A Comparative Study on Handwritten Digit Recognizer Using CNN and ANN

M.V.Dharshini

B.Tech Artificial Intelligence and Data Science,

Sri Eshwar College of Engineering,

Coimbatore, TamilNadu-641202

dharshini.mv2022ai-ds@sece.ac.in

R.Mahima Tejeshwini

B.Tech Artificial Intelligence and Data Science,

Sri Eshwar College of Engineering,

Coimbatore, TamilNadu-641202

mahimatejeshwini.r2022ai-ds@sece.ac.in

P.Mageshwaran

B.Tech Artificial Intelligence and Data Science,

Sri Eshwar College of Engineering,

Coimbatore, TamilNadu-641202

mageshwaran.p2022ai-ds@sece.ac.in

We extend our sincere gratitude to Dr. S. Sumathi, M.E., Ph.D., Associate Professor in the Department of Artificial Intelligence and Data Science at Sri Eshwar College of Engineering, for her invaluable guidance, expertise, and unwavering support throughout the research process. Her insightful feedback and encouragement greatly enriched the quality of this work.

ABSTRACT

Handwriting recognition, whether offline or online, poses intricate challenges in accurately interpreting characters due to variations and distortions influenced by diverse community styles of handwriting. In the machine learning domain, Convolutional Neural Networks (CNNs) and Artificial Neural Networks (ANNs) play pivotal roles. CNNs, with their translation invariance and spatial hierarchy capture, excel in image recognition and computer vision. Conversely, ANNs showcase versatility in approximating continuous functions and adaptability for generalization to new data, demonstrating proficiency in capturing intricate relationships within complex datasets.

This research paper conducts a comparative analysis of CNNs and ANNs for handwritten digit recognition, aiming to explore their unique strengths, weaknesses, and applicability to this specific task. The objective is to contribute valuable insights that guide the selection of the most effective model for digit recognition in real-world scenarios. By delving into the nuanced aspects of each model, the study seeks to provide practical considerations for decision-makers navigating the complexities of handwritten digit recognition

KEYWORD

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

INTRODUCTION

The advent of the COVID-19 pandemic has underscored the pivotal role of digitalization in both educational and business domains, prompting organizations to leverage digital tools to navigate the challenges presented by the global health crisis. Features like Auto Classification and Text Reader have proven indispensable in optimizing document management processes, fostering heightened efficiency and accessibility amid the dynamic landscape of the pandemic.

Within the realm of image classification, handwritten digit recognition poses a classic challenge, demanding an effective categorization of digits into corresponding labels (0-9). In addressing this challenge, neural network models, notably Convolutional Neural Networks (CNNs), emerge as potent tools, demonstrating the ability to tackle classification tasks with efficacy.

As organizations increasingly embrace machine learning approaches for handwritten digit recognition, the task involves training machines with past datasets and known class labels. Diverse machine learning models, including artificial neural networks (ANNs), CNNs, decision tree classifiers, and improved chain code histogram features, are applied to this complex problem. However, despite extensive research, a perfect approach for recognizing handwritten digits remains elusive.

Handwriting recognition, whether offline or online, presents a nuanced perspective on the challenges inherent in accurately interpreting and distinguishing characters. The variation and distortion within handwritten character sets, influenced by different community styles of handwriting, contribute to the complexity of this recognition task.

In the realm of machine learning, both CNNs and ANNs play crucial roles. CNNs, distinguished by their translation invariance and adeptness in capturing spatial hierarchies, excel in tasks like image recognition and computer vision. On the other hand, ANNs, with their versatility in approximating continuous functions and adaptability for generalization to new data, demonstrate proficiency in capturing intricate relationships within complex datasets.

This research paper seeks to delve into the comparative analysis of CNNs and ANNs for handwritten digit recognition. By exploring their individual strengths, weaknesses, and applicability to this specific task, the aim is to contribute insights that guide the selection of the most effective model for digit recognition in a real-world context.

Artificial Neural Networks (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, 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. Each connection between neurons has an associated weight, and the network learns by adjusting these weights based on the input data. 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.

ANNs excel at tasks involving pattern recognition, classification, regression, and decision-making. They are versatile and can be applied to a wide range of applications, from image and speech recognition to natural language processing. Deep Learning, a subset of machine learning, often involves deep neural networks with multiple hidden layers, allowing for the extraction of intricate features from complex data.

METHODOLOGY OF ANN

Very Simple neural network with no hidden layers:-

Introduction:

Leveraging MNIST Dataset, In this research project, we utilized the MNIST dataset, a well-known collection of 28x28 pixel grayscale images depicting handwritten digits along with their respective labels. The dataset serves as a benchmark for various machine learning tasks.

Library Imports and Dataset Inspection

To kick off our exploration, we imported essential libraries, such as TensorFlow, Keras, and Matplotlib. With these tools, we loaded the MNIST dataset and conducted an initial examination of its shape and content. This step provided crucial insights into the underlying structure of our input data.

Data Visualization with Matplotlib:-

Matplotlib was employed to visually represent the pixel values of the first training image through a matrix plot. Simultaneously, we cross-verified the associated label to ensure alignment between the input data and corresponding labels, a critical aspect in supervised learning.

Normalization of Pixel Values:-

Recognizing the importance of preparing data for neural network training, we normalized the pixel values to a range between 0 and 1. This preprocessing step is pivotal for enhancing the stability and efficiency of the training process, contributing to the convergence of optimization algorithms.

Flattening Images for Neural Network Input:-

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.

Neural Network Model Construction:-

Using the Keras library, we constructed a neural network model. The model featured a sequential structure with a single dense layer, consisting of 10 units to represent the possible digits (0 through 9). The sigmoid activation function was applied to the dense layer. Subsequently, the model was compiled, incorporating the Adam optimizer, sparse categorical crossentropy loss function, and accuracy as the evaluation metric.

Training and Evaluation:-

The model underwent training, mapping input images to their corresponding digit labels over multiple epochs. Evaluation on the test data ensued, providing insights into the model's performance. Metrics such as loss and accuracy were employed to quantify the effectiveness of the trained model.

Conversion of Predictions and Confusion Matrix Analysis:-

Predicted probabilities were converted into class labels, facilitating a more interpretable understanding of model outputs. TensorFlow's confusion_matrix function was employed to compute a confusion matrix, a valuable tool for assessing classification model performance by highlighting counts of true positives, true negatives, false positives, and false negatives for each class.

Very Simple neural network with hidden layers:-

Introduction to Seaborn and Figure Initialization:-

To make our model evaluations clearer, we used Seaborn, a handy tool for showing data visually. We kicked things off by creating a picture with specific dimensions (10 inches by 7 inches). This step set the stage for a useful heatmap, a visual representation of our model's performance using a confusion matrix.

Seaborn Heatmap Generation and Annotation:-

Seaborn's heatmap feature played a key role in turning our confusion matrix into a visually understandable chart. We added numbers to the chart using the annot=True setting, making it more precise. The fmt='d' setting ensured these numbers appeared as whole numbers, making the heatmap a helpful tool for understanding.

Neural Network Model Definition and Compilation:-

Shifting from visuals to the model's structure, we designed a new neural network with two layers. The first hidden layer, with 100 units and activated by ReLU, helped the model recognize patterns better. The next layer, with 10 units and using the sigmoid activation function, aimed at making the model good at classifying things. After setting up the model, we put it all together by choosing the right tools—like the Adam optimizer, sparse categorical crossentropy loss function, and accuracy as our main measure of success.

Model Training and Evaluation:-

With our model ready, we put it through a training session using flattened data from our training set. Checking how well it did on our test data gave us important insights into whether our model could understand new patterns it hadn't seen before.

Predictions, Confusion Matrix, and Visual Enhancement:-

We let our model make predictions on our test data and turned those guesses into categories. This led to a new confusion matrix, showing us how our model was doing after learning. To make all of this easier to understand, we created a new picture and used Seaborn to turn our confusion matrix into a more readable chart. Adding labels made it clear which categories we were talking about.

Conclusion:-

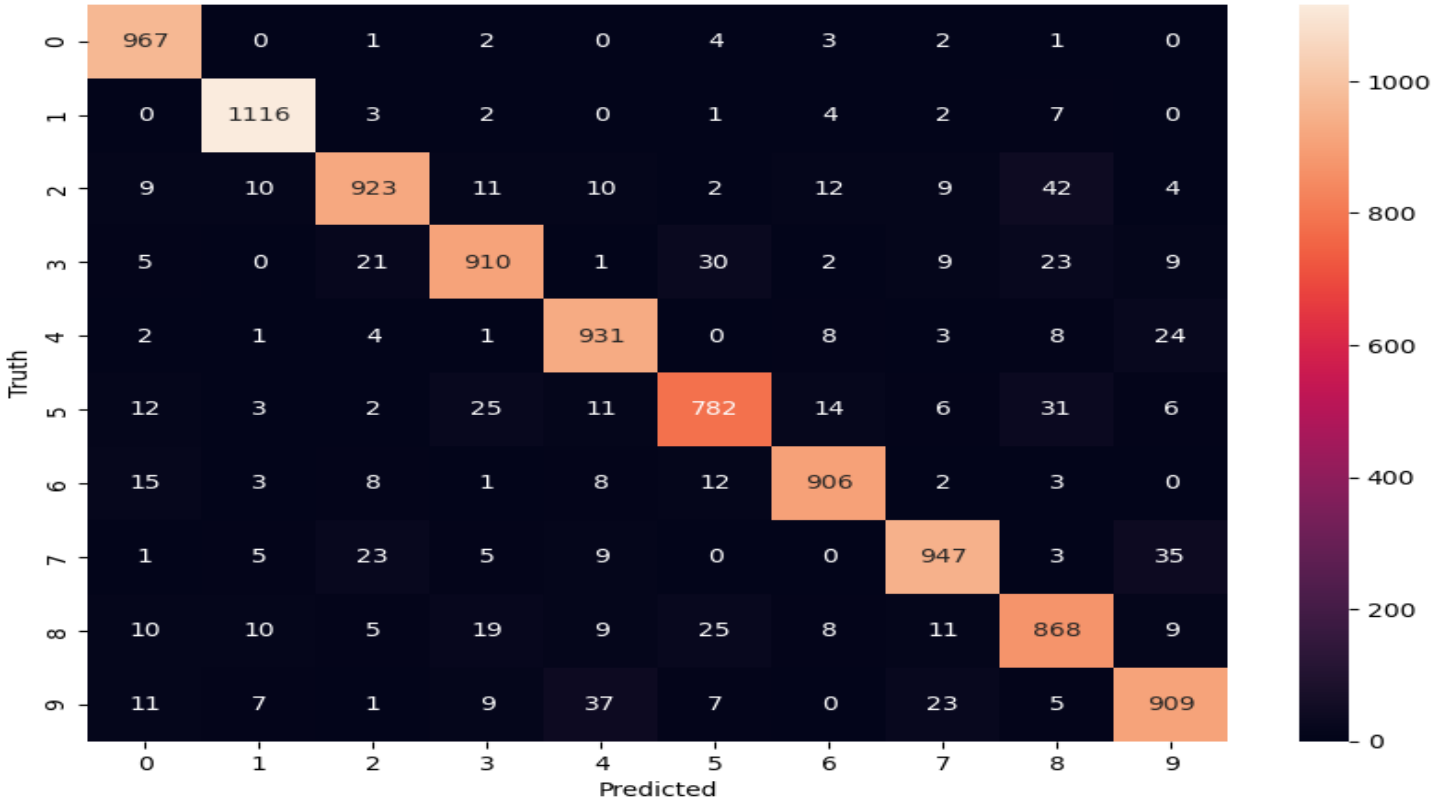
The experimentation with Artificial Neural Networks (ANNs) for digit recognition reveals a noteworthy contrast in performance based on the presence of hidden layers. Without a hidden layer, the ANN achieves an accuracy of 95.72%, while the introduction of a hidden layer significantly enhances performance, yielding an accuracy of 97.49%. This outcome underscores the pivotal role of hidden layers in capturing intricate features and patterns within digit images, leading to a substantial improvement in recognition accuracy. The findings emphasize the importance of thoughtful architecture design in ANNs for digit recognition tasks, with the incorporation of hidden layers proving instrumental in achieving higher accuracy rates.

Differences in with hidden layer without hidden layer:

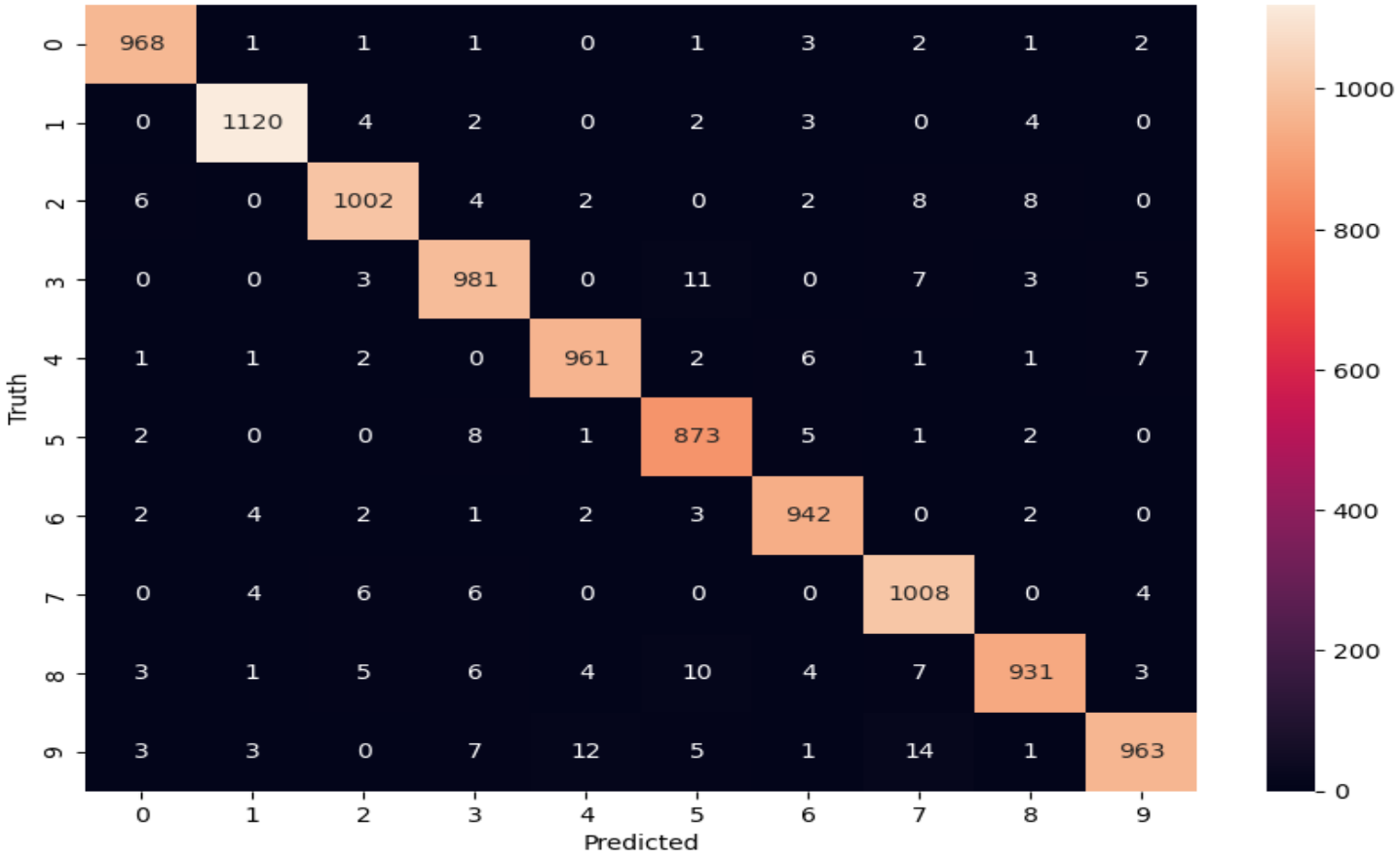
	With	Without
Neural Network Architecture	Uses a neural network with two dense layers (100 units with ReLU activation and 10 units with sigmoid activation).	Uses a simpler neural network with only one dense layer (10 units with sigmoid activation).
Visualization	Plots two heatmaps of confusion matrices using Seaborn, before and after training the neural network.	Displays an image using Matplotlib (plt.matshow) and prints an AxesImage object, but does not include visualization of the confusion matrix.
Model Predictions	After training, predicts labels and computes a confusion matrix for the updated model.	Predicts labels and computes a confusion matrix for the initial model.
Activation Functions	Uses ReLU activation for the hidden layer and sigmoid activation for the output layer.	Uses sigmoid activation for the only layer in the model.

Confusion Matrix

Without Hidden Layer:-



With Hidden Layer:-



Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a class of deep neural networks designed for processing and analyzing visual data, making them highly effective for tasks like image recognition, object detection, and image classification. The core concept behind CNNs is to mimic the visual processing that occurs in the human brain.

CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The key operations within a CNN are convolution and pooling. In the convolutional layers, the network applies filters (also called kernels) to the input data to detect features like edges, textures, or patterns. These filters slide over the input, and their weights are learned during the training process, allowing the network to automatically extract meaningful features.

Pooling layers follow convolutional layers, reducing the spatial dimensions of the input data and retaining essential information. This helps in reducing computation while maintaining the network's ability to capture important features. The output from these layers is then flattened and fed into fully connected layers, enabling the network to make predictions or classifications based on the learned features.

What sets CNNs apart is their ability to capture hierarchical representations of features, starting from simple edges and shapes in the early layers to complex and abstract features in deeper layers. This hierarchical feature extraction makes CNNs robust to variations in scale, orientation, and position, contributing to their success in image-related tasks. The training of CNNs involves optimizing the network's parameters, such as weights and biases, through backpropagation and gradient descent, ensuring the network learns to recognize relevant patterns in the input data.

METHODOLOGY OF CNN

Introduction:-

In this research, we harness the potential of Convolutional Neural Networks (CNNs) in conjunction with the MNIST dataset. The MNIST dataset, a collection of 28x28 pixel grayscale images representing handwritten digits.

Library Imports and Dataset Inspection:-

We're importing essential libraries for our project, including TensorFlow for building and training neural networks, Matplotlib for data visualization, NumPy for numerical computations, and Seaborn for advanced data visualization and analysis.

Visualizing Data with Matplotlib

In the Matplotlib visualization section, we utilize the versatile Matplotlib library to transform raw pixel values and numerical data into informative graphical representations. These visual insights provide a deeper understanding of the MNIST dataset, revealing the distinctive features of handwritten digits

Normalization

The code begins by loading the MNIST dataset, containing grayscale images of handwritten digits, into variables `X_train` and `X_test`. To ensure data consistency, pixel values are converted to 32-bit floating-point numbers and then normalized by dividing by 255, scaling them within the [0, 1] range. Furthermore, the data's shape is adjusted using `np.expand_dims` to match the CNN model's input format, enabling successful model training and evaluation.

Transforming Images for CNN Input

The "Transforming Images for CNN Input" stage is pivotal for adapting the MNIST dataset to the Convolutional Neural Network's (CNN) expectations. It commences by ensuring that the images are 28x28 pixels in grayscale, aligning them with the CNN's requisites.Next, the code reshapes the 2D image arrays into 3D, introducing a single channel per image. This transformation is crucial as it prepares the data to be seamlessly ingested by the convolutional layers within the CNN. This optimization streamlines the model's performance and enhances its ability to capture intricate image features.

Constructing the Convolutional Neural Network

In Keras, we use its tools to create a Convolutional Neural Network with a sequential structure. The model includes convolutional and pooling layers for extracting image features. It concludes with a fully connected dense layer containing 10 units to represent the digits from 0 to 9. We use the softmax activation function for clarity in classification. The model is compiled with the Adam optimizer, categorical crossentropy loss function, and accuracy as the metric.

Training and Validation:-

During this phase, the CNN model is trained on the preprocessed training data over 50 epochs, with 30% of the data set aside for validation. EarlyStopping and ModelCheckpoint callbacks are employed to enhance the training process. EarlyStopping monitors validation accuracy and halts training if it stops improving for four consecutive epochs with a minimum change of 0.01.

ModelCheckpoint saves the best model based on validation accuracy. The model's parameters are iteratively adjusted to minimize categorical cross-entropy loss using the Adam optimization algorithm. This process fine-tunes the model's parameters, allowing it to learn from the training data and continuously evaluate its performance on unseen validation data. The end goal is to achieve a well-trained model capable of accurately recognizing handwritten digits.

Decoding Predictions and Unraveling Confusion:-

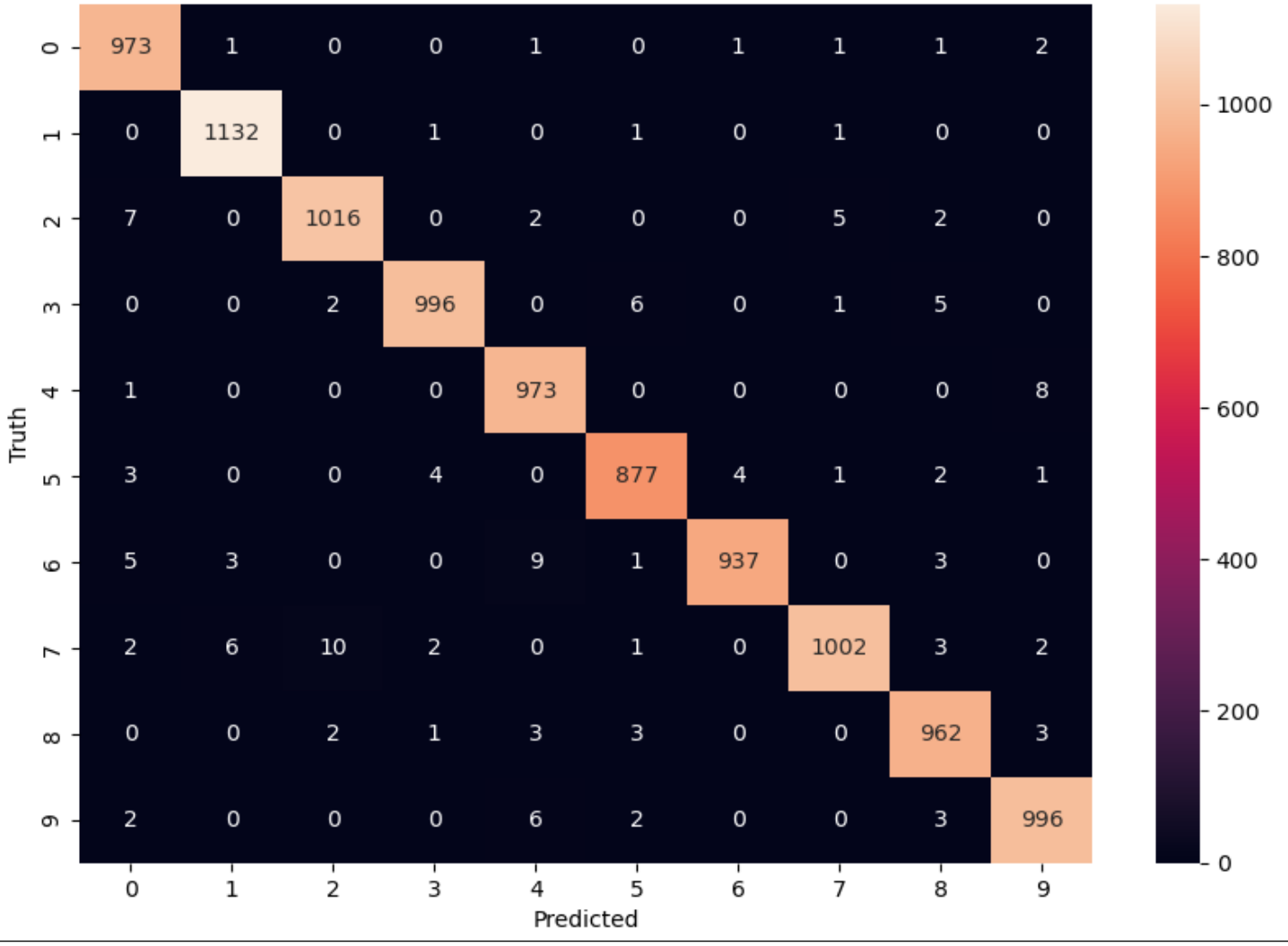
The model's predictions are converted from probabilities into interpretable class labels. TensorFlow's confusion_matrix function is employed to construct a confusion matrix. It provides valuable insights by categorizing the model's predictions into true positives, true negatives, false positives, and false negatives for each class. This comprehensive analysis helps in assessing how effectively the model is classifying different digits and identifying areas where it may be making errors.

Additionally, a heatmap visualization of the confusion matrix is generated using Matplotlib and Seaborn, allowing for a more intuitive and visual assessment of the model's performance. This phase serves as a critical step in evaluating the model's accuracy and its ability to correctly classify handwritten digits.

Conclusion :-

In conclusion, our research encompasses a comprehensive journey, including data exploration, thorough data preparation, building the convolutional model, rigorous training, validation, and performance assessment. This holistic approach, empowered by Convolutional Neural Networks, demonstrates its effectiveness in recognizing handwritten digits, underscoring its role in image recognition and classification.

Confusion Matrix (CNN)



OVERALL CONCLUSION

This research delves into the optimization of digit recognition models through a meticulous exploration of two prominent neural network architectures: Convolutional Neural Networks (CNN) and Artificial Neural Networks (ANN). The achieved results underscore significant accomplishments in accuracy metrics, shedding light on the distinct strengths of each model in the context of digit recognition.

The CNN model demonstrated notable efficacy by achieving a commendable accuracy of 98.30% on the training dataset. Even more impressively, its performance on the test dataset soared to an accuracy of 98.64%. This remarkable result speaks to the CNN's inherent capacity to generalize well to previously unseen data. The model's proficiency in automatically extracting hierarchical features and discerning spatial patterns from digit images plays a pivotal role in its exceptional performance.

Conversely, the ANN model exhibited commendable accuracy, registering an impressive 98.42% on the training dataset. However, the model faced a slightly more pronounced challenge in generalizing to new data, evidenced by its test dataset accuracy of 97.49%. This nuanced discrepancy between training and test accuracies suggests a potential limitation in the ANN's adaptability to the broader variability inherent in unseen datasets.

In summation, this research underscores the superiority of the CNN architecture for digit recognition tasks, emphasizing its robustness in capturing intricate features within digit images and its ability to generalize effectively. The findings underscore the critical importance of selecting a model architecture aligned with the specific intricacies of the task at hand. The CNN emerges as the more potent choice for digit recognition, providing a solid foundation for future endeavors in the realm of image classification and pattern recognition.

Drawbacks of CNN over ANN in Digit Recognition

Limited Applicability: CNNs are highly specialized for image processing tasks, making them less versatile than ANNs. ANNs can be used for a wide range of tasks, including text and structured data analysis, whereas CNNs are primarily designed for visual data.

Data Dependency: CNNs require a large amount of labelled image data for training, which may not be readily available for all applications. ANNs can sometimes work with smaller datasets and are more adaptable to various data types.

Complexity: CNNs can be complex to design and fine-tune. The architecture of CNNs, with multiple layers and hyper parameters, can require extensive experimentation to achieve optimal performance.

Computational Resources: CNNs demand significant computational resources, including powerful GPUs or TPUs for training. ANNs, especially shallow ones, are less computationally intensive.

Over fitting: CNNs can still be prone to over fitting, especially when working with limited data. Careful regularization techniques are required to mitigate this issue.

Interpretability: CNNs are often seen as "black boxes" because it's challenging to interpret how they make decisions. ANNs can be more interpretable for some tasks, making them preferable in cases where understanding the model's reasoning is crucial.

Complexity in Pre-processing: CNNs may require intricate pre-processing steps, such as data augmentation, to enhance their performance. ANNs are typically more forgiving in terms of data pre-processing.

Difficulty in Transfer Learning: Transfer learning, a technique where pre-trained models are fine-tuned for new tasks, can be more challenging with CNNs compared to ANNs, particularly when dealing with domains vastly different from the source data.

Limited Robustness to Noise: CNNs are sensitive to noise and perturbations in images. Small changes in the input image can result in significant variations in the output. ANNs can be more robust to noisy input data.

Complexity in Deployment: Deploying CNN models can be more complex, as they often require specialized infrastructure and libraries to run efficiently. ANNs are typically easier to deploy.

When and Where to use CNN

Convolutional Neural Networks (CNNs) are particularly well-suited for tasks that involve image and spatial data analysis. Here are some common scenarios and applications where CNNs are used:

Image Classification: CNNs are widely used for classifying images into various categories. Applications include recognizing objects, animals, plants, and more. CNNs are employed in platforms like social media to automatically tag and categorize uploaded images.

Object Detection: CNNs can identify and locate objects within images. This is useful in self-driving cars for detecting pedestrians and other vehicles, in security systems for recognizing intruders, and in medical imaging for detecting anomalies.

Facial Recognition: CNNs are used for recognizing and verifying faces, enabling applications like unlocking smartphones, identifying criminals in surveillance footage, and personalizing marketing messages.

Image Segmentation: CNNs can segment an image into different regions or objects, which is valuable in medical imaging (segmenting tumors), autonomous driving (identifying road lanes), and more.

Artificial Intelligence in Games: CNNs are employed in game development for object recognition and decision-making by non-player characters (NPCs).

Video Analysis: CNNs are used to process video data, identifying actions and objects in real time. This is applied in video surveillance, content recommendations, and autonomous vehicles.

Natural Language Processing (NLP): In NLP tasks involving text classification of documents containing images (e.g., invoices, legal documents), CNNs can be used to process and extract information from text within images.

Augmented Reality (AR): In AR applications, CNNs help recognize objects or features in the physical environment to overlay virtual content on the real world.

Quality Control: CNNs can be employed in manufacturing and production lines to inspect and ensure the quality of products. They can detect defects, impurities, or anomalies in items like electronic components or food products.

Emotion Recognition: CNNs are used for emotion recognition from facial expressions, which has applications in market research, mental health, and user experience design.

FUTURE WORK

The research suggests future work in optimizing digit recognition models through:-

Architecture Refinement:- Enhance CNN and ANN architectures for better performance.

Hyperparameter Tuning:- Systematically optimize parameters for improved accuracy.

Data Augmentation:- Explore advanced techniques to augment training datasets.

Ensemble Methods:- Investigate combining predictions from multiple models.

Transfer Learning:-Assess the benefits of leveraging pre-trained models for digit recognition.

Real-world Applications:- Extend applications to document processing and scene character recognition.

Robustness and Interpretability:-Explore model robustness and interpretability, crucial for real-world deployment.

Scalability and Efficiency:-Evaluate scalability and efficiency, optimizing model size without compromising accuracy.

REFERENCES

1.A Comparative Study on Handwritten Digit Recognizer using Machine Learning Technique by R.Kanniga Devi , G.Elizabeth Rani (published on 2019).

2.Handwritten Digit Recognition using Machine and Deep Learning Algorithms by Samay Pashine , Ritik Dixit , Rishika Kushwah (published on 2020).

3.Hybrid CNN-SVM Classifier for Handwritten Digit Recognition by Savita Ahlawat , Amit Choudhary (published on 2020).

Project Reference:-

Mageshwaran17. (2023). Digit Recognizer.
https://github.com/Mageshwaran17/Digit_Recognizer.git