INFOSYS SPRINGBOARD INTERNSHIP

Project:

Handwritten Digit Recognition Model

Members

Rudraksh Gupta

Sneha Kushwaha

Yash Nilesh Shah

Mentor

Narendra Kumar

COMPARISON

Aspect	LeNet-5	CNN	Logistic Regression
Architecture	A classic convolutional neural network with two convolutional layers followed by subsampling layers and fully connected layer	Typically involves a more complex architecture with more layers and parameters, which can be tailored to specific tasks.	A simple linear model without hidden layers, used as a baseline.
Number of Parameters	Moderate	Varies (typically more than LeNet-5)	Few
Training Complexity	Moderate complexity in training due to convolutional layers.	High complexity, potentially requiring careful tuning of many hyperparameters	Simple training process.
Training Time	Takes a moderate amount of time to train.	Can take a long time to train, especially with larger and deeper architectures.	Trains very quickly.
Accuracy (%)	Achieves the highest accuracy at 99%.	Achieved 85% accuracy, which might indicate suboptimal architecture or training.	Achieved 92% accuracy, which is impressive for a linear model.
Computational Resources	Requires moderate computational resources.	Requires high computational resources.	Requires high computational resources.
Strengths	More resource-intensive than logistic regression	Can be very resource-intensive and harder to optimize	Limited in capacity for handling complex tasks compared to neural networks.

Weaknesses	Requires more computational	Can be computationally expensive,	Limited capacity for complex tasks
Use Cases	Image classification,	Advanced image classification,	Simple tasks, baseline comparison,

Disclaimer - All the analysis done in this file is based on the programmed code published on the dedicated GitHub repository. For more details, please visit this website

Detailed Comparisions

Model 1: LeNet5

1. Introduction

The LeNet-5 model, designed by Yann LeCun in 1998, is one of the earliest convolutional neural networks and has been widely used for handwritten digit recognition. This report analyzes the implementation and performance of the LeNet-5 model on the MNIST dataset, detailing its architecture, training process, and evaluation metrics.

2. Dataset Overview

The MNIST dataset is a collection of 70,000 grayscale images of handwritten digits, each of size 28x28 pixels. It is partitioned into:

- **Training Set**: 60,000 images
- **Test Set**: 10,000 images

The dataset serves as a standard benchmark for image classification tasks, providing a basis for comparing different machine learning models.

3. Data Preprocessing

Effective preprocessing is essential to ensure the data is in the right format for the neural network. The preprocessing steps include:

- **Normalization**: Scaling pixel values to the range [0, 1] to facilitate faster training and convergence.

```
x_train_full = x_train_full / 255.0
x_test = x_test / 255.0
```

- **Splitting**: Dividing the dataset into training, validation, and test sets to monitor and evaluate the model's performance.

```
x_valid, x_train = x_train_full[:5000], x_train_full[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
```

- **Reshaping**: Adding a channel dimension to the images to match the input shape required by the LeNet-5 model.

```
x_train = x_train[..., np.newaxis]
x_valid = x_valid[..., np.newaxis]
x_test = x_test[..., np.newaxis]
```

- **One-Hot Encoding**: Converting the labels to one-hot encoded vectors for multi-class classification.

```
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_valid = tf.keras.utils.to_categorical(y_valid, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

4. Model Architecture

The LeNet-5 model is structured as follows:

- **Input Layer**: (28, 28, 1)
- **Convolutional Layer 1**: 6 filters, kernel size (5, 5), activation function: relu. This layer extracts basic visual features from the input.

```
model = models.Sequential([
    layers.Conv2D(6, (5, 5), activation='relu', input_shape=(28, 28, 1)),
])
```

- **Average Pooling Layer 1**: Reduces the spatial dimensions of the feature maps, retaining important information while reducing computational complexity.

```
layers.AveragePooling2D(),
```

- **Convolutional Layer 2**: 16 filters, kernel size (5, 5), activation function: relu. This layer captures more detailed features.

```
layers.Conv2D(16, (5, 5), activation='relu'),
layers.AveragePooling2D(),
```

- **Flatten Layer**: Converts the 2D feature maps into a 1D vector for input into fully connected layers.

```
layers.Flatten(),
```

- **Dense Layer 1**: 120 units, activation function: relu. This fully connected layer combines features in a non-linear fashion.

```
layers.Dense(120, activation='relu'),
```

- **Dense Layer 2**: 84 units, activation function: relu. Another fully connected layer for feature combination.

```
layers.Dense(84, activation='relu'),
```

- **Output Layer**: 10 units, activation function: softmax. Outputs a probability distribution over the 10 digit classes.

```
layers.Dense(10, activation='softmax')
```

5. Training Process

Training the LeNet-5 model involves optimizing its weights to minimize the loss function. Key training details include:

- **Optimizer**: Adam optimizer, known for its adaptive learning rate and efficiency.

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

- **Loss Function**: categorical_crossentropy, suitable for multi-class classification problems with one-hot encoded labels.
- **Metrics**: Accuracy is used to monitor the model's performance during training.
- **Epochs**: The model is trained for 10 epochs.

```
history = model.fit(x_train, y_train, epochs=10,
validation_data=(x_valid, y_valid))
```

- **Batch Size**: The model processes 128 samples at a time, balancing training speed and memory usage.

6. Evaluation Metrics

The model's performance is evaluated using the test set. Key metrics include:

- **Validation Accuracy**: Monitored during training to prevent overfitting.
- **Test Accuracy**: The final accuracy on the test set, reflecting the model's generalization capability.

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('\nTest accuracy:', test_acc)
```

7. Conclusion and Recommendations

The LeNet-5 model, despite being a pioneering architecture, remains effective for handwritten digit recognition tasks. Its structured layers efficiently extract and combine features to achieve high accuracy. For future work, consider:

- **Exploring Deeper Architectures**: Investigate more complex models to potentially improve performance.
- **Hyperparameter Optimization**: Experiment with different learning rates, batch sizes, and other hyperparameters.
- **Data Augmentation**: Introduce techniques like rotation, scaling, and translation to increase the robustness of the model.
- **Evaluation on Complex Datasets**: Test the model on datasets with more variability and noise to assess its robustness and generalizability.

Model 2: Logistic Regression

1. Introduction:

This model utilizes logistic regression to classify handwritten digits from the MNIST dataset. Logistic regression is a straightforward and interpretable linear model commonly used for binary classification but can be extended to multi-class problems using techniques like one-vs-rest or softmax regression.

2. Dataset:

- **Dataset:** The MNIST dataset, a widely used dataset for benchmarking machine learning algorithms.
- **Classes:** 10 distinct classes representing digits from 0 to 9.
- **Samples:** 60,000 training images and 10,000 test images.
- **Image Size:** Each image is 28x28 pixels.
- **Preprocessing:**
    - The pixel values were scaled using StandardScaler, which standardizes features by removing the mean and scaling to unit variance. This helps in achieving better model performance and faster convergence.

3. Model Training:

- **Algorithm:** Logistic Regression
- **Hyperparameters:**
    - **tol:** 0.1 (Tolerance for stopping criteria)
    - **solver:** 'lbfgs' (An optimizer in the family of quasi-Newton methods)
    - **max_iter:** Increased from the default to ensure better convergence.
- **Training Duration:** Not specified, but logistic regression is generally faster to train compared to neural networks.
- **Computational Resources:** Not specified, but logistic regression is less computationally intensive and can be trained on a standard CPU.

4. Evaluation:

- **Cross-Validation Accuracy:** Provided, indicating the model's performance on different subsets of the training data. Cross-validation helps in assessing the model's robustness and generalizability.
- **Test Accuracy:** The model achieved a high accuracy on the test set, demonstrating its effectiveness for this classification task.
- **Performance Metrics:** Detailed metrics such as precision, recall, F1-score, and confusion matrix were not provided, but overall accuracy was reported.

5. Visualizations:

- **Sample Digit:** Displayed a sample digit using matplotlib.
- **Predictions:** Demonstrated predictions for a sample digit and 10 random digits from the test set, showcasing the model’s performance.

6. Conclusion:

- **Pros:**
  - Logistic regression is simple and easy to implement, making it suitable for quick prototyping and baseline models.
  - It requires less computational resources compared to more complex models like CNNs.
  - Logistic regression models are interpretable, allowing for easy understanding of the feature contributions.
- **Cons:**
  - Logistic regression may not capture complex patterns in the data as effectively as CNNs.
  - It tends to have lower accuracy compared to more sophisticated models for image classification tasks.
- **Recommendations:**
  - Experiment with regularization techniques, such as L1 or L2 regularization, to prevent overfitting and improve generalization.
  - Explore feature engineering techniques to enhance model performance, such as adding polynomial features or using dimensionality reduction methods like PCA.

Model 3: Convolution Neural Network

1. Introduction

Handwritten digit recognition is a fundamental problem in the field of computer vision and machine learning. This report focuses on a Convolutional Neural Network (CNN) model designed to classify digits from the MNIST dataset. The CNN model leverages convolutional layers to automatically and adaptively learn spatial hierarchies of features from input images, making it particularly effective for image recognition tasks.

2. Dataset Overview

The MNIST dataset is a benchmark dataset in machine learning, comprising 70,000 grayscale images of handwritten digits from 0 to 9. Each image is of size 28x28 pixels. The dataset is split into:

- **Training Set**: 60,000 images
- **Test Set**: 10,000 images

The dataset is widely used for training and evaluating image processing systems and serves as an excellent starting point for exploring the capabilities of neural networks.

3. Data Preprocessing

Data preprocessing steps are crucial for ensuring that the input data is in the right format and range for the neural network to process efficiently. The preprocessing steps include:

- **Normalization**: The pixel values of the images are scaled to fall within the range [0, 1]. This normalization step helps in faster convergence during training.

```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

- **Reshaping**: The images are reshaped to include a single channel, transforming the shape from (28, 28) to (28, 28, 1). This is necessary because the CNN expects a 3D input.

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

- **Subset Selection**: A subset of 100 images from the training set is selected for initial training.

```
x = x_train[:100]
y = y_train[:100]
```

4. Model Architecture

The CNN model architecture designed for this task consists of multiple layers, each with specific functions:

- **Convolutional Layer 1**: Applies 6 filters of size (3, 3) with the tanh activation function. This layer extracts basic features such as edges.

```
model.add(Conv2D(filters=6, kernel_size=(3, 3), activation="tanh",
input_shape=(28, 28, 1)))
```

- **Average Pooling Layer 1**: Reduces the dimensionality of the feature maps while retaining the important information.

```
model.add(AveragePooling2D())
```

- **Convolutional Layer 2**: Applies 12 filters of size (3, 3) with the tanh activation function. This layer extracts more complex features from the input.

```
model.add(Conv2D(filters=12, kernel_size=(3, 3), activation="tanh"))
```

- **Average Pooling Layer 2**: Further reduces the dimensionality of the feature maps.

```
model.add(AveragePooling2D())
```

- **Flatten Layer**: Converts the 2D feature maps into a 1D vector, which can be fed into fully connected layers.

```
model.add(Flatten())
```

- **Dense Layer 1**: A fully connected layer with 50 units and the tanh activation function. This layer learns combinations of features that represent digit classes.

```
model.add(Dense(50, activation="tanh"))
```

- **Output Layer**: A fully connected layer with 10 units (one for each digit class) and the softmax activation function, which outputs a probability distribution over the 10 classes.

```
model.add(Dense(10, activation="softmax"))
```

5. Training Process

Training the CNN involves optimizing its parameters to minimize the loss function using a chosen optimizer. Key aspects of the training process include:

- **Optimizer**: Adam optimizer is used for its efficiency and adaptive learning rate properties.

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

- **Loss Function**: sparse_categorical_crossentropy is chosen because the target labels are integers.
- **Metrics**: Accuracy is used to evaluate the model's performance.
- **Epochs**: The model is trained for 10 epochs.

```
model.fit(x_train, y_train, epochs=10)
```

- **Batch Size**: The model processes 32 samples at a time before updating the model parameters.

6. Evaluation Metrics

After training, the model's performance is evaluated using the test set. The key metric is accuracy:

- **Test Accuracy**: The model achieves an accuracy of approximately 98% on the test set.

```
score = model.evaluate(x_test, y_test)
print(score)
```

7. Inference Example

An inference example is provided to demonstrate the model's capability to predict unseen data:

- A test image is displayed, and the model predicts the digit class with high confidence.

```
plt.imshow(x_test[3].reshape(28, 28), cmap="gray")
plt.show()

result = model.predict([x_test[3].reshape(1, 28, 28, 1)])
print(np.argmax(result))
```

Conclusion and Recommendations

The LeNet-5 model exhibits excellent performance on the MNIST dataset, achieving high accuracy. Its architecture is relatively simple, making it suitable for recognizing isolated handwritten digits. Future work could explore:

- **Expanding the Training Set**: Using more images for training to improve generalization.
- **Hyperparameter Tuning**: Optimizing learning rate, batch size, and other hyperparameters for better results.
- **Evaluating on More Complex Datasets**: Testing the model on datasets with more variability to assess robustness.