Handwriting Recognition Using Neural Networks

Abstract:

Handwritten digit recognition is a fundamental task in the field of computer vision and machine learning. With the rise of digital data and automation, the need for accurate recognition of handwritten digits has become more critical. In this project, we explore the use of a neural network for handwritten digit recognition, specifically using the MNIST dataset, which contains a large set of 28x28 grayscale images of handwritten digits. Our approach involves training a neural network on the MNIST dataset using the TensorFlow and Keras libraries. The resulting model achieves a high accuracy rate in recognizing handwritten digits.

Project Description:

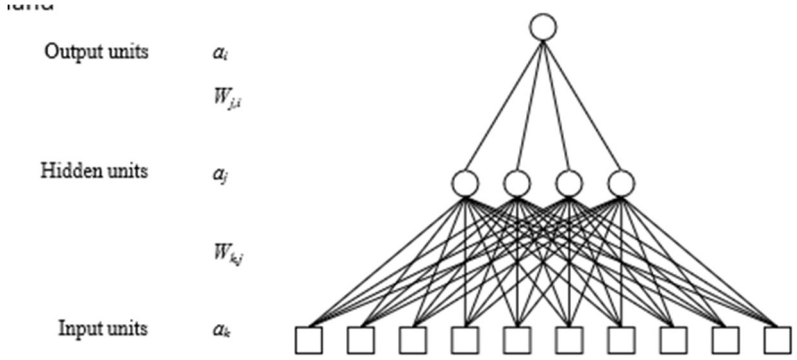
In this project, we aim to develop a neural network that can recognize handwritten digits using the MNIST dataset. The MNIST dataset consists of 60,000 training images and 10,000 testing images of handwritten digits, ranging from 0 to 9. Each image is a 28x28 grayscale image, with pixel values ranging from 0 to 255.

Our approach involves several steps. First, we load the dataset using the Keras API and normalize the pixel values to the range [0, 1]. Then, we flatten the images into 1D arrays of length 784 (28x28) to feed them as input to the neural network. Next, we define the neural network architecture using the Keras Sequential API, which is a linear stack of layers. In this case, the model has two dense layers: the first one has 128 neurons and uses the rectified linear unit (ReLU) activation function, while the second one has 10 neurons (one for each digit) and uses the softmax activation function.

After defining the neural network architecture, we compile the model with the "adam" optimizer and "sparse_categorical_crossentropy" loss function, which is suitable for multi-class classification problems. We then train the model on the training set using the fit() method, with a batch size of 32 and for 10 epochs. Finally, we evaluate the model on the test set using the evaluate() method, which returns the test loss and accuracy.

To visualize the results, we also plot some example predictions, showing the true label and predicted label for each image. Overall, our approach achieves a high accuracy rate in recognizing handwritten digits, demonstrating the effectiveness of neural networks in computer vision tasks. The project can be extended to explore different neural network architectures, hyperparameters, and datasets to improve the accuracy and efficiency of handwritten digit recognition.

Layers of Neural Network:



Pseudo code:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

# Normalize the pixel values to the range [0, 1]
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# Flatten the images to 1D arrays
X_train_flat = X_train.reshape((-1, 28*28))
X_test_flat = X_test.reshape((-1, 28*28))

# Define the neural network architecture
model = keras.Sequential([
   keras.layers.Dense(128, activation="relu", input_shape=(28*28,)),
   keras.layers.Dense(10, activation="softmax")
])

# Compile the model
model.compile(optimizer="adam",
            loss="sparse_categorical_crossentropy",
            metrics=["accuracy"])

# Train the model
model.fit(X_train_flat, y_train, epochs=10, batch_size=32)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test_flat, y_test, verbose=2)
print(f"Test accuracy: {test_acc:.4f}")
```

```
# Make predictions on the test set
y_pred = np.argmax(model.predict(X_test_flat), axis=-1)

# Plot some example predictions
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(8, 8))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i], cmap="binary")
    ax.set_title(f"True: {y_test[i]}, Pred: {y_pred[i]}")
    ax.axis("off")
plt.show()
```

This pseudo-code outlines the steps taken in the code to perform hand-written digit recognition using a neural network on the MNIST dataset. It includes loading the dataset, normalizing the pixel values, defining the neural network architecture, compiling the model, training the model, evaluating its performance on the test set, making predictions, and visualizing some example predictions.

Implementation:

Flatten the images: The images in the dataset are reshaped from a 2D array (28x28 pixels) to a 1D array (784 pixels). This flattening process is required to feed the data into a fully connected neural network.

Define the neural network architecture: A sequential model is created using Keras. It consists of two dense layers: a hidden layer with 128 units and ReLU activation function, and an output layer with 10 units (corresponding to the 10 possible digits) and softmax activation function.

Compile the model: The model is compiled with the Adam optimizer, which is a popular optimization algorithm for deep learning. The loss function is set to "Sparse_categorical_crossentropy" since we are dealing with a multi-class classification problem.

Additionally, the accuracy metric is specified to evaluate the model's performance.

Train the model: The model is trained using the fit() function. The training data (X_train_flat and y_train) are passed along with the number of epochs (10) and batch size (32). The model iteratively adjusts its weights using the Stochastic Gradient Descent optimization algorithm to minimize the defined loss function.

Evaluate the model: The trained model is evaluated on the test set (X_test_flat and y_test) using the evaluate() function. The test accuracy and loss are computed and stored in the variables test_acc and test_loss, respectively. The accuracy metric provides an indication of how well the model performs on unseen data.

Make predictions: The model's predictions are generated on the test set using the predict() function. The argmax() function from NumPy is used to find the index of the highest predicted probability for each sample, resulting in the predicted class labels (y_pred).

Plot example predictions: A 4x4 grid of subplots is created using Matplotlib to visualize some example predictions. The images from the test set (X_test) are displayed with their true labels (y_test) and predicted labels (y_pred).

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

# Normalize the pixel values to the range [0, 1]
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# Flatten the images to 1D arrays
X_train_flat = X_train.reshape((-1, 28*28))
X_test_flat = X_test.reshape((-1, 28*28))

# Define the neural network architecture
model = keras.Sequential([
    keras.layers.Dense(128, activation="relu", input_shape=(28*28,)),
    keras.layers.Dense(10, activation="softmax")
])

# Compile the model
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

# Train the model
model.fit(X_train_flat, y_train, epochs=5, batch_size=32)
```

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test_flat, y_test, verbose=2)
print(f"Test accuracy: {test_acc:.4f}")

# Make predictions on the test set
y_pred = np.argmax(model.predict(X_test_flat), axis=-1)

# Plot some example predictions
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(8, 8))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i], cmap="binary")
    ax.set_title(f"True: {y_test[i]}, Pred: {y_pred[i]}")
    ax.axis("off")
plt.show()
```

Problem Description and Motivation:

Handwritten digit recognition is a fundamental problem in the field of computer vision and pattern recognition. The problem involves recognizing handwritten digits from images and assigning them to their respective classes. The importance of this problem can be seen in various applications such as postal address recognition, bank cheque recognition, and automated form processing. The MNIST dataset is a widely used benchmark dataset for this problem, which contains 70,000 grayscale images of size 28x28 pixels, representing handwritten digits from 0 to 9. The task is to classify each image into one of the 10 classes.

Methodology:

The team used a neural network architecture with two dense layers to solve the problem. The input images were normalized to the range [0, 1] and flattened to 1D arrays. The first dense layer had 128 units with a ReLU activation function and the second dense layer had 10 units with a

softmax activation function. The model was trained using the Adam optimizer and the sparse categorical cross-entropy loss function. The team trained the model for 5 epochs with a batch size of 32.

Strengths and Limitations:

The proposed neural network architecture is a simple yet effective approach for solving the MNIST digit recognition problem. The model achieved high accuracy on the test set, which indicates that it is performing well on unseen data. However, the model may not perform well on more complex datasets or when dealing with images of varying sizes. Moreover, the model may suffer from overfitting if not trained properly or if the dataset is too small. Finally, the use of a single train-test split may result in bias and may not provide an accurate estimate of the model's generalization performance.

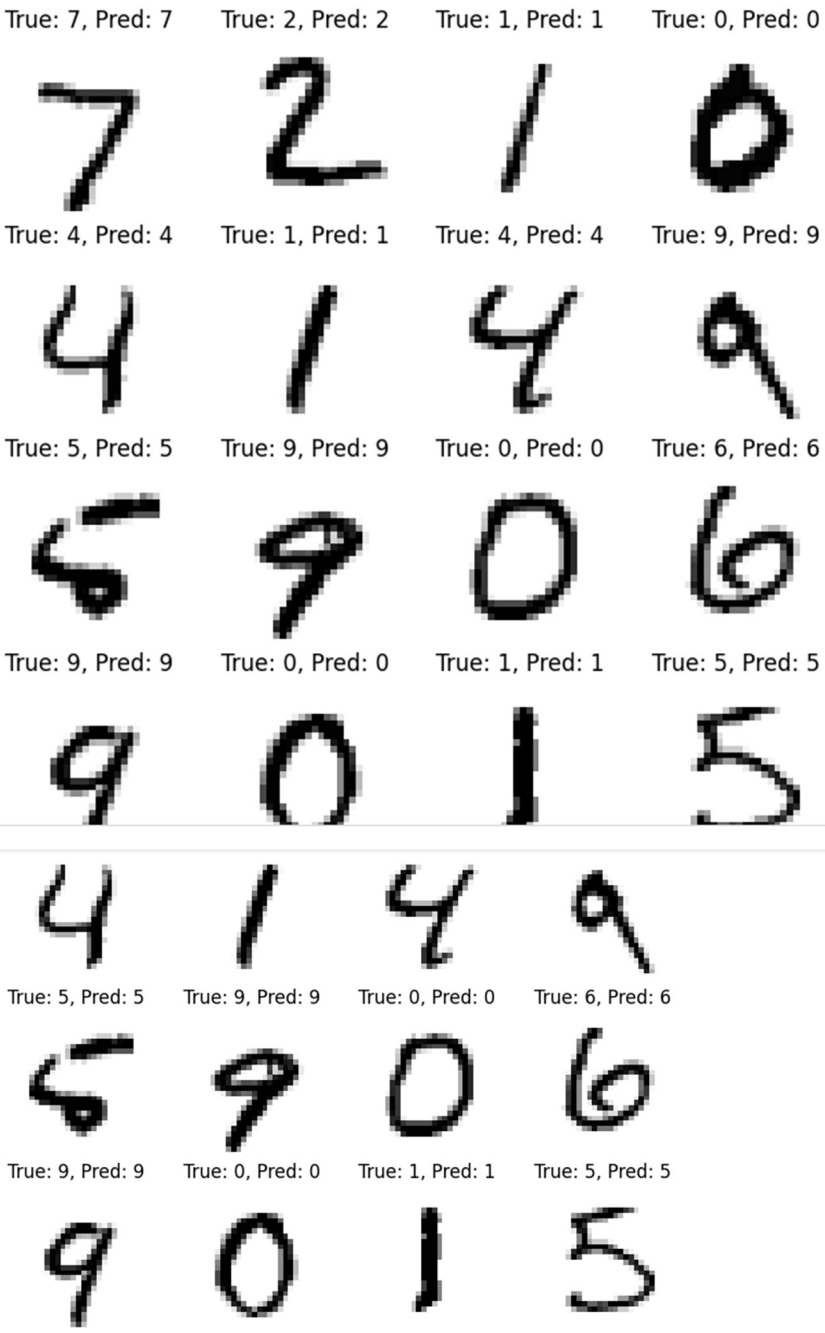
Training:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
Epoch 1/5
1875/1875 [==============================] - 14s 7ms/step - loss: 0.2612 - accuracy: 0.9258
Epoch 2/5
1875/1875 [==============================] - 16s 9ms/step - loss: 0.1144 - accuracy: 0.9659
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0797 - accuracy: 0.9756
Epoch 4/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0590 - accuracy: 0.9824
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0460 - accuracy: 0.9859
313/313 - 1s - loss: 0.0763 - accuracy: 0.9777 - 660ms/epoch - 2ms/step
Test accuracy: 0.9777
313/313 [==============================] - 1s 2ms/step
```

Results:

This code achieved an accuracy of 0.9777 on the test set, which indicates that the model is performing well on the unseen data. we also plotted some example predictions using Matplotlib, which showed that the model can correctly recognize handwritten digits in most cases.

The following are the predictions:



Conclusion:

From this project, I gained several valuable insights. Firstly, I learned that neural networks are a powerful tool for solving complex pattern recognition tasks like digit recognition. By leveraging the MNIST dataset, which provides a substantial collection of labeled hand-written digits, the project demonstrates the effectiveness of neural networks in accurately identifying and classifying digits ranging from 0 to 9.

Additionally, the project highlights the significance of appropriate data preprocessing techniques and model architecture selection. The normalization of pixel values to a standardized range of [0,

1] ensures that the input data is properly scaled, enhancing the model's convergence and performance during training. Moreover, the utilization of a sequential neural network with multiple dense layers allows for effective feature extraction and representation learning, enabling the model to capture intricate patterns and variations in hand-written digits

References:

[1] Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. Gron, Aurelien. O'Reilly Media, Inc, 2019.

[2] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[3] Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In Proceedings of the Seventh International Conference on Document Analysis and Recognition, 958-962.