Handwritten Numeral Recognition

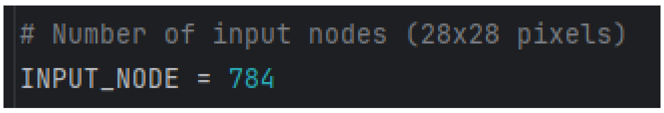
Wang Ruitao
5321BG01

Introduction

This program is aimed to use a neural network to recognize handwritten digits using the MNIST dataset. Here is an introduction explaining each module used in the program.
MNIST Dataset: The MNIST dataset is a widely-used dataset in machine learning and deep learning for benchmarking classification models. It contains 70,000 images of handwritten digits (0-9), with each image being 28x28 pixels. In this program, MNIST is used as the training and evaluation dataset for developing a neural network that can recognize handwritten numerals.
TensorFlow: TensorFlow is an open-source machine learning library developed by Google, providing a flexible and powerful framework for creating neural networks. In this program, TensorFlow is used for defining, training, saving the model and inference.
NumPy: NumPy is a powerful numerical library for Python that facilitates efficient manipulation of arrays and matrices. In this program, NumPy is used for reshaping and scaling image data from the MNIST dataset.
Pillow: The Pillow library, imported as Image from PIL, is used for image processing. The program uses it to convert the input images into a grayscale format and resize them to 28x28 pixels, ensuring that they match the dimensions expected by the neural network.
OpenCV (cv2): OpenCV is an open-source computer vision and image processing library. In this program, OpenCV is used to read input images in grayscale mode, apply thresholding for noise removal, and prepare the images for digit recognition.

Model Definition

In the Model Definition part of the program, a three-layer fully connected neural network is used to recognize handwritten digits.
Input Layer:
This layer receives the input data.
Each input image has dimensions of 28x28 pixels, which are flattened into a vector of length 784. Hence, the input layer has 784 nodes.
This layer essentially serves as a conduit to feed data into the network

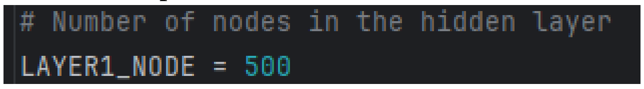


```
# Number of input nodes (28x28 pixels)
INPUT_NODE = 784
```

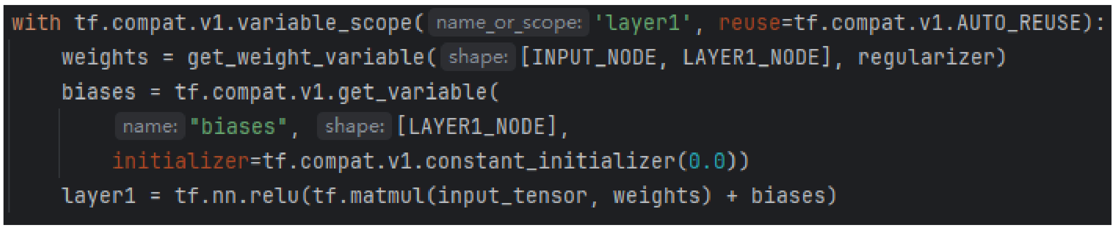
Hidden Layer
The hidden layer is crucial in allowing the network to learn complex patterns in the data. This layer has 500 nodes in the given model.

A ReLU (Rectified Linear Unit) activation function is applied to the linear combination of the inputs and weights to introduce non-linearity.

The weights and biases for this layer are learned during training, allowing it to transform the input data into a representation that makes classification easier.



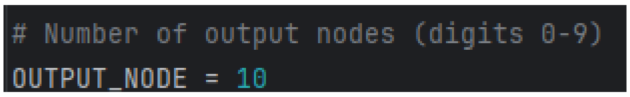
```
# Number of nodes in the hidden layer
LAYER1_NODE = 500
```



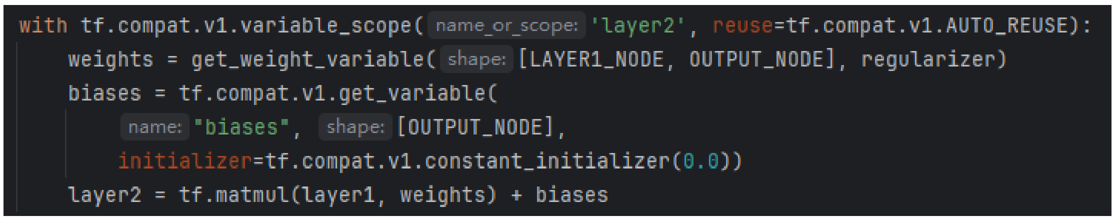
```
with tf.compat.v1.variable_scope( name_or_scope: 'layer1', reuse=tf.compat.v1.AUTO_REUSE):
    weights = get_weight_variable( shape: [INPUT_NODE, LAYER1_NODE], regularizer)
    biases = tf.compat.v1.get_variable(
        name: "biases", shape: [LAYER1_NODE],
        initializer=tf.compat.v1.constant_initializer(0.0))
    layer1 = tf.nn.relu(tf.matmul(input_tensor, weights) + biases)
```

Output Layer

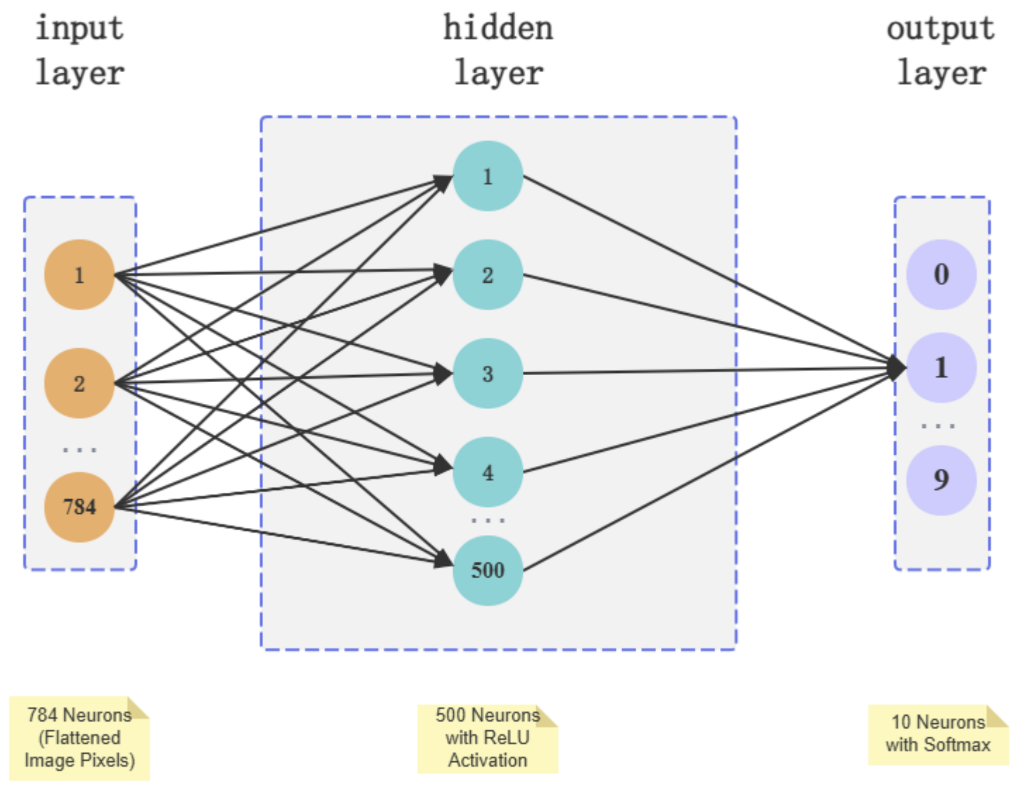
The output of this layer is passed to a softmax function during loss calculation to produce a probability distribution over the possible classes (digits 0-9).



```
# Number of output nodes (digits 0-9)
OUTPUT_NODE = 10
```



```
with tf.compat.v1.variable_scope( name_or_scope: 'layer2', reuse=tf.compat.v1.AUTO_REUSE):
    weights = get_weight_variable( shape: [LAYER1_NODE, OUTPUT_NODE], regularizer)
    biases = tf.compat.v1.get_variable(
        name: "biases", shape: [OUTPUT_NODE],
        initializer=tf.compat.v1.constant_initializer(0.0))
    layer2 = tf.matmul(layer1, weights) + biases
```



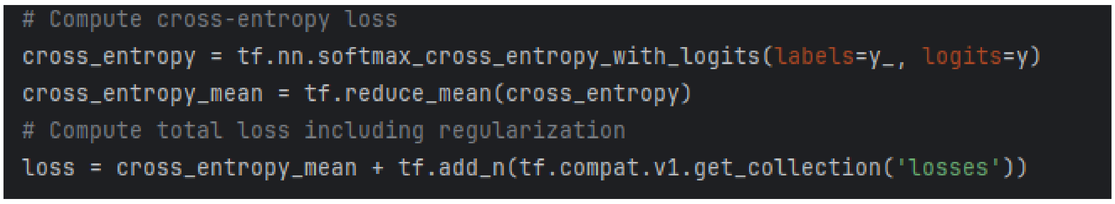
Training process

Data Preparation: Load, normalize, reshape, and one-hot encode the MNIST dataset.

Define Input Placeholders: The placeholders are used to feed data into the network.

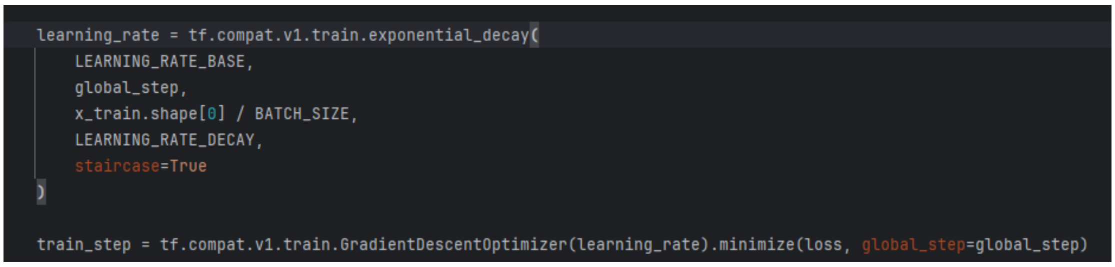
Model Forward Propagation: Using the forward propagation function (mnist_inference.inference) to get predictions.

Loss Calculation: The loss function is defined to measure the difference between the true labels (y_) and the predicted labels (y)using cross-entropy loss.



```
# Compute cross-entropy loss
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y)
cross_entropy_mean = tf.reduce_mean(cross_entropy)
# Compute total loss including regularization
loss = cross_entropy_mean + tf.add_n(tf.compat.v1.get_collection('losses'))
```

Optimization: Gradient descent is used to minimize the loss function.



```
learning_rate = tf.compat.v1.train.exponential_decay(
    LEARNING_RATE_BASE,
    global_step,
    x_train.shape[0] / BATCH_SIZE,
    LEARNING_RATE_DECAY,
    staircase=True
)

train_step = tf.compat.v1.train.GradientDescentOptimizer(learning_rate).minimize(loss, global_step=global_step)
```

Training Loop: Iterate over batches of data, perform training, and update model parameters.

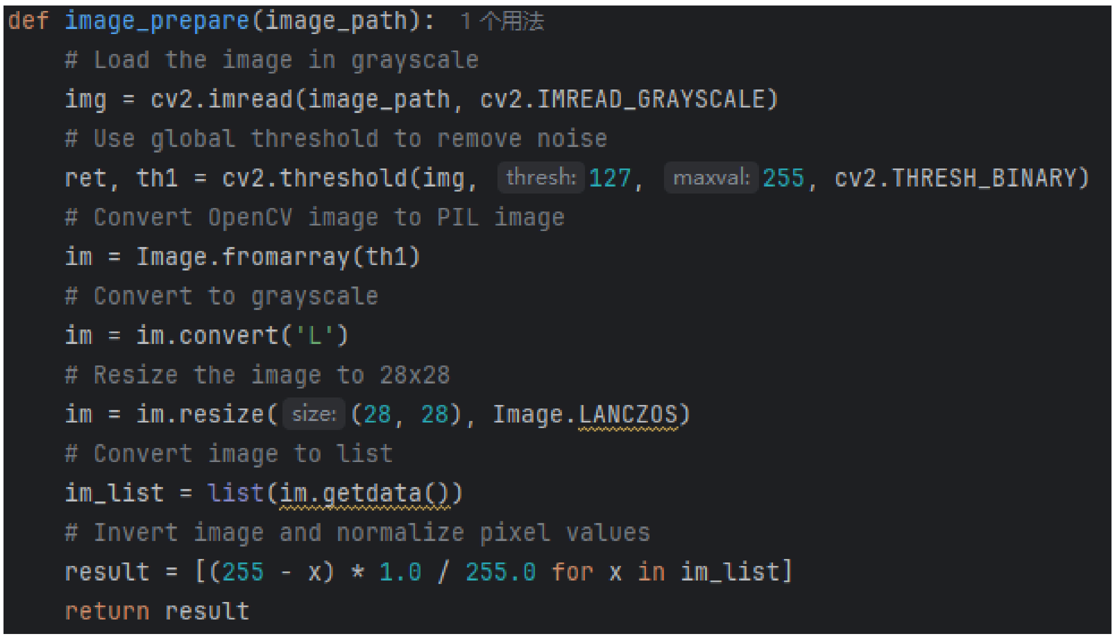
Model Saving: The trained model is saved for future use.

Evaluation

The model is evaluated on validation or custom input images to make predictions. Moving averages of the variables are used to make predictions more stable.

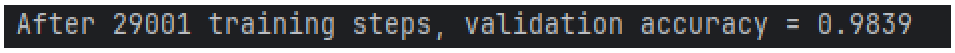
Data preprocessing

Normalize, reshape, and prepare images for input to the network.



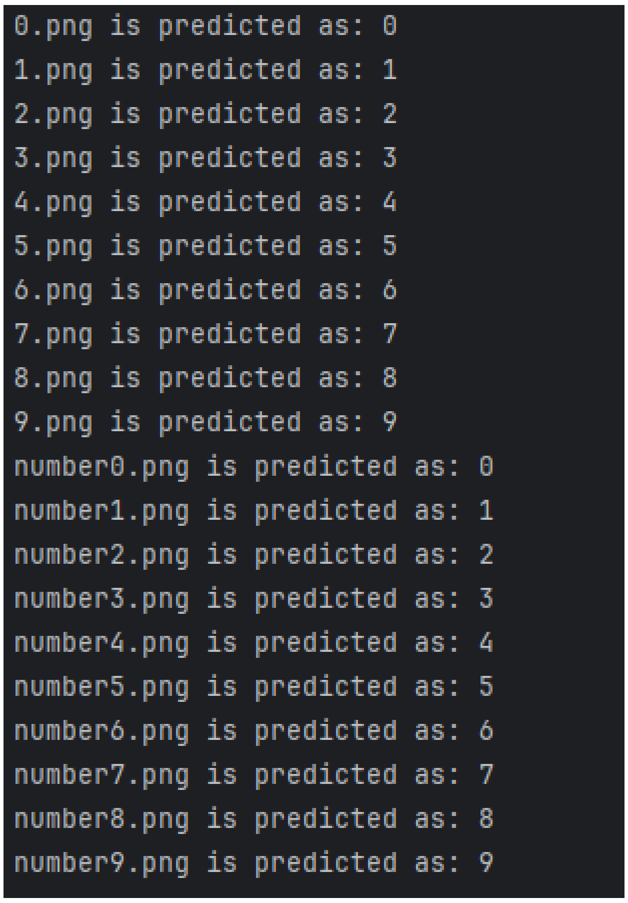
```python
def image_prepare(image_path):  1 个用法
    # Load the image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # Use global threshold to remove noise
    ret, th1 = cv2.threshold(img,  thresh: 127,  maxval: 255, cv2.THRESH_BINARY)
    # Convert OpenCV image to PIL image
    im = Image.fromarray(th1)
    # Convert to grayscale
    im = im.convert('L')
    # Resize the image to 28x28
    im = im.resize( size: (28, 28), Image.LANCZOS)
    # Convert image to list
    im_list = list(im.getdata())
    # Invert image and normalize pixel values
    result = [(255 - x) * 1.0 / 255.0 for x in im_list]
    return result
```

Results



```
After 29001 training steps, validation accuracy = 0.9839
```

Running the mnist_eval.py file is to calculate the accuracy of the test images on the mnist data set.



```
0.png is predicted as: 0
1.png is predicted as: 1
2.png is predicted as: 2
3.png is predicted as: 3
4.png is predicted as: 4
5.png is predicted as: 5
6.png is predicted as: 6
7.png is predicted as: 7
8.png is predicted as: 8
9.png is predicted as: 9
number0.png is predicted as: 0
number1.png is predicted as: 1
number2.png is predicted as: 2
number3.png is predicted as: 3
number4.png is predicted as: 4
number5.png is predicted as: 5
number6.png is predicted as: 6
number7.png is predicted as: 7
number8.png is predicted as: 8
number9.png is predicted as: 9
```

Save the picture to be tested in the picture folder and run the app.py file to output the test results.

Discussion

This week, I conducted my first attempt to complete a full machine learning project, and it has been both challenging and rewarding. I spent a lot of time learning, researching, and asking friends for guidance, and I'm quite satisfied with the result I managed to achieve.

Throughout the process, I faced numerous challenges. One of the toughest moments was figuring out how to test my model with my own handwritten images. Initially, I had no idea how to input custom images into the trained model. After several times of rereading the model-building process, I realized that the input node requires a vector of length 784. Thus, I needed to convert my images into the same format, which led me to the image_prepare () function in app.py. Using the image processing library PIL, I converted my images to grayscale, resized them to 28x28 pixels, and then reshaped them into a 784-length array. Then I finally got my own results.

Another difficulty was the poor accuracy when testing with my own handwritten images. While the model achieved 98.39% accuracy on the MNIST dataset, it hardly could correctly identify my own handwritten digits, most of which were incorrectly classified as the digit 8. After asking for help from a friend who majored in CS, I learned that this issue was because the MNIST images are black digits on a white background, while my own images were white digits on a black background. To fix this, I needed to invert the grayscale of my test images and apply binarization to reduce noise for better results.

This programming journey has been incredibly interesting and has sparked a deeper interest in the field of machine learning. Despite the difficulties, I totally enjoyed the process and am eager to learn more.

Reference

[1] https://www.tensorflow.org/
[2] 《Tensorflow：实战 Google 深度学习框架》, 郑泽宇,顾思宇,2017.3
[3] https://en.m.wikipedia.org/wiki/MNIST_database