{(2)}$: 10×10, $b^{(1)}$: 10×1, and $b^{(2)}$: 10×1. The ReLU activation function is used in the hidden layer and the Softmax activation function in the output layer. The model is trained with a learning

rate of 0.0001 for 1000 iterations, achieving the highest accuracy of 9.066% as shown in figure 12.

3 ANN with Different Hidden Layer Size

This ANN configuration increases the hidden layer size to 256 neurons, with weights initialized using the Kaiming method. The dimensions are $W^{(1)}$: 256×784, $W^{(2)}$: 10×256, $b^{(1)}$: 256×1, and $b^{(2)}$: 10×1. The ReLU activation function is used in the hidden layer and the Softmax activation function in the output layer. The model is trained with a learning rate of 0.001 for 1000 iterations, achieving the highest accuracy of 15.39% as shown in figure 10.

4 ANN with 2 Hidden Layers

This ANN includes two hidden layers with 512 and 128 neurons respectively. The weights and biases are randomly initialized. The dimensions are $W^{(1)}$: 512×784, $W^{(2)}$: 128×512, $W^{(3)}$: 10×128, $b^{(1)}$: 512×1, $b^{(2)}$: 128×1, and $b^{(3)}$: 10×1. The ReLU activation function is used in the hidden layers, and the Softmax activation function in the output layer. The model is trained with a learning rate of 0.001 for 1000 iterations, achieving a highest accuracy of 15% as shown in figure 11.

5 ANN with Different Activation Function

This ANN employs the tanh activation function in the hidden layer and the Sigmoid activation function in the output layer. The weights and biases are randomly initialized. The dimensions remain $W^{(1)}$: 10×784, $W^{(2)}$: 10×10, $b^{(1)}$: 10×1, and $b^{(2)}$: 10×1. Figures 7 and 6 show the plots of the tanh function and its derivative, and the Sigmoid function, respectively. The model is trained with a learning rate of 0.001 for 1000 iterations, achieving the highest accuracy of 6.656% as shown in figure 8.

6 ANN with Combined Parameters

This ANN features two hidden layers with 512 and 128 neurons, and weights initialized using the Kaiming method. The dimensions are $W^{(1)}$: 512×784, $W^{(2)}$: 128×512, $W^{(3)}$: 10×128, $b^{(1)}$: 512×1, $b^{(2)}$: 128×1, and $b^{(3)}$: 10×1. The LeakyReLU activation function is used in the hidden layers and the Sigmoid activation function in the output layer. Regularization is applied with $\lambda = 0.02$. The model is trained with a learning rate of 0.001 for 5000 iterations, achieving the highest accuracy of 86.061% as shown in figure 13.

Accuracies obtained for different parameters are displayed in table 2.

B PROBLEM 2: CIFAR-10 Dataset

The CIFAR-10 dataset contains 32x32 pixel color images across 10 different classes, making it more complex than MNIST. Figure 3 shows sample images from this dataset. The input data is transformed into a 2-dimensional format, transposed, and normalized.

1 Complex ANN for CIFAR-10

The ANN for CIFAR-10 consists of 3072 neurons in the input layer, 128 neurons in the hidden layer, and 10 neurons in the output layer. Weights are initialized using the Kaiming method, and biases are randomly initialized. The dimensions are $W^{(1)}$: 128×3072, $W^{(2)}$: 128×128, $W^{(3)}$: 10×128, $b^{(1)}$: 128×1, $b^{(2)}$: 128×1, and $b^{(3)}$: 10×1. The LeakyReLU and Tanh activation functions are used in the hidden layers and the Sigmoid activation function in the output layer. Regularization is applied with $\lambda = 0.02$. The model is trained with a learning rate of 0.01 for 500 iterations, achieving the highest accuracy of 26.326% as shown in figure 14.

The CIFAR-10 dataset is more complex than MNIST due to the larger input size and the diversity of the images. As a result, the simple ANN used here does not perform as well. The lower accuracy indicates the need for more advanced models, such as Convolutional Neural Networks (CNNs), which are better suited for handling complex image data.

VI DISCUSSION AND ANALYSIS

The experiments conducted on the MNIST and CIFAR-10 datasets aimed to explore the performance of various ANN configurations under different initialization methods, activation functions, and hidden layer structures. Below

is a summary of each model and its result analysis.

1 Simple ANN

The simple ANN with randomly initialized weights and biases achieved a maximum accuracy of 13.278% on the MNIST dataset after 1000 iterations. The accuracy versus iteration graph reveals that the model's accuracy improves steadily during the initial stages of training, reflecting effective learning from the training data. This upward trend continues until around 600 iterations, at which point the accuracy peaks. Following this, the accuracy begins to decline, suggesting that the model is overfitting. Overfitting occurs when the model excessively learns from the training data, including its noise and outliers, which hampers performance on unseen data. To address this issue, the model could benefit from regularization techniques like dropout or strategies such as early stopping to prevent further overfitting and ensure better generalization.

2 ANN with Kaiming Weight Initialization

Introducing Kaiming weight initialization aimed to enhance convergence, yet the model achieved a lower accuracy of 9.066%. This result suggests that while Kaiming initialization is known to benefit deeper networks by improving weight scaling, its effects on shallow networks with fewer neurons may be less significant. The second graph illustrates the accuracy of an ANN with Kaiming initialization over 1000 iterations. The accuracy starts at around 0.05 and shows a consistent, linear increase, reaching approximately 0.09 by the end of the training. This pattern indicates that Kaiming initialization provides a more stable starting point, facilitating a steadier and more reliable improvement in accuracy compared to the random initialization method used previously.

3 ANN with Different Hidden Layer Size

This ANN configuration, which includes a hidden layer of 256 neurons with Kaiming weight initialization and ReLU activation, achieved a maximum accuracy of 15.39% after 1000 iterations. The graph depicting the model's performance shows a gradual increase in accuracy from around 13.2% to approximately 15.4% over the training period. This steady but slow improvement suggests that the model is learning, albeit at a slow pace. Potential reasons for the slow convergence may include the complexity of the learning task, suboptimal hyperparameters, or issues such as vanishing or exploding gradients. To enhance performance, further investigation into the model's architecture, hyperparameters, and data characteristics would be necessary.

4 ANN with 2 Hidden Layers

This ANN features two hidden layers with 512 and 128 neurons respectively, and employs ReLU activation for the hidden layers and Softmax for the output layer, with weights and biases randomly initialized. Despite training for 1000 iterations with a learning rate of 0.001, the model achieved a maximum accuracy of only 15%. The corresponding graph reveals that accuracy starts at around 11.5% and gradually improves to about 15% over 500 iterations, suggesting steady but slow progress. The slow rate of improvement may be due to the complexity of the problem, suboptimal hyperparameters, potential issues with vanishing or exploding gradients, or limitations in the training data. To better understand and address these issues, it would be beneficial to explore different hyperparameter settings, analyze the loss function over time, reassess the model architecture, and evaluate the quality and quantity of the training data.

5 ANN with Different Activation Function

The graph illustrates the accuracy of an ANN over 1000 iterations using a specific initialization function, possibly the dfunc method. The accuracy begins at around 0.056 and gradually rises to approximately 0.066, indicating a steady but slow improvement in performance. This suggests that the initialization method contributes to a stable training process, though convergence is relatively slow. Switching to the tanh activation function for

the hidden layer and Sigmoid for the output layer resulted in an accuracy of 6.656%. This lower accuracy underscores the significant impact of activation functions on model performance, revealing that ReLU and Softmax activation functions are more effective for this dataset and architecture compared to Tanh and Sigmoid.

6 ANN with Combined Parameters

Combining various improvements—two hidden layers with 512 and 128 neurons, Kaiming initialization, LeakyReLU activation function, and regularization—resulted in a significant accuracy of 86.061% on the MNIST dataset. This outcome demonstrates that meticulous selection and integration of initialization methods, activation functions, and regularization techniques can markedly enhance model performance. The third graph illustrates the accuracy versus iteration for this well-performing model. Initially, from 0 to 1000 iterations, the accuracy increases sharply, reflecting the model's rapid learning of data patterns. From 1000 to 4000 iterations, the accuracy continues to improve more gradually, indicating fine-tuning of model parameters. Finally, from 4000 to 5000 iterations, the accuracy levels off, suggesting that the model has reached a high proficiency and is effectively generalizing to new data. This learning curve is indicative of a well-optimized model, benefiting from a balanced combination of learning rate, network architecture, and regularization techniques.

7 ANN for CIFAR-10 Dataset

The CIFAR-10 dataset presented a more complex challenge, requiring a more sophisticated ANN with 3072 input neurons, 128 hidden neurons, LeakyReLU and Tanh activation functions, and regularization, achieving an accuracy of 26.326% after 500 iterations. This lower accuracy compared to the MNIST dataset reflects CIFAR-10's greater complexity and diversity, highlighting the need for more advanced architectures like CNNs. The second graph illustrates the training process of this CIFAR ANN model, showing an initial phase with significant accuracy fluctuations from 0 to 200 iterations, indicating struggles with stable learning patterns, possibly due to high learning rates or inappropriate hyperparameters. From 200 to 400 iterations, while oscillations persist, there is a general upward trend suggesting gradual learning. Between 400 and 500 iterations, the accuracy curve stabilizes with consistent improvement, indicating that the model is starting to converge toward a better understanding of the data. Further hyperparameter optimization could reduce initial instability and accelerate convergence.

Overall, these experiments highlight the critical role of network architecture, initialization methods, activation functions, and regularization in training ANNs. Properly tuning these parameters is essential for achieving higher accuracy and better generalization, especially when dealing with more complex datasets.

VII CONCLUSION

In conclusion, the various ANN configurations tested on the MNIST dataset revealed significant insights into the impact of network architecture and initialization techniques on model performance. The initial simple ANN with random weight initialization yielded a modest accuracy of 13.278%, highlighting the importance of initialization in achieving effective learning. The introduction of Kaiming initialization, though beneficial for deeper networks, did not substantially improve performance in shallow networks, achieving a lower accuracy of 9.066%. Conversely, expanding the network with larger hidden layers or additional layers resulted in modest improvements, with accuracies reaching up to 15% and 15.39%, respectively. The most notable enhancement came from combining two hidden layers with Kaiming initialization, LeakyReLU activation, and regularization, achieving a significant accuracy of 86.061%. This underscores the effectiveness of a carefully chosen combination of techniques in optimizing model performance.

The CIFAR-10 dataset, with its increased complexity, demonstrated the limitations of

these ANN configurations, achieving a maximum accuracy of 26.326%. The complexity of CIFAR-10 necessitates more advanced architectures, such as CNNs, to effectively capture the intricate patterns within the data. The training process of the CIFAR ANN revealed initial instability, with accuracy fluctuations indicating challenges in achieving stable learning. However, as training progressed, the model began to converge, showing consistent improvement. This suggests that while the ANN configuration made some progress, further optimization and advanced network architectures are required to handle the complexities of CIFAR-10 more effectively.

VIII APPENDICES

A Equations

Some of the equations used in this paper are as follows:

Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Softmax Function

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2}$$

where x_i is the i^{th} element of the input vector x and n is the number of elements in x.

ReLU

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3}$$

Tanh Function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4}$$

LeakyReLU

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{5}$$

where x denotes the input vector and α is a small constant.

Accuracy

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total Instances}} \tag{6}$$

where,
True Positive TP = Model predicted correctly for true data
True Negative TN = Model predicted correctly for false data
Total Instances = True Positive + False Positive + True Negative + False Negative i.e. total data

B Figures

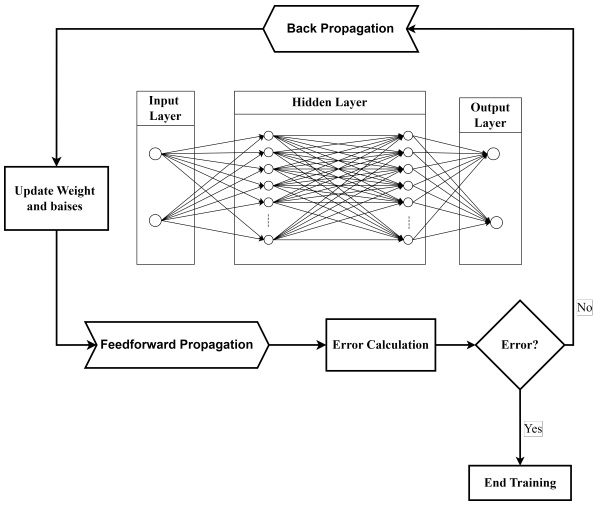

Figure 1: System Block Diagram

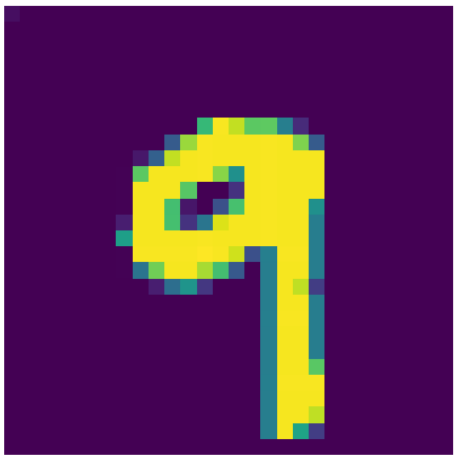

Figure 2: Example of MNIST image

Figure 3: Example of MNIST image

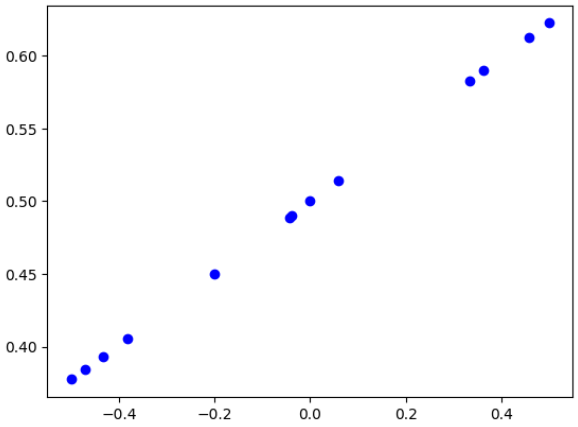

Figure 6: Visualization of Sigmoid function

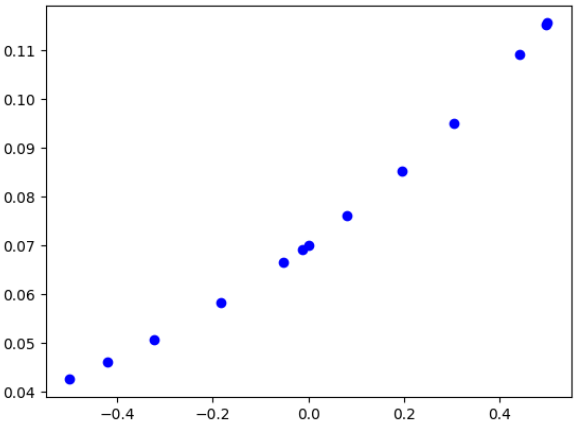

Figure 4: Visualization of Softmax function

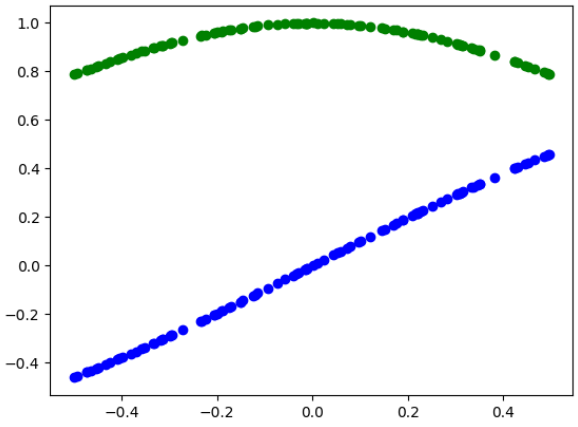

Figure 7: Visualization of Tanh function and its derivative

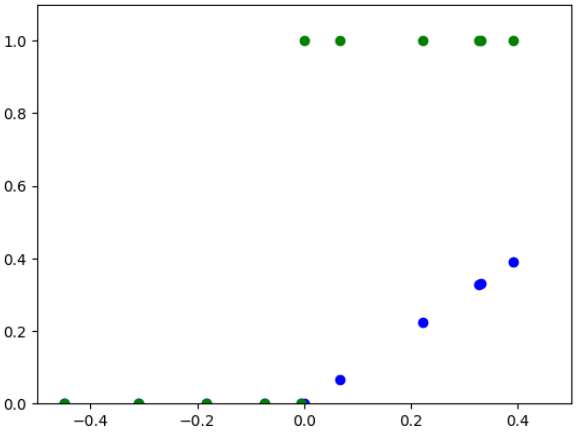

Figure 5: Visualization of ReLU function and its derivative

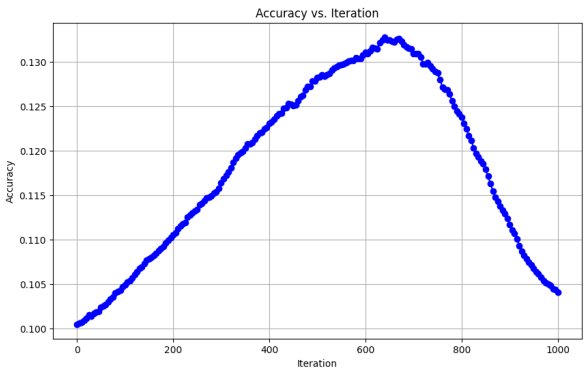

Figure 8: MNIST Accuracy plot using simple ANN

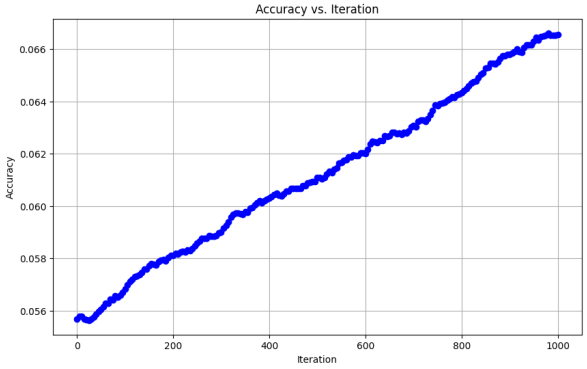

Figure 9: MNIST Accuracy plot using different activation functions

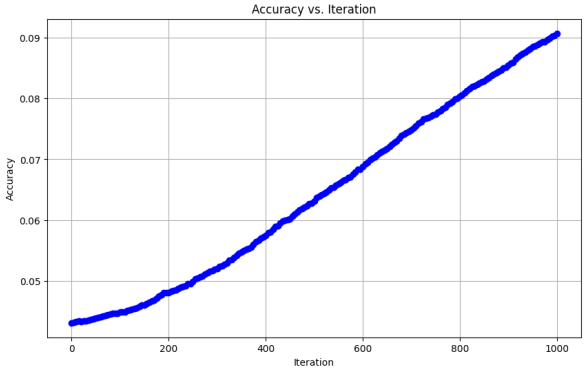

Figure 12: MNIST Accuracy plot using Kaiming Weight Initialization

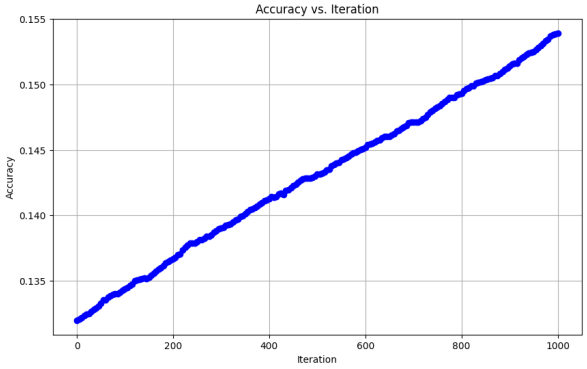

Figure 10: MNIST Accuracy plot using different number of hidden layers

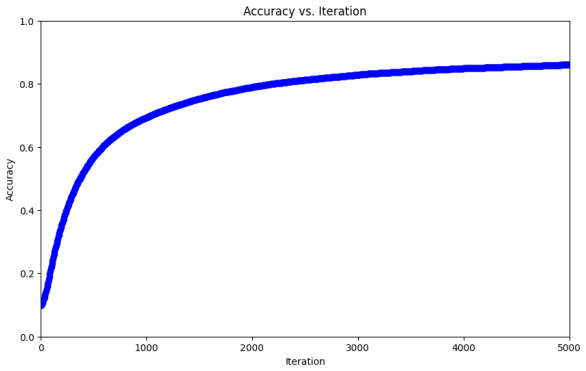

Figure 13: Best MNIST Accuracy plot using different training parameters

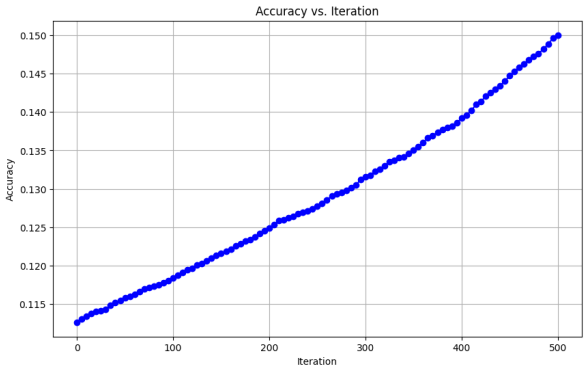

Figure 11: MNIST Accuracy plot using additional hidden layers

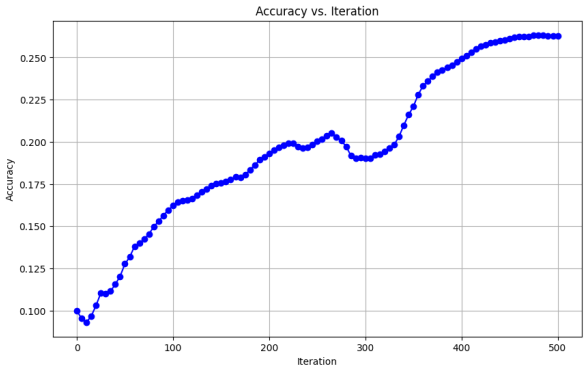

Figure 14: CIFAR Accuracy plot using different training parameters

C Tables

Table 1: Accuracy analysis on MNIST dataset

Functions/Parameters	Iteration	Accuracy
Basic	1000	0.13278
Different Activation Functions	1000	0.06656
Kaiming Initialization	1000	0.09066
Different Hidden Layer Size	1000	0.15393
Multiple Hidden Layers	500	0.15002
Combined	5000	0.86061

Table 2: Accuracy analysis on CIFAR dataset

Functions/Parameters	Iteration	Accuracy
Combined	500	0.26326

D Backpropagation Derivation

We consider a neural network with one input layer, one hidden layer, and one output layer. The hidden layer uses the ReLU activation function, and the output layer uses the Softmax activation function. The cost function is the cross-entropy loss for a multi-class classification problem with 10 classes.

1 Forward Propagation

Let: $\mathbf{A^{(0)}} \in R^{784\times m}$: input vector

There are 784 features (e.g., pixels in a flattened 28x28 image) and m examples in the input vector.

$\mathbf{W^{(1)}} \in R^{10\times 784}$: weights between the input layer and the hidden layer

There are 784 input features, and the hidden layer has 10 units.

$\mathbf{b^{(1)}} \in R^{10\times 1}$: biases for the hidden layer

There are 10 units in the hidden layer, so there are 10 bias values.

$\mathbf{W^{(2)}} \in R^{10\times 10}$: weights between the hidden layer and the output layer

There are 10 units in the hidden layer and 10 units in the output layer (for the 10 classes).

$\mathbf{b^{(2)}} \in R^{10\times 1}$: biases for the output layer

There are 10 output units (for the 10 classes), so there are 10 bias values.

The output of the hidden layer (before activation) is:

$$\mathbf{Z^{(1)}} = \mathbf{W^{(1)}}\mathbf{A^{(0)}} + \mathbf{b^{(1)}} \tag{7}$$

The activation of the hidden layer using ReLU is:

$$\mathbf{A^{(1)}} = \text{ReLU}(\mathbf{Z^{(1)}}) = \max(0, \mathbf{Z^{(1)}}) \tag{8}$$

The shape of $\mathbf{Z^{(1)}}$ obtained is $\mathbf{10 \times m}$. So, the shape of $\mathbf{A^{(1)}}$ is also $\mathbf{10 \times m}$.

The output of the output layer (before activation) is:

$$\mathbf{Z^{(2)}} = \mathbf{W^{(2)}}\mathbf{A^{(1)}} + \mathbf{b^{(2)}} \tag{9}$$

The activation of the output layer using Softmax is:

$$\mathbf{A^{(2)}} = \text{Softmax}(\mathbf{Z^{(2)}}) = \frac{e^{\mathbf{Z^{(2)}}}}{\sum_{j=1}^{k} e^{\mathbf{Z_j}^{(2)}}} \tag{10}$$

The shape of $\mathbf{Z^{(2)}}$ obtained is $\mathbf{10 \times m}$. So, the shape of $\mathbf{A^{(2)}}$ is also $\mathbf{10 \times m}$.

2 Cross-Entropy Loss

For a single training example, the cross-entropy loss is given by:

$$L = -\sum_{i=1}^{k} y_i \log(A_i^{(2)}) \tag{11}$$

where y_i is the true label (one-hot encoded) and $A_i^{(2)}$ is the predicted probability for class i.

3 Back Propagation

Gradient of the Loss with respect to the Output Layer Activation

First, we calculate the gradient of the loss L with respect to the activation $\mathbf{A^{(2)}}$:

$$\frac{\partial L}{\partial A_i^{(2)}} = -\frac{y_i}{A_i^{(2)}} \tag{12}$$

3.1.Gradient of the Loss with respect to the Output Layer Input

Next, we use the chain rule to find the gradient with respect to the input of the softmax function $\mathbf{Z^{(2)}}$:

$$\frac{\partial L}{\partial Z_i^{(2)}} = \sum_{j=1}^{k} \frac{\partial L}{\partial A_j^{(2)}} \frac{\partial A_j^{(2)}}{\partial Z_i^{(2)}} \tag{13}$$

The gradient of the softmax function is:

$$\frac{\partial A_j^{(2)}}{\partial Z_i^{(2)}} = A_j^{(2)}(\delta_{ij} - A_i^{(2)}) \tag{14}$$

where δ_{ij} is the Kronecker delta.
Thus:

$$\frac{\partial L}{\partial Z_i^{(2)}} = \sum_{j=1}^{k} -\frac{y_j}{A_j^{(2)}} A_j^{(2)}(\delta_{ij} - A_i^{(2)}) \tag{15}$$

$$\frac{\partial L}{\partial Z_i^{(2)}} = A_i^{(2)} - y_i \tag{16}$$

So, in matrix form, we get:

$$\frac{\partial L}{\partial Z_i^{(2)}} = A^{(2)} - Y \tag{17}$$

where $\mathbf{A^{(2)}}$ is the matrix for all classes and $\mathbf{Y}$ is the matrix of actual output for all classes.

3.2.Gradient of the Loss with respect to the Weights and Biases of the Output Layer

The gradient with respect to the weights $\mathbf{W}^{(2)}$ and biases $\mathbf{b}^{(2)}$ are:

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{Z}^{(2)}} \cdot (\mathbf{A}^{(1)})^T \tag{18}$$

$$\frac{\partial L}{\partial \mathbf{b}^{(2)}} = \frac{\partial L}{\partial \mathbf{Z}^{(2)}} \tag{19}$$

To compute the gradient of the loss L with respect to the weights $\mathbf{W}^{(2)}$ and biases $\mathbf{b}^{(2)}$ of the output layer, we use the chain rule.
Using the chain rule, the gradient with respect to $\mathbf{W}^{(2)}$ is:

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{Z}^{(2)}} \cdot \frac{\partial \mathbf{Z}^{(2)}}{\partial \mathbf{W}^{(2)}} \tag{20}$$

Since:

$$\frac{\partial \mathbf{Z}^{(2)}}{\partial \mathbf{W}^{(2)}} = \mathbf{A}^{(1)} \tag{21}$$

It follows that:

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{Z}^{(2)}} \cdot (\mathbf{A}^{(1)})^T \tag{22}$$

We take the transpose of $\mathbf{A}^{(1)}$ for matrix multiplication.

Using the chain rule, the gradient of the loss with respect to $\mathbf{b}^{(2)}$ is:

$$\frac{\partial L}{\partial \mathbf{b}^{(2)}} = \frac{\partial L}{\partial \mathbf{Z}^{(2)}} \cdot \frac{\partial \mathbf{Z}^{(2)}}{\partial \mathbf{b}^{(2)}} \tag{23}$$

Since:

$$\frac{\partial \mathbf{Z}^{(2)}}{\partial \mathbf{b}^{(2)}} = \mathbf{I} \tag{24}$$

Here, $\mathbf{I}$ denotes the identity matrix.
It follows that:

$$\frac{\partial L}{\partial \mathbf{b}^{(2)}} = \frac{\partial L}{\partial \mathbf{Z}^{(2)}} \tag{25}$$

3.3.Gradient of the Loss with respect to the Hidden Layer Activation

Next, we propagate the gradients to the hidden layer. The gradient of the loss with respect to the hidden layer activation $\mathbf{a}^{(1)}$ is:

$$\frac{\partial L}{\partial \mathbf{A}^{(1)}} = (\mathbf{W}^{(2)})^T \cdot \frac{\partial L}{\partial \mathbf{Z}^{(2)}} \tag{26}$$

3.4.Gradient of the Loss with respect to the Hidden Layer Input

The ReLU function's gradient is:

$$\frac{\partial A_i^{(1)}}{\partial Z_i^{(1)}} = \begin{cases} 1 & \text{if } Z_i^{(1)} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

Thus:

$$\frac{\partial L}{\partial \mathbf{Z}^{(1)}} = \frac{\partial L}{\partial \mathbf{A}^{(1)}} \cdot^* \text{ReLU}'(\mathbf{Z}^{(1)}) \tag{28}$$

where $\cdot^*$ denotes element-wise multiplication.

3.5.Gradient of the Loss with respect to the Weights and Biases of the Hidden Layer

Finally, the gradient with respect to the weights $\mathbf{W}^{(1)}$ and biases $\mathbf{b}^{(1)}$ are:

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{1}{m} \cdot \frac{\partial L}{\partial \mathbf{Z}^{(1)}} \cdot (\mathbf{A}^{(0)})^T \tag{29}$$

$$\frac{\partial L}{\partial \mathbf{b}^{(1)}} = \frac{1}{m} \cdot \sum \frac{\partial L}{\partial \mathbf{Z}^{(1)}} \tag{30}$$

The gradient with respect to $\mathbf{W}^{(1)}$ is:

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial L}{\partial \mathbf{Z}^{(1)}} \cdot \frac{\partial \mathbf{Z}^{(1)}}{\partial \mathbf{W}^{(1)}} \tag{31}$$

Since:

$$\frac{\partial \mathbf{Z}^{(1)}}{\partial \mathbf{W}^{(1)}} = \mathbf{A}^{(0)} \tag{32}$$

It follows that:

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial L}{\partial \mathbf{Z}^{(1)}} \cdot (\mathbf{A}^{(0)})^T \tag{33}$$

Considering m examples:

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{1}{m} \cdot \left(\frac{\partial L}{\partial \mathbf{Z}^{(1)}} \cdot (\mathbf{A}^{(0)})^T \right) \tag{34}$$

The gradient of the loss with respect to $\mathbf{b}^{(1)}$ is:

$$\frac{\partial L}{\partial \mathbf{b}^{(1)}} = \frac{1}{m} \cdot \sum \frac{\partial L}{\partial \mathbf{Z}^{(1)}} \tag{35}$$

New Weights for $\mathbf{W}^{(1)}$:

$$\mathbf{W}^{(1)}_{\text{new}} = \mathbf{W}^{(1)} - \frac{\eta}{m} \cdot \left(\frac{\partial L}{\partial \mathbf{Z}^{(1)}} \cdot (\mathbf{A}^{(0)})^T \right) \tag{36}$$

where η denotes learning rate. **New Weights for $\mathbf{W}^{(2)}$:**

$$\mathbf{W}^{(2)}_{\text{new}} = \mathbf{W}^{(2)} - \eta \cdot \left(\frac{\partial L}{\partial \mathbf{Z}^{(2)}} \cdot (\mathbf{A}^{(1)})^T \right) \tag{37}$$

New Biases for $\mathbf{b}^{(1)}$:

$$\mathbf{b}^{(1)}_{\text{new}} = \mathbf{b}^{(1)} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}^{(1)}} \tag{38}$$

Substituting the gradient:

$$\mathbf{b}^{(1)}_{\text{new}} = \mathbf{b}^{(1)} - \eta \cdot \frac{\partial L}{\partial \mathbf{Z}^{(1)}} \tag{39}$$

New Biases for $\mathbf{b}^{(2)}$:

$$\mathbf{b}^{(2)}_{\text{new}} = \mathbf{b}^{(2)} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}^{(2)}} \tag{40}$$

Substituting the gradient:

$$\mathbf{b}^{(2)}_{\text{new}} = \mathbf{b}^{(2)} - \eta \cdot \frac{\partial L}{\partial \mathbf{Z}^{(2)}} \tag{41}$$

This process repeats until the stopping criteria are met or the maximum number of iterations is completed.
This is the complete derivation of the back-propagation algorithm.

REFERENCES

[1] J. Zupan, "Introduction to artificial neural network (ann) methods: What they are and how to use them," *Acta Chimica Slovenica*, vol. 41, 01 1994.

[2] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.08458

Kristina Ghimire is currently pursuing her undergraduate degree in Computer Engineering at IOE, Thapathali Campus. Her research interests encompass various areas, including data mining, machine learning, and deep learning.(THA077BCT023)

Punam Shrestha is currently pursuing her undergraduate degree in Computer Engineering at IOE, Thapathali Campus. Her research interests encompass various areas, including data mining, and web development.(THA077BCT038)