🧠 Project Title: **Handwritten Digit Recognition Using KNN and CNN**

🔷 Project Overview - Faith

The goal of this project was to solve a classification task using machine learning. We worked on the MNIST dataset, which consists of thousands of handwritten digit images (from 0 to 9). Our task was to build models that could recognize the correct digit from each image and generate predictions for an unseen test set provided by Kaggle.

This type of task is a classic example of image classification, and it has real-world applications in areas like digitizing handwritten forms, postal mail sorting, and bank check processing.

🔷 Step 1: Importing Libraries

We started by importing the essential libraries for:

- Data manipulation – using numpy and pandas
- Data visualization – using matplotlib.pyplot and seaborn
- Model building – using Scikit-learn for KNN and Keras (TensorFlow backend) for CNN

These libraries gave us all the tools needed for loading data, exploring patterns, building models, and evaluating results.

🔷 Step 2: Loading and Exploring the Data

We loaded two datasets:

- train.csv: Contains labeled images (each 28x28 image is flattened into 784-pixel values + a label column).
- test.csv: Contains only the image data without labels.

We used pandas to explore the shape of the data and check for missing values. There were no null values, which meant the data was clean and ready for use.

🔷 **Step 3: Data Preprocessing**

🧼 We:

 Separated features and labels:

- X = pixel values (784 columns per image)
- y = labels (digits 0–9)

 Normalized the pixel values by dividing them by 255.0 so they fall between 0 and 1. This makes training faster and more stable because the model doesn't need to deal with large values.

 Reshaped image data for the CNN model into a 3D format (28, 28, 1) — which represents height, width, and grayscale channel.

 One-hot encoded the labels (for CNN) using to_categorical(y) so that each label becomes a 10-element vector (e.g., 3 becomes [0 0 0 1 0 0 0 0 0 0]).

 Split the dataset into training and validation sets using an **80/20 split**:

🍼 **Model 1: K-Nearest Neighbors (KNN) – Kay Ali**

We started with a classical machine learning model, KNN, which is simple but surprisingly effective.

KNN doesn't actually learn in the traditional sense. Instead, it stores the entire training dataset. When it sees a new image, it finds the 5 closest images (neighbors) in the training set and classifies the new image based on a majority vote among those neighbors.

📊 **KNN Results:**

- Accuracy: ~93% on the validation set
- Strengths: Easy to implement, intuitive
- Weaknesses: Slow at prediction time and less efficient with high-dimensional image data

KNN gave us a baseline — a way to understand how a simple model performs. It helped us appreciate the value of more complex deep learning models later.

🧪 Model 2: Convolutional Neural Network (CNN) - Richards

After KNN, we moved on to a deep learning model, specifically a Convolutional Neural Network (CNN), which is designed to work well with image data.

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

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(10, activation="softmax"))
```

🛠️ How It Works:

- Convolutional layer: Learns small patterns (edges, shapes) in images.
- MaxPooling: Reduces image size to improve efficiency and reduce overfitting.
- Flatten + Dense: Connect learned patterns to output classes.
- Dropout: Prevents overfitting by randomly turning off neurons during training.
- Softmax Output: Outputs probability for each of the 10 digit classes.

📊 CNN Results:

- Accuracy: ~98% on the validation set
- Strengths: Excellent at recognizing image features
- Weaknesses: More complex and requires more time to train

✅ Why We Used It:

CNN is considered the best practice for image classification tasks. It's far more capable than KNN in terms of accuracy and generalization, especially on large datasets like MNIST.

Final Submission File - **Faith**

After training the CNN, we used it to generate predictions on the test dataset. We created a CSV file for Kaggle submission using the format:

```
submission = pd.DataFrame({
    "ImageId": range(1, len(predictions) + 1),
    "Label": predictions
})
submission.to_csv("submission.csv", index=False)
```

This file had:

- ImageId: Numbers from 1 to the total number of test images
- Label: Predicted digit (0–9)

✅ Final Thoughts

Model Validation Accuracy Notes

KNN	~93%	Simple, good baseline
CNN	~98%	Advanced, better accuracy and generalization

We successfully built and compared two models for digit recognition. While KNN was a solid starting point, the CNN significantly outperformed it, proving how deep learning excels in image classification tasks.