HANDWRITTEN DIGIT IMAGE CLASSIFICATION

Jay Malhotra (E23BCAU0050)

1. Introduction

Handwritten digit recognition is a multi-class classification problem in computer vision and machine learning, with applications ranging from postal mail sorting to form digitization. This project uses K-Nearest Neighbors (KNN), Artificial Neural Network (ANN), and Random Forest (RF) classifiers to achieve reliable handwritten digit classification on the MNIST dataset, a widely used benchmark for image classification.

2. Objectives

1. **Model Training:** Train and evaluate three machine learning models, KNN, ANN and Random Forest, on the MNIST dataset.
2. **Prediction Interface:** Create a user-friendly system for classifying handwritten digits from uploaded images.

3. Dataset

3.1. Initial Local CSV Dataset

The project initially used a locally stored version of our own manual dataset in .csv format. While this dataset provided labeled training and testing data, it introduced several challenges:

1. **Data Preprocessing Overhead:** The dataset required manual reshaping, normalization, and scaling.
2. **Risk of Inconsistencies:** Errors could arise from improper preprocessing steps.
3. **Limited Features:** The dataset lacked built-in tools for augmentation and normalization.

3.2. Transition to TensorFlow's Built-In MNIST Dataset

To streamline preprocessing and reduce the risk of errors, the project transitioned to TensorFlow's built-in MNIST dataset:

1. **Preprocessed and Standardized:** The dataset is provided as 28x28 grayscale images, eliminating the need for extensive preprocessing.
2. **Ease of Integration:** TensorFlow's tools ensured smooth compatibility with models.
3. **Data Augmentation:** TensorFlow's ecosystem supports image augmentation to enhance model generalization.

The TensorFlow MNIST dataset improved reliability and reduced preprocessing complexity, allowing more focus on model performance and prediction accuracy.

4. Method/Model Selection and Development

4.1. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric supervised learning algorithm used for classification and regression tasks. In KNN, predictions are made by identifying the k closest labeled examples (neighbors) in the training dataset to a given input.

Key Features:

- **Instance-Based Learning**: KNN stores all training data and makes predictions based on proximity, rather than building an explicit model.
- **Distance Metric**: The closeness of neighbors is typically determined using Euclidean distance.

- **Majority Voting**: For classification, the most common label among the k-nearest neighbors is assigned to the test data.

Example:
For digit recognition:

- If k=5 and the 5 nearest neighbors to a test digit include 3 labeled as '2' and 2 labeled as '3', the digit will be classified as '2' (majority vote).

Advantages:

- Simple and easy to implement.
- Effective for small datasets with clear separation between classes.

Disadvantages:

- Computationally expensive for large datasets.
- Sensitive to irrelevant features and the choice of k.

In this project, KNN was chosen for its robustness and ease of implementation, achieving high accuracy on random test samples and specific digits.

4.2. Artificial Neural Network (ANN)

Artificial Neural Network (ANN) is a computational model inspired by the structure and functioning of biological neural networks in the human brain. It is widely used in machine learning for complex tasks like classification, regression, and pattern recognition.

Key Features:

- **Structure**: ANNs consist of layers of interconnected nodes (neurons), including:
  - **Input Layer**: Receives input features.
  - **Hidden Layers**: Perform complex computations by applying weights and biases.
  - **Output Layer**: Produces the final predictions (e.g., classification probabilities).
- **Activation Functions**: Non-linear functions like ReLU (Rectified Linear Unit) and softmax are applied to introduce non-linearity and handle complex patterns in data.
- **Learning Mechanism**: ANNs are trained using algorithms which adjusts weight and bias by minimizing a loss function.

Example:
For digit recognition:

- Input features represent pixel values of a digit image.
- Hidden layers process the features to learn patterns.
- Output layer predicts the digit (0–9) with associated probabilities.

Advantages:

- **Powerful**: Capable of handling large datasets and learning complex patterns.
- **Adaptability**: Suitable for tasks with high-dimensional input data like images, audio, and text.

Disadvantages:

- **Resource-Intensive**: Requires significant computational resources and time for training.
- **Prone to Overfitting**: Needs regularization and sufficient data to generalize well.

In this project, ANN demonstrated high accuracy on the test dataset and random samples. Due to its ability to learn complex patterns through hidden layers and handling large

datasets, it is no surprise that this model performed the best out of the 3 in both training and testing phase.

4.3. Random Forest (RF)

Random Forest is an ensemble machine learning algorithm that combines multiple decision trees to produce a more robust and accurate model. It is primarily used for classification and regression tasks.

Key Features:

1. **Ensemble Learning**:
    - Random Forest creates a "forest" of decision trees, where each tree is trained on a random subset of the data (bagging) and random features (feature randomness).
    - The final prediction is made by aggregating the outputs of all the trees.
2. **Feature Randomness**:
    - At each split in the decision tree, Random Forest considers a random subset of features rather than all features, reducing the likelihood of overfitting.
3. **Bootstrap Aggregation (Bagging)**:
    - Each tree is trained on a different bootstrap sample (random sampling with replacement) from the training data, making the model more resilient to noise and overfitting.

Advantages:

- **High Accuracy**: Performs well on many datasets due to the ensemble approach.
- **Robustness**: Handles missing data, noisy features, and non-linear relationships well.
- **Feature Importance**: Provides insights into which features are most influential in predictions.

Disadvantages:

- **Complexity**: Requires significant computational resources for large datasets due to many trees.
- **Interpretability**: Individual trees are interpretable, but the entire forest can be harder to understand.

Example:

For digit recognition:

- Each decision tree might classify a digit differently based on its subset of data and features.
- Random Forest aggregates the predictions from all the trees to make the final decision, improving accuracy and reducing the variance.

In this project, Random Forest provided strong performance, achieving high accuracy on the test dataset and handling most random samples correctly. However, it struggled with a specific test digit, predicting incorrectly in 1/10 cases, showcasing its limitation in certain edge cases.

5. Implementation Details

5.1. Data Preprocessing

MNIST images were flattened into vectors of size 784 (28x28 pixels) and normalized to values between 0 and 1.

5.2. Prediction Interface
The system offers a choice between KNN, ANN, and Random Forest for digit prediction. Users upload grayscale images of handwritten digits, which are preprocessed into MNIST-compatible formats (28x28, normalized). Predictions are visualized with confidence scores.

5.3. Visualization
The code generates bar plots displaying confidence scores for each digit class (0-9). It also calculates the confusion matrix and visualises it.

6. Challenges and Solutions

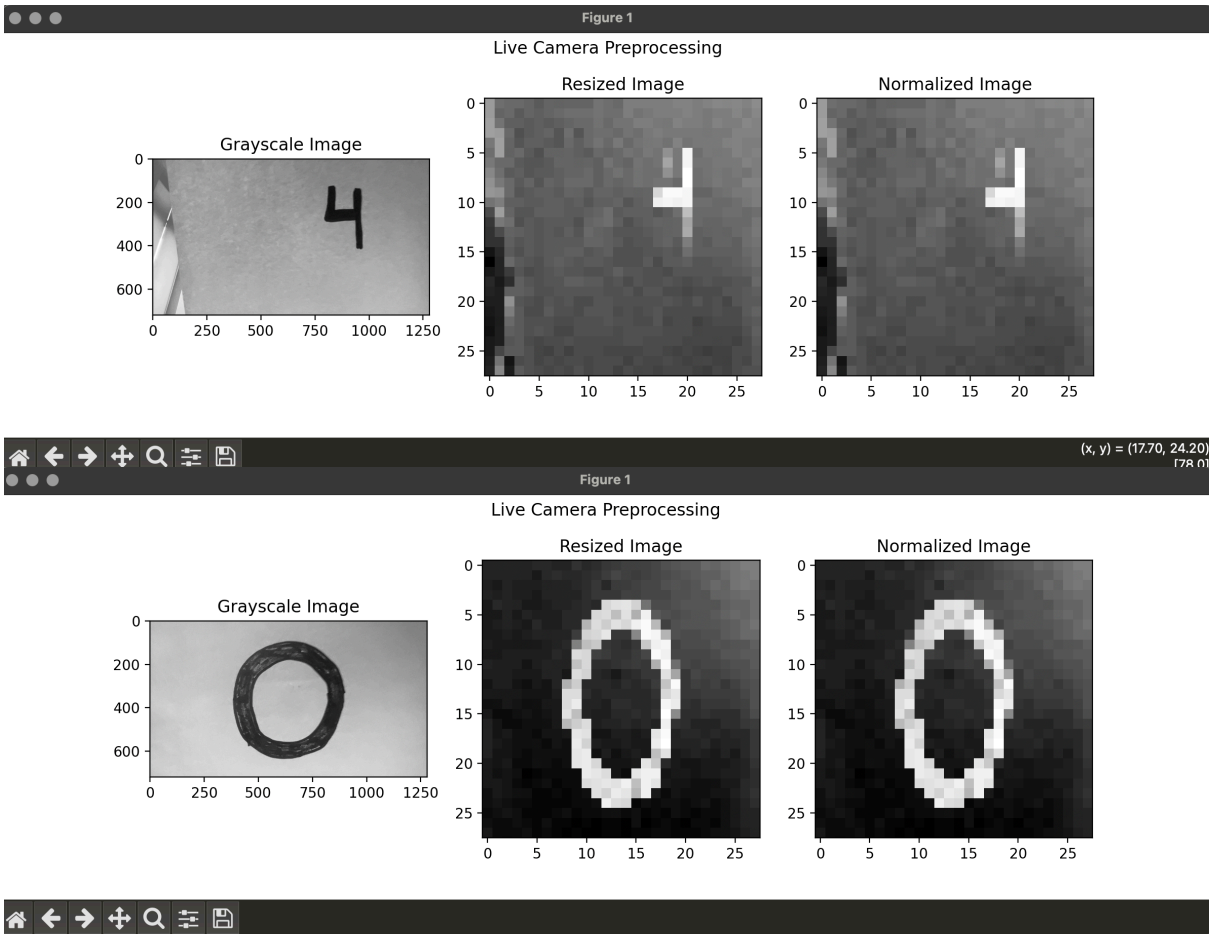
6.1. Local Dataset Issues
One of the initial challenges was working with the local CSV-based dataset. While the dataset was easy to load and use, it required extensive preprocessing steps such as reshaping, normalization, and scaling. These steps were prone to errors, such as incorrect scaling or improper reshaping, which significantly impacted model performance. Additionally, manual preprocessing increased the complexity of the pipeline, making the workflow inefficient. To address these issues, we transitioned to TensorFlow's built-in MNIST dataset, which comes preprocessed and standardized as 28x28 grayscale images. This change eliminated the need for manual preprocessing, ensured consistency across inputs, and allowed us to use TensorFlow's optimized data handling tools, streamlining the entire process.

6.2. Dropping the Voting Classifier
The initial implementation included a Voting Classifier to combine the strengths of Random Forest, KNN, and ANN. While this ensemble approach improved training accuracy, its performance during real-world predictions was inconsistent. The classifier struggled to balance the localized accuracy of KNN with the broader decision-making of Random Forest, often leading to misclassifications on user-uploaded images, particularly noisy or distorted ones. Additionally, the ANN classifier, despite its high training accuracy, struggled with certain real-world predictions, further reducing the effectiveness of the Voting Classifier. Debugging these inconsistencies proved time-consuming, and the increased complexity reduced the system's reliability. Consequently, we opted to drop the Voting Classifier, focusing on standalone models for Random Forest, KNN, and ANN. This decision simplified the architecture and enhanced prediction reliability without significantly compromising accuracy.

6.3. Dropping SVM
SVM was initially included due to its strong theoretical performance. SVM with the RBF kernel achieved the best metrics during evaluation, but its computational cost was very high. Training SVM on the large MNIST dataset required significant time and resources, and predictions were slow, making it unsuitable for real-time applications. These limitations led us to exclude SVM in favor of Random Forest, which offered a better balance of computational efficiency, accuracy, and reliability for practical use cases.

6.4. Transition from Live Camera to Image Upload

The live camera feature was initially included for real-time digit recognition but encountered significant challenges. The preprocessing required (grayscaling, resizing, inverting colors) was prone to errors and often incompatible with the MNIST dataset's format. For example, live camera images often had inverted colors (black digits on a white background) or noise, which complicated predictions. These issues were difficult to address consistently, leading to unreliable results.



To resolve this, we transitioned to an image upload interface that assumes preprocessed, MNIST-compatible grayscale images. This approach provided users with better control over input quality, significantly improving the system's reliability and usability.

7. Metrics Used for Model Evaluation

To effectively evaluate the performance of our models, the following metrics were used:

1. **Accuracy**: The proportion of correctly classified digits out of the total number of digits. While accuracy gives a good overall sense of model performance, it does not account for class imbalances or specific misclassification types.
2. **Precision**: The fraction of true positive predictions out of all predictions made for a specific class. Precision is particularly relevant when the cost of false positives is high.
3. **Recall**: The fraction of true positive predictions out of all actual instances of a specific class. Recall is important when the cost of false negatives is high.

4. **F1 Score**: The harmonic mean of precision and recall. F1 Score balances the trade-off between precision and recall and is particularly useful when both false positives and false negatives need to be minimized.
5. **Confusion Matrix**: A visualization of model predictions compared to true labels, showing counts of true positives, true negatives, false positives, and false negatives. It provides detailed insights into which classes are being misclassified.

8. Results and Discussion

Writing the code:

```python
import tensorflow as tf

print('\nLoading MNIST Data...')
mnist = tf.keras.datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], -1) / 255.0
X_test = X_test.reshape(X_test.shape[0], -1) / 255.0
```

Using tensorflow to download MNIST dataset and split it into training and testing sets.

Flattening each 28x28 image into a 1D array of size 784 (784 features) and normalizing pixel values for model input

```python
from keras.api.utils import to_categorical

X_train_reshaped = X_train.reshape(-1, 28, 28, 1)
X_test_reshaped = X_test.reshape(-1, 28, 28, 1)
y_train_oh = to_categorical(y_train, 10)
y_test_oh = to_categorical(y_test, 10)
```

Reshape data into 28x28x1 format specifically for ANN.

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

def calculate_metrics(model, X_test, y_test, model_type):
    if model_type == 'ann':
        y_pred = model.predict(X_test).argmax(axis=1)
    else:
        y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f"\nMetrics for {model_type.upper()}:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("\nConfusion Matrix:")
    print(conf_matrix)
```

Evaluating model's performance on test data.

Weighted average ensures classes with more samples have larger influence on overall score.

```
    plt.matshow(conf_matrix, cmap='coolwarm')
    plt.title(f'{model_type.upper()} Confusion Matrix')
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

    return accuracy, precision, recall, f1, conf_matrix

def train_model(model_type):
    if model_type == 'random_forest':
        from sklearn.ensemble import RandomForestClassifier
        print('\nTraining Random Forest Classifier with n_estimators=300 and max_depth=30...')
        model = RandomForestClassifier(n_estimators=300, max_depth=30, n_jobs=-1, random_state=42)
        model.fit(X_train, y_train)
        calculate_metrics(model, X_test, y_test, model_type)

    elif model_type == 'knn':
        from sklearn.neighbors import KNeighborsClassifier
        print('\nTraining KNN Classifier with n_neighbors=5...')
        model = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
        model.fit(X_train, y_train)
        calculate_metrics(model, X_test, y_test, model_type)

    elif model_type == 'ann':
        from keras.api.models import Sequential
        from keras.api.layers import Dense, Flatten
        print('\nTraining ANN Model...')
        model = Sequential([
            Flatten(input_shape=(28, 28, 1)),
            Dense(128, activation='relu'),
            Dense(64, activation='relu'),
            Dense(10, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        model.fit(X_train_reshaped, y_train_oh, epochs=10, batch_size=128, verbose=1)
        calculate_metrics(model, X_test_reshaped, y_test, model_type)

    else:
        print('Invalid model type selected. Exiting...')
        sys.exit()
    return model
```

Visualizing Confusion Matrix.

Train a Random Forest Classifier with 300 decision trees, a max depth of 30, parallel processing, and a fixed random state.

Train a K-Nearest Neighbour Classifier with K=5 on the training data.

Train an Artificial Neural Network with 2 hidden layers having 128 neurons and 64 neurons.

```python
print("\nSelect the model to train:")
print("1. Random Forest")
print("2. KNN")
print("3. ANN")

model_choice = input("Enter your choice (1/2/3): ").strip()

if model_choice == '1':
    model_type = 'random_forest'
elif model_choice == '2':
    model_type = 'knn'
elif model_choice == '3':
    model_type = 'ann'
else:
    print("Invalid choice. Defaulting to Random Forest.")
    model_type = 'random_forest'

```

Let's user choose between the 3 models.

```python
model = train_model(model_type)

if model_type != 'ann':
    import pickle
    with open('MNIST_model.pickle', 'wb') as f:
        pickle.dump(model, f)
    with open('MNIST_model.pickle', 'rb') as f:
        model = pickle.load(f)

```

Train and save selected model.

```python
from PIL import Image

def preprocess_image(image_path):
    try:
        print(f"\nPreprocessing image: {image_path}")
        img = Image.open(image_path)
        img = img.resize((28, 28))
        img_array = np.array(img)
        img_array = img_array / 255.0
        img_array = img_array.flatten().reshape(1, -1) if model_type != 'ann' else img_array.reshape(1, 28, 28, 1)
        return img_array, img
    except Exception as e:
        print(f"Error processing image: {e}")
        return None, None

```

Resize input image to 28x28.

Normalize image by scaling pixel values from original range 0 to 255, to 0 to 1

Flatten and reshape image to match input format based on model selection.

```python
def upload_and_predict():
    while True:
        print("\nEnter the full path of the image (png, jpg, jpeg) or type 'exit' to quit:")
        image_path = input("Image Path: ").strip()

        if image_path.lower() == 'exit':
            print("Exiting the prediction loop. Goodbye!")
            break

```

Allow user to enter file path of image.

```python
        processed_img_array, processed_img = preprocess_image(image_path)

        if processed_img_array is not None:
            if model_type == 'ann':
                probabilities = model.predict(processed_img_array)[0]
                predicted_digit = np.argmax(probabilities)
            else:
                probabilities = model.predict_proba(processed_img_array)[0]
                predicted_digit = model.predict(processed_img_array)[0]

```

Process the uploaded image and predict digit based on probabilities.

```
            formatted_probabilities = [f"{prob:.4f}" for prob in probabilities]
            print(f"\nThe predicted digit is: {predicted_digit}")
            print(f"Prediction probabilities for each digit: {formatted_probabilities}")

            plt.imshow(processed_img, cmap='gray')
            plt.title(f"Predicted Digit: {predicted_digit}")
            plt.axis('off')
            plt.show()

            plt.bar(range(10), probabilities)
            plt.title("Prediction Probabilities")
            plt.xlabel("Digit")
            plt.ylabel("Probability")
            plt.xticks(range(10))
            plt.show()
        else:
            print("Failed to process the image. Please check the file path and format.")

```

```
def visualize_samples():
    print("\nVisualizing some random test samples...")
    indices = np.random.randint(0, len(X_test), 10)
    for i in indices:
        img = X_test[i].reshape(28, 28) * 255.0
        if model_type == 'ann':
            probabilities = model.predict(X_test_reshaped[i].reshape(1, 28, 28, 1))[0]
            predicted = np.argmax(probabilities)
        else:
            probabilities = model.predict_proba(X_test[i].reshape(1, -1))[0]
            predicted = model.predict(X_test[i].reshape(1, -1))[0]

        formatted_probabilities = [f"{prob:.4f}" for prob in probabilities]
        print(f"Actual: {y_test[i]} | Predicted: {predicted} | Probabilities: {formatted_probabilities}")

        plt.title(f"Actual: {y_test[i]} | Predicted: {predicted}")
        plt.imshow(img, cmap='gray')
        plt.axis('off')
        plt.show()

upload_and_predict()
visualize_samples()

```

Randomly selects 10 samples from test dataset, visualize and predicts the digit based on selected model.

Results:

Using ANN:

```
(myenv) (base) jay@Jays-MacBook-Air Project % python -u "/Users/jay/Desktop
earning/Project/digit_classification_project.py"

Loading MNIST Data...

Select the model to train:
1. Random Forest
2. KNN
3. ANN
Enter your choice (1/2/3): 3

Training ANN Model...
```

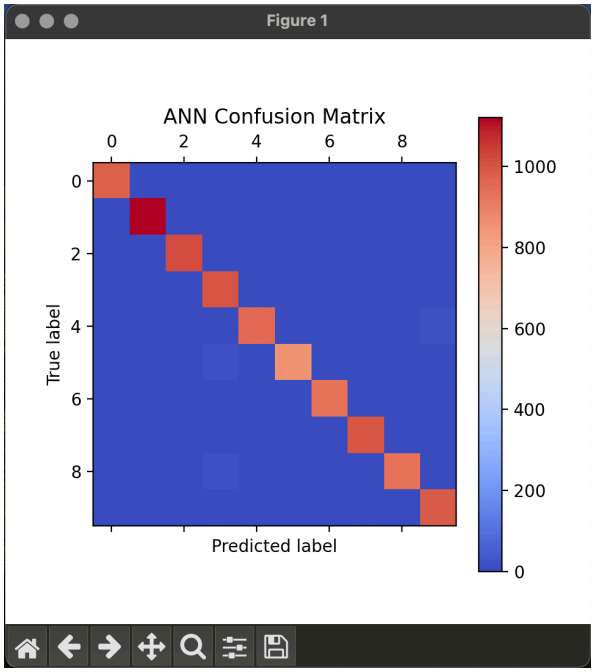


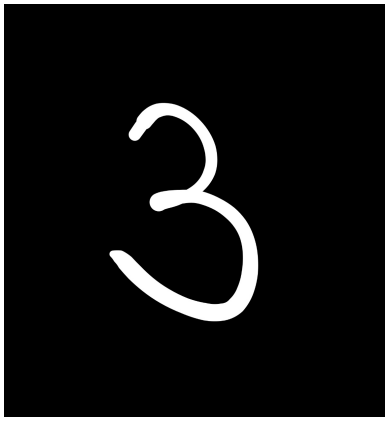
Figure 1
ANN Confusion Matrix
0
2
4
6
8
True label
Predicted label
1000
800
600
400
200
0

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8256 - loss: 0.6096
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9563 - loss: 0.1467
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9729 - loss: 0.0932
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9787 - loss: 0.0684
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9837 - loss: 0.0519
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9876 - loss: 0.0411
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9900 - loss: 0.0337
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9923 - loss: 0.0270
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9930 - loss: 0.0226
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9944 - loss: 0.0184
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 369us/step
```

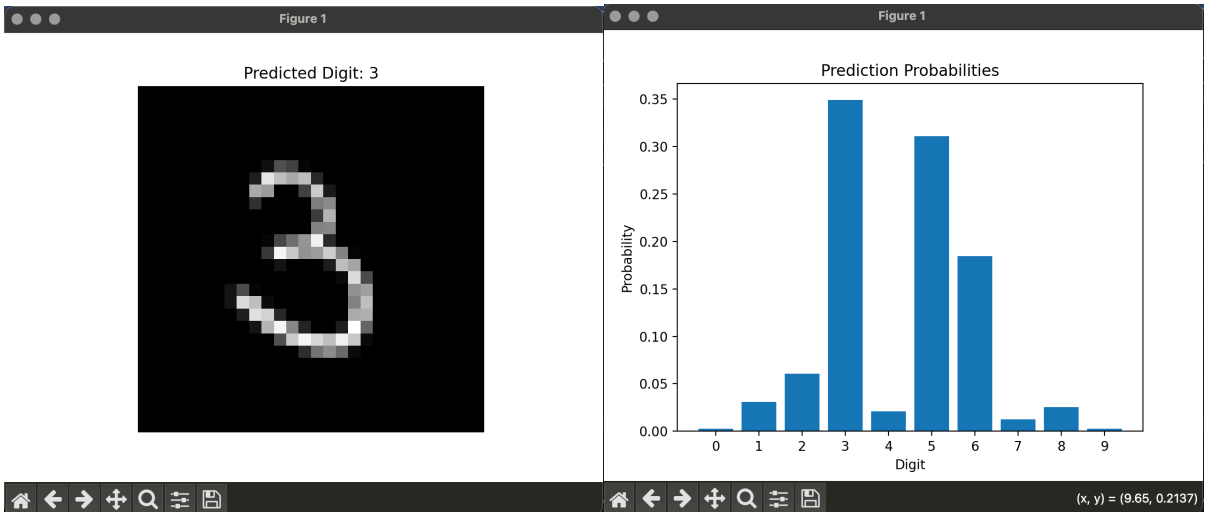
```
Metrics for ANN:
Accuracy: 0.9773
Precision: 0.9776
Recall: 0.9773
F1 Score: 0.9773

Confusion Matrix:
[[ 971    0    0    1    0    2    1    1    2    2]
 [   0 1122    4    1    0    1    1    2    4    0]
 [   1    0 1013    8    2    0    0    4    4    0]
 [   0    0    5 1002    0    0    0    2    1    0]
 [   0    1    5    3  954    0    0    3    2   14]
 [   2    1    0   18    3  862    1    1    3    1]
 [   3    3    3    1    6    9  932    0    1    0]
 [   1    2    9    7    1    0    0  999    2    7]
 [   4    0    8   20    1    2    1    2  934    2]
 [   2    2    0    8    3    3    0    5    2  984]]
```

```
Enter the full path of the image (png, jpg, jpeg) or type 'exit' to quit:
Image Path: /Users/jay/Downloads/3.jpeg

Preprocessing image: /Users/jay/Downloads/3.jpeg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step

The predicted digit is: 3
Prediction probabilities for each digit: ['0.0027', '0.0308', '0.0608', '0.3
6']
```

Actual Digit:

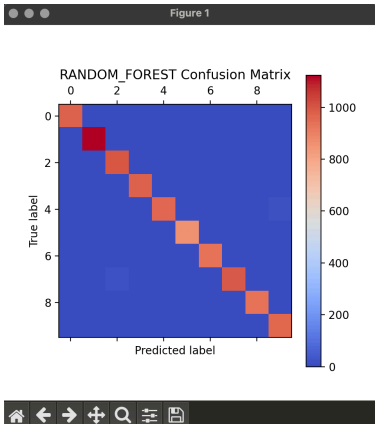


Predicted Digit:



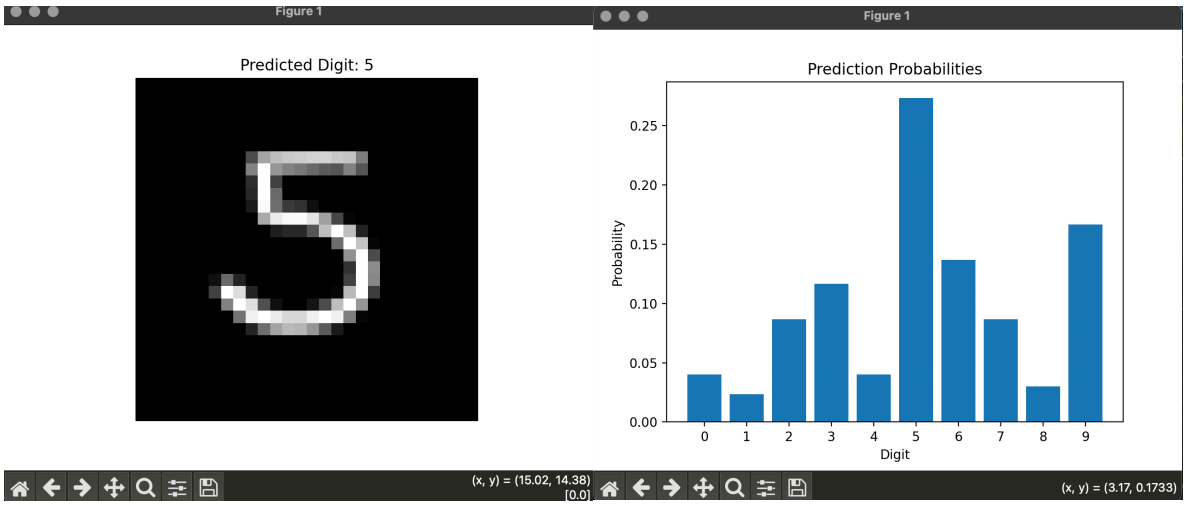
Using Random Forest:

```
Enter your choice (1/2/3): 1

Training Random Forest Classifier with n_estimators=300 and max_depth=30...

Metrics for RANDOM_FOREST:
Accuracy: 0.9716
Precision: 0.9716
Recall: 0.9716
F1 Score: 0.9716

Enter the full path of the image (png, jpg, jpeg) or type 'exit' to quit:
Image Path: /Users/jay/Downloads/5.jpg

Preprocessing image: /Users/jay/Downloads/5.jpg

The predicted digit is: 5
Prediction probabilities for each digit: ['0.0400', '0.0234', '0.0867', '0
7']
```



Actual Digit:

Predicted Digit:

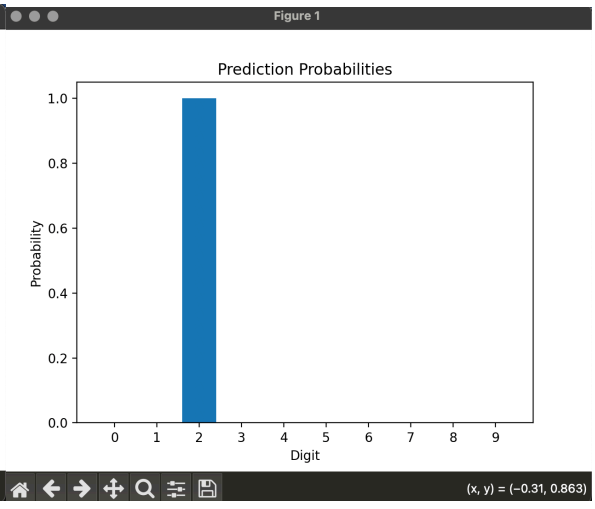


Using KNN:

```
Enter your choice (1/2/3): 2

Training KNN Classifier with n_neighbors=5...

Metrics for KNN:
Accuracy: 0.9688
Precision: 0.9690
Recall: 0.9688
F1 Score: 0.9687
Enter the full path of the image (png, jpg, jpeg) or type 'exit' to quit:
Image Path: /Users/jay/Downloads/2.jpeg

Preprocessing image: /Users/jay/Downloads/2.jpeg

The predicted digit is: 2
Prediction probabilities for each digit: ['0.0000', '0.0000', '1.0000', '0.0
0']
```

Actual Digit:

Predicted Digit:



The three models were evaluated on their performance using the MNIST dataset. The following key results were observed:

1. **Model Metrics**:
    - KNN achieved an accuracy of 96.88%, precision of 96.90%, recall of 96.88%, and an F1 score of 96.87%.
    - ANN achieved an accuracy of 97.73%, precision of 97.76%, recall of 97.73%, and an F1 score of 97.73%.
    - Random Forest achieved an accuracy of 97.16%, precision of 97.16%, recall of 97.16%, and an F1 score of 97.16%.
2. **Random Test Sample Predictions**:
    - Both KNN and ANN were able to correctly classify 10 out of 10 random test samples selected from the test set.
    - Random Forest, however, correctly classified 9 out of 10 random test samples.
3. **Prediction on User-Tested Digits**:
    - The test digits selected for evaluation included 2, 3, and 5. While both KNN and ANN provided correct predictions for these digits, the Random Forest classifier misclassified digit 5, resulting in incorrect outputs.
4. **Observations**:
    - ANN showed the highest metrics in accuracy, precision, recall, and F1 score. It demonstrated great generalization for some unseen test digits, classifying all 10 out of 10 test samples correctly.
    - KNN, although slightly behind ANN in metrics, consistently predicted all test samples correctly, including user-tested digits, which highlights its robustness on smaller datasets or less complex patterns.
    - Random Forest showed stable performance but struggled with one of the random samples, indicating possible sensitivity to noise or variations in data.

9. Insights and Considerations

- **KNN Strengths**:
    - KNN consistently classified both test samples and random examples correctly, underscoring its effectiveness for this dataset. However, its computational cost may increase for larger datasets due to the need to calculate distances for every test sample. This makes it suitable for smaller datasets or scenarios with sufficient computational resources for inference.
- **ANN Performance**:
    - ANN demonstrated the best metrics overall, achieving the highest accuracy and correctly classifying all test samples and random examples. This project highlights ANN's ability to handle complex patterns effectively. However, the number of passes made through the dataset (epoch) could be further reduced as after a certain number of passes, the accuracy was more or less the same.
- **Random Forest Performance**:
    - Random Forest remains a reliable ensemble method, balancing accuracy with interpretability. The misclassification for a random digit noted earlier was due to an edge case where the digit **8** was not written properly, making it inherently ambiguous even for human interpretation. It performed well

overall but may need further fine-tuning of hyperparameters to handle subtle variations in data.

10. Conclusion

Overall Best Model: Based on the experiments, ANN (Artificial Neural Network) emerged as the overall best-performing model. With the highest accuracy of 97.73%, it excelled in recognizing handwritten digits when tested on the MNIST dataset. Its adaptability to more complex features and non-linear decision boundaries makes it the most reliable for general cases. However, it is computationally more expensive and takes longer to train compared to the other models.

Best for Specific Conditions:

- **KNN (k=5)**:
  - **Best for Simplicity and Small Datasets**: KNN is ideal when computational resources are limited or when the dataset size is smaller. It achieved 96.88% accuracy, which is slightly lower than ANN, but it consistently performed well during real-world predictions, correctly classifying all test samples.
  - **Strengths**: Minimal training time, no model assumptions.
  - **Weaknesses**: Slower during predictions as it computes distances from all training points.
- **Random Forest**:
  - **Best for Real-World Generalization**: With an accuracy of 97.16%, Random Forest provided a strong balance between simplicity and performance. It is computationally efficient during predictions and handles noise and outliers better than KNN.
  - **Strengths**: Robust to overfitting due to ensembling, interpretable.
  - **Weaknesses**: Slightly lower performance compared to ANN and missed 1 out of 10 test digits during random prediction trials.

To conclude, for most general use cases and when computational power is available, ANN should be the preferred model due to its high accuracy and ability to capture complex patterns. For quick and reliable predictions, especially when computational resources are limited, Random Forest is a strong alternative. In low-resource environments or small datasets, KNN is the simplest and most interpretable model, providing consistently good results without requiring extensive preprocessing or parameter tuning.

11. Future Work

1. Integrate Convolutional Neural Networks (CNNs) for enhanced accuracy and robustness.
2. Expand to multi-digit classification for handwritten sentences or words.
3. Deploy as a web application for broader accessibility.

12. References

Mamgai, Utkarsh Shaw Tania Rishab. “Handwritten Digit Recognition Using Neural Network.” *GeeksforGeeks*, 30 Sept. 2024, www.geeksforgeeks.org/handwritten-digit-recognition-using-neural-network/. Accessed 20 Nov. 2024.

"Image Classification | Tensorflow Core." *TensorFlow*, 3 Apr. 2024, www.tensorflow.org/tutorials/images/classification. Accessed 20 Nov. 2024.

Nithyashree. "Image Classification Using Machine Learning." *Analytics Vidhya*, 12 June 2024, www.analyticsvidhya.com/blog/2022/01/image-classification-using-machine-learning/. Accessed 20 Nov. 2024.