**Comparative Analysis on Handwritten Digit Recognizer Using CNN and AN**

***Abstract:* Handwriting recognition, whether online and offline, faces difficult problems in correctly deciphering characters because different handwriting styles lead to distortions and differences in handwriting patterns. Artificial neural networks (ANNs) and convolutional neural networks (CNNs) are fundamental to machine learning. When it comes to capturing spatial patterns and translation invariance, Convolutional Neural Networks (CNNs) do exceptionally well. Artificial Neural Networks (ANNs), on the other hand, manage intricate data linkages and quickly adjust to new knowledge.**

**In the context of handwritten digit recognition, this research study compares and contrasts CNNs and ANNs in order to investigate their respective advantages, disadvantages, and suitability for this particular task. Our suggested technique provides an overall accuracy of 98.30% for CNN and 98.64 percent for Train and 98.42% for ANN.**

Keywords— Handwriting Recognition, Convolutional Neural Networks (CNN), Artificial Neural Networks (ANN), MNIST Dataset.

1. Introduction

With a 2023 report revealing a 40% year-on-year increase in AI and ML adoption by enterprises. Algorithms drive this surge, with 82% of companies utilizing machine learning algorithms to understand data, fueling predictions of a global market value reaching $96 billion by 2025.

In several industries, increased use of deep learning algorithms is likely to enable more efficient use of time and energy. It is most commonly used in virtual assistance, voice-controlled remote controls and new technologies such as autonomous vehicles. The global deep learning market size is projected to grow from $17.60 billion in 2023 to $188.58 billion by 2030, at a CAGR of 40.3% during the forecast period[1].

Handwritten digit recognition holds immense significance in today's world due to its diverse applications across various sectors. As it also plays a pivotal role in historical document digitization and analysis, preserving and deciphering invaluable manuscripts and records. Additionally, its application in authentication and security, such as signature

verification for legal documents or bank transactions, underscores its role in ensuring data integrity and preventing fraud. The versatility of handwritten digit recognition spans industries, contributing significantly to efficiency, accuracy, and data management in today's multifaceted world.

In order to enable machines to comprehend abstract patterns and make predictions in domains like language processing and financial forecasting—which account for 95% of neural network applications—artificial neural networks (ANNs) mimic the neural connections seen in the brain and learn from data.

CNNs, commonly referred to as ConvNets, are multi-layered neural networks that are mostly utilized for object detection and image processing. It is extensively utilized for anomaly detection, time series forecasting, medical image processing, and satellite image identification. In 2012, CNNs had an enormous spike in popularity (which still to this day) following the achievement of state-of-the-art performance in the ImageNet challenge by a CNN named AlexNet.

2. Literature Review

Numerous studies explore the MNIST handwritten dataset's reliability. Saeed Al Mansoori's work on” Intelligent Handwritten Digit Recognition using Artificial Neural Network”[1] utilized 5000 MNIST samples, achieving 99.32% accuracy via gradient descent back-propagation and feed-forward testing with a Multilayer Perceptron (MLP) Neural Network.

Vijayalaxmi R Rudraswamimath and Bhavanishankar K employed SVM, KNN, RFC, and CNN, attaining 98.72% (CNN) and 96.67% (KNN) accuracy in "Handwritten Digit Recognition using CNN"[2]. Their approach allowed users to provide input by either uploading the image of the digit or using data from the MNIST dataset.

Drishti Beohar and Akhtar Rasool's[3] "Handwritten Digit Recognition of MNIST dataset using Deep Learning" compared ANN and CNN, with the help of PyCharm with Python 3.7. They utilized backpropagation, Gradient Descent, reLU activations, categorical cross-entropy loss, and ADAM optimizer on 60,000 images (10 epochs, batch size 200), resulting in an average baseline error of 1.31% for ANN and 0.91% for CNN.

Authors Ritik Dixit, Rishika Kushwah, and Samay Pashine[4] of “Handwritten Digit Recognition using Machine and Deep Learning Algorithms” explored handwritten digit recognition using SVM, CNNs, and MLP with the MNIST dataset. SVM performed well on simpler datasets but it is not capable of classifying complex and ambiguous images accurately. Overall accuracies: SVM (Train: 99.98%, Test: 93.77%), MLP (Train: 99.92%, Test: 98.85%), CNN (Train: 99.53%, Test: 99.31%), trained over 30 epochs.

Tsehay Admassu Assegie[5] presented an article titled "Handwritten digit recognition using decision tree classification”. The results showed that number 1 had the highest accuracy rate of 93.73%, while number 8 had the lowest accuracy rate of 84.12%.Overall, the model achieved an accuracy of 83.4%.

BM Vinjit, Mohit Kumar Bhojak, Sujit Kumar, and Gitanjali Nikam created a "Handwritten digit recognition module using Convolutional Neural Network (CNN)" [6] on MNIST. The CNN featured specialized architecture, with 32 filters in the first convolution layer and 64 in the next two, achieving 99.36% training and 99.15% testing accuracy.

Kh Tohidul Islam, Ghulam Mujtaba, Dr. Ram Gopal Raj, and Henry Friday Nweke[7] developed a “Handwritten digit recognition system using artificial neural networks (ANN)” Trained on MNIST, the model, with a single hidden layer, achieved an impressive 99.60% overall classification accuracy, showcasing ANN's efficacy in recognizing handwritten digits accurately.

The paper [8], “A Comparative Study of Different Deep Learning Model for Recognition of Handwriting Digits” by Pronab Ghosh, Atqiya Abida Anjum, Asif Karim, Masum Shah Junayed, Md. Zahid Hasan, Khan Md. Hasib, Al Nahian Bin Emran, compares ANNs and CNNs using 60,000 MNIST training images and 10,000 for testing. They employed Multi-layered, fully connected neural networks (NN) with 10 and 12 hidden layers for handwritten digit (HD) recognition. Results showed ANN with 10 layers at 99.10%, 12 layers at 99.34%, and CNN at 99.70% overall accuracy.

3. Proposed Solution

3.1 Dataset:

The **MNIST** database (**Modified National Institute of Standards and Technology** database) is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger NIST Special Database 3 (digits written by employees of the United States Census Bureau) and Special Database 1 (digits written by high school students) which contain monochrome images of handwritten digits [10].

Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset, the original dataset was a set of 128x128 binary images, processed into 28x28 grayscale images. It is a labeled dataset that pairs images of hand-written numerals with the name of the respective numeral, it can be also used in supervised learning to train classifiers. It contains numbers from 0 to 9 and a sample of this dataset is presented below in the figure 1.1.

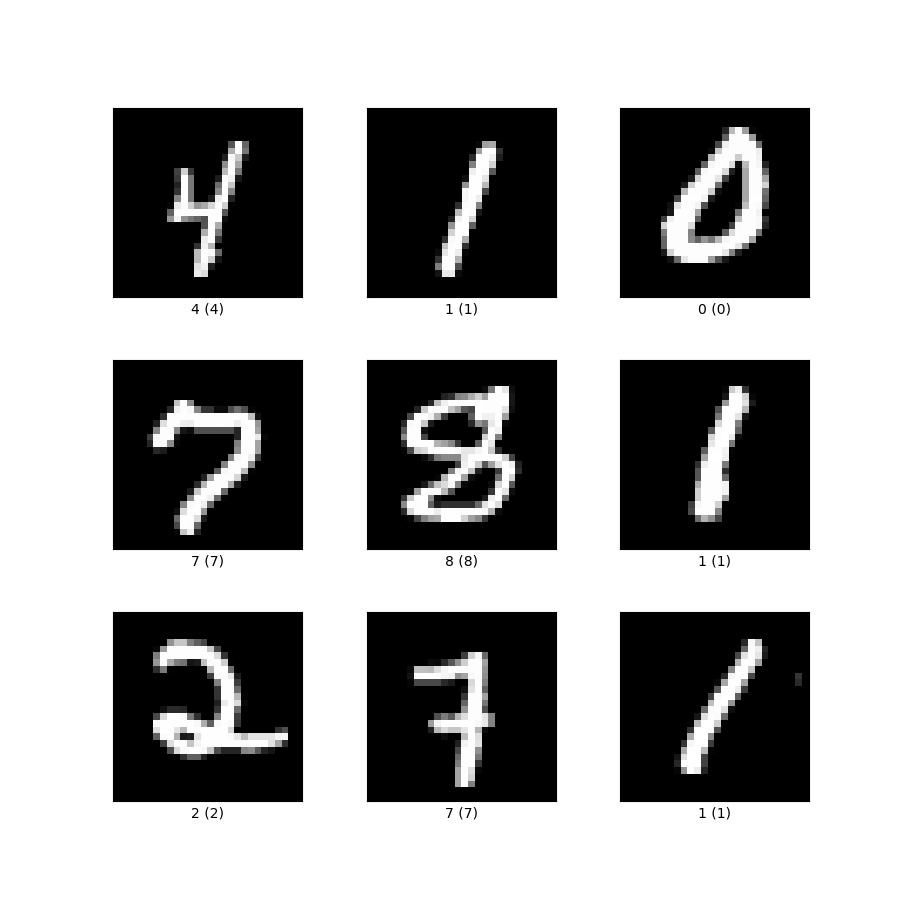
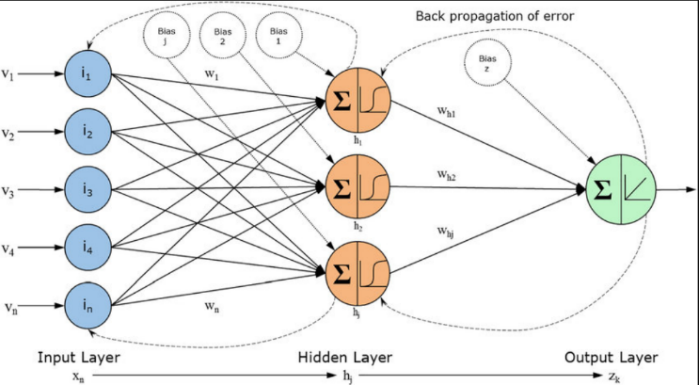


Figure 1

3.2 Artificial Neural Network (ANN):

Artificial Neural Networks (ANNs) are a fundamental concept in the field of artificial intelligence and machine learning. Inspired by the structure and function of biological neural networks in the human brain[11], ANNs are computational models designed to simulate the learning processes that occur in the human nervous system. ANNs consist of interconnected nodes, also known as artificial neurons or perceptrons, organized into layers.

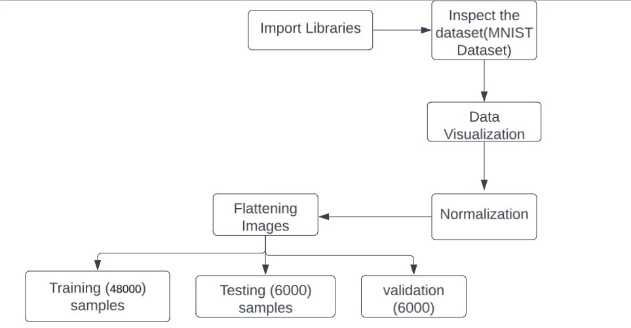
The basic structure comprises an input layer, one or more hidden layers, and an output layer, a sample of ANN with single input, hidden and output layer is given in the figure 1.2 below.



***Figure 2***

Each connection between neurons has an associated weight, and the network learns by adjusting these weights based on the input data[12]. The learning process involves feeding input data through the network, computing an output, comparing it to the desired output, and adjusting the weights accordingly through a process known as backpropagation.

The basic workflow[13] of ANNs is given below in the figure1.3

****

***Figure 3***

To start our exploration, we imported the necessary libraries such as TensorFlow, Keras and Matplotlib. Using these tools, we loaded the MNIST dataset and conducted the study preliminary examination of its form and content.

Recognizing the importance of preparing the data for training neural networks, we performed normalizing pixel values between 0 and 1. This preprocessing step is crucial for improving stability and efficiency of the training process and contributes to uniformity optimization algorithms.

To suit the requirements of certain neural network architectures, particularly those with dense layers, we flattened the images into 1D arrays. This transformation simplified the input structure, optimizing the model's performance. We experimented with various optimizers and activation functions in the creation of a neural network model using the Keras library. Initially, we tested the model without any hidden layer.

**3.2.1 Very Simple neural network with no hidden layers:**

**Pseudo code:**

* **use sequential function in keras library to create input layer**
* **set learning rate of adadelta optimizer**
* **configure the model with adadelta optimize using sparse categorical crossentropy as loss function.**
* **monitor the training and testing accuracy**
* **initiate the training process using trainig data for 10 epochs.**

When training our model, we used the Adadelta optimizer for 10 epochs. The model performed commendably, achieving an accuracy of 92.5% on the training dataset and 93.8% on the testing dataset. To optimize the performance of the model, we chose a sigmoid activation function. Specifically, implementing the sigmoid function resulted in an accuracy of 92.6% in training and a slightly better accuracy of 93.68% in testing.

On the other hand, when we experimented with ReLU's leaky activation feature, the model's performance decreased significantly, recording an accuracy of 43.3% during training and 45.80% in testing. These results highlight the effectiveness of the sigmoid activation function in improving the overall performance of our model.

**3.2.2 With one hidden layer:**

In our single hidden layer model architecture, we examined various activation functions to determine their impact on training and testing accuracy. ReLU's implementation of leak activation resulted in a remarkable training accuracy of 97.8%, with a slightly lower but still impressive testing accuracy of 97.5%. ReLU's parametric activation function showed even higher performance, achieving a remarkable training accuracy of 99.33% and an equally impressive test accuracy of 98.08%.

Meanwhile, the standard ReLU activation function also showed excellent performance, offering 98.93% training accuracy and 97.80% testing accuracy. Notably, ReLU's parametric activation function was the most efficient of the three, producing the highest training accuracy of 99.33% and an impressive testing accuracy of 98.08%. This highlights the importance of the choice of activation function for optimizing the overall performance of the model.

**3.2.3 With two hidden layer:**

In our neural network model with two hidden layers, we used the parametric activation function ReLU and Adadelta optimizer. The training phase resulted in an exceptional accuracy of 99.48%, demonstrating the high learning ability of the model. During testing, the model maintained a high accuracy of 97.94%, which confirmed its generalization ability. The choice of activation function, which can fit and capture complex patterns in the data, contributed significantly to the overall success of the model.

**3.2.4 With three hidden layer:**

**Pseudo code:**

* **use sequential function in keras library to create input layer**
* **create first layer with parametric relu using dense function in keras library**
* **create second layer with parametric relu using dense function in keras library**
* **create the last layer with sigmoid activation function**

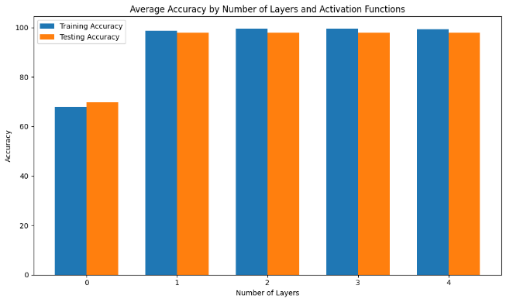
When building our neural network model, we increased its complexity by including three hidden layers using the parametric activation function ReLU and adadelta optimizer. The training phase delivered an impressive 99.47% accuracy, reflecting the model's ability to capture complex patterns in the training data set. During testing, the model demonstrated a high accuracy of 97.98%, highlighting its strong ability to generalize beyond training data. Consistent use of the ReLU parametric activation function across multiple hidden layers improved the model's performance in terms of learning and accurately predicting from unseen data.

**3.2.5 With four hidden layer:**

In the design of our neural network model, we augmented its complexity by introducing four hidden layers, accompanied by the Adadelta optimizer and the parametric ReLU activation function. Throughout the training process, the model exhibited exceptional learning capabilities, achieving a remarkable training accuracy of 99.45%. This high level of accuracy underscores the model's adeptness at capturing intricate patterns within the training dataset.

Subsequently, during the testing phase, the model demonstrated strong generalization, maintaining a commendable accuracy of 97.84%. The combination of the Adadelta optimizer and the ReLU parametric activation function on multiple hidden layers synergistically contributed to the overall performance of the model and highlighted its suitability for both training and accurately predicting results from unpublished data.

The below graph gives information between no. of layers versus accuracy.



***Figure 4 No. of Layers vs. Accuracy***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Activation function | Optimizer | Layer | Train data accuracy | Test data accuracy |
| Sigmoid | Adadelta | 0 | 92.5% | 93.8% |
| Leaky ReLU | Adadelta | 0 | 43.3% | 45.8% |
| Leaky ReLU | Adadelta | 1 | 97.8% | 97.5% |
| Parametric ReLU | Adadelta | 1 | 99.3% | 98.08% |
| Gated ReLU | Adadelta | 1 | 98.93% | 97.8% |
| Swish | Adadelta | 1 | 98.8% | 97.9% |
| Parametric ReLU | Adadelta | 2 | 99.48% | 97.94% |
| Parametric ReLU | Adadelta | 3 | 99.47% | 97.98% |
| Parametric ReLU | Adadelta | 4 | 99.43% | 97.8% |

***Table 1.1***

From the above table 1.1 we can infer that while using adadelta optimizer, parametric relu activation function in 3 hidden layers we got the highest accuracy.

To optimize a neural network model with three hidden layers, we conducted experiments with multiple optimizers. Using the Adadelta optimizer in combination with ReLU's parametric activation function produced impressive results with a training accuracy of 99.47% and a corresponding testing accuracy of 97.98%. Switching to the RMSprop optimizer maintained excellent performance and achieved a training accuracy of 99.15% and a test accuracy of 97.7%, showing the model's ability to generalize well.

Similarly, the SGD optimizer combined with the parametric activation function ReLU produced satisfactory results with a training accuracy of 97.10% and a testing accuracy of 97.35%. Research on the Nadam optimizer further improved the model, achieving a training accuracy of 99.08% and a testing accuracy of 97.83%, demonstrating its effectiveness in improving the learning capabilities of the model.

While the Adagrad optimizer produced competitive results, it demonstrated a training accuracy of 97.70% and a testing accuracy of 97.15%. These results highlight the varying impact of the choice of optimizer on model performance and highlight the importance of careful tuning of hyper parameters to achieve optimal results.

In our attempt to refine a neural network model that has maximum accuracy using the Adadelta optimizer and the parametric activation function ReLU, we conducted experiments with different learning rates. With a learning rate of 0.01, the model achieved a training accuracy of 90.08%, with a corresponding testing accuracy of 91.38%.Increasing the learning rate to 0.1 then led to a significant improvement in performance, showing a training accuracy of 96.47% and a testing accuracy of 96.6%.

Other experiments with learning rate 1.0 achieved remarkable results, training accuracy increased to 99.48%, and testing accuracy reached an impressive 98.07%. These results highlight the sensitivity of model performance to the choice of learning rate, with higher rates indicating the ability for accelerated learning and higher accuracy on both training and test datasets.

While using different optimizers for parametric ReLU activation function we obtained various results and it is represented in the below table 1.2

|  |  |  |  |
| --- | --- | --- | --- |
| Activation function | Optimizer | Train data accuracy | Test data accuracy |
| Parametric ReLU | RMSprop | 99.15% | 97.70% |
| Parametric ReLU | SGP | 97.10% | 97.35% |
| Parametric ReLU | Nadam | 99.08% | 97.83% |
| Parametric ReLU | Adagrad | 97.7% | 97.10% |

***Table 1.2***

3.3 Convolutional Neural Network (CNN):

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers[14].



***Figure 5 Simple CNN architecture [14]***

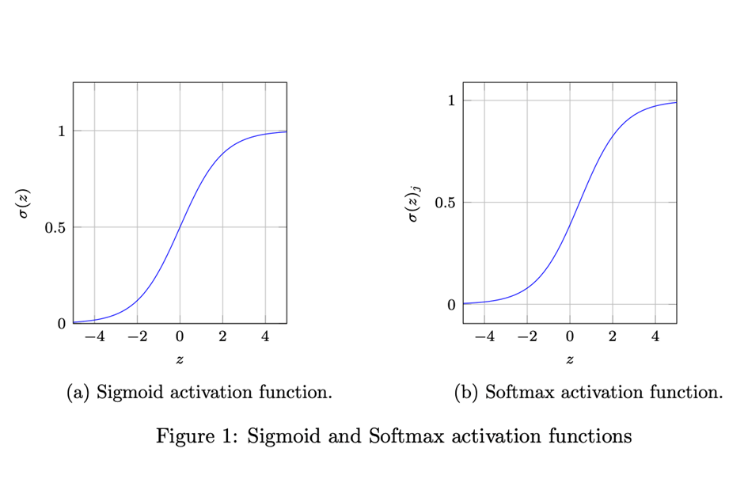
**3.3.1 Initial CNN Configuration with Sigmoid Activation**

In our exploration of convolutional neural networks (CNN), we initiated the model with a configuration comprising 32 kernels or features, each with a kernel size of 3x3. The activation function employed in this instance was the sigmoid function. However, the initial training and testing accuracies were notably low, recording 11.40% and 10.5%, respectively.

This outcome suggests that the chosen configuration may not be optimal for capturing the underlying patterns in the data. Further refinement of the CNN architecture, including adjustments to the number of kernels, kernel size, or activation function, may be necessary to enhance the model's ability to learn and generalize effectively.

**3.3.2 Improved Accuracy with Softmax Activation**

In response to the low accuracy initially observed in our convolutional neural network (CNN) model with 32 3x3 kernels and a sigmoid activation function, we made an improvement by replacing the activation function with a softmax. This adjustment led to a significant improvement in results: training accuracy increased to 98.78% and testing accuracy reached an impressive 98.4%.



***Figure 6 sigmoid and softmax activation function [10]***

The use of softmax activation functions significantly improved the model's ability to capture complex patterns in the data, demonstrating the importance of appropriate activation functions for the success of CNN architectures. These results highlight the iterative nature of model optimization and the impact that thoughtful adjustments to key parameters can have on the overall performance of a neural network.

In the refinement of our convolutional neural network (CNN) model, we configured it with 32 kernels of size 3x3 and implemented the rectified linear unit (ReLU) activation function. This modification yielded remarkable results, with the training accuracy surging to an impressive 99.90%, indicative of the model's proficiency in learning complex patterns within the training dataset. During testing, the model maintained a high accuracy level of 98.84%, emphasizing its robust generalization to unseen data.

Additionally, the computational time for training the model was efficiently managed, completing the process in a mere 3 minutes. This combination of high accuracy and efficient computational performance underscores the effectiveness of the chosen configuration and highlights the importance of selecting appropriate parameters to achieve optimal results in CNN architectures.

**3.3.3 Enhanced Performance with ReLU Activation**

In our ongoing effort to optimize the Convolutional Neural Network (CNN) model, we have further increased its complexity by increasing the number of kernels to 64, each 3x3 in size, while increasing the activation function of the rectified linear unit (ReLU). This fix resulted in an exceptional training accuracy of 99.95%, demonstrating the model's improved ability to recognize complex patterns in the training data set.

During testing, the model maintained impressive accuracy, achieving an accuracy rate of 98.95% and showing strong generalization to previously unseen data. However, the training computation time increased to 8 minutes due to the increased complexity and larger number of kernels.

This trade-off highlights the careful consideration required to balance accuracy and computational efficiency when fine-tuning CNN architectures to achieve specific goals.

Continuing our quest for optimal performance in the convolutional neural network (CNN) model, we further heightened its complexity by increasing the number of kernels to 128, each with a size of 3x3, while retaining the rectified linear unit (ReLU) activation function. This adjustment resulted in an impressive training accuracy of 99.93%, showcasing the model's advanced ability to discern intricate patterns within the training dataset.

During testing, the model maintained a robust accuracy of 98.95%, demonstrating its effectiveness in generalizing to previously unseen data. However, this increased complexity came at the cost of an extended computational time for training, requiring 15 minutes. This trade-off underscores the need for careful consideration when choosing the model architecture, as the balance between accuracy and computational efficiency becomes crucial in meeting specific requirements and constraints.

**3.3.4 Optimization with 64 Kernels and Max Pooling**

To refine the Convolutional Neural Network (CNN) architecture, we configured a single convolutional layer with 64 kernels, each 4x4 in size, and enabled the maximum pooling layer. The results of this setup were exceptional, with training accuracy of up to 99.94%, highlighting these ability to capture complex patterns in the training data set.

During testing, the model showed strong generalization and achieved an impressive accuracy of 99.06%. Despite the increased complexity due to the larger kernel size, the computing time required for training remained an effective 5 minutes. This combination of high precision and efficient computing power highlights the effectiveness of the architecture and the selected parameters to obtain a balanced and efficient CNN model.

**3.3.5 Advanced Architecture: Dual Convolutional Layers with Max Pooling**

We used 64 kernels of size 4x4 in the first convolutional layer, followed by a maximum pooling layer of pool size 3x3. Then in the second convolution layer we used 32 kernels of size 3x3, combined with another maximum pooling layer of pool size 2x2. This configuration of the produced impressive results with 99.70% training accuracy and demonstrated the model's ability to learn complex patterns from the training data set. Despite the increased depth and complexity due to the additional convolution layer and maximum pooling operations, the computing time required for training remained efficient at 5.5 minutes.

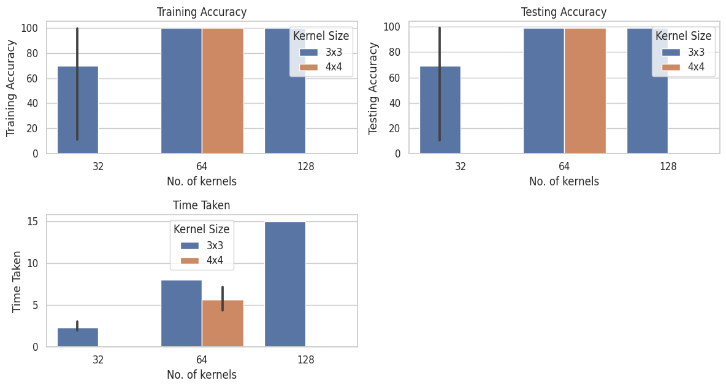
**Pseudo code:**

* **Initialize a sequential model, add a Conv2D layer with 64 filters, 4x4 kernel, ReLU activation, same padding.**
* **Perform max pooling with a 3x3 window size.**
* **Add another 2D convolutional layer with 32 filters using a 3x3 kernel and ReLU activation.**
* **Employ same padding to retain spatial information.**
* **Conduct max pooling with a 2x2 window size.**
* **Flatten the output to 1D tensor**
* **Add dense layers of 100, 50, 25 neurons respectively with ReLU activation.**
* **Final dense layer with 10 neurons using softmax activation (for 10-class classification).**

In the optimization of our convolutional neural network (CNN) architecture, we incorporated two convolutional layers, each followed by a max-pooling layer, to enhance feature extraction and spatial down-sampling. The first convolutional layer employed 64 kernels of size 4x4, with subsequent max-pooling using a pool size of 3x3. The second convolutional layer featured 32 kernels of size 3x3, followed by another max-pooling layer with a pool size of 2x2.

Despite the increased depth and complexity introduced by the additional convolutional layer and max-pooling operations, the computational time for training remained reasonably efficient at 8.2 minutes. This amalgamation of accuracy, architectural sophistication, and computational efficiency exemplifies the effectiveness of the chosen CNN configuration in achieving superior performance.

In the refinement of our convolutional neural network (CNN) architecture, we integrated two convolutional layers, each complemented by a max-pooling layer to enhance feature extraction and spatial down-sampling. The first convolutional layer featured 64 kernels of size 4x4, with subsequent max-pooling utilizing a pool size of 3x3. The second convolutional layer incorporated 32 kernels of size 3x3, followed by another max-pooling layer with a pool size of 2x2.



***Figure 7***

**Pseudo code:**

**# Additional Convolutional Layer with Padding**

**keras.layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', padding='valid'),**

This configuration resulted in remarkable performance, with a training accuracy of 99.72%, affirming the model's capacity to discern intricate patterns within the training dataset. During testing, the model maintained a high accuracy level of 99.06%, showcasing robust generalization to previously unseen data. Notably, the computational time for training remained efficient at 4 minutes, highlighting the effectiveness of the chosen CNN configuration in achieving superior accuracy and computational efficiency

Below table1.3 gives a brief view on different no. of kernals, kernel size and activation function.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. of kernals | Kernal Size | Activation function | Training accuracy | Testing accuracy | Time taken |
| 32 | 3x3 | Sigmoid | 11.40% | 10.5% | 2 minutes |
| 32 | 3x3 | Softmax | 98.78% | 98.4% | 2 minutes |
| 32 | 3x3 | ReLU | 99.90% | 98.84% | 3 minutes |
| 64 | 3x3 | ReLU | 99.95% | 98.95% | 8 minutes |
| 128 | \  3x3 | ReLU | 99.93% | 98.95% | 15 minutes |
| 64 | 4x4 | ReLU | 99.94% | 99.06% | 5 minutes |
| 64 (1st layer), 32 (2nd layer) | 4x4 (1st layer), 3x3 (2nd layer) | ReLU | 99.70% | 99.08% | 5.5 minutes |
| 64 (1st layer), 32 (2nd layer) | 4x4 (1st layer), 3x3 (2nd layer) | ReLU  (with padding = same) | 99.72% | 99.11% | 8.2 minutes |
| 64 (1st layer), 32 (2nd layer) | 4x4 (1st layer), 3x3 (2nd layer) | ReLU  (with padding = valid) | 99.72% | 99.06% | 4 minutes |

***Table 1.3***

***4. Result Discussion***

Here, we compare the performance of an Artificial Neural Network (ANN) and a Convolutional Neural Network (CNN) used for image categorization in table 1.4. These architectures are distinguished by the table that follows, which provides information on their configurations, optimizers, accuracy rates, computing efficiency, and other aspects. Gaining knowledge of these architectures' subtleties can help determine which machine learning applications are most suited for them.

|  |  |
| --- | --- |
| **ANN** | **CNN** |
| **Performance Comparison** | |
| The ANN reaches its peak accuracy using three hidden layers. The first two layers employ Parametric ReLU, and the final layer uses Softmax activation. It's optimized with Adadelta (learning rate: 1.0). | The CNN's highest accuracy is achieved via a setup involving a convolutional layer followed by two max-pooling layers. The initial layer holds 64 4x4 kernels with a 3x3 pooling size, followed by a 32-kernel layer with 2x2 pooling. 'Same' padding is used, activating ReLU, and Adadelta optimizes it. |
| **Training Accuracy** | |
| 99.48% | 99.72% |
| **Testing Accuracy** | |
| 98.07% | 99.11% |
| **Computational Time** | |
| Time taken by the model to train is 30.2 seconds. | Time taken by the model to train is 4 minutes and14 seconds. |
| **Confusion matrix** | |
| [[584 0 0 0 0 0 0 0 1 2]  [ 0 628 1 0 0 0 1 0 0 0]  [ 1 1 597 1 0 0 0 0 0 0]  [ 0 0 0 625 0 0 0 0 1 1]  [ 0 3 0 0 587 0 0 1 0 4]  [ 0 1 0 8 0 533 4 0 3 0]  [ 2 0 0 0 0 0 569 0 0 0]  [ 2 3 0 0 2 0 0 658 0 3]  [ 3 0 0 0 0 1 0 0 593 0]  [ 1 0 0 3 4 0 0 0 2 566]] | [[578 0 4 0 0 0 1 1 1 2]  [ 0 623 1 1 0 0 3 1 1 0]  [ 2 2 593 0 0 0 0 1 2 0]  [ 1 0 3 614 0 3 0 0 6 0]  [ 1 4 1 0 580 0 0 2 0 7]  [ 0 0 2 9 0 523 9 0 2 4]  [ 1 0 0 0 0 0 569 0 1 0]  [ 0 2 0 1 1 0 0 661 0 3]  [ 0 2 1 1 2 0 1 1 587 2]  [ 0 2 1 2 5 4 0 3 1 558]] |
| **HEAT MAP** | |
| **Trained with 10 epochs** | **Trained with 10 epochs** |

***Table 1.4***

From our results, the merits and demerits of ANN and CNN is observed. Firstly we can discuss about the merits of both ANN and CNN

**4.1.1. Merits of ANN**

1. Parametric ReLU Activation: The use of Parametric ReLU activation allows the network to learn the optimal values for the alpha parameters, introducing flexibility in the activation function.
2. Sequential Model: The Sequential API in Keras simplifies the construction of neural networks, making it easy to define a linear stack of layers.
3. Custom Optimizer: The code specifies a custom Adadelta optimizer with a learning rate of 1.0, providing control over the optimization process.
4. Sparse Categorical Crossentropy: The choice of sparse categorical crossentropy as the loss function is suitable for integer-encoded target labels, common in classification tasks.
5. Training Routine: The model is trained on the flattened input data for 10 epochs, facilitating quick experimentation and model evaluation

**4.1.2 Merits of CNN**

1. Convolutional Neural Network (CNN): The use of Conv2D and MaxPooling2D layers allows the model to capture spatial hierarchies and patterns in image data, making it suitable for image classification tasks.
2. Padding and Pooling: The inclusion of padding='same' in convolutional layers and max-pooling layers helps maintain spatial dimensions and reduce information loss at the borders.
3. Sequential Model: The Sequential API simplifies the model architecture, making it easy to understand and modify. It follows a sequential flow of layers, enhancing readability.
4. Custom Optimizer: The code employs a custom Adadelta optimizer with a learning rate of 1.0, offering control over the optimization process during training.
5. Sparse Categorical Crossentropy: The choice of sparse categorical crossentropy as the loss function is suitable for integer-encoded target labels, common in classification tasks.

**4.1.3 Demerits of ANN**

1. Parametric ReLU Overhead: Using Parametric ReLU may introduce additional parameters to learn, potentially leading to longer training times and increased memory requirements.
2. Fixed Learning Rate: The Adadelta optimizer is configured with a fixed learning rate of 1.0, which might not be optimal for all datasets or converge efficiently in certain cases.
3. Limited Model Complexity: The model architecture is relatively simple with only dense layers and lacks convolutional or recurrent layers, which may limit its ability to capture complex hierarchical patterns.

**4.1.3 Demerits of CNN**

1. Fixed Learning Rate: The Adadelta optimizer is configured with a fixed learning rate of 1.0, which might not be optimal for all datasets or converge efficiently in certain cases.
2. Limited Comments: The code lacks detailed comments, making it less clear for someone else to understand the purpose and choices made during model construction and training.
3. No Validation Set: The training routine does not include a separate validation set for monitoring model performance on unseen data during training.
4. Limited Model Complexity: The model architecture is relatively simple with only dense layers, which may limit its ability to capture complex hierarchical patterns in certain datasets.
5. Epochs and Data: The number of epochs (10) and the training data (X\_train, y\_train) are fixed, and adapting these parameters may be necessary for optimal model training, depending on the dataset and task.

***5. Conclusion***

By comparing convolutional neural networks (CNN) and artificial neural networks (ANN) for handwritten digit recognition, the study examined their strengths and limitations. Both CNNs and ANNs offer clear advantages: CNNs that can capture spatial patterns in images showed strength in recognizing handwritten digits.

In contrast, ANNs demonstrated adaptability to complicated data relationships but struggled with complex hierarchical patterns in handwritten digits. The research yielded remarkable insights into activation functions and model architecture. Parametric activation of ReLU in ANN enabled high accuracy, but at the cost of higher computational cost. In contrast, different kernel sizes and activation functions in CNNs contributed to differentiated accuracy and training time.

However, both models faced challenges due to fixed learning rates and model complexity limitations that prevented optimal performance in certain scenarios. While CNNs essentially excelled at capturing hierarchies and spatial patterns in handwritten digits, ANNs demonstrated their adaptability to different data relationships. However, both models had limitations of fixed learning rates, limited model complexity, and insufficient comments in the code, suggesting that further investigation and refinement of the model is required to achieve higher accuracy and versatility in digit recognition tasks handwritten.

***6.References***

1. https://www.fortunebusinessinsights.com/deep-learning-market-107801
2. Saeed Al Mansoori, “*Intelligent Handwritten Digit Recognition using Artificial Neural Network*,” ISSN : 2248-9622, Vol. 5, Issue 5, ( Part -3) May 2015, DOI: 10.13140/RG.2.1.2466.0649
3. Vijayalaxmi R Rudraswamimath, Bhavanishankar K, “*Handwritten Digit Recognition using CNN*”, Volume 4, Issue 6, June – 2019
4. Drishti Beohar, Akhtar Rasool, “*Handwritten Digit Recognition of MNIST dataset using Deep Learning state-of-the-art Artificial Neural Network (ANN) and Convolutional Neural Network (CNN)”,* DOI: 10.1109/ESCI50559.2021.939687AISSMS Institute of Information Technology, Pune, India. Mar 5-7, 2021
5. Ritik Dixit, Rishika Kushwah, Samay Pashine, *”Handwritten Digit Recognition using Machine and Deep Learning Algorithms”,*
6. Tsehay Admassu Assegie, *“Handwritten digits recognition with decision tree classification: a machine learning approach”,*
7. B M Vinjit, Mohit Kumar Bhojak, Sujit Kumar and Gitanjali Nikam, *“Implementation of Handwritten Digit Recognizer using CNN”*
8. Kh Tohidul Islam, Ghulam Mujtaba, Dr. Ram Gopal Raj, Henry Friday Nweke, *“Handwritten Digits Recognition with Artificial Neural Network”*
9. Pronab Ghosh, Atqiya Abida Anjum, Asif Karim, Masum Shah Junayed, Md. Zahid Hasan, Khan Md. Hasib, Al Nahian Bin Emran, *“A Comparative Study of Different Deep Learning Model for Recognition of Handwriting Digits”*
10. <https://paperswithcode.com/dataset/mnist>
11. *“I.A Biological Basis of Artificial Neural Networks”,* [Encyclopedia of Physical Science and Technology (Third Edition)](https://www.sciencedirect.com/referencework/9780122274107/encyclopedia-of-physical-science-and-technology), 2003
12. [Unraveling the Power of Neural Networks.docx - Title: Unraveling the Power of Neural Networks Introduction Neural networks inspired by the human | Course Hero](https://www.coursehero.com/file/216601805/Unraveling-the-Power-of-Neural-Networksdocx/)
13. [www.researchgate.net/figure/Workflow-diagram-of-the-artificial-neural-network-algorithm-developed-by-Lancashire-et\_fig3\_323312622](http://www.researchgate.net/figure/Workflow-diagram-of-the-artificial-neural-network-algorithm-developed-by-Lancashire-et_fig3_323312622)
14. [Introduction to Convolution Neural Network - GeeksforGeeks](https://origin.geeksforgeeks.org/introduction-convolution-neural-network/?ref=lbp)

**Make sure to remove all placeholder and explanatory text from the template when you add your own text. This text should not be here in the final version!**