Handwritten Digit Recognition Using CNN

Abstract:

CNN (Convolutional Neural Network) is a type of computer program inspired by how our brains process visual information. It's really good at understanding and recognizing pictures.

Imagine you want a computer to recognize cats in photos. A CNN does this by looking at different parts of the image bit by bit, learning to recognize features like edges, textures, and patterns. It then combines these features to understand the bigger picture, like the shape of a cat's face or body.

Think of it like how you recognize things—you might notice the ears, eyes, and tail of a cat to say, "Ah, that's a cat!" Similarly, a CNN breaks down images into smaller pieces, learns from them, and puts it all together to make sense of what's in the picture. It's really handy for tasks like identifying objects in photos or videos.

Digit recognition is a fundamental task in computer vision, with applications ranging from automated document processing to character recognition in various domains. Convolutional Neural Networks (CNNs) have emerged as a powerful tool for tackling this challenge. In this study, we explore the effectiveness of CNNs in digit recognition by leveraging their ability to automatically learn hierarchical features from input images. The proposed CNN architecture involves convolutional layers that extract essential features, pooling layers that reduce spatial dimensions, and fully connected layers for recognizing and classifying digits. Through training on labeled digit datasets, the CNN learns to discern unique patterns and shapes associated with different digits. Experimental results demonstrate the superior performance of the CNN model in accurately and efficiently recognizing digits, showcasing its potential for practical applications in character recognition systems. The findings underscore the significance of CNNs in advancing the state-of-the-art in digit recognition, offering a robust and adaptable solution for diverse real-world scenarios.

AI (Artificial intelligence):

Artificial intelligence (AI) in simple words is basically making a computer do the work that traditionally requires the human brain. AI has the ability to take in large amounts of data unlike the human and uses that data to recognize patterns, make decisions, and give judgment. In this AI we have a subset which is called Machine learning. ML is used to make computers to learn to behave as humans. This is done by two ways, supervised learning in which the computer is given a set of input data and the required output for it. Now it uses ML to learn the algorithm to understand how that particular input gives this particular output. Now the unsupervised learning is when input data is provided but with no output, so the ML has to learn to analyze and clutter the datasets into categories.

ML (Machine Learning):

ML (Machine Learning) is the branch of computer science which helps the machines to learn without being programmed. A program learns from data given. It performs tasks and measures accuracy and tells if it's performance at doing tasks improved with the experience or not. We use algorithms and other techniques in Machine Learning instead of doing the programming part. Machines learn from past experiences and examples. A model can be built based on their past experiences, so that it can be used to predict the new values. If the question or the problem is too large and difficult to solve, then it can be used. It can also help finding the answers to questions based on the analysis of data. It requires very less time to find important things when a large amount of data is given. It can also solve very complex problems as machines can learn faster than humans and they may even exceed humans in some fields. As a result, its demand is rising continuously. Machine Learning is catching up with cloud computing and big data as it can solve many difficult problems with ease. It's used in many applications.

Neural network:

A neural network is a computational model inspired by the structure and functioning of the human brain. It is composed of interconnected nodes, also known as neurons or artificial neurons, organized in layers. Neural networks are used for machine learning and artificial intelligence tasks, particularly for pattern recognition and decision-making.

Here are the key components of a neural network:

- Neurons (Nodes): Neurons are the fundamental units of a neural network. Each neuron receives one or more inputs, processes them using a weighted sum, and passes the result through an activation function to produce an output.
- Layers: Neurons are organized into layers. The three main types of layers are:
- Input Layer: This layer receives the initial input data.
- Hidden Layers: These layers process information and learn patterns. Neural networks can have multiple hidden layers.
- Output Layer: The final layer produces the network's output, which is the result of the model's computation.
- Weights and Biases: Each connection between neurons has an associated weight, representing the strength of the connection. Additionally, each neuron has a bias term. During training, these weights and biases are adjusted to optimize the network's performance.
- Activation Function: Neurons use an activation function to introduce non-linearity into the network. Common activation functions include sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU).
- Feedforward and Backpropagation: In a feedforward process, data passes through the network layer by layer, producing an output. Backpropagation is the training process where errors are calculated, and the weights and biases are adjusted to minimize these errors. This process is often done using optimization algorithms like gradient descent.

Neural networks have demonstrated remarkable capabilities in various domains, including image and speech recognition, natural language processing, and decision-making. Different types of

neural networks, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are designed for specific tasks and data types.

CNN (Convolutional Neural Network):

CNN stands for Convolutional Neural Network. It is a class of deep neural networks commonly used in computer vision tasks such as image recognition, object detection, and image classification. CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input data.

Key components of a CNN include:

- Convolutional Layers: These layers apply convolutional operations to the input data, using filters or kernels to extract features. Convolutional operations involve sliding these filters over the input data to detect patterns like edges, textures, or other important features.
- Pooling Layers: Pooling layers down sample the spatial dimensions of the input volume, reducing the computational complexity and the number of parameters. Max pooling is a common type of pooling layer, which retains the maximum values within each region.
- Activation Functions: Non-linear activation functions, such as ReLU (Rectified Linear Unit), are applied to introduce non-linearity into the network. This allows the model to learn complex relationships and patterns in the data.
- Fully Connected Layers: These layers connect every neuron in one layer to every neuron in the next layer, allowing the network to make predictions based on the learned features.

CNNs have proven to be highly effective in image-related tasks due to their ability to automatically learn hierarchical features from the data. They have been extended and adapted for use in various other domains, such as natural language processing and speech recognition, demonstrating their versatility in different applications.

Handwritten digit recognition using Convolutional Neural Networks:

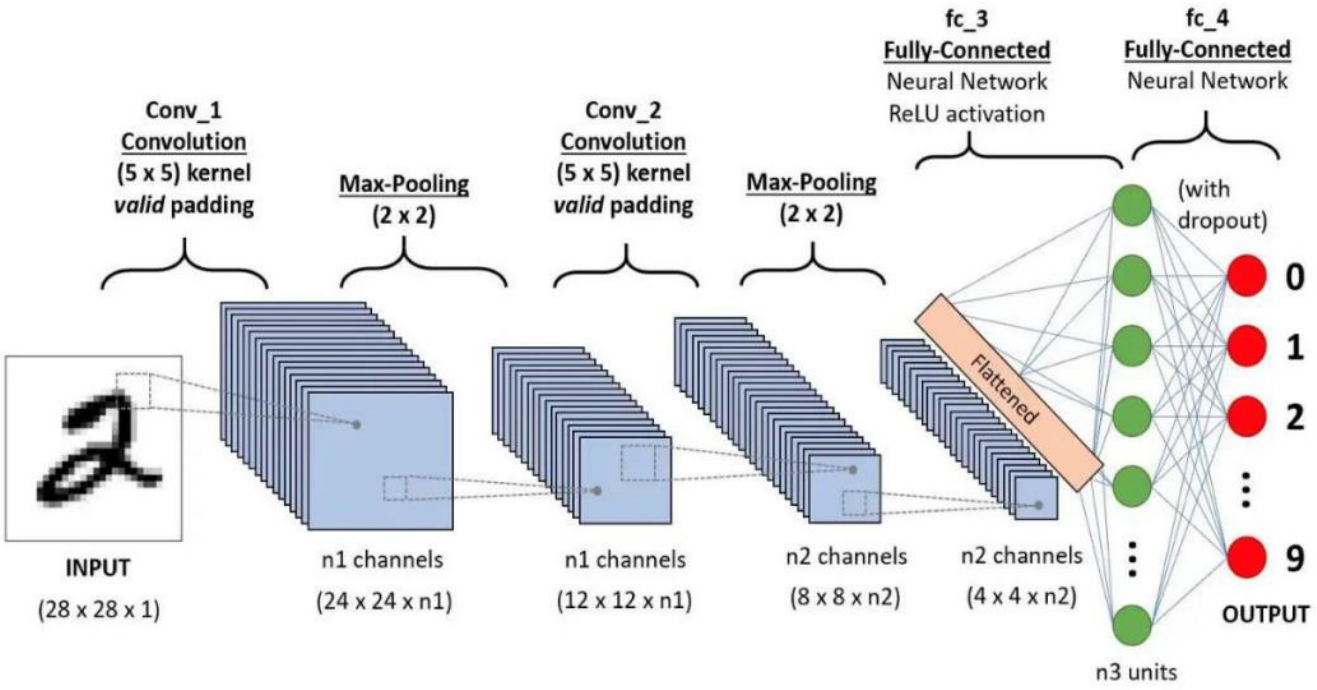


Figure1: General look of handwritten digit recognition using CNNs:

Handwritten digit recognition using Convolutional Neural Networks (CNNs) involves several key steps. Let's break it down:

1. Data Preparation:
   - Dataset: Gather a dataset of handwritten digits. A popular choice is the MNIST dataset, which consists of 28x28 pixel grayscale images of handwritten digits (0 through 9).
2. Input Representation:
   - Image Pixels: Represent each image as a grid of pixel values. In the case of MNIST, each image is 28x28 pixels, forming a 2D array.

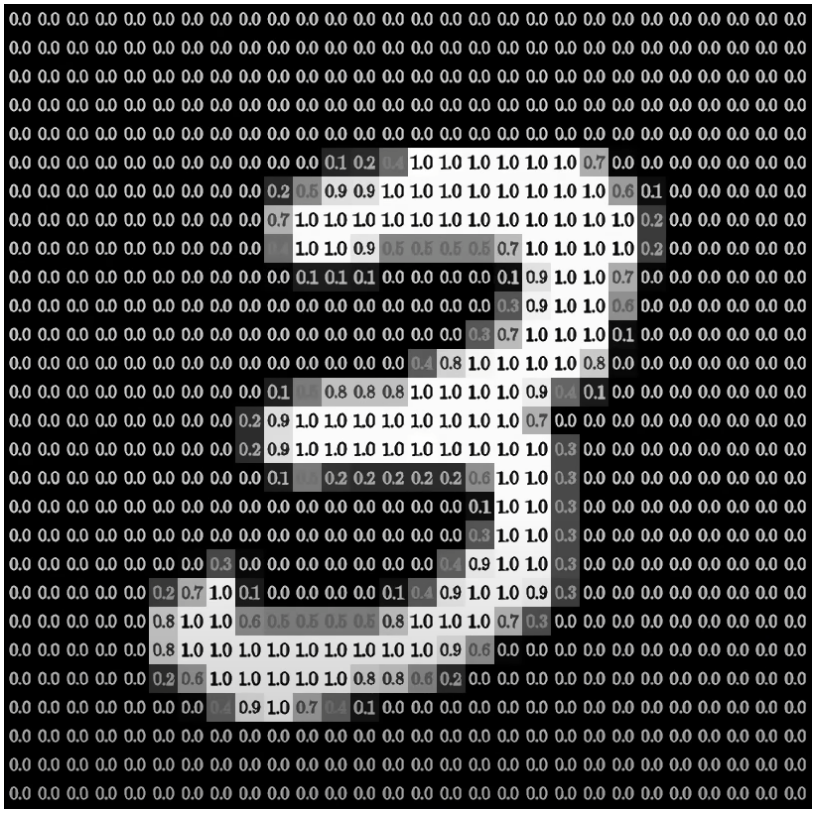
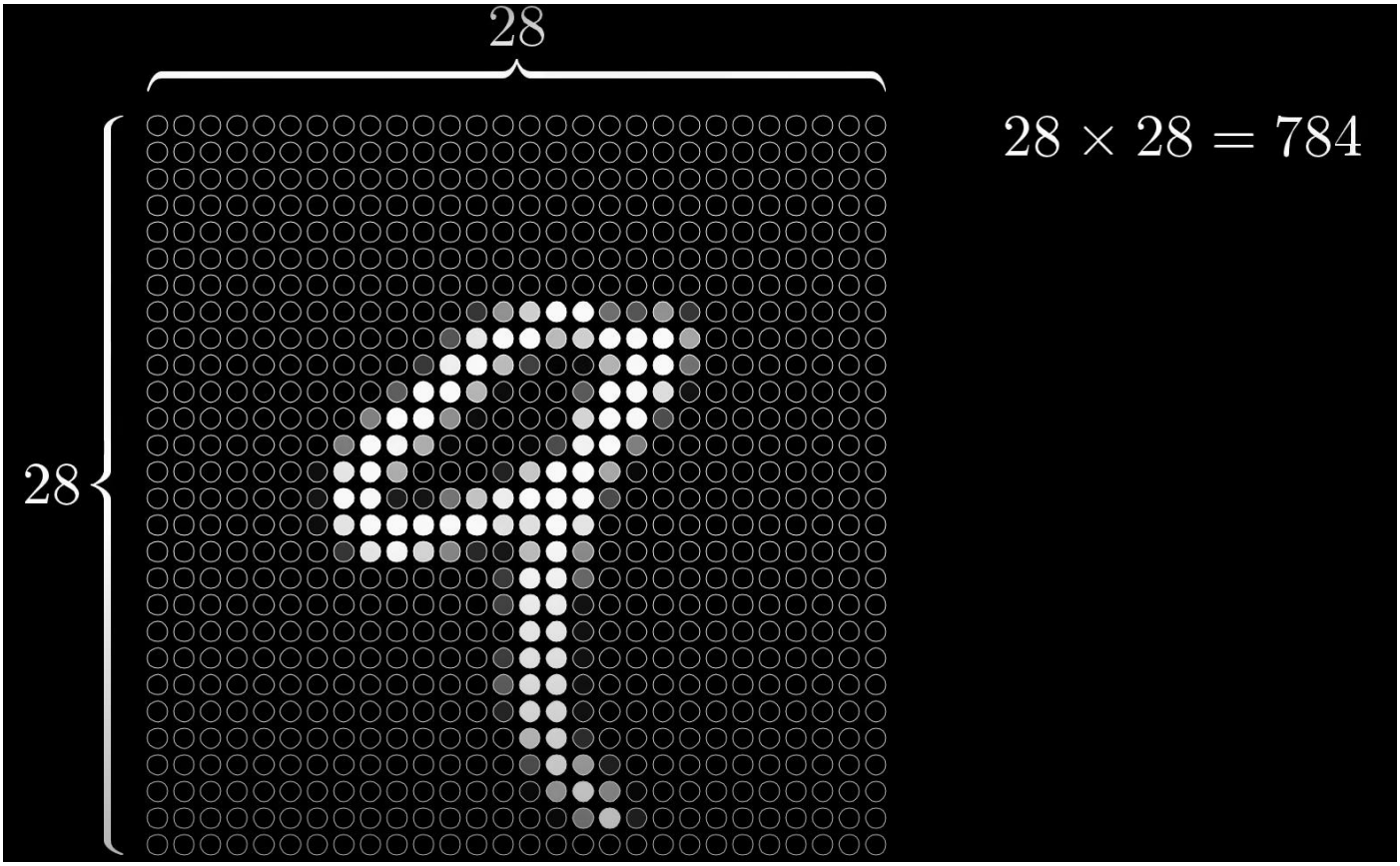


*Figure2: 28*28-digit image representation as a grid of pixel values:*

3. CNN Architecture:
   - Input Layer: The first layer takes the pixel values of the image. (*Figure3 & 4*)
   - Convolutional Layers: These layers apply filters to the input image, learning features like edges or curves. (*Figure5*)
   - Activation Function: After convolution, apply a non-linear activation function like ReLU to introduce non-linearity. (*Figure6*)
   - Pooling Layers: Down sample the spatial dimensions to reduce computational load. (*Figure6*)
   - Flatten Layer: Flatten the output from the previous layers into a 1D array. (*Figure6*)
   - Fully Connected Layers: Connect each neuron to every neuron in the next layer. (*Figure7*)
   - Output Layer: Produce the final output, representing the predicted digit. (*Figure7*)



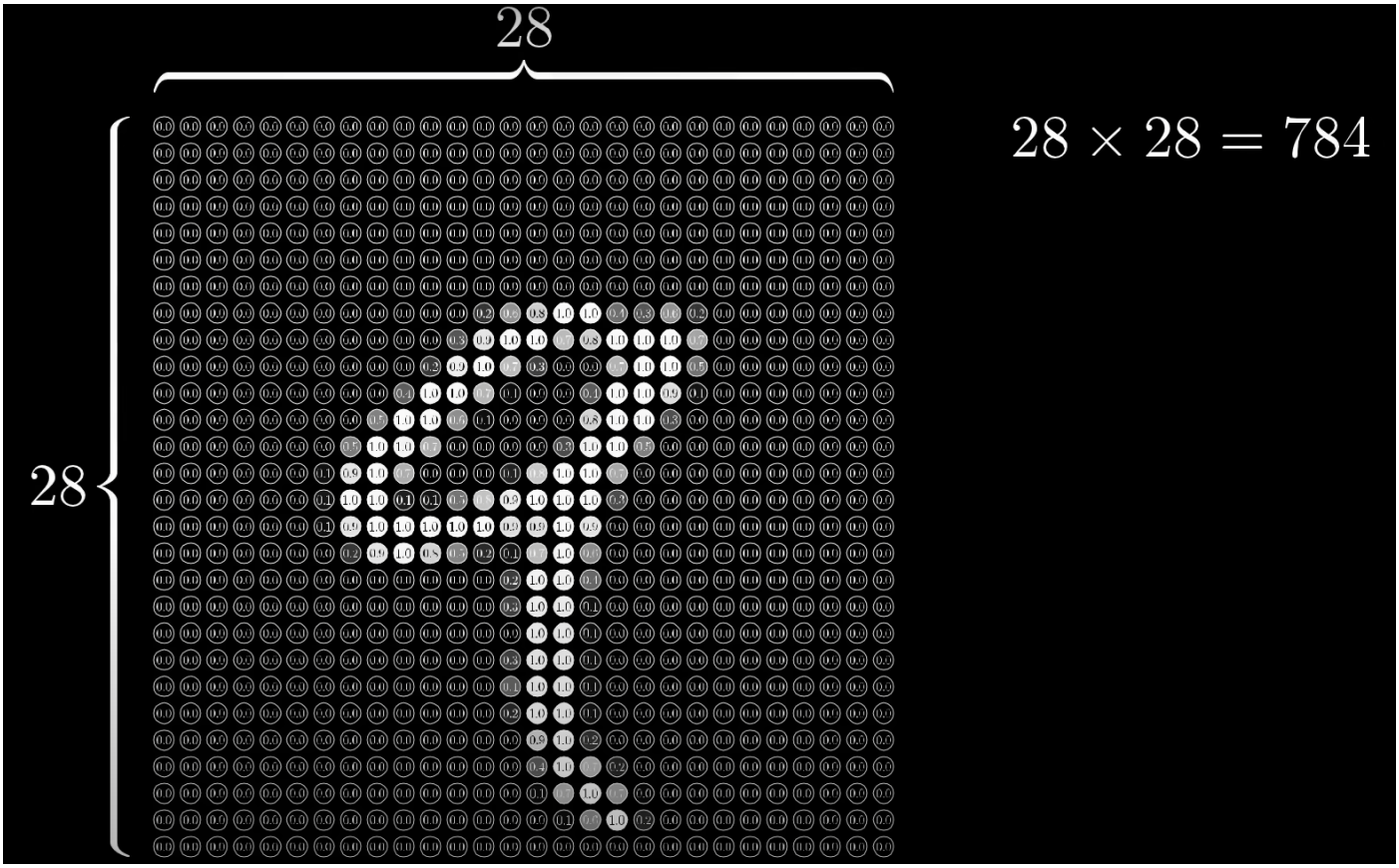


*Figure3 & 4: 28*28-digit image representation as a grid of pixel values and the total of inputs:*

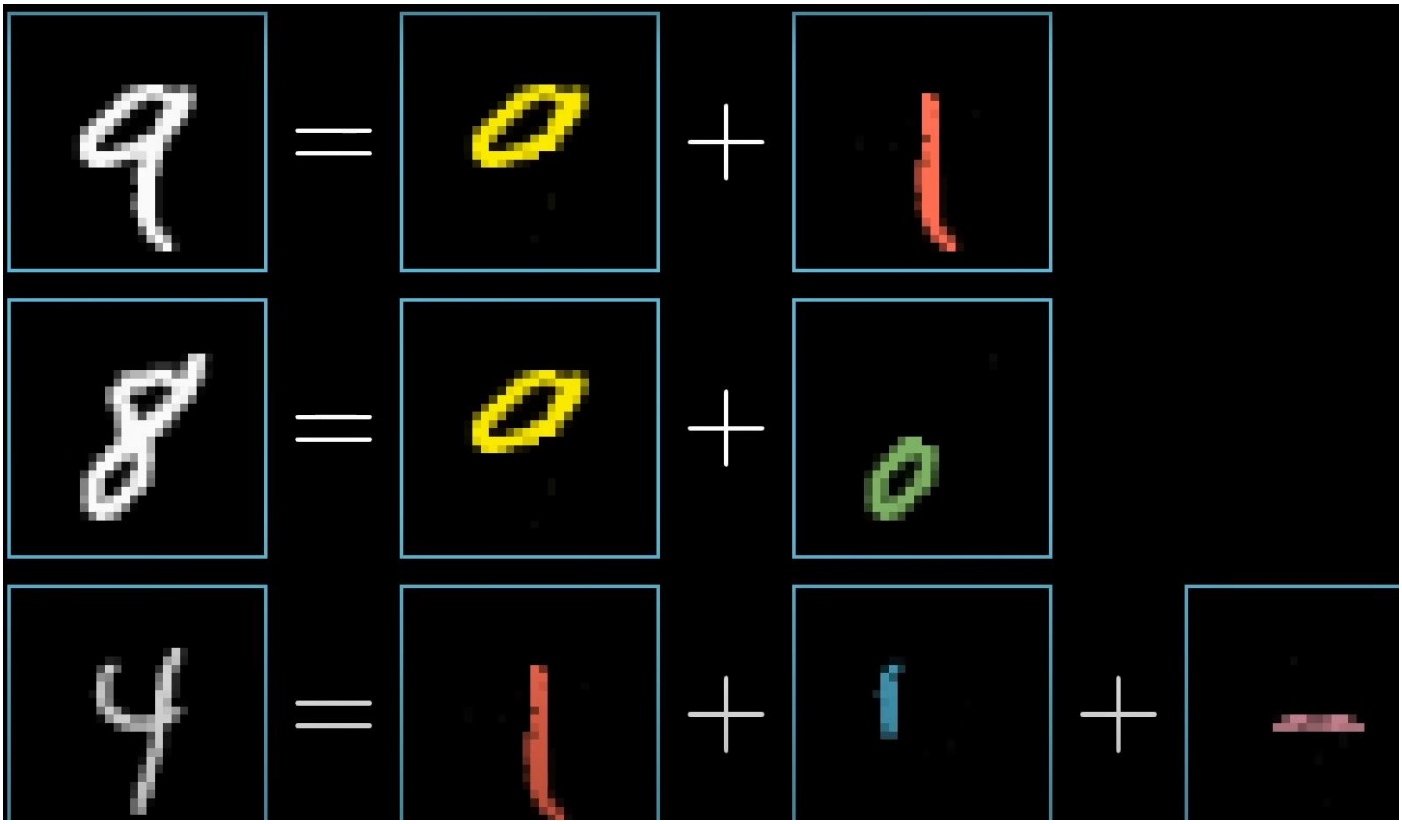


Figure5: results of the convolutional layers:

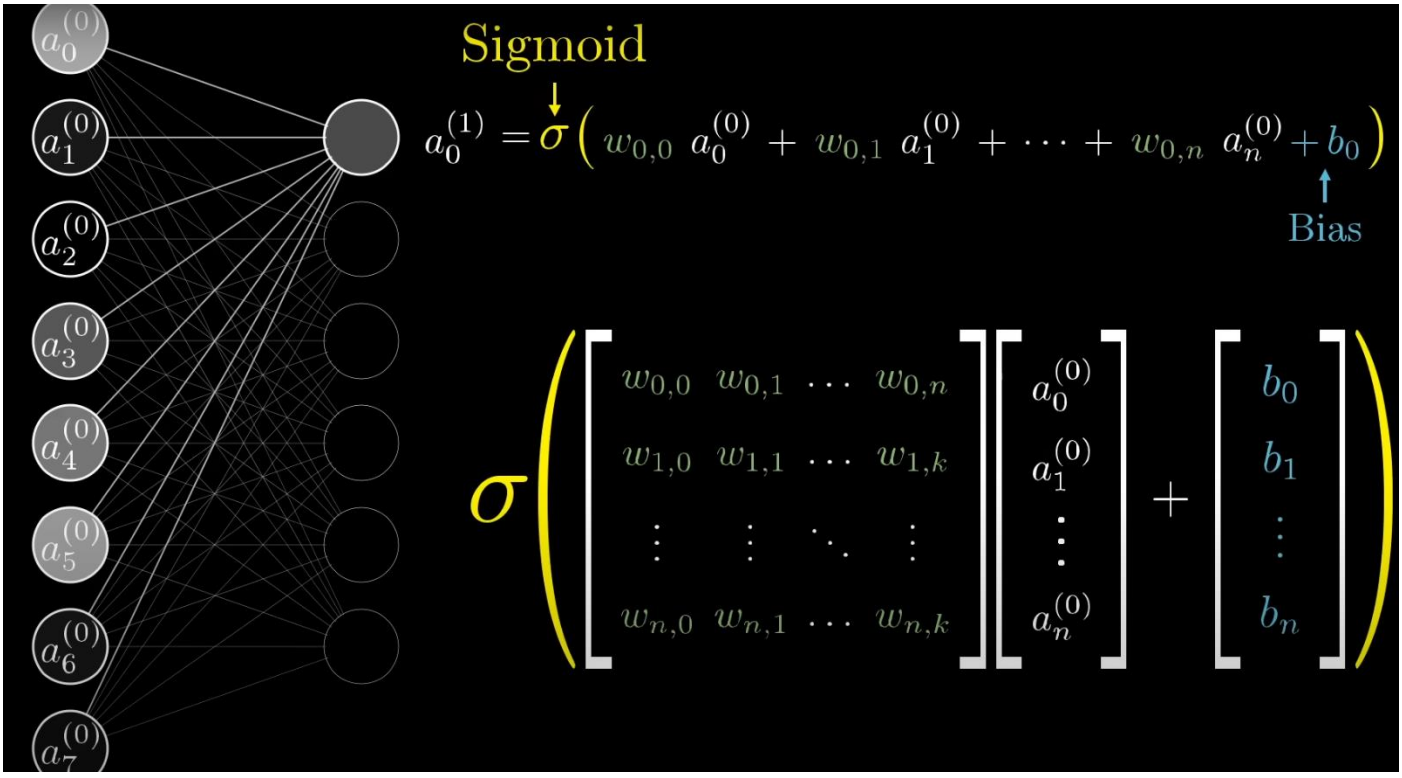


Figure6: results of the activation function, pooling layers, flatten layer:

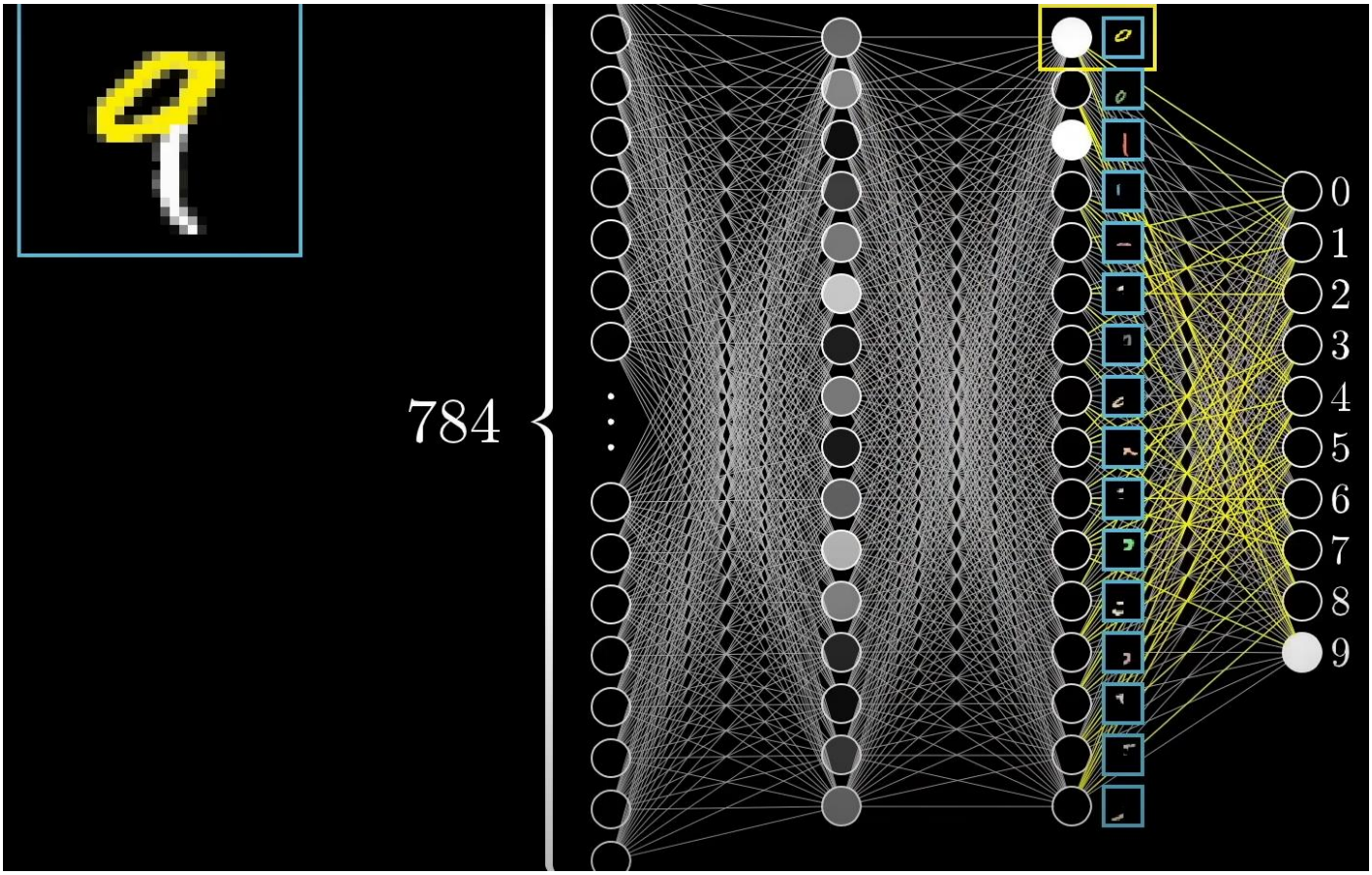


Figure7: results of the fully connected layers and the final output:

4. Training:
   - Loss Function: Use a loss function (e.g., cross-entropy loss) to measure the difference between predicted and actual digit labels.
   - Optimizer: Utilize an optimizer (e.g., stochastic gradient descent) to adjust the network's weights and biases.
   - Backpropagation: Iteratively update weights and biases based on the training data, minimizing the loss.
5. Testing and Prediction:
   - Feedforward: Pass new handwritten digit images through the trained network.
   - Output: The output layer produces a probability distribution over the possible digits.
   - Prediction: Choose the digit with the highest probability as the predicted digit.
6. Evaluation:
   - Accuracy: Measure how well the model performs on a separate set of validation or test data.
   - Adjustments: Fine-tune the model if needed, considering factors like learning rate or model architecture.
7. Deployment:
   - Application: Once satisfied with the model's accuracy, deploy it to recognize handwritten digits in real-world scenarios.

In summary, a CNN learns to recognize handwritten digits by processing pixel values through convolutional and fully connected layers, adjusting its parameters during training, and ultimately

producing accurate predictions for new digit images. The power of CNNs lies in their ability to automatically learn hierarchical features, making them well-suited for image recognition tasks like handwritten digit recognition.

Implementation of Handwritten Recognition:

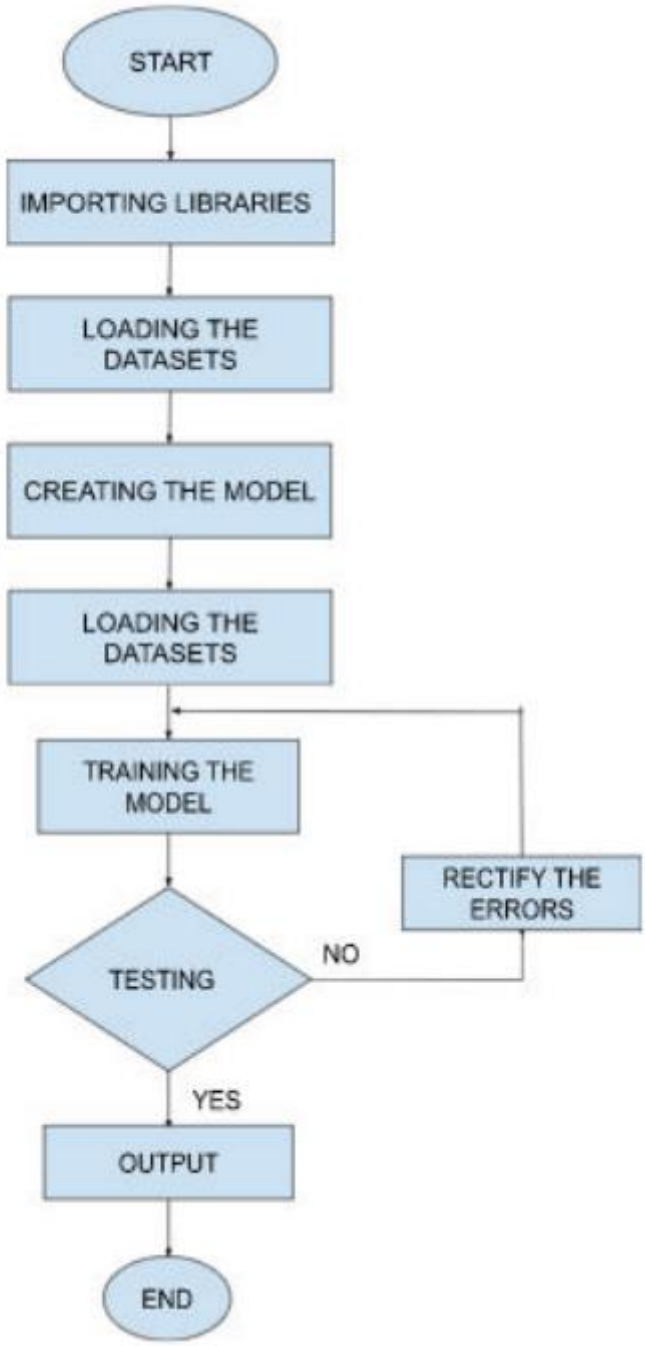


Figure8: methodology used in implementation of handwritten recognition

Libraries Used:

1. os:
   - Purpose: For file management.

- Explanation: Used to interact with the operating system, particularly for checking the existence of image files in a specific directory.

2. cv2 (OpenCV):
   - Purpose: To load, read, and process images.
   - Explanation: OpenCV is a popular computer vision library. In this code, it is used to read images in grayscale and perform operations like inverting the colors.
3. numpy (np):
   - Purpose: For numerical computations with arrays.
   - Explanation: Numpy is a powerful library for numerical operations. Here, it's used to manipulate arrays, particularly for inverting the colors of the image.
4. matplotlib.pyplot (plt):
   - Purpose: For visualizing digits.
   - Explanation: Matplotlib is a plotting library. It's used here to display the images of handwritten digits.
5. tensorflow (tf):
   - Purpose: For deep learning.
   - Explanation: TensorFlow is a popular deep learning library. It's used to build, train, and evaluate the neural network for handwritten digit recognition.

Code Sections:

1. Loading and Preprocessing Data:
   - Explanation: The MNIST dataset, consisting of images and labels for handwritten digits, is loaded and split into training and testing sets. The pixel values of the images are normalized to a range between 0 and 1.
2. Building the Neural Network Model:
   - Explanation: A Sequential model is created. The input layer is flattened, followed by two dense layers with rectified linear unit (ReLU) activation functions. The final layer has 10 neurons with a softmax activation function for digit classification.
3. Compiling the Model:
   - Explanation: The model is compiled using the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the metric. This defines how the model will be trained and evaluated.
4. Training the Model:
   - Explanation: The model is trained on the training data for three epochs (iterations over the entire dataset).

```
1875/1875 [==============================] - 8s 3ms/step - loss: 0.2638 - accuracy: 0.9236
Epoch 2/3
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1080 - accuracy: 0.9667
Epoch 3/3
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0732 - accuracy: 0.9770
```

Figure9: model training results

5. Saving and Loading the Model:
    - Explanation: The trained model is saved to a file ('handwrittendigits.model') and later loaded back for evaluation.
6. Model Evaluation:
    - Explanation: The loaded model is evaluated on the test data, and the loss and accuracy metrics are printed.

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0959 - accuracy: 0.9704
Loss: 0.0959160327911377
Accuracy: 0.9703999757766724
```

Figure10: accuracy of the model

7. Digit Recognition Loop:
    - Explanation: A loop is initiated to predict digits from images ('digits/digit{image_number}.png'). Each image is loaded, inverted, and passed through the trained model. The predicted digit and the image are displayed using Matplotlib.
8. Incrementing Image Number:
    - Explanation: The loop increments the image number for the next iteration.

Note:

- Ensure the 'digits' directory exists and contains the digit images in the specified format (e.g., 'digit1.png', 'digit2.png').
- The model is trained for a small number of epochs (3) in this example. In practice, more epochs may be needed for better accuracy.
- The script assumes images are in PNG format, and the images are expected to be grayscale. Adjustments may be needed for different formats or color images.

This code essentially demonstrates the process of training a simple neural network to recognize handwritten digits using the MNIST dataset and then applying it to predict digits from custom images.

```python
import os #for file management
import cv2 #to load, read and process images
import numpy as np #for calculus (arrays)
import matplotlib.pyplot as plt #for digits visualisation
import tensorflow as tf #for deep learning

# Load the MNIST dataset (labeled dataset of handwritten digits)
mnist = tf.keras.datasets.mnist
# The MNIST dataset is a set of 28x28 grayscale images of handwritten digits
# and their labels (0-9)

# Split the dataset into training and testing sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# x_train and x_test are arrays of grayscale images
# y_train and y_test are arrays of labels (digits)

# Normalize the data
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
# Normalize the data (pixels are between 0 and 1 instead of 0 and 255)
# we ignore the other RGB values because the images are grayscale
# This helps the neural network to learn better

# Create the model
model = tf.keras.models.Sequential()
# Sequential model is a linear stack of layers

# Add the first layer (input layer)
model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
# Flatten layer is used to flatten the input (28x28) into a vector (784)
model.add(tf.keras.layers.Dense( units: 128, activation=tf.nn.relu))
# Dense layer is a fully connected layer (all neurons are connected)
# 128 neurons
# Activation function: Rectified Linear Unit (relu)
model.add(tf.keras.layers.Dense( units: 128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense( units: 10, activation=tf.nn.softmax))
# 10 neurons
# Activation function: Softmax (probability distribution)
# Softmax is used because we want to predict the probability of each class (each digit)
# The sum of all probabilities is 1
# The highest probability is the predicted class (the digit with the highest probability)
```

```python
# The highest probability is the predicted class (the digit with the highest probability)

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Optimizer: Adam (adaptive moment estimation)
# Loss function: sparse_categorical_crossentropy
# Metrics: accuracy
# The loss function is used to measure how well the model did on training and then tries to improve on it using the optimizer (Adam)
# The metrics is used to monitor the training and testing steps (accuracy)

# Train the model
model.fit(x_train, y_train, epochs=3)
# Epochs: number of iterations over the entire dataset

# Save the model
model.save('handwrittendigits.model')

# Load the model
model = tf.keras.models.load_model('handwrittendigits.model')

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)

print('Loss:', loss)
print('Accuracy:', accuracy)

image_number = 1
# Predict the digit
while os.path.isfile(f"digits/digit{image_number}.png"):
# Check if the file exists
    try:
        # Load the image
        img = cv2.imread(f"digits/digit{image_number}.png")[:,:,0]
        # Load the image as grayscale
        # [:,:,0] to get the first channel (grayscale)

        img = np.invert(np.array([img]))
        # Invert the image (black background and white digit)
        # Convert the image to a numpy array
        # The neural network accepts numpy arrays as input
        # The neural network expects a batch of images as input
```

```
        img = np.invert(np.array([img]))
        # Invert the image (black background and white digit)
        # Convert the image to a numpy array
        # The neural network accepts numpy arrays as input
        # The neural network expects a batch of images as input
        # The batch size is 1 (one image)
        # The image is 28x28 pixels

        prediction = model.predict(img)
        # Predict the digit
        # The prediction is an array of probabilities (10 probabilities)
        print("Image number:", image_number)
        print("The digit is probably a:", np.argmax(prediction))
        print("\n")
        # Print the digit with the highest probability

        plt.imshow(img[0], cmap=plt.cm.binary)
        # Display the image
        plt.show()

    except:
        print("Error")
    finally:
        image_number += 1
        # Increment the image number
```

Figure11, 12 & 13: code source of the implementation (well commented):

Test and results:

*Inputs 28*28-digit pictures*

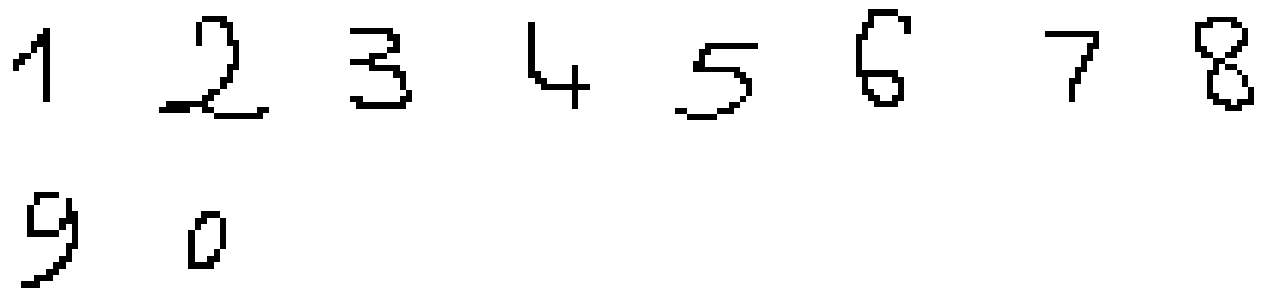


*Figure14: 28*28-digit pictures:*

The output

```
1/1 [==============================] - 0s 126ms/step
Image number: 1
The digit is probably a: 1
1/1 [==============================] - 0s 31ms/step
Image number: 2
The digit is probably a: 2
1/1 [==============================] - 0s 30ms/step
Image number: 3
The digit is probably a: 3
1/1 [==============================] - 0s 30ms/step
Image number: 4
The digit is probably a: 4
1/1 [==============================] - 0s 29ms/step
Image number: 5
The digit is probably a: 5
1/1 [==============================] - 0s 37ms/step
Image number: 6
The digit is probably a: 6
1/1 [==============================] - 0s 34ms/step
Image number: 7
The digit is probably a: 7
1/1 [==============================] - 0s 35ms/step
Image number: 8
The digit is probably a: 8
1/1 [==============================] - 0s 37ms/step
Image number: 9
The digit is probably a: 9
1/1 [==============================] - 0s 35ms/step
Image number: 10
The digit is probably a: 5
```

Figure15: the output:

Limitations and areas for improvement:

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0959 - accuracy: 0.9704
Loss: 0.0959160327911377
Accuracy: 0.9703999757766724
```

Figure10: accuracy of the model

While the provided code is a good starting point for handwritten digit recognition, there are several limitations and areas for improvement:

1. Limited Training Epochs:
   - Issue: The model is trained for only three epochs, which might be insufficient for the model to fully converge and learn intricate patterns in the data.
   - Improvement: Increase the number of training epochs to allow the model to learn more complex representations. Experiment with different epoch values to find an optimal balance between accuracy and training time.
2. Simple Model Architecture:
   - Issue: The neural network consists of only two dense layers. For more challenging tasks, a deeper and more complex architecture might be necessary.
   - Improvement Experiment with deeper architectures, additional convolutional layers, and regularization techniques to enhance the model's capacity to capture intricate features in the data.
3. Limited Data Augmentation:
   - Issue: No data augmentation is applied during training, which might limit the model's ability to generalize well to variations in handwriting styles.
   - Improvement: Implement data augmentation techniques (e.g., rotation, scaling, and translation) to artificially increase the diversity of the training dataset and improve the model's robustness.
4. Evaluation on a Single Test Set:
   - Issue: The model is evaluated on a single test set, and the performance metrics may vary based on the particular split of the dataset.
   - Improvement: Perform cross-validation or split the data into separate training, validation, and test sets. This provides a more robust assessment of the model's generalization performance.
5. No Dropout or Batch Normalization:
   - Issue: The model lacks dropout layers or batch normalization, which might lead to overfitting, especially with a limited amount of data.
   - Improvement: Introduce dropout layers or batch normalization to prevent overfitting and improve the model's generalization to new data.
6. Hyperparameter Tuning:
   - Issue: The code uses default hyperparameters for the optimizer, learning rate, and layer sizes.
   - Improvement: Experiment with different hyperparameter values to find optimal settings. Techniques like learning rate schedules or adaptive learning rate methods may also be beneficial.
7. Handling Edge Cases:
   - Issue: The code does not handle potential errors during image loading or prediction, leading to potential issues with edge cases.
   - Improvement: Implement proper error handling to ensure robustness, especially when dealing with diverse images or unexpected situations.
8. Visualization and Interpretability:

- Issue: The code lacks visualization of misclassified samples or interpretability tools, making it challenging to understand where the model struggles.
- Improvement: Visualize misclassified samples, use techniques like Grad-CAM for interpretability, and consider incorporating confusion matrices or precision-recall curves for a deeper understanding of model performance.

By addressing these limitations and incorporating suggested improvements, the code can be enhanced to achieve better accuracy and robustness in handwritten digit recognition. Continuous experimentation and tuning are essential for optimizing model performance in real-world scenarios.

Conclusion:

In this implementation, we created a neural network to recognize a handwritten digit using TensorFlow and Keras. using the MNIST dataset, our model can recognize handwritten digits which are being input to the model. The performance of CNN for the handwritten digit is accurate. The method works well, and the loss percentage is less with all those training sessions. The only difficulty here is the noise in the image, but with the training it has, it tries to achieve the best possible output. The model's performance and accuracy were tested after training the model for 50 epochs. With the results given from this work we are more confident in finding other ways to make this better and to make it easier for complex data like converting handwritten paragraphs into text. Through this research work we understood all the mechanisms used to identify handwritten data. We understand the importance of hand recognition as it is easy for the user to write data on paper and use handwritten data recognition to convert it into text instead of the typing it on keyboard. Further it is recommended to implement on edge computing platforms like Raspberry Pi 4 system for actual usage.