SRH University Heidelberg

# Task 1:
# Handwritten Digit classification based on the Convolutional Neural Network in Computer Vision


| | |
|---|---|
| Author: | Nikhil Kumar |
| Matriculation Number: | 11037894 |
| Email Address: | nikhil.kumar@stud.hochschule-heidelberg.de |
| Date of Submission: | 1st June 2024 |


Under the Guidance of Professor
Dr. Ing Milan Gnjatovic

# Aim of the Task:

This task aims to develop an accurate and robust system for classifying handwritten digits (0-9) using a Convolutional Neural Network (CNN). This involves designing and training a CNN model on a dataset like MNIST, ensuring effective data preprocessing, and experimenting with various CNN architectures. The goal is to optimize the model through hyperparameter tuning and implement techniques to avoid overfitting, such as dropout and data augmentation. The performance of the model will be evaluated using metrics like accuracy, precision, recall, and F1score. The final objective is to achieve high classification accuracy on the test dataset and provide comprehensive documentation of the entire process, including insights into the model's effectiveness and any challenges encountered.

# Objective :

The objective of this project is to implement a classification system for handwritten digits using a Convolutional Neural Network (CNN), specifically tailored for the MNIST dataset. The primary goals include developing a robust and efficient CNN model capable of accurately identifying handwritten digits (0-9) based on their pixel intensity values. This involves preprocessing the MNIST dataset, designing and training the CNN, and evaluating its performance in terms of accuracy, precision, recall, and F1-score. Through this endeavor, a comprehensive understanding of applying deep learning methods to image classification tasks will be gained, specifically in the context of handwritten digit recognition using the MNIST dataset.

**The specific objectives of this task include:**

1. **Data Preprocessing:**

- Normalize the pixel values of the images to a suitable range (0 to 1).
- Reshape the images to fit the input requirements of the CNN.

2. **CNN Architecture Design:**

- Design a CNN architecture suitable for image classification tasks. This typically includes layers such as convolutional layers, pooling layers, and fully connected layers.
- Experiment with different architectures to find the most effective one for this specific task.

3. **Model Training:**

- Train the CNN on the training dataset using appropriate loss functions (e.g., categorical cross-entropy) and optimization algorithms (e.g., Adam, SGD).
- Implement techniques to avoid overfitting, such as dropout, data augmentation, or early stopping.

4. **Model Evaluation:**

- Evaluate the performance of the trained CNN on a separate test dataset.
- Use metrics such as accuracy, precision, recall, and confusion matrix to assess the model's performance.

5. **Hyperparameter Tuning:**

- Experiment with different hyperparameters like batch size, number of epochs, different optimizers, cost function, and activation functions to optimize the model's performance.

# Introduction :

In recent years, the proliferation of deep learning techniques, particularly Convolutional Neural Networks (CNN), has revolutionized the field of computer vision. One of the seminal tasks in this domain is the classification of handwritten digits, a fundamental problem in pattern recognition with diverse applications ranging from automated mail sorting to bank check processing. Leveraging the power of CNN, this project aims to tackle this task by developing a robust and efficient system for classifying handwritten digits based on pixel-level information.

The MNIST dataset, a benchmark dataset in the field of machine learning, serves as the foundation for this endeavour. Comprising thousands of grayscale images of handwritten digits ranging from 0 to 9, MNIST presents a challenging yet tractable problem for CNN. By harnessing the hierarchical features learned through successive convolutional and pooling layers, CNN offers a promising approach to extracting discriminative features from raw pixel data, enabling accurate digit classification.

This project not only focuses on achieving high classification accuracy but also delves into the intricacies of CNN architecture design, data preprocessing techniques, and model evaluation strategies. Through rigorous experimentation and analysis, insights will be gleaned into the effectiveness of CNN in handling image classification tasks, particularly in the context of handwritten digit recognition. Ultimately, the goal is to develop a sophisticated CNN model capable of accurately deciphering handwritten digits, contributing to advancements in both computer vision and pattern recognition domains.

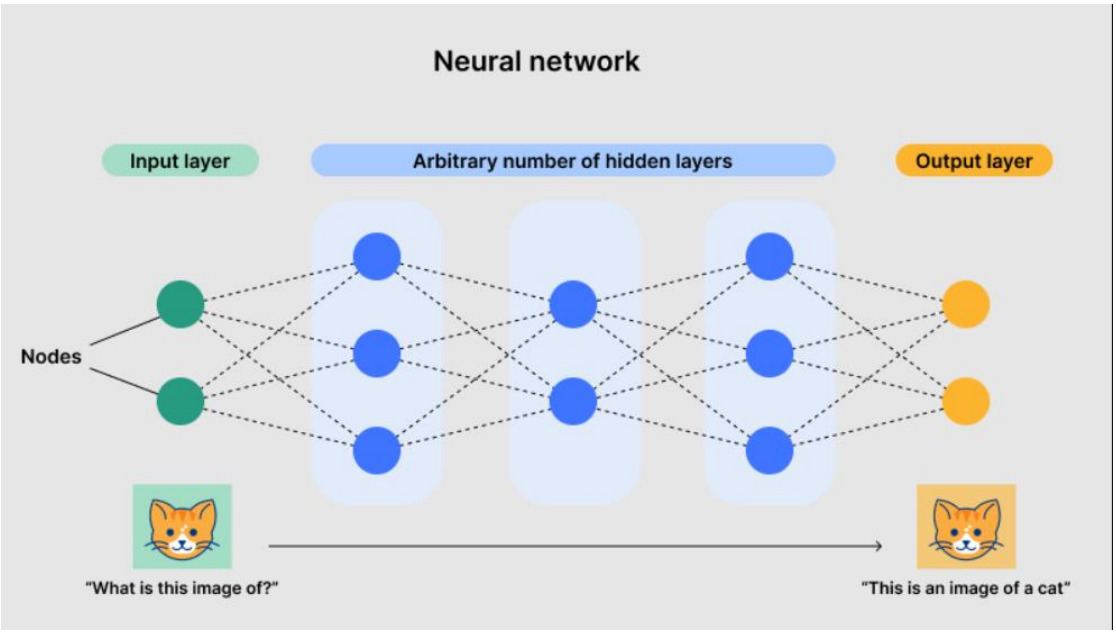
# What is a Neural Network?

A neural network is a method in artificial intelligence that teaches computers to process data in a manner inspired by the human brain. This type of machine learning, known as deep learning, utilizes interconnected nodes or neurons in a layered structure that resembles the human brain, creating an adaptive system that enables computers to learn from their mistakes and continuously improve. [1b]

By mimicking the way biological neurons work together to identify phenomena, weigh options, and arrive at conclusions, neural networks attempt to solve complex problems, such as summarizing documents or recognizing faces, with greater accuracy. [1a]

One of the best-known examples of a neural network is **Google's search algorithm.**
Neural networks are sometimes called artificial neural networks (ANN's) or simulated neural networks (SNNs). They are a subset of machine learning and at the heart of deep learning models.[1a]

## A Neural Network is depicted in the figure below.



[1d]

# Why are neural networks important?

Neural networks can help computers make intelligent decisions with limited human assistance. This is because they can learn and model the relationships between input and output data that are nonlinear and complex. For instance, they can do the following tasks. . [1b]

Make generalizations and inferences
Neural networks can comprehend unstructured data and make general observations without explicit training. [1b]

For instance, they can recognize that two different input sentences have a similar meaning:

- Can you tell me how to make the payment?
- How do I transfer money?

A neural network would know that both sentences mean the same thing. Or it would be able to broadly recognize that **Baxter Road is a place**, but **Baxter Smith is a person's name**. [1b]

# Where Neural Networks are used?

## Computer vision:

Computer vision is the ability of computers to extract information and insights from images and videos. With neural networks, computers can distinguish and recognize images similar to humans. Computer vision has several applications, such as the following: [1b]

- Visual recognition in self-driving cars so they can recognize road signs and other road users
- Content moderation to automatically remove unsafe or inappropriate content from image and video archives
- Facial recognition to identify faces and recognize attributes like open eyes, glasses, and facial hair
- Image labeling to identify brand logos, clothing, safety gear, and other image details.[1b]

## Speech recognition:

Neural networks can analyze human speech despite varying speech patterns, pitch, tone, language, and accent. Virtual assistants like Amazon Alexa and automatic transcription software use speech recognition to do tasks like these:

- Assist call center agents and automatically classify calls
- Convert clinical conversations into documentation in real time.
- Accurately subtitle videos and meeting recordings for wider content reach. [1b]

## Natural language processing:

Natural language processing (NLP) is the ability to process natural, human-created text. Neural networks help computers gather insights and meaning from text data and documents. [1b]

NLP has several use cases, including in these functions:

- Automated virtual agents and chatbots
- Automatic organization and classification of written data
- Business intelligence analysis of long-form documents like emails and forms
- Indexing of key phrases that indicate sentiment, like positive and negative comments on social media
- Document summarization and article generation for a given topic. [1b]

# How do neural networks work?

**The human brain is the inspiration behind neural network architecture**. Human brain cells, called neurons, form a complex, highly interconnected network and send electrical signals to each other to help humans process information. Similarly, an artificial neural network is made of artificial neurons that work together to solve a problem. **Artificial neurons are software modules, called nodes, and artificial neural networks are software programs** or algorithms that, at their core, use computing systems to solve mathematical calculations. [1b]

## A simple example of Neural Network architecture is described below:

In general, a neural network contains one input layer, one or more hidden layers, and one output layer.[1c]

1. **Input Layer:**

   - Information from the outside world enters the artificial neural network from the input layer. Input nodes process the data, analyze or categorize it, and pass it on to the next layer. [1b]
   - Neurons in the input layer serve to encode a vector of scalar values that represent the input to a neural network (i.e., each neuron encodes a scalar value and has no input).
   - Therefore, an input layer containing n0 neurons can be represented as a vector of n0 scalar values. [1c]

$$a^{[0]} = \begin{bmatrix} a_1^{[0]} \\ a_2^{[0]} \\ \vdots \\ a_{n0}^{[0]} \end{bmatrix} , \qquad [1c]$$

where superscript [0] represents the index of a given layer, i.e., [0] for the input layer, [1] for the first hidden layer, and goes on.

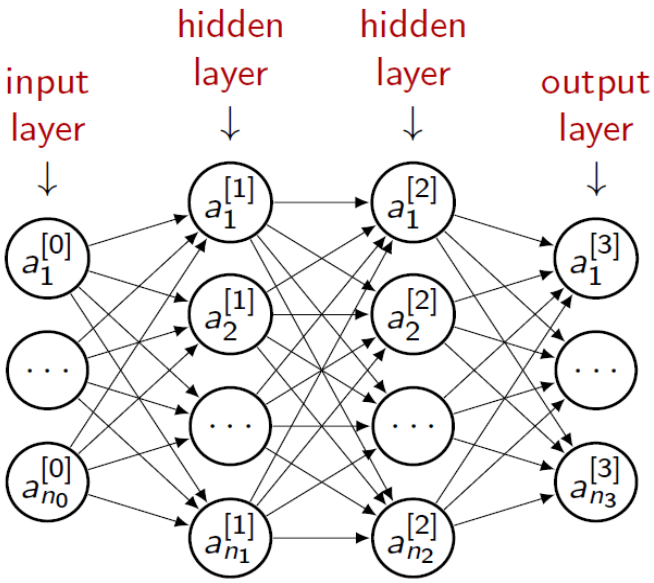
2. **Hidden Layer :**

   Hidden layers take their input from the input layer or other hidden layers. Artificial neural networks can have a large number of hidden layers. Each hidden layer analyses the output from the previous layer processes it further, and passes it on to the next layer.[1b]

3. **Output Layer:**

   The output layer gives the final result of all the data processing by the artificial neural network. [1b] If we perform a binary classification, the output layer may contain only

one node, and its value may represent the probability of one class versus another, if we perform a multinomial classification, the number of neurons in the output layer may be equal to the number of classes. [1c]




[1c]

**The figure above illustrates the example of Neural Network architecture**

# Other Types of Neural Networks:

## 1. Feed-forward neural networks:

A feedforward neural network is the simplest type of neural network where information moves in one direction—from input nodes, through hidden nodes (if any), to output nodes, with no cycles or loops in the network. Commonly used for straightforward tasks like image classification and pattern recognition, every node in one layer is connected to every node in the next layer. This network processes data from the input to the output and uses a feedback process to improve predictions over time.[1b]

## 2. Recurrent neural networks:

This complex neural network model works by saving the output generated by its processor nodes and feeding them back into the algorithm. This process enables recurrent neural networks to enhance their prediction capabilities.
In this neural network model, each node behaves like a memory cell. These cells work to ensure intelligent computation and implementation by processing the data they receive. However, what sets this model apart is its ability to recollect and reuse all processed data. [1e]

A strong feedback loop is one of the critical features of a recurrent neural network. These neural network solutions can 'self-learn' from their mistakes. If an incorrect prediction is made, the system learns from feedback and strives to make the correct prediction while passing the data through the algorithm the second time.
Recurrent neural networks are commonly used in text-to-speech applications and for sales forecasting and stock market predictions. [1e]

### 3. Modular neural networks:

Modular neural networks feature a series of independent neural networks whose operations are overseen by an intermediary. Each independent network is a 'module' that uses distinct inputs to complete a particular part of the larger network's overall objective.

The modules do not communicate with one another or interfere with each other's processes while computation occurs. This makes performing extensive and complex computational processes more efficient and quick. [1e]

# What is Hand digit classification?

Handwritten digit classification in machine learning refers to the task of automatically recognizing and categorizing handwritten digits into their respective classes ,typically digits from 0 to 9. This task falls under the broader category of image classification, where the goal is to train a model to correctly identify and classify images based on their visual features.

The process of handwritten digit classification involves the following steps:

- **Data Collection and Preprocessing:** Acquiring a dataset of handwritten digit images, such as the MNIST dataset, which consists of thousands of grayscale images of digits. The images are usually preprocessed to standardize size, orientation, and intensity, making them suitable for analysis.

- **Feature Extraction:** Extracting meaningful features from the digit images that can be used by the machine learning model for classification. In the case of handwritten digits, common features include pixel intensities derived from image processing techniques.

- **Model Training:** Using a machine learning algorithm to train a classification model on a labeled dataset. Popular algorithms for this task include neural networks like Gaussian naive Bayes classifiers, and convolutional neural networks – CNNs.

- **Model Evaluation:** Assessing the performance of the trained model using evaluation metrics such as accuracy, precision, recall, and F1-score. The model is typically evaluated on a separate test set to measure its generalization ability.

# MNIST Dataset:

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. [1g]

MNIST dataset is a dataset of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau. Each image is labeled with the digit it represents. This set has been studied so much that it is often called the **"Hello World"** of Machine Learning: whenever people come up with a new classification algorithm, they are curious to see how it will perform on MNIST. Whenever someone learns Machine Learning, sooner or later they tackle MNIST. There are 70,000 images, and each image has 784 features. This is because each image is 28×28 pixels, and each feature simply represents one pixel's intensity, from 0 (white) to 255 (black). [1f]

[1f]

The above image shows some digits from the MNIST dataset.

Always make a test set and put it aside before carefully going over the data. The MNIST dataset is actually already divided into two sets: and the first 60,000 images are a training set and the last 10,000 images are a test set. [1f]

# Convolutional Neural Networks

If we want to apply a neural network to image classification, it would be convenient to represent a digital image as a vector that can be presented to the input layer.
However, applying this vector directly to a fully connected feedforward neural network would not be an appropriate solution.
The main problems can be summarized as follows:

- The number of neurons in the input layer of a given neural network is constant. On the other hand, the input image may vary in size, which implies that the vector representing a given input will also vary in size.
- Even if all input images were of constant size, the generated neural network would not be necessarily robust to even small translations of the input image (e.g., shifting pixels of a training set image by a small amount in the same direction).
- The number of parameters in such a neural network would be significant. For, each neuron in the first hidden layer is assigned M*N+1 parameters (i.e., M*N weights for each pixel of the input image and a bias).

To overcome these problems, the classification of images is performed by a special type of neural network **Convolutional Neural Networks**. [1h]

## What is Convolution in CNN?

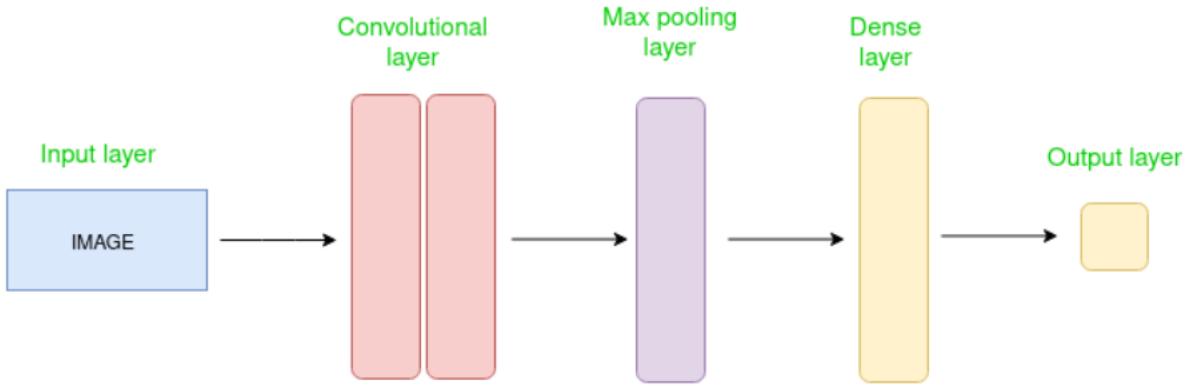
Convolution in the context of Convolutional Neural Networks (CNNs) is a mathematical operation used to extract features from input data, such as images. It involves applying a filter (also known as a kernel) over the input data to produce a feature map. The filter slides over the input data, and at each position, it performs element-wise multiplication and then sums the results to create a single value in the feature map. This process captures local patterns such as edges, textures, and other relevant features, allowing the network to learn spatial hierarchies and structures in the data. Convolution operations help preserve the spatial relationship between pixels by learning image features using small input data squares, making CNNs particularly effective for tasks involving images and visual data.

## What is Convolutional Neural Networks?

A Convolutional Neural Network (CNN) is a type of deep learning model specifically designed for processing structured grid data, such as images. CNNs are composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply convolution operations to the input data, extracting hierarchical features that become progressively more complex in deeper layers. Pooling layers reduce the dimensionality of the feature maps, making the model computationally efficient and more robust to spatial variations in the input. Fully connected layers at the end of the network integrate the extracted features to perform classification or regression tasks. CNNs are widely used in computer vision applications, including image recognition, object detection, and image segmentation, due to their ability to automatically and adaptively learn spatial hierarchies of features from raw image data.

# Architecture of Convolutional Neural Networks

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers. [1i]




[1i]

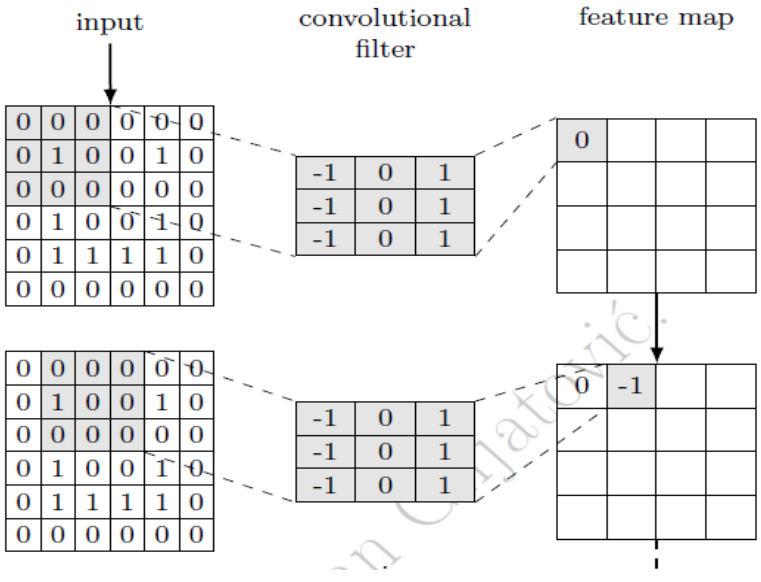
The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent. [1i]

# Layers of Convolutional Neural Networks

1. **Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. [1i]

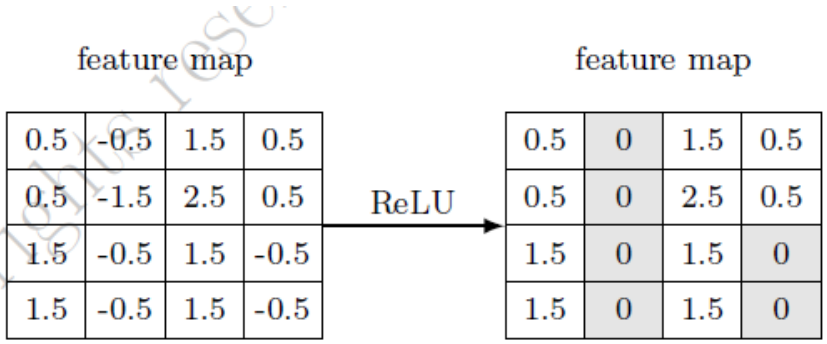
2. **Convolutional Layers:** A filter is convolved with the input image and the bias term is added. The filter is usually of dimension 3X3 and its elements are estimated by means of the backpropagation algorithm. The resulting matrix is referred to as the feature map.

Below is the figure that shows how the filters or kernels are applied to the input image in the convolution layer. [1h]




[1h]

3. **Activation or Detector Layer:** The feature map is passed through a nonlinear activation function, the Rectified Linear Unit function -RELU. [1h]



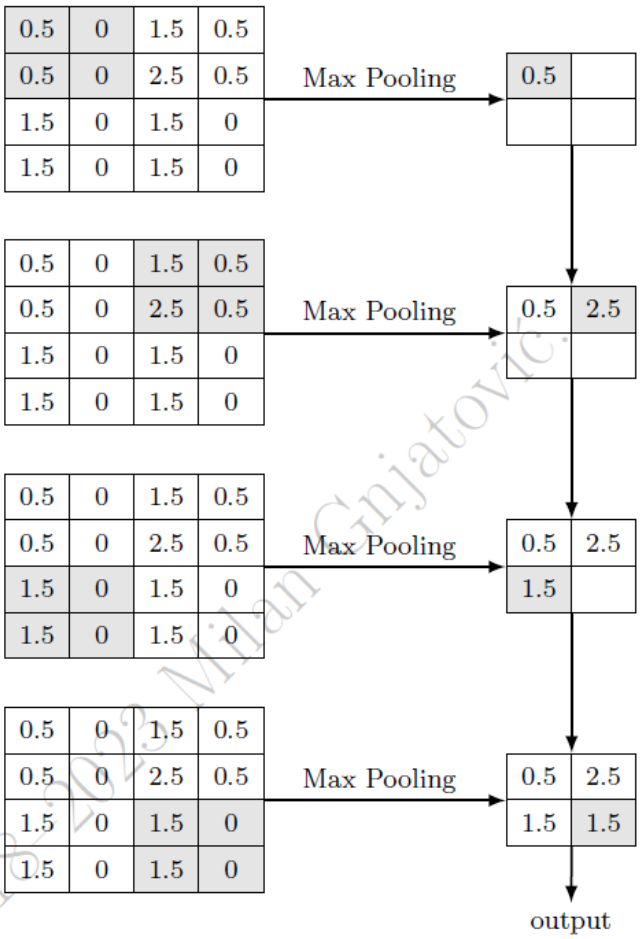

| | | | |
|---|---|---|---|
| 0.5 | -0.5 | 1.5 | 0.5 |
| 0.5 | -1.5 | 2.5 | 0.5 |
| 1.5 | -0.5 | 1.5 | -0.5 |
| 1.5 | -0.5 | 1.5 | -0.5 |

| | | | |
|---|---|---|---|
| 0.5 | 0 | 1.5 | 0.5 |
| 0.5 | 0 | 2.5 | 0.5 |
| 1.5 | 0 | 1.5 | 0 |
| 1.5 | 0 | 1.5 | 0 |

[1h]

4. **Pooling layer:** An additional filter is convolved with the feature map obtained in the previous stage. It should be noted that this filter is slid over the feature map without overlapping with itself. This stage often applies the Max Pooling function, which returns the maximum value in a rectangular neighborhood covered by the filter. Additional pooling functions are the Mean Pooling, weighted average of a rectangular neighborhood, L2 norm of a rectangular neighborhood, etc. In the general case, pooling generates a representation that is approximately invariant to small translations of the input. This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. [1h]

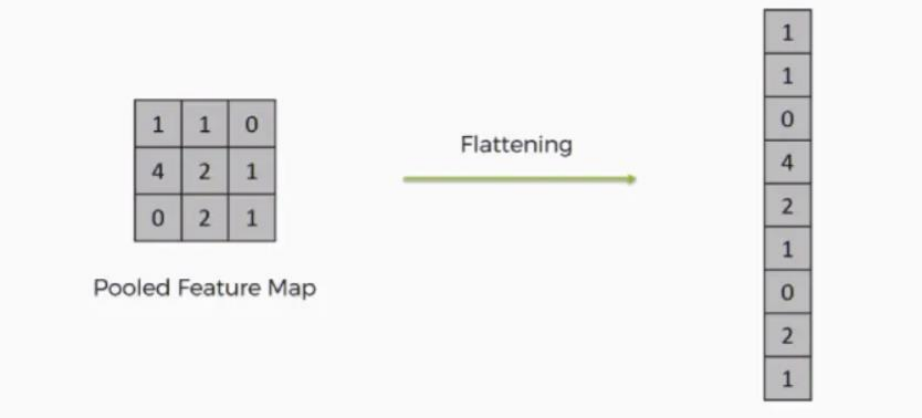
An illustration of the use of max pooling is provided below.




[1h]

**Max Pooling:** Takes the maximum value in each patch of the feature map.
**Average Pooling:** Takes the average value of each patch.

6. **Flattening Layer:** Flattening is used to convert all the resultant 2-dimensional arrays from pooled feature maps into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer to classify the image. [1j]




[1j]

7. **Fully Connected Layer:** The Fully Connected (FC) layer consists of weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture. [1j]

8. **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or SoftMax which converts the output of each class into the probability score of each class. [1i]

# Important Hyperparameters of Convolution Neural Networks:

1. **Padding**: Padding in the context of Convolutional Neural Networks (CNNs) refers to the process of adding extra pixels to the border of an input image. This is done for several reasons, including preserving the spatial dimensions of the input after the convolution operation and allowing for better feature extraction at the edges of the image. Padding is an important hyperparameter in the design of convolutional layers.

**Types of Padding:**

**a) Valid Padding (No Padding):**

- No additional pixels are added to the input image.
- The dimensions of the output feature map are smaller than the input.
- Only applies the filter where it completely overlaps with the input.

Example: If you have a 5x5 input and a 3x3 filter, the output will be (5-3+1) x (5-3+1) = 3x3.

**b) Same Padding or Zero Padding to Maintain Size:**

- Padding is added so that the output feature map has the same spatial dimensions as the input.
- This is achieved by adding enough zeros around the border of the input.
- Ensures the filter can be applied to all regions of the input, including the borders.

Example: If you have a 5x5 input and a 3x3 filter, the output will be padded so the result is still 5x5.

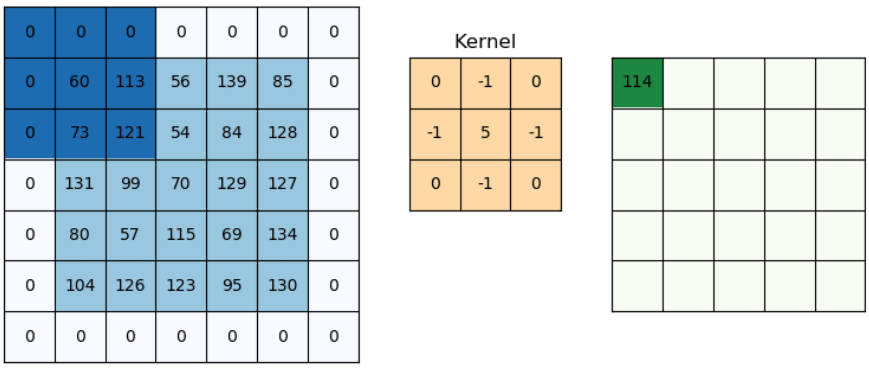
**c) Full Padding:**

- Adds enough padding such that the filter can cover all regions of the input, even those where it extends beyond the original boundaries.
- Results in an output feature map that is larger than the input.
- Less commonly used compared to "valid" and "same" padding.

**Practical Example:**

Consider a 5x5 input image and a 3x3 filter.

- **Valid Padding:** No extra pixels added. The filter slides over the input, resulting in a 3x3 feature map.
- **Same Padding**: Padding added to maintain input dimensions. The output feature map will also be 5x5.



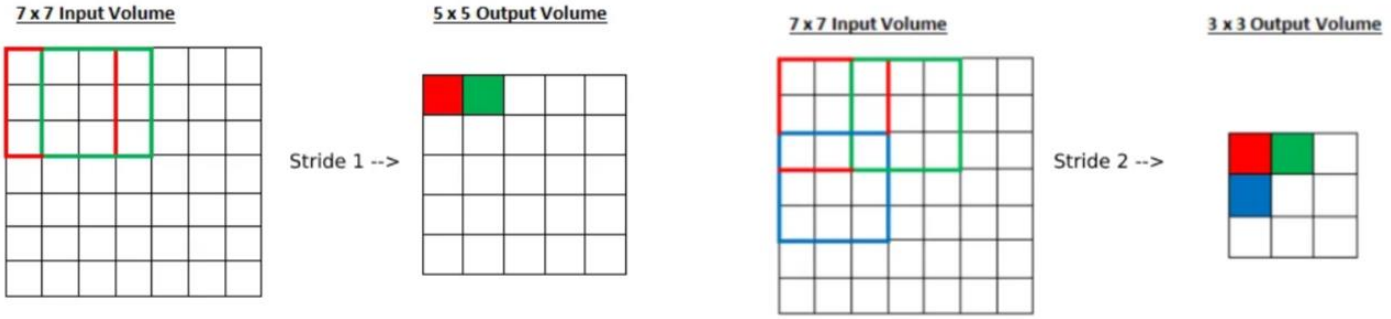

[1j]

The above figure illustrates how calculation is done for the padded layer.

**Default Behavior of Padding:**

If padding is not explicitly mentioned in the configuration of a convolutional layer, most frameworks default to no padding, meaning no padding is added. So explicitly padding is to be mentioned.

2. **Stride:** Stride is a parameter of the neural network's filter that modifies the amount of movement over the image. we had stride 1 so it will take one by one. If we give stride 2 then it will take value by skipping the next 2 pixels. [1j]




**Default Behavior of Strides:**
if strides are not explicitly mentioned in the configuration of a convolutional layer or a pooling layer, a default stride value is typically applied.

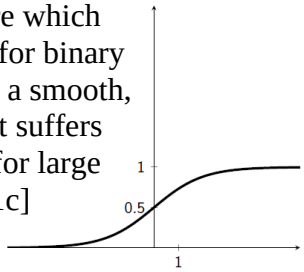
- In **Convolution Layers**, using most deep learning frameworks such as TensorFlow, Keras, and PyTorch, the **default stride value for convolutional layers is 1**.
- This means that the filter will move one pixel at a time both horizontally and vertically across the input**.**
- Similarly, **in pooling layers** (like max pooling or average pooling), **the default stride value is often equal to the size of the pooling window**.
- For instance, if you use a 2x2 pooling window, the default stride will typically be 2.

3. **Activation Function:** An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction.[1j]

There are several commonly used activation functions such as -

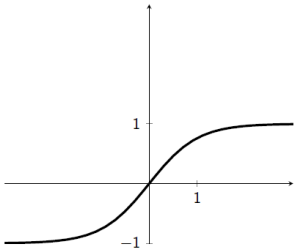
- **The sigmoid function -** For a binary classification in the CNN model. The sigmoid function is a mathematical function defined as shown in figure which maps any real-valued number to a value between 0 and 1,making it useful for binary classification tasks in neural networks. It has an S-shaped curve, making it a smooth, non-linear function that can help model complex relationships. However, it suffers from the vanishing gradient problem, where gradients become very small for large positive or negative inputs, slowing down the training of deep networks. [1c]

$$g(z) = \frac{1}{1 + e^{-z}}$$

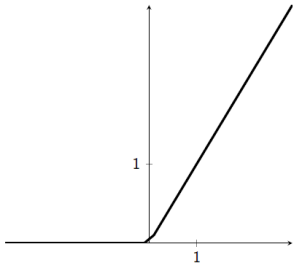



- **The hyperbolic tangent (tanh) function -** the tanh function, which is similar to the sigmoid function, is applied more often as an activation function (e.g., in recurrent neural networks). [1c] The only difference is that it is symmetric around the origin. The range of values, in this case, is from -1 to 1.[1j]

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



$$g(z) = \max(0, z) = \begin{cases} z, & \text{if } z > 0, \\ 0, & \text{otherwise} \end{cases}$$




- **The Rectified Linear Unit (ReLU) function -** The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.[1j] It is widely used in neural networks because it helps mitigate the vanishing gradient problem, allowing models to learn faster and perform better. However, ReLU can suffer from the "dying ReLU" problem, where neurons can become inactive and only output zero, especially when learning rates are high.[1c]

- **The Softplus function -** The Softplus function is similar to ReLU but significantly less applied since it requires a more complex calculation.[1c]

- **The Softmax function -** It is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.

  ReLU of tanh are often used as activation functions for hidden layers and softmax for the output layer.

4. **Optimizer:** An optimizer in neural networks is an algorithm that adjusts the model's weights to minimize the loss function during training. It determines how the model updates its parameters based on the gradients of the loss function concerning the weights and biases**.** Common optimizers include Gradient Descent, Adam, and RMSprop, each with its method of navigating the parameter space to improve model performance.
   Gradient descent is an optimization algorithm used to minimize the loss function in machine learning models, including neural networks. It iteratively adjusts the model's parameters, such as weights and biases, to find the optimal set that reduces the error between the predicted and actual values. The process begins with an initial set of parameters, and in each iteration, the algorithm computes the gradient, which is the vector of partial derivatives of the loss function concerning the parameters. This gradient indicates the direction and rate of the steepest ascent; however, since the goal is to minimize the loss, the parameters are updated in the opposite direction of the gradient, effectively descending the slope of the loss function. The step size of each update is determined by the learning rate, a hyperparameter that needs to be carefully chosen: too large, and the algorithm may overshoot the minimum; too small, and the convergence may be very slow or get stuck in local minima. Variants of gradient descent, such as stochastic gradient descent (SGD), which updates parameters for each training example, and mini-batch gradient descent, which updates parameters using small batches of data, help balance the tradeoff between convergence speed and accuracy.
   Advanced versions like Adam, RMSprop, and Adagrad incorporate adaptive learning rates and momentum to accelerate convergence and improve performance on complex, high-dimensional data.

5. **Loss Functions:** A loss function in neural networks quantifies the difference between the predicted output and the actual target values. It provides a measure of how well or poorly the model is performing, guiding the optimization process. Common loss functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks.

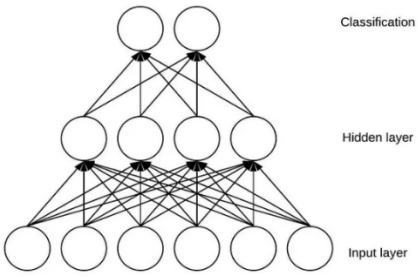
**For Regression**

- **Mean Squared Error (MSE):**
  - ❖ Measures the average of the squares of the errors, that is, the average squared difference between the estimated values and the actual value.
  - ❖ This is one of the simplest and most effective cost or loss functions that we can use. It can also be called the **quadratic cost function or the sum of squared errors**.
- **Mean Absolute Error (MAE):**
  - ❖ Measures the average of the absolute differences between predicted values and actual values.
- **Huber Loss:**
  - ❖ Combines the best properties of MAE and MSE by being less sensitive to outliers than MSE and more sensitive to small errors than MAE.

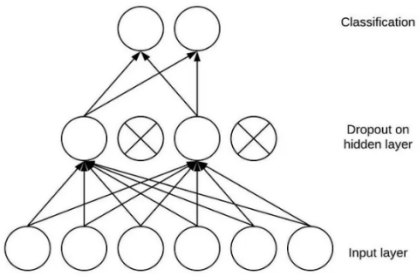
**For Classification**

- **Binary Cross entropy:**
  - ❖ Used for binary classification problems.
  - ❖ Measures the performance of a classification model whose output is a probability value between 0 and 1.
- **Categorical Cross entropy:**
  - ❖ Used for multi-class classification problems.
  - ❖ Measures the performance of a classification model whose output is a probability distribution across multiple categories.
- **Sparse Categorical Cross entropy:**
  - ❖ Similar to categorical cross-entropy but for integer labels instead of one-hot encoded labels.

6. **Dropout:** Another typical characteristic of CNNs is a Dropout layer. The Dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others.[1j]




Without Dropout




With Dropout

7. **Epochs:** Epochs in neural networks refer to the number of complete passes(Iterations) through the entire training dataset during the training process. Each epoch allows the model to learn and adjust its weights based on the training data. Multiple epochs can improve model accuracy by iteratively refining the weights.

8. **Verbose:** It is a parameter that controls the level of logging and output during the training process. Setting verbose to 0 means no output, 1 provides progress bars, and 2 gives more detailed logging for each epoch, helping users monitor the training process.

9. **Batch Size:** Batch size is the number of training samples processed before the model updates its internal parameters. Smaller batch sizes offer more frequent updates and can lead to a more stable training process. Larger batch sizes make better use of hardware acceleration and can improve training speed but might require more memory.

# Evaluation Measures:

Evaluation measures for Gaussian Naive Bayes (GNB) on the MNIST dataset are important for assessing the performance of the classifier in a multi-class classification task, such as recognizing handwritten digits. Here are the evaluation measures commonly used for GNB on the MNIST dataset:

1. **Confusion Matrix:** A much better way to evaluate the performance of a classifier is to look at the confusion matrix. The confusion matrix is a tabular representation that summarizes the performance of a classification model by presenting the counts of true positive, true negative, false positive, and false negative predictions for each class. The general idea is to count the number of times instances of class A are classified as class B. Each row in a confusion matrix represents an actual class, while each column represents a predicted class. For example, to know the number of times the classifier confused images of 5s with 3s, you would look in the 5th row and 3rd column of the confusion matrix. A perfect classifier would have only true positives and true negatives, so its confusion matrix would have nonzero values only on its main diagonal (top left to bottom right ) [1f]

2. **Accuracy:** Accuracy measures the proportion of correctly classified samples out of the total number of samples. It is calculated as:
   Accuracy = Number of correct predictions/ Total Number of predictions.

3. **Precession:** You can find a lot of information in the confusion matrix, but occasionally you might want a shorter metric. An interesting one to look at is the accuracy of the positive predictions; this is called the precision of the classifier. [1f]
   Precision measures the ability of the classifier to correctly identify positive samples (true positives) out of all samples predicted as positive. It is calculated as:

$$P = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} .$$

[1k]

   Precision is useful when the cost of false positives is high.

4. **Recall:** Recall measures the ability of the classifier to correctly identify positive samples (true positives) out of all actual positive samples. It is calculated as:

$$R = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \cdot \quad [1k]$$

5. **F1 Score**: It is often convenient to combine precision and recall into a single metric called the F1 score, in particular, if you need a simple way to compare two classifiers. The F1 score is the harmonic mean of precision and recall. Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values. As a result, the classifier will only get a high F1 score if both recall and precision are high.[1f]
   This Harmonic mean provides a balance between the two metrics.
   It is calculated as:

$$F_1 = \frac{2}{P^{-1} + R^{-1}} = \frac{2PR}{P + R} \cdot \text{ [1k]}$$

6. **Macro-average Metrics:** In multi-class classification tasks like MNIST, macro-average metrics compute the average of individual metrics (precision, recall, F1-score) calculated for each class. This approach treats all classes equally and is useful for assessing overall classifier performance across multiple categories.

# Explaining the Program Functionality (What is going in the code):

Each step of what is happening in the programme is explained below, following the theoretical explanation. Having completed numerous trials, I have experimented with various hyperparameters and functions. Select a programme that produces quality results and explain it here. (Program File: CV_Task1_Trail2_NikhilKumar.ipynb)

## 1. Data Loading and Preprocessing

a) **Data Import:** The program begins by importing the MNIST dataset from local CSV files, containing images of handwritten digits and their corresponding labels. The training and testing data are loaded into separate Pandas Data Frames.

b) **Feature and Label Separation:** The labels (digit values) are separated from the images in both the training and testing datasets. The image data are then reshaped to the required format for the CNN, which is (number of images, 28, 28, 1). This format represents the height, width, and channel (grayscale) of the images.

c) **One-Hot Encoding:** The labels are converted to one-hot encoded vectors. This step transforms the labels into a binary matrix representation, which is necessary for the categorical classification task.

## 2. Building the CNN Model

a) **Model Initialization:** A Sequential model is initialized using TensorFlow's Keras API. This type of model is appropriate for stacking layers sequentially.

b) **Adding Convolutional Layers:** Eight convolutional layers are added to the model. Each layer has different hyperparameters such as the number of filters, kernel size, and activation function. The first four layers are followed by MaxPooling layers, which down-sample the feature maps to reduce spatial dimensions and computation.

c) **Skipping Pooling in Later Layers:** For the last four convolutional layers, pooling is omitted to avoid reducing the spatial dimensions excessively, which might result in losing critical information.

d) **Flattening Layer:** After the convolutional layers, a flattened layer is added to convert the 2D feature maps into a 1D vector, preparing it for the fully connected layers.

e) **Fully Connected Layer:** A Dense layer with 200 neurons and ReLU activation is added to learn complex representations.

f) **Output Layer:** The final Dense layer has 10 neurons with a softmax activation function, which outputs the probabilities for each of the 10-digit classes.

## 3. Model Compilation and Training

a) **Compilation:** The model is compiled with the Adam optimizer, which is an advanced gradient descent optimization algorithm. The loss function used is a **mean squared error** (MSE), and the performance metric is accuracy. This is the function where the model is prepared for training.

b) **Training:** The model is trained on the training dataset for 12 epochs with a batch size of 500. During training, the model's performance on both the training and validation (testing) datasets is logged for each epoch.

## 4. Evaluation and Metrics Calculation

a) **Final Training Accuracy:** After training, the final accuracy on the training data is printed to assess the model's performance during training.

b) **Prediction:** The trained model is used to predict the classes of the test images. The predicted probabilities are converted to class labels.

c) **Confusion Matrix:** A confusion matrix is computed to show the performance of the classification model. It provides a detailed breakdown of true positive, false positive, true negative, and false negative predictions for each class.

5. **Performance Metrics:**

- **Accuracy:** The overall percentage of correctly classified instances.
- **Precision for each class:** The ratio of true positive predictions to the sum of true positive and false positive predictions for each class.
- **Macroaverage Precision:** The average precision across all classes.
- **Recall for each class:** The ratio of true positive predictions to the sum of true positive and false negative predictions for each class.
- **Macroaverage Recall:** The average recall across all classes.
- **F1 Score for each class:** The harmonic mean of precision and recall for each class.
- **Macroaverage F1 Score:** The average F1 score across all classes.

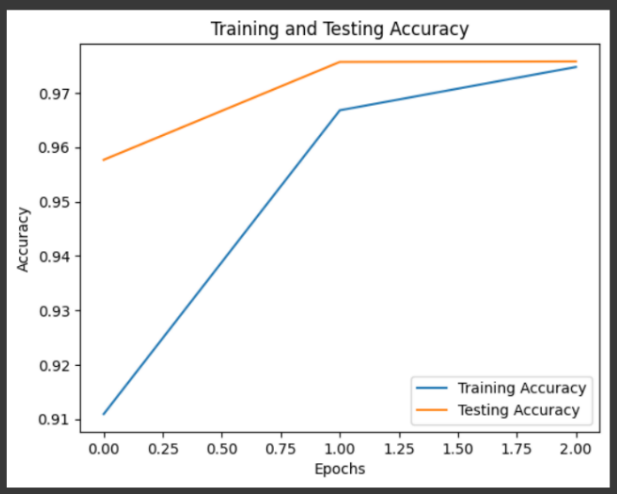
6. **Visualization**

A plot of the training and testing accuracy over the epochs is generated to visualize the model's performance and to observe how well it generalizes to the test data.

# Output Results of Various Trails with Observations:

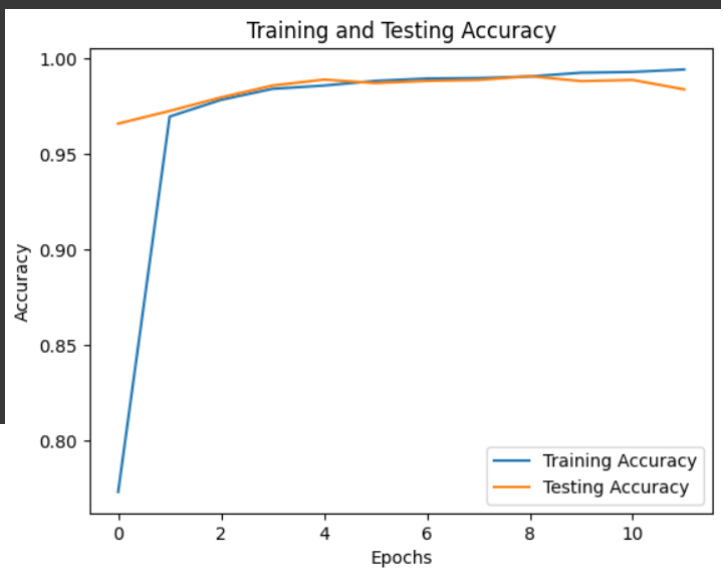
**Trial 1:- With preprocessing Normalization, All 3 Convolutional Layers with Same Hyperparameters. Activation Function - Relu, Optimizer - Adam, Loss function – mean squared error. Batch Size = 32, Epochs = 3.**
**The time Taken for the Whole execution is approximately around 3 to 5 minutes**

```
Trial 1 :-With preprocessing Normalization, All 3 Convolutional Layers with Same HyperParameters
 Activitation Function - Relu, Optimizer - Adam, Loss function - mean_squared_error.
 Batch Size = 32, Epochs = 3. Time Taken for Whole execution is approximately around 3 to 5 minutes

 Also the detailed logging of the training process is Shown Below:
Epoch 1/3
1875/1875 [==============================] - 55s 28ms/step - loss: 0.0130 - accuracy: 0.9109 - val_loss: 0.0066 - val_accuracy: 0.9577
Epoch 2/3
1875/1875 [==============================] - 44s 24ms/step - loss: 0.0051 - accuracy: 0.9668 - val_loss: 0.0038 - val_accuracy: 0.9757
Epoch 3/3
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0040 - accuracy: 0.9748 - val_loss: 0.0036 - val_accuracy: 0.9758

Training accuracy: 0.9747662544250488
313/313 [==============================] - 2s 6ms/step

 Confusion Matrix:
[[ 960    1    4    1    2    3    4    4    1    0]
 [   0 1125    1    4    0    2    0    3    0    0]
 [   0    1 1015    1    1    0    1   12    1    0]
 [   0    0    4 1000    0    1    0    5    0    0]
 [   1    2    0    0  971    0    2    3    0    3]
 [   2    0    2   14    1  866    1    3    2    1]
 [   3    6    1    0   17    4  926    0    1    0]
 [   0    1   16    2    0    1    0 1007    0    0]
 [   0    1    6   13    8    4    0    9  922   11]
 [   3    0    0    8   11    9    1   10    2  965]]

 Accuracy: 0.9757975797579758

 Precision for each class:
[0.99071207 0.98944591 0.96758818 0.95877277 0.96043521 0.97303371
 0.99037433 0.95359848 0.99246502 0.98469388]

 Macroaverage Precision: 0.9761119565287855

 Recall for each class:
[0.97959184 0.99118943 0.98352713 0.99009901 0.98879837 0.97085202
 0.96659708 0.9805258  0.94661191 0.95639247]

 Macroaverage Recall: 0.975418505233641

 F1 score for each class:
[0.98512057 0.9903169  0.97549255 0.97418412 0.97441044 0.97194164
 0.97834126 0.9668747  0.96899632 0.97033685]

 Macroaverage F1 Score:
 0.9756015355157827
```



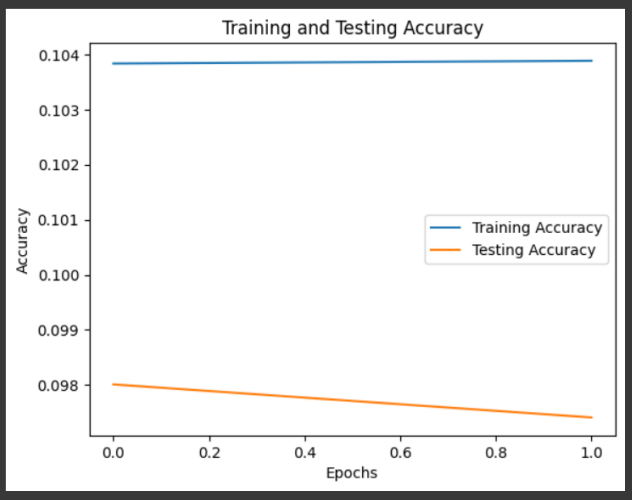
**Trial 2:- NO  Normalization of the dataset, 8 Convolutional Layers with all Different Hyperparameters and without Pooling in the last 4 layers.**
**Activation Function - Relu, Optimizer - Adam, Loss function – mean squared error.**
**Batch Size = 500, Epochs = 12. The time Taken for the Whole execution is approximately 20 to 22 minutes**

```
Trial 2 :- NO  Normalization of dataset, 8 Convolutional Layers with all Different HyperParameters and without Pooling in last 4 layers.
 Activitation Function - Relu, Optimizer - Adam, Loss function - mean_squared_error.
 Batch Size = 500, Epochs = 12. Time Taken for Whole execution is approximately around 20 to 22 minutes

 Also the detailed logging of the training process is Shown Below (Without Progress Bar):
Epoch 1/12
120/120 - 116s - loss: 0.0292 - accuracy: 0.7734 - val_loss: 0.0052 - val_accuracy: 0.9660 - 116s/epoch - 967ms/step
Epoch 2/12
120/120 - 111s - loss: 0.0047 - accuracy: 0.9697 - val_loss: 0.0042 - val_accuracy: 0.9726 - 111s/epoch - 928ms/step
Epoch 3/12
120/120 - 114s - loss: 0.0034 - accuracy: 0.9784 - val_loss: 0.0031 - val_accuracy: 0.9797 - 114s/epoch - 947ms/step
Epoch 4/12
120/120 - 111s - loss: 0.0025 - accuracy: 0.9842 - val_loss: 0.0022 - val_accuracy: 0.9859 - 111s/epoch - 921ms/step
Epoch 5/12
120/120 - 113s - loss: 0.0022 - accuracy: 0.9859 - val_loss: 0.0016 - val_accuracy: 0.9890 - 113s/epoch - 946ms/step
Epoch 6/12
120/120 - 112s - loss: 0.0018 - accuracy: 0.9883 - val_loss: 0.0021 - val_accuracy: 0.9872 - 112s/epoch - 935ms/step
Epoch 7/12
120/120 - 112s - loss: 0.0016 - accuracy: 0.9895 - val_loss: 0.0019 - val_accuracy: 0.9883 - 112s/epoch - 932ms/step
Epoch 8/12
120/120 - 114s - loss: 0.0016 - accuracy: 0.9898 - val_loss: 0.0018 - val_accuracy: 0.9888 - 114s/epoch - 951ms/step
Epoch 9/12
120/120 - 112s - loss: 0.0015 - accuracy: 0.9906 - val_loss: 0.0015 - val_accuracy: 0.9908 - 112s/epoch - 932ms/step
Epoch 10/12
120/120 - 109s - loss: 0.0012 - accuracy: 0.9926 - val_loss: 0.0017 - val_accuracy: 0.9882 - 109s/epoch - 907ms/step
Epoch 11/12
120/120 - 115s - loss: 0.0011 - accuracy: 0.9930 - val_loss: 0.0016 - val_accuracy: 0.9888 - 115s/epoch - 962ms/step
Epoch 12/12
120/120 - 115s - loss: 9.2649e-04 - accuracy: 0.9943 - val_loss: 0.0025 - val_accuracy: 0.9839 - 115s/epoch - 959ms/step

Training accuracy: 0.9942832589149475
313/313 [==============================] - 6s 17ms/step

 Confusion Matrix:
[[ 971    0    6    0    0    0    1    0    2    0]
 [   0 1126    3    2    0    0    0    4    0    0]
 [   0    0 1030    0    0    0    0    2    0    0]
 [   0    0    5 1003    0    1    0    1    0    0]
 [   3    0    4    0  950    0    1    2    3   19]
 [   1    0    0   11    0  879    1    0    0    0]
 [   5    1    0    0    1    6  942    0    3    0]
 [   1    1   19    1    0    0    0 1004    1    0]
 [   1    0    8    3    0    2    0    0  960    0]
 [   3    0    3    9    2    5    0    6    8  973]]

 Accuracy: 0.9838983898389839

  Precision for each class:
 [0.9857868  0.99822695 0.9554731  0.97473275 0.99685205 0.98432251
  0.9968254  0.98527969 0.9825998  0.98084677]

  Macroaverage Precision: 0.9840945807804194

  Recall for each class:
 [0.99081633 0.99207048 0.99806202 0.99306931 0.96741344 0.98542601
  0.98329854 0.97760467 0.98562628 0.96432111]

  Macroaverage Recall: 0.9837708190277284

  F1 score for each class:
 [0.98829517 0.9951392  0.97630332 0.9838156  0.98191214 0.98487395
  0.99001576 0.98142717 0.98411071 0.97251374]

  Macroaverage F1 Score: 0.9838406763992188
```



**Trial 3:- NO Normalization, 8 Convolutional Layers with all Different Hyperparameters and added strides in Pooling Layer, Implemented dropout Technique**
**Batch Size = 1, Epochs = 2. The time Taken for the Whole execution is approximately around 40 minutes to 1 hour. Activation Function - Relu, Optimizer - Adam, Loss function – mean squared error.**

```
Trial 3 :- NO  Normalization, 8 Convolutional Layers with all Different HyperParameters and added strides in Pooling Layer,Implemented droupOut Techinque
 Batch Size = 1, Epochs = 2.  Time Taken for Whole execution is approximately around 40 to 50 minutes
Activitation Function - Relu, Optimizer - Adam, Loss function - mean_squared_error.

 Also the detailed logging of the training process is Shown Below:
Epoch 1/2
59999/59999 [==============================] - 2086s 35ms/step - loss: 0.0902 - accuracy: 0.1038 - val_loss: 0.0900 - val_accuracy: 0.0980
Epoch 2/2
59999/59999 [==============================] - 2065s 34ms/step - loss: 0.0901 - accuracy: 0.1039 - val_loss: 0.0900 - val_accuracy: 0.0974

Training accuracy: 0.10388506203889847
313/313 [==============================] - 8s 24ms/step

 Confusion Matrix:
[[   0    0    0    0    0    0    0    0  980    0]
 [   0    0    0    0    0    0    0    0 1135    0]
 [   0    0    0    0    0    0    0    0 1032    0]
 [   0    0    0    0    0    0    0    0 1010    0]
 [   0    0    0    0    0    0    0    0  982    0]
 [   0    0    0    0    0    0    0    0  892    0]
 [   0    0    0    0    0    0    0    0  958    0]
 [   0    0    0    0    0    0    0    0 1027    0]
 [   0    0    0    0    0    0    0    0  974    0]
 [   0    0    0    0    0    0    0    0 1009    0]]

 Accuracy: 0.09740974097409741

 Precision for each class:
[0.         0.         0.         0.         0.         0.
 0.         0.         0.09740974 0.        ]

 Macroaverage Precision: 0.009740974097409741

 Recall for each class:
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]

 Macroaverage Recall: 0.1

 F1 score for each class:
[0.         0.         0.         0.         0.         0.
 0.         0.         0.17752666 0.        ]

 Macroaverage F1 Score: 0.017752665633828487
```

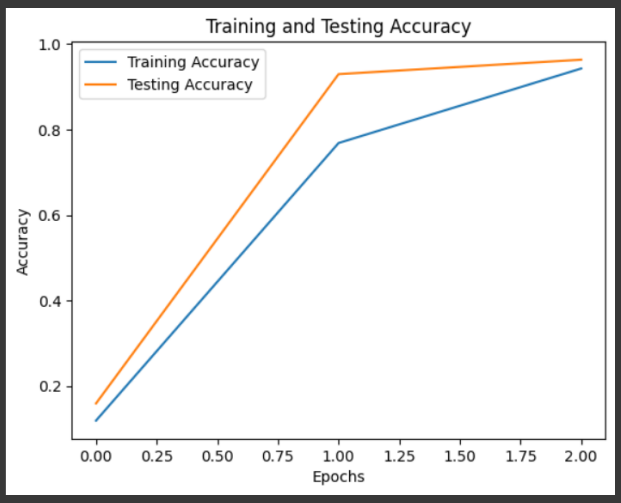



**Trial 4:- With preprocessing Normalization, All 8 Convolutional Layers with different hyperparameters. Different Activation Functions are used, Optimizer - SGD, Loss function – categorical crossentropy. Batch Size = 890, Epochs = 3**

```
Trial 4 :-With preprocessing Normalization, All 8 Convolutional Layers with different HyperParameters
 Different Activitation Function are used, Optimizer - SGD, Loss function - categorical_crossentropy.
 Batch Size = 890, Epochs = 3

 Also the detailed logging of the training process is Shown Below:
Epoch 1/3
68/68 [==============================] - 56s 804ms/step - loss: 2.3180 - accuracy: 0.0993 - val_loss: 2.3085 - val_accuracy: 0.1032
Epoch 2/3
68/68 [==============================] - 53s 784ms/step - loss: 2.3065 - accuracy: 0.1030 - val_loss: 2.3034 - val_accuracy: 0.1135
Epoch 3/3
68/68 [==============================] - 53s 786ms/step - loss: 2.3031 - accuracy: 0.1124 - val_loss: 2.3018 - val_accuracy: 0.1135

Training accuracy: 0.11236853897571564
313/313 [==============================] - 3s 9ms/step

 Confusion Matrix:
[[   0  980    0    0    0    0    0    0    0    0]
 [   0 1135    0    0    0    0    0    0    0    0]
 [   0 1032    0    0    0    0    0    0    0    0]
 [   0 1010    0    0    0    0    0    0    0    0]
 [   0  982    0    0    0    0    0    0    0    0]
 [   0  892    0    0    0    0    0    0    0    0]
 [   0  958    0    0    0    0    0    0    0    0]
 [   0 1027    0    0    0    0    0    0    0    0]
 [   0  974    0    0    0    0    0    0    0    0]
 [   0 1009    0    0    0    0    0    0    0    0]]

 Accuracy: 0.11351135113511351

 Precision for each class:
[0.         0.11351135 0.         0.         0.         0.
 0.         0.         0.         0.        ]

 Macroaverage Precision: 0.011351135113511351

 Recall for each class:
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

 Macroaverage Recall: 0.1

 F1 score for each class:
[0.         0.20388001 0.         0.         0.         0.
 0.         0.         0.         0.        ]

 Macroaverage F1 Score: 0.02038800071851985
```

**Trial 5 :- All 3 Convolutional Layers with the Same Hyperparameters.**
**Different Activation Functions are used, Optimizer - RMSProp, Loss function - Huber.**
**Batch Size = 32, Epochs = 3. The time Taken for Whole execution is approximately around 3 to 4 minutes**

```
Trial 5 :-All 3 Convolutional Layers with Same HyperParameters
 Different Activitation Function are used , Optimizer - RMSProp, Loss function - Huber.
 Batch Size = 32, Epochs = 3. Time Taken for Whole execution is approximately around 3 to 4 minutes

 Also the detailed logging of the training process is Shown Below:
Epoch 1/3
1875/1875 [==============================] - 69s 35ms/step - loss: 0.0449 - accuracy: 0.1190 - val_loss: 0.0446 - val_accuracy: 0.1592
Epoch 2/3
1875/1875 [==============================] - 63s 33ms/step - loss: 0.0160 - accuracy: 0.7687 - val_loss: 0.0054 - val_accuracy: 0.9299
Epoch 3/3
1875/1875 [==============================] - 61s 33ms/step - loss: 0.0043 - accuracy: 0.9430 - val_loss: 0.0029 - val_accuracy: 0.9637

Training accuracy: 0.9429990649223328
313/313 [==============================] - 3s 11ms/step

 Confusion Matrix:
[[ 959    0    2    0    0    6    2    5    3    3]
 [   0 1129    2    1    0    1    1    0    1    0]
 [   3    2  990    5    3    0    2   20    6    1]
 [   0    0    9  969    0    3    0   20    6    3]
 [   1    1    3    0  940    0    5    2    1   29]
 [   6    0    0   13    1  848    8    3    8    5]
 [  10    4    1    0    6    5  927    0    5    0]
 [   1    5   19    6    0    1    0  986    4    5]
 [   0    3    3    6    6    2    3   13  924   14]
 [   6    2    0    5    5    5    0   11   11  964]]

 Accuracy: 0.9636963696369637

 Precision for each class:
[0.97261663 0.98516579 0.96209913 0.9641791  0.97814776 0.97359357
 0.9778481  0.93018868 0.95356037 0.94140625]

 Macroaverage Precision: 0.963880539215217

 Recall for each class:
[0.97857143 0.99471366 0.95930233 0.95940594 0.95723014 0.95067265
 0.96764092 0.9600779  0.9486653  0.95540139]

 Macroaverage Recall: 0.9631681640061446

 F1 score for each class:
[0.97558494 0.9899167  0.96069869 0.9617866  0.96757591 0.9619966
 0.97271773 0.94489698 0.95110654 0.94835219]

 Macroaverage F1 Score: 0.9634632887901364
```




# Observations

1. **Execution Time:** The execution time for Trials 1 and 5 was around 3-5 minutes due to their small, 3-layer convolutional architecture, despite varying hyperparameters like optimizers, loss functions, and activation functions. In contrast, Trials 2, 3, and 4 had much higher execution times. Trial 2 used 8 convolutional layers and trained for 12 epochs, increasing complexity and duration. Trial 3's batch size of 1 led to inefficient, frequent weight updates, hence increasing the training time. Trial 4 used the slower-converging SGD optimizer with categorical_crossentropy, further extending training time.
2. **Accuracy:** In Trial 1, the accuracy was impressive at 97.5% due to its small and simple CNN architecture. However, Trial 2 was the best, achieving an accuracy of 99.42% with more layers and epochs. Trials 3 and 4 had poor accuracy due to the inefficiency of a batch size of 1 and the slower convergence of the SGD optimizer. Trial 5 also performed well with an accuracy of 96%, with RMSprop optimizer and Huber loss function. Overall, I would say Trial 2 stands out as the most effective.

**Important Takeaways from the Overall Observations of the Various Trials:**

- The best performance was observed in Trial 2 with 8 convolutional layers, no normalization, and extensive training, achieving the highest accuracy and robust performance metrics.
- Normalization generally improved the training and validation accuracy, as seen in Trials 1 and 4 compared to Trials 3 and 5.
- The choice of activation functions, optimizers, and loss functions significantly impacted the model's performance. For example, Trial 3's poor performance can be attributed to its unconventional setup (batch size of 1 and dropout technique) and loss function choice.
- Larger batch sizes and more epochs typically resulted in better performance but increased training time significantly.

# Understanding the Output (Just with random example)

1. **Epoch Progress:**
    - Epoch 1/10 indicates the first epoch out of a total of 10 epochs.
    - loss: Training loss after the epoch.
    - accuracy: Training accuracy after the epoch.

2. **Training Metrics:**

- 1875/1875 [==============================] - 64s 33ms/step indicates the number of batches processed (1875 batches), the total time taken for the epoch (64s), and the time per step (33ms/step).

3. **Loss and Accuracy:**

- loss: 0.0071 - Accuracy: 0.9531 shows the loss and accuracy for the training data after the first epoch.

4. **Validation Metrics:**

- val_loss: 0.0028 - val_accuracy: 0.9833 shows the loss and accuracy for the validation data after the first epoch.
- val_loss: Validation loss after the epoch.
- val_accuracy: Validation accuracy after the epoch

# Conclusion:

In conclusion, the implemented Convolutional Neural Network (CNN) effectively classifies handwritten digits from the MNIST dataset, achieving satisfactory performance metrics with and without dataset normalization depending on different trials and with a complex architecture comprising eight convolutional layers. The model demonstrates robust learning capabilities by leveraging the Adam optimizer and mean squared error as the loss function, as evidenced by the detailed training logs and evaluation metrics, including accuracy, precision, recall, and F1 scores. The visualization of training and validation accuracy further underscores the model's proficiency in generalizing to unseen data, making it a promising approach for handwritten digit recognition tasks.

# Reference:


1. **Text Reference:**

a) https://www.ibm.com/topics/neural-networks#:~:text=One%20of%20the%20best%2Dknown,heart%20of%20deep%20learning%20models.
b) https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a,that%20resembles%20the%20human%20brain.
c) http://gnjatovic.info/misc/feedforward.neural.networks.pdf
d) https://cf-assets.www.cloudflare.com/slt3lc6tev37/1wkNx98skWwkKAw2XExpQe/33505b0b82e3156fc042bca42a1a2034/neural-network-diagram.png
e) https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/
f) https://github.com/yanshengjia/ml-road/blob/master/resources/Hands%20On%20Machine%20Learning%20with%20Scikit%20Learn%20and%20TensorFlow.pdf Book - Hands-on machine learning with Scikit-learn and TensorFlow. (Since the book was referred from a library, I'm sharing the GitHub reference link for the same book in PDF format.)
g) https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118
h) Lecture Notes - Gnjatovic.image.processing.pdf
i) https://www.geeksforgeeks.org/introduction-convolution-neural-network/
j) https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243

k) Lecture Notes - http://gnjatovic.info/misc/naive.bayesian.classification.pdf

2. **Source Code Reference :**

a) Code Snippets
b) Lecture Notes