Topic: Digit Recognizer

# Handwritten Digit Recognition using KNN and CNN: A Machine Learning Approach

- Adole Faith Ene
- Akaabiam Terlumun Richards
- Abbas Ummu-Kulthum Karima Ali

## Abstract

This project addresses the challenge of handwritten digit recognition using two distinct machine learning techniques: the K-Nearest Neighbors (KNN) algorithm and Convolutional Neural Networks (CNN). The dataset used is the MNIST dataset, which contains thousands of grayscale images of handwritten digits. We began with the KNN model as a baseline for performance comparison and then implemented a CNN for improved accuracy and efficiency. The goal was to build models capable of classifying digit images accurately and submitting predictions to the Kaggle Digit Recognizer competition. This report explores the dataset, preprocessing techniques, model construction, evaluation, and final submission, along with insights into the performance of each approach. Our analysis provides a deeper understanding of model selection, training methodologies, and the power of deep learning in visual data recognition tasks.

## Table of Contents

## Introduction

The ability to recognize handwritten digits is a classic problem in computer vision and machine learning. It has real-world applications in areas such as postal mail sorting, form processing, automatic bank check reading, and digitizing historical archives. The field has seen a significant evolution over the past decades, from basic algorithms like KNN to complex architectures like deep neural networks.

This project focuses on solving this classification problem using the MNIST dataset and comparing two different learning techniques: K-Nearest Neighbors (KNN) and Convolutional Neural Networks (CNN). By understanding and evaluating both models, we aim to appreciate the evolution from traditional machine learning algorithms to advanced deep learning techniques. We also examine how preprocessing steps, model architecture, training duration, and evaluation techniques influence model performance.

In the realm of digit recognition, the MNIST dataset provides an ideal benchmark, enabling the comparison of multiple algorithms on a well-structured and normalized dataset. We use KNN to establish a simple yet strong baseline, then shift to CNN to harness the power of deep learning in extracting hierarchical spatial features. This report documents the end-to-end process, from data loading and visualization to model evaluation and result submission on Kaggle.

## 1. Brief History of MNIST, CNN, and KNN

**MNIST Dataset:**
The MNIST (Modified National Institute of Standards and Technology) dataset was created by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. It includes 70,000 images of handwritten digits, each image being 28x28 pixels in grayscale. The dataset is split into 60,000 training samples and 10,000 testing samples. It has become a benchmark for evaluating machine learning and image processing algorithms. Its popularity is due to its simplicity, clean formatting, and broad applicability to classification tasks.

MNIST originated from the NIST dataset, which included binary images of digits taken from U.S. Census Bureau employees and American high school students. MNIST was curated to reduce the inconsistencies of the original dataset, providing a more uniform and cleaned-up version better suited for training machine learning algorithms. Today, it is a de facto standard for testing algorithms for image classification and has inspired new datasets like EMNIST and Fashion-MNIST.

**K-Nearest Neighbors (KNN):**
KNN is one of the most basic yet effective classification algorithms. It was introduced by Evelyn Fix and Joseph Hodges in 1951. It is a non-parametric, instance-based learning algorithm where classification is based on the majority vote of the k closest neighbors. The simplicity and intuitiveness of KNN make it a strong candidate for establishing a baseline performance in classification problems.

The core idea is to classify a data point based on how its neighbors are classified. Distance metrics like Euclidean, Manhattan, or Minkowski are typically used to compute closeness. Despite its simplicity, KNN is surprisingly powerful when the dataset is relatively small, and the feature space is well-behaved.

**Convolutional Neural Networks (CNN):**
CNNs are a class of deep neural networks introduced in the late 1980s by Yann LeCun. They are specifically designed to process and recognize patterns in visual data. CNNs have revolutionized the field of computer vision, particularly after the success of models like LeNet-5, AlexNet, VGG, and ResNet. They are capable of learning spatial hierarchies through convolutional layers, pooling layers, and dense layers. CNNs mimic the way human vision perceives patterns, leading to exceptional performance in image classification and detection tasks.

LeNet-5, developed by LeCun in 1998, was one of the first CNNs used for digit recognition, and it laid the groundwork for modern CNN architectures. With the advent of GPUs and large labeled datasets, CNNs have scaled dramatically, now powering systems in facial recognition, autonomous vehicles, and medical imaging.

## 2. Dataset Overview

The MNIST dataset used in this project contains 70,000 grayscale images of handwritten digits from 0 to 9. Each image is 28x28 pixels in size. The training set contains 60,000 labeled images, and the test set consists of 10,000 unlabeled images provided by Kaggle for evaluation. These digits are centered and size-normalized, simplifying preprocessing and allowing direct comparison between classifiers. The pixel values range from 0 to 255, with 0 representing black and 255 representing white.

We begin our work by importing this dataset and visualizing some sample digits to better understand the structure and content. This also helps ensure that the images are loaded correctly.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

train.head()
```

The dataset includes a 'label' column, which contains the digit each image represents. The remaining 784 columns represent pixel values (28x28=784). For preprocessing, we separate the label from the pixel data and normalize the pixel values by dividing by 255.0.

## 3. Data Preprocessing

Before training any machine learning model, data preprocessing is a crucial step to ensure the dataset is clean, consistent, and suitable for analysis. For this project, we started by splitting the dataset into feature variables (X) and target labels (y). The pixel intensity values were normalized to a range of 0 to 1 by dividing by 255. This normalization step helps models train more efficiently and achieve better convergence.

```
X = train.drop('label', axis=1)
y = train['label']

# Normalize pixel values
X = X / 255.0
test = test / 255.0
```

To evaluate the models, we also split the training data into training and validation sets. This allows us to test model performance before using the final test dataset.

```
from sklearn.model_selection import train_test_split

# 80% training and 20% validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2,
random_state=42)
```

After splitting, we reshaped the input arrays to match the input structure required by CNNs. CNNs require input in the form of image dimensions (28, 28, 1), where the last dimension represents the single grayscale channel.

```
# Reshape for CNN input
X_train_cnn = X_train.values.reshape(-1, 28, 28, 1)
X_valid_cnn = X_valid.values.reshape(-1, 28, 28, 1)
test_cnn = test.values.reshape(-1, 28, 28, 1)
```

We also encoded the target labels into categorical format, which is required when using categorical crossentropy as the loss function in CNNs.

```
from tensorflow.keras.utils import to_categorical

y_train_cat = to_categorical(y_train, num_classes=10)
y_valid_cat = to_categorical(y_valid, num_classes=10)
```

These preprocessing steps prepare the data for efficient and accurate training with both KNN and CNN models.

## Model 1: K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is one of the simplest and most effective algorithms for classification problems. It classifies new data points based on the majority class among its 'k' nearest neighbors. In this project, we implemented KNN using the KNeighborsClassifier from the sklearn.neighbors library.

We used a value of **k = 5**, which is a common starting point in KNN models. This means that the model looks at the five nearest training samples to determine the class of a new input. The Euclidean distance metric was used by default to determine the closeness between data points.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Create the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the model
knn.fit(X_train, y_train)

# Predict on validation set
y_pred_knn = knn.predict(X_valid)

# Accuracy
knn_accuracy = accuracy_score(y_valid, y_pred_knn)
print(f"KNN Validation Accuracy: {knn_accuracy * 100:.2f}%")
```

The KNN model achieved an accuracy of **93.00%** on the validation set. This performance is impressive for such a simple, non-parametric algorithm. However, KNN can be computationally expensive when dealing with large datasets, as it stores all training samples and performs distance calculations during prediction.

After verifying its performance, we generated the final predictions for the test dataset provided by Kaggle:

```
y_test_knn = knn.predict(test)

submission_knn = pd.DataFrame({
    "ImageId": list(range(1, len(y_test_knn)+1)),
    "Label": y_test_knn
})

submission_knn.to_csv("submission_knn.csv", index=False)
```

This submission file contains the predicted labels for each image in the test dataset. It is formatted according to Kaggle’s Digit Recognizer competition requirements, with an ImageId column and a Label column.

## Model 2: Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are deep learning models that have become the standard for image classification tasks. Unlike traditional machine learning algorithms, CNNs automatically learn features from image data through convolutional layers, making them ideal for tasks such as digit recognition.

In this project, we constructed a simple CNN using TensorFlow and Keras. The model architecture consisted of two convolutional layers followed by max-pooling, a dropout layer to prevent overfitting, and two dense layers for classification.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout

# CNN model architecture
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

The model uses ReLU activation functions for the convolutional layers to introduce non-linearity and a softmax activation function at the output to generate probabilities for the 10 digit classes.

We compiled the model using the Adam optimizer and categorical crossentropy as the loss function, which is appropriate for multi-class classification problems:

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

We trained the model for 10 epochs with a batch size of 64. Training progress was monitored using the validation dataset.

```
history = model.fit(X_train_cnn, y_train_cat, epochs=10, batch_size=64,
validation_data=(X_valid_cnn, y_valid_cat))
```

After training, we evaluated the model on the validation set:

```
val_loss, val_accuracy = model.evaluate(X_valid_cnn, y_valid_cat)
print(f"CNN Validation Accuracy: {val_accuracy * 100:.2f}%")
```

The CNN achieved an accuracy of approximately **98.60%**, significantly outperforming the KNN model. The use of convolutional layers allowed the model to capture complex spatial features from the images, which contributed to the improved performance.

We then generated predictions on the test dataset and prepared the submission file:

```
cnn_predictions = model.predict(test_cnn)
y_test_cnn = np.argmax(cnn_predictions, axis=1)

submission_cnn = pd.DataFrame({
    "ImageId": list(range(1, len(y_test_cnn) + 1)),
    "Label": y_test_cnn
})

submission_cnn.to_csv("submission_cnn.csv", index=False)
```

## 4. Results and Comparison

After training and evaluating both models, we compared their performances based on validation accuracy and computational efficiency.

**K-Nearest Neighbors (KNN)**

- **Model Simplicity**: Very easy to implement with minimal training time.
- **Accuracy**: Achieved a validation accuracy of **93.00%**.
- **Drawbacks**: Computationally expensive at prediction time due to lazy learning. Performance decreases with larger datasets.

**Convolutional Neural Network (CNN)**

- **Model Complexity**: Requires more resources to build and train.
- **Accuracy**: Achieved a validation accuracy of **98.60%**.
- **Advantages**: Learns spatial hierarchies and patterns, highly effective for image data.

| **Metric** | **KNN** | **CNN** |
|---|---|---|
| Accuracy | 93.00% | 98.60% |
| Training Time | Minimal | Moderate |
| Prediction | Slow (distance calc) | Fast (feed-forward) |
| Complexity | Low | High |

The CNN model significantly outperformed the KNN model in terms of accuracy. This demonstrates the advantage of deep learning in capturing complex spatial features from images. Despite the longer training time, CNN is more scalable and efficient for large-scale classification tasks.

## 5. Submission File Structure

After both models were trained and evaluated, we prepared prediction files for submission to the Kaggle competition. Kaggle expects a CSV file with two columns: ImageId and Label.

```
For both models, we formatted the submission file as follows:
submission = pd.DataFrame({
    "ImageId": list(range(1, len(y_test_predictions) + 1)),
    "Label": y_test_predictions
})

submission.to_csv("submission.csv", index=False)
```

The submission files were:

- submission_knn.csv for the KNN model.
- submission_cnn.csv for the CNN model.

## 6. Model Evaluation Techniques

To ensure that the models performed well and were not overfitting or underfitting, several evaluation techniques were applied:

- **Train-Test Split**: The dataset was split into 80% training and 20% validation sets. This provided a reliable way to measure model performance on unseen data.
- **Accuracy Score**: We calculated accuracy as the ratio of correctly predicted labels to the total number of predictions:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true, y_pred)
```

- **Confusion Matrix**: Though not shown in detail here, confusion matrices could be used to identify which digits were misclassified most often.
- **Model History Plot** (CNN): The training history object provided loss and accuracy for each epoch. These were plotted to visualize training progress and detect overfitting.

```python
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

These techniques helped us choose the best model (CNN) based on empirical evidence and ensured the reliability of our results.

## Conclusion

In this project, we implemented and compared two models—KNN and CNN—for handwritten digit recognition using the MNIST dataset. The KNN model provided a strong baseline with 93.00% accuracy but lacked the sophistication and learning capacity of the CNN. On the other hand, the CNN model demonstrated superior performance, achieving 98.60% accuracy and generalizing better to unseen data.

Through data preprocessing, careful model selection, and evaluation techniques, we developed a robust pipeline for digit classification. This project illustrates how traditional machine learning models like KNN can be useful for initial benchmarks, while deep learning models like CNN offer significant performance gains in image recognition tasks.

The skills and understanding gained from this project are applicable to a wide range of real-world problems in computer vision and pattern recognition.

## References

- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE.
- Kaggle Digit Recognizer Competition: https://www.kaggle.com/c/digit-recognizer/data
- MNIST Dataset Homepage: http://yann.lecun.com/exdb/mnist/