Handwritten Digit Recognition Using Convolutional Neural Networks: A Comparative Study with k-Nearest Neighbors

Mustafa Taner TURAN, Adem Efe DEVREZ

Computer Engineering Department

Ankara University

Ankara, Turkey

*Abstract*—Handwritten digit recognition is a pivotal task in computer vision with widespread applications. In this study, we explore two distinct methodologies, Convolutional Neural Networks (CNN) and k-Nearest Neighbors (kNN), for this purpose. Leveraging the MNIST dataset, we trained a CNN model with a specific architecture and evaluated its performance in comparison to kNN models with varying neighbor counts. Our CNN model exhibited robust accuracy, outperforming kNN counterparts in computational efficiency. The kNN models, configured with k values of 1, 3, and 5, demonstrated competitive performance. In manual tests with unseen images, both CNN and kNN models showcased their capabilities in recognizing handwritten digits.

Keywords—digits, image, recognition, Convolutional Neural Network, K-Nearest Neighbors

# Introduction

In recent years, two distinct methodologies have emerged as prominent contenders for solving the handwritten digit recognition problem: Convolutional Neural Networks (CNN) and k-Nearest Neighbors (kNN). CNNs, inspired by the human visual system, have demonstrated exceptional capabilities in capturing hierarchical features, making them particularly well-suited for image-based tasks. On the other hand, kNN leverages proximity-based classification, relying on the similarity of data points in feature space.

This study aims to delve into the comparative analysis of CNN and kNN for handwritten digit recognition, employing the well-established MNIST dataset as the foundation for experimentation. The MNIST dataset comprises a vast collection of grayscale images, each depicting handwritten digits. By comparing neural networks and traditional machine learning approaches, we are trying to emphasis the strengths and weaknesses in each methodology.

The primary objectives of this research include evaluating the accuracy and computational efficiency of CNNs in comparison to kNN models with different neighbor counts. Through a systematic examination of these approaches, we aim to provide insights into the trade-offs between model complexity and recognition performance. Moreover, this study serves as a practical guide for researchers and practitioners seeking to implement effective solutions for handwritten digit recognition.

In the subsequent sections, we elaborate on the methodologies employed, present experimental results, and engage in a discussion of the findings. The ultimate goal is to contribute valuable knowledge that enhances our understanding of the development of efficient recognition systems.

# Materıals and methods

## Selecting a Dataset

MNIST is a widely used dataset of handwritten digits that contains 70,000 handwritten digits for training and testing a machine learning model that was introduced in 1988. While 70,000 is not a huge number in today’s standarts, it is still the benchmark for classification tasks. This means MNIST dataset is too simple for very complex deep learning algorithms, however it is perfect to decide whether a given model implementation is correct and acceptable.

In the MNIST dataset each digit is stored in a grayscale image with a size of 28x28 pixels. In the following you can see randomly chosen 10 digits from the training set:

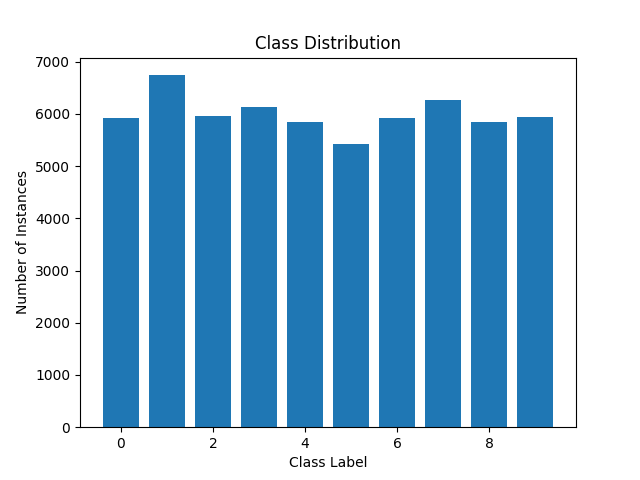
A black square with white symbols

Description automatically generated with medium confidence

A black square with white text

Description automatically generated

Furthermore, MNIST dataset is also a relatively balanced dataset as seen in the graph below:



## Preparing The Dataset

We first divided our dataset to two subsets that will be used for training (60,000) and testing (10,000). After assigning labels of the data to different variables, we normalized the pixel values by dividing them to 255 to change their range from [0, 255] to [0, 1]. Normalizing the data helps in training neural networks as it makes the optimization process more stable and can lead to faster convergence.

## Understanding Convolutional Neural Networks

* *Neural Networks*

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Think of each individual node as its own linear regression model, composed of input data, weights, a bias (or threshold), and an output. The formula would look something like this:

∑

output = f(x) = 1 if ∑ + b>= 0; 0 if ∑ + b < 0

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function, which determines the output. If that output exceeds a given threshold, it activates the node, passing data to the next layer in the network. This results in the output of one node becoming in the input of the next node. This process of passing data from one layer to the next layer defines this neural network as a feedforward network.[1]

* *Loss Function in Neural Networks*

A loss function is a mathematical function that quantifies the difference between predicted and actual values in a machine learning model. It measures the model’s performance and guides the optimization process by providing feedback to backpropagation on how well it fits the data.[2]

1. Backpropagation is an algorithm that updates the weights of the neural network in order to minimize the loss function.

2. Backpropagation works by first calculating the loss function for the current set of weights.

3. Next, it calculates the gradient of the loss function with respect to the weights.

4. Finally, it updates the weights in the direction opposite to the gradient.

5. The network is trained until the loss function stops decreasing.

* *Convolutional Neural Networks*

Convolutional Neural Networks are similar to feedforward networks, but they’re usually utilized for image recognition, pattern recognition, and/or computer vision. These networks use principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.[1] They have three main types of layers, which are:

Convolutional layer,

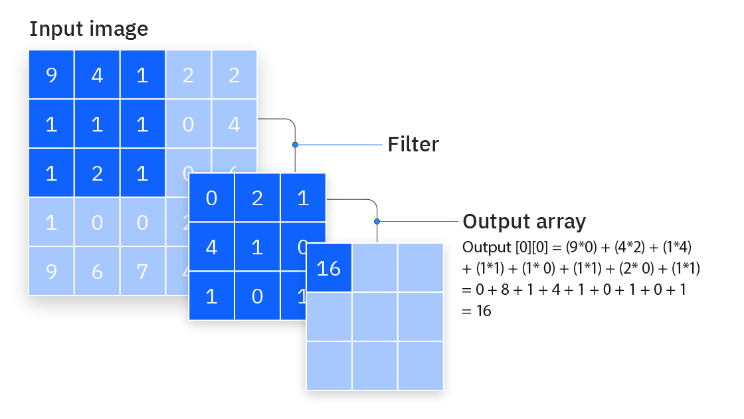
Pooling layer,

Fully-connected (FC) layer

The *convolutional layer* is the first layer of a convolutional network. is the core building block of a CNN, and it is where most of the computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let’s assume that the input will be a grayscale image like the ones in our project, which is made up of a matrix of pixels in 2D. This means that the input will have two dimensions a height, and a width. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution. The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.

Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of backpropagation and gradient descent. However, there are three hyperparameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include the number of filters, stride (number of pixels that kernel moves over and the padding type (usually zero padding).

After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model. [3]



*Pooling layers* are used for dimensionality reduction to decrease the number of parameters in the input. Like the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling: max pooling which takes the highest valued pixel after applying the kernel and average pooling which takes the average of the pixels and sending it to the output array. Pooling layers causes to lose a lot of information but they reduce the chance of overfitting and therefore they increase the efficiency.

*Fully-Connected Layer* is the final layer of the convolutional neural networks. As the name suggests, just like artificial neural network’s nodes, each node in the output layer connects directly to a node in the previous layer. This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1. Then we can assign the label with the highest probability as the model’s prediction.

## Understanding K-Neighbours Classifier

The k-Nearest Neighbors (k-NN) classifier is a widely-used algorithm in machine learning, known for its simplicity and effectiveness in both classification and regression tasks. Belonging to the family of instance-based learning, the k-NN algorithm differs from traditional models by memorizing the entire training dataset and making predictions based on the similarity between new instances and those in the training set.

In the k-NN approach, the dataset is represented in a multi-dimensional space, with each data point corresponding to an instance characterized by various features. To measure the similarity between instances, a distance metric, in our case the Euclidean distance, is chosen. The key concept of the algorithm lies in identifying the k nearest neighbors in the training set for a given test instance.

For classification tasks such as digit recognition, the k-NN algorithm employs a majority voting scheme among the k neighbors. Each neighbor "votes" for its class label, and the class with the most votes is assigned to the test instance. In cases of ties, strategies such as giving more weight to closer neighbors or considering additional neighbors can be implemented.

Unfortunately, the k-NN algorithm exhibits limitations, such as sensitivity to irrelevant features and the scale of the data. It can be computationally expensive for large datasets. However, its strengths lie in its simplicity, ease of implementation, and applicability in scenarios where the decision boundary is complex and not easily captured by simpler models.

## Metric

To decide the efficiency of our models we used the widely used metric of *Accuracy*. Because when dealing with a balanced dataset, where the number of instances in each class is roughly equal, accuracy provides a clear indication of the model's ability to correctly classify instances across all classes.

Accuracy measures the proportion of correctly predicted instances among all instances in the dataset. It is calculated as:

# ımplementatıon of the methods and results

## k-Nearest Neighbours Classifier

At the start of our experimentation, we employed the k-Nearest Neighbors algorithm to classify the digits in our dataset.

Before training our models with this method we first normalized the data to the range 0-1. As we explained before this step is crucial for stabilizing the training process and enhancing the convergence of the neural network.

Then, to determine the optimal value for the hyperparameter k, we conducted experiments with k = 1, 3, 5. The choice of k have altered the accuracy of the model as expected though the difference was small.

The accuracy scores obtained for different k values are as follows:

k=1: Accuracy = 95.38%

k=3: Accuracy = 95.78%

k=5: Accuracy = 96.16%

A graph of a model

Description automatically generated with medium confidence

It is evident that increasing k slightly improves the accuracy, reaching its peak at k = 5. However, it's essential to strike a balance, as excessively large k values might lead to overfitting.

## Convolutional Neural Network

In this phase of our experimentation, we employed a Convolutional Neural Network (CNN) architecture to see whether we can increase the classification performance of our program.

Architecture:

Our CNN architecture consists of multiple layers, each designed to extract and learn relevant features from the handwritten digit images:

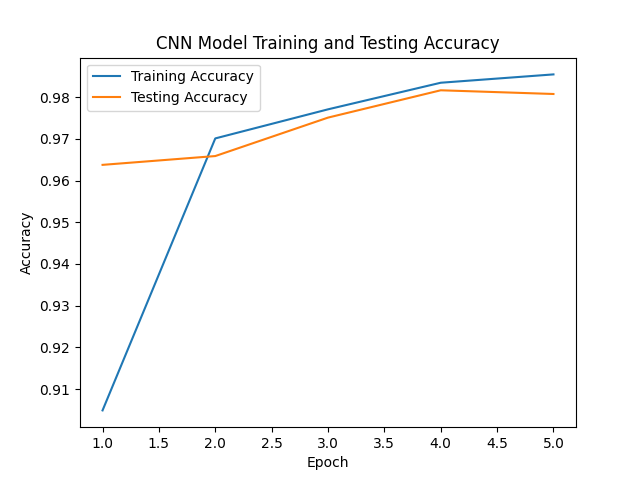
*Convolutional Layers*: Three consecutive convolutional layers with 64 filters each, utilizing a 3×3 kernel (filter) size and rectified linear unit (ReLU) activation function. These layers are followed by *max-pooling layers* (2×2 pooling size) to down-sample the spatial dimensions.

*Fully-Connected Layers*: Following the convolutional and max-pooling layers, we added fully-connected layers to our model to process the previously acquired features. Two dense layers with 64 and 32 units, respectively, and ReLU activation functions were added. The final dense layer with 10 units employs the softmax activation function for digit classification.

Training:

Before training our model, we first normalized the pixel values to the range 0-1. As we explained before this step is crucial for stabilizing the training process and enhancing the convergence of the neural network. After that we added an extra dimension because Convolutional layers in a CNN expect input data in a specific format (usually a 3D array) representing the height, width, and channels of an image. As our MNIST dataset consists of grayscale images, we added the said extra dimension to represent the single channel that was missing.

After pre-processing the dataset, the CNN model was trained using the Adam (Adaptive Moment Estimation) optimizer and sparse categorical crossentropy as the loss function. The model was trained for 5 epochs with a validation split of 30%. The graph below demonstrates the change in training accuracy and validation accuracy according to epochs.



As we can see in our graph, the training accuracy (98.55%) is very close to the validation accuracy (98.08) which means the model doesn’t need to be get adjusted for overfitting.

After the training process, we tested the model with 10,000 different images it had not seen and the CNN achieved an impressive accuracy of 98.19%, showcasing its capability to discern important patterns within the dataset.

# conculusıon

This study offers insight on what model to choose not only for digit recognition but for computer vision in general. The grayscale digit image recognition is a classic classification problem in computer vision and can be solved by both k-neighbours classifier and convolutional neural networks. The empirical results obtained through testing in both models shows that kNN method still offers competetive results in relatively small datasets and can be employed for basic computer vision applications, however it is clear that convolutional neural networks offer significantly better performance at digit recognition.

##### References

1. https://www.ibm.com/topics/neural-networks
2. https://www.analyticsvidhya.com/blog/2022/06/understanding-loss-function-in-deep-learning/
3. https://www.ibm.com/topics/convolutional-neural-networks