Recognition of Handwritten Digits using Convolutional Neural Network

Akhil Kukkadapu[1], Mr. Mohd Faisal[2], Adharsh Nandigama[3], K. Tharun Chary[4]
[1,3,4]*Department of Artificial Intelligence and Machine Learning, Sphoorthy Engineering College, Hyderabad, India*
[2]*Assistant Professor, Department of CSE (AI & ML) Sphoorthy Engineering College, Hyderabad, India*

***Abstract*— Handwritten digit recognition is a critical task with widespread applications and ranging from automated postal sorting to digitizing historical documents. In this paper and we propose a robust approach for handwritten digit recognition leveraging Convolutional Neural Networks (CNNs). CNNs have demonstrated exceptional performance in image related tasks and making them well suited for the intricate patterns present in handwritten digits. Our methodology involves the construction of a deep neural network architecture specifically designed for the complexities of handwritten digit recognition. The proposed model employs multiple convolutional layers to capture hierarchical features of the input images and followed by pooling layers for spatial dimension reduction. Additionally, and fully connected layers are incorporated to enable global learning and feature integration.**

Keywords—handwritten digit recognition, deep learning, convolutional neural network

I. INTRODUCTION

Handwritten digit recognition plays a pivotal role in the realm of pattern recognition and computer vision and finding applications in various domains such as postal services and document digitization and automated data entry. As technology advances and the need for accurate and efficient methods to decipher handwritten digits becomes increasingly crucial. Convolutional Neural Networks (CNNs) have emerged as powerful tools for image related tasks and demonstrating remarkable success in tasks such as object detection and image classification.
In this paper and we present a comprehensive exploration of the application of CNNs for handwritten digit recognition and aiming to enhance the start of the art methodologies. Despite the success of traditional methods and handwritten digit recognition remains a challenging problem due to variations in writing styles and diverse datasets and image distortions. CNNs inspired by the visual processing in the human brain and have shown unparalleled capabilities in capturing hierarchical features from images. The hierarchical and spatially invariant features extracted by CNNs make them well suited for the intricate patterns inherent in handwritten digits.
In this introductory section and we outline the motivation behind our research and provide an overview of existing methodologies and highlight the significance of our proposed CNN based approach. We anticipate that our findings will contribute to the ongoing discourse in the field of image recognition and providing valuable insights for researchers and practitioners working on handwritten character recognition and paving the way for enhanced solutions in real world applications.
This paper introduces the recognition of handwritten digits (0 to 9) from the well-known MNIST dataset using the TensorFlow framework (library) and Python as the programming language, along with its associated libraries. Upon inputting a specific digit, the system is designed to recognize and display the results with a high level of accuracy.

II. DATA SOURCE MODULE

In this paper and the primary dataset employed for experimentation is the MNIST (Modified National Institute of Standards and Technology) dataset. This dataset has gained widespread recognition and adoption within the field of deep learning and computer vision. It comprises a comprehensive collection of handwritten digits and with each digit meticulously size normalized and centred within a fixed image size of 28 by 28 pixels.
By utilizing' the MNIST dataset in our study and we aim to leverage its standardized format to rigorously assess the performance and effectiveness of our proposed

methodology for handwritten digit recognition. The consistency in digit presentation within the dataset facilitates a fair and objective comparison of results and enabling us to draw meaningful insights into the capabilities and limitations of employed techniques.

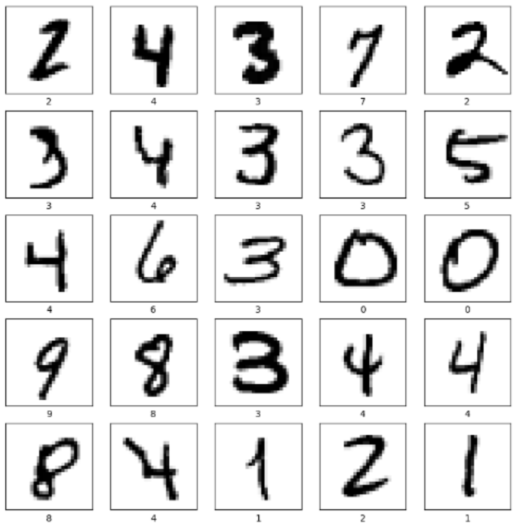


Fig1 MNIST Dataset

The MNIST dataset encompasses a collection of 70,000 images portraying handwritten digits and each contributed by different individuals and providing a diverse range of writing styles for comprehensive evaluation. In our approach, we judiciously allocated 60,000 of these handwritten images for the training phase of our model. Subsequently we reserved the remaining 10,000 images for testing phase. This deliberate division allows us to expose our model to a substantial variety of handwritten digits during the training process and promoting robust learning. The segregated testing set serves as an independent benchmark to assess the model's generalization and performance on unseen data and ensuring a thorough evaluation of its ability to recognize handwritten digits accurately.

III. DIGITAL IDENTIFICATION MODULE

A. *Pre Processing*

When we are required to build a predictive model, we have to look and manipulate the data before we start modelling which includes multiple preprocessing steps such as adjustments to image attributes such as colour of images changing, adjusting size of images, visualizing the image dataset and converting them to vector form from categorical. These steps help us in preprocessing of our dataset.

We have converted our image from RGB to Gray scale for easy computation by dividing with 255-pixel value to get the value between 0-1. In this process we are also converting the data from categorical

B. *Principle of Convolutional neural network*

Following the completion of the pre-processing phase, the subsequent pivotal step involves the creation of the Convolutional Neural Network (CNN) model. A CNN is a specialized neural network architecture designed for image-related tasks. Our CNN model is constructed with four hidden layers, each playing a distinct role in the feature extraction process, ultimately enabling the model to make accurate predictions. The sequential layers of the CNN include: (a) Convolutional Layer (b) ReLu Layer (c) Pooling Layer (d) Fully Connected Layer. The rationale behind employing a CNN lies in its inherent ability to automatically identify significant features without requiring explicit human guidance. This advantageous characteristic positions CNNs as powerful tools for image recognition tasks, making them particularly well-suited for handwritten digit recognition in our study. Identify applicable funding agency here. If none, delete this text box.

1. *Convolutional Layer*

The Convolutional layer is a straightforward implementation of a filter that serves as an activation function in neural networks. Its primary function involves extracting features from an input image by filtering various aspects of the image to generate a feature map. These features encompass attributes such as location and strength. As the filter traverses the entire image, the value of each pixel is computed, contributing to the creation of the feature map. This process allows the Convolutional layer to systematically identify and capture distinct features within the input image.

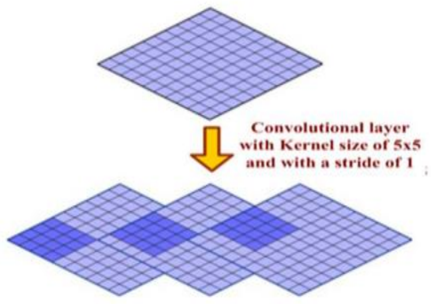


Fig 2 Convolutional Layer

2. *ReLu Layer*

In simpler terms, the ReLu layer's job is to get rid of any negative pixel values in an image and replace them with zero. This step is taken to prevent the accumulation of pixel values, ensuring that only positive values contribute to the overall image representation.

3. *Pooling Layer*

The primary role of this layer is to reduce the size of the image. This reduction is implemented to enhance computational speed and decrease overall computational cost. Essentially, what this layer does is employ a 2 x 2 matrix with a specified stride (movement from one pixel to another) of 1. This matrix window is then moved across the entire image, and within each window, the highest value is retained. This process is repeated for every part of the image. To illustrate, if the matrix before the pooling layer was 4 x 4, after the pooling layer, the image matrix is condensed to a 2 x 2 matrix.

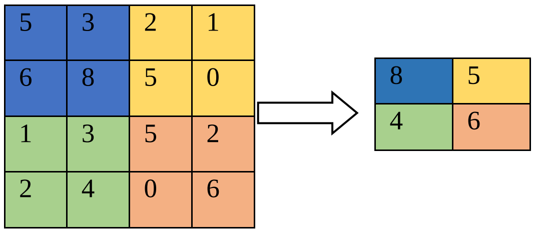
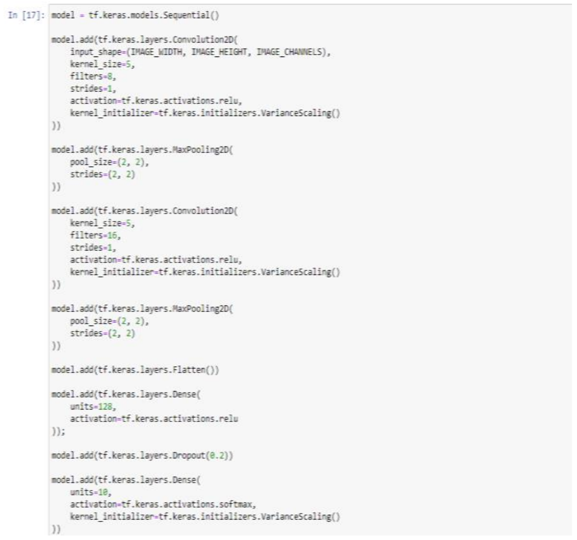


Fig 3 Pooling Layer

4. *Fully Connected Layer*

This is the last layer of CNN. This is the part where the actual classification happens. All the matrix from the pooling layer is stacked up here and put into a single list. The values which are higher are the points of prediction for the given image.



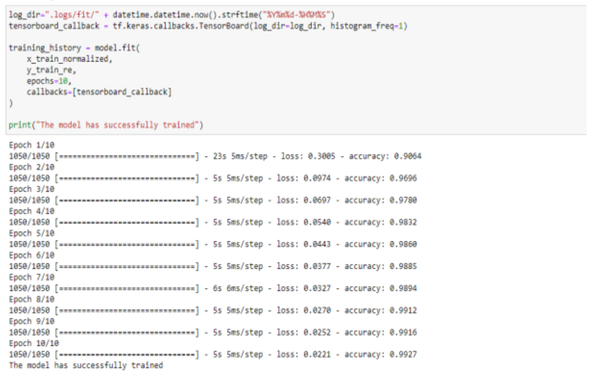
```
In [17]: model = tf.keras.models.Sequential()

         model.add(tf.keras.layers.Convolution2D(
             input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS),
             kernel_size=5,
             filters=8,
             strides=1,
             activation=tf.keras.activations.relu,
             kernel_initializer=tf.keras.initializers.VarianceScaling()
         ))

         model.add(tf.keras.layers.MaxPooling2D(
             pool_size=(2, 2),
             strides=(2, 2)
         ))

         model.add(tf.keras.layers.Convolution2D(
             kernel_size=5,
             filters=16,
             strides=1,
             activation=tf.keras.activations.relu,
             kernel_initializer=tf.keras.initializers.VarianceScaling()
         ))

         model.add(tf.keras.layers.MaxPooling2D(
             pool_size=(2, 2),
             strides=(2, 2)
         ))

         model.add(tf.keras.layers.Flatten())

         model.add(tf.keras.layers.Dense(
             units=128,
             activation=tf.keras.activations.relu
         ));

         model.add(tf.keras.layers.Dropout(0.2))

         model.add(tf.keras.layers.Dense(
             units=10,
             activation=tf.keras.activations.softmax,
             kernel_initializer=tf.keras.initializers.VarianceScaling()
         ))
```

Fig 4 CNN Model

5. The depicted image showcases the CNN model along with the incorporation of essential libraries. The 'Sequential' function is employed to organize all the processes sequentially, ensuring a step-by-step flow. Additionally, the 'Dropout' code is utilized to selectively exclude values from the dataset, effectively mitigating overfitting.

C. *Classification of Recognition results*

After completed the model construction, the next phase involves model evaluation. In this segment of the project, it is crucial to verify whether the constructed model aligns with our expectations. However, prior to the evaluation process, the model must undergo training on the designated evaluation dataset.



```
log_dir=".logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

training_history = model.fit(
    x_train_normalized,
    y_train_re,
    epochs=10,
    callbacks=[tensorboard_callback]
)

print("The model has successfully trained")
```

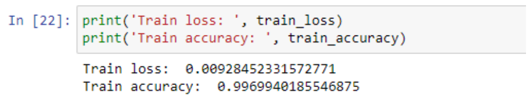
```
Epoch 1/10
1050/1050 [==============================] - 23s 5ms/step - loss: 0.3005 - accuracy: 0.9064
Epoch 2/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0974 - accuracy: 0.9696
Epoch 3/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0697 - accuracy: 0.9780
Epoch 4/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0540 - accuracy: 0.9832
Epoch 5/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0443 - accuracy: 0.9860
Epoch 6/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0377 - accuracy: 0.9885
Epoch 7/10
1050/1050 [==============================] - 6s 6ms/step - loss: 0.0327 - accuracy: 0.9894
Epoch 8/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0270 - accuracy: 0.9912
Epoch 9/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0252 - accuracy: 0.9916
Epoch 10/10
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0221 - accuracy: 0.9927
The model has successfully trained
```

Fig 5 Training & Testing of CNN Model

In the image above, the training and testing phases of my model are distinctly illustrated. To prevent the model from repeatedly learning the same values, I opted for 10 epochs. A batch size of 200 was chosen, considering the ten classes (representing digits 0-9), and the training process unfolded over 60,000 images. Concurrently, validation occurred on a separate set of 10,000 images. The initiation of the process reveals insights into validation accuracy and validation loss. Notably, as each epoch progresses, there is a discernible enhancement in validation accuracy. Following the completion of the evaluation, the model's performance on the test datasets becomes apparent.

IV. RESULTS

The outcomes of testing the MNIST transcribed digit dataset using different configurations of CNN models have been documented and analyzed. The findings validate the impact of diverse architectural parameters on the performance of our recognition system. In this experiment, a training parameter with a learning rate of 0.009 and 10 epochs was employed. The highest achieved recognition accuracy, specifically with a CNN architecture featuring four layers, stands at an impressive 99.69% for the dataset.



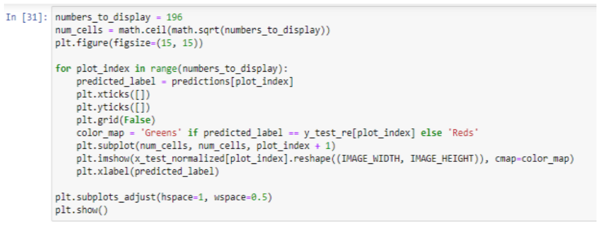
```
In [22]: print('Train loss: ', train_loss)
         print('Train accuracy: ', train_accuracy)

         Train loss:  0.00928452331572771
         Train accuracy:  0.9969940185546875
```

Fig 7 Evaluation

The objective of this study is to comprehensively explore the various configurations of CNN architecture to achieve optimal recognition accuracy for the MNIST dataset. In general, the proposed model, featuring a CNN design with three layers, has demonstrated superior recognition accuracy, reaching 99.69 % when utilizing the Adam optimizer. This observation underscores the effectiveness of the chosen model architecture in achieving high accuracy on the MNIST dataset.

V. PREDICTION

This is the final part of my project where we will check if our model is predicting the digits accurately and with what percentage it is correctly predicting it. So, what I have done here is I have extracted some of the images from the test dataset and kept it into a folder.



```
In [31]: numbers_to_display = 196
         num_cells = math.ceil(math.sqrt(numbers_to_display))
         plt.figure(figsize=(15, 15))

         for plot_index in range(numbers_to_display):
             predicted_label = predictions[plot_index]
             plt.xticks([])
             plt.yticks([])
             plt.grid(False)
             color_map = 'Greens' if predicted_label == y_test_re[plot_index] else 'Reds'
             plt.subplot(num_cells, num_cells, plot_index + 1)
             plt.imshow(x_test_normalized[plot_index].reshape((IMAGE_WIDTH, IMAGE_HEIGHT)), cmap=color_map)
             plt.xlabel(predicted_label)

         plt.subplots_adjust(hspace=1, wspace=0.5)
         plt.show()
```

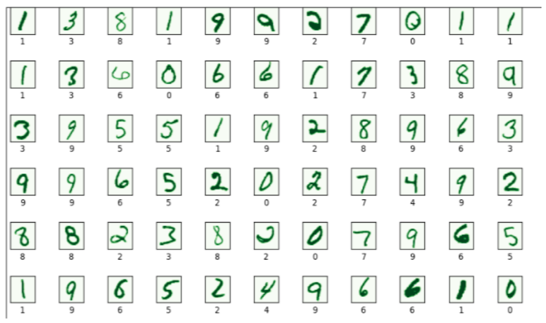


Fig 8 Prediction

The evident success of our model is highlighted by its impressive prediction rate, achieving accuracy up to 99% on the test dataset images. Every image from the test dataset was correctly predicted, underscoring the robust performance of the model. With this validation, the focus now shifts to the next phase, where the model's predictions will be extended to user-provided images.

VI. CONCLUSION

In this study, aimed at enhancing the performance of transcribed digit recognition, we evaluated variations of a convolutional neural network to streamline the process, eliminating the need for complex pre-processing, costly feature extraction, and intricate ensemble approaches seen in traditional recognition systems. Extensive assessments using the MNIST dataset emphasized the significance of various hyperparameters. Our findings underscored the essential role of fine-tuning hyperparameters in optimizing CNN performance. Remarkably, we achieved a recognition rate of 99.89% with the Adam optimizer for the MNIST database, surpassing all previously reported results.

The experiments clearly illustrate the impact of increasing the number of convolutional layers in the CNN architecture on transcribed digit recognition performance. The novelty of this work lies in its thorough exploration of all CNN architecture parameters, yielding the best recognition accuracy for the MNIST dataset. Notably, this accuracy outperforms those achieved by peer researchers utilizing pure CNN models. While some researchers employed ensemble CNN network models for improved accuracy, it came at the expense of increased computational complexity. The present work achieved comparable accuracy without sacrificing computational efficiency.

Looking ahead, future research avenues may explore various CNN architectures, particularly hybrid CNNs like CNN-RNN and CNN-HMM models, as well as domain-specific recognition systems. Evolutionary algorithms could be investigated to optimize CNN learning parameters, such as the number of layers, learning rate, and convolutional channel sizes. These endeavors would contribute to advancing the field of digit recognition and pave the way for more efficient and accurate models.