上机实验二：基于卷积神经网络的MNIST手写体数字识别

计卓2101 高僖 U202115285

数据预处理

对于MNIST手写数字数据集，我进行了如下预处理：

1. **加载数据集：** 使用TensorFlow的Keras库加载MNIST数据集，其中包括训练集和测试集。
2. **数据形状调整：** 调整图像的形状以适应模型的输入。原始图像是28x28的灰度图像，将其调整为大小为(28, 28, 1)的张量。
3. **数据类型转换：** 将图像的像素值从整数转换为浮点数，并进行归一化处理，使得像素值在0到1之间。
4. **标签独热编码：** 对训练集和测试集的标签进行独热编码处理，以便与模型输出的格式匹配。

网络结构

构建了一个CNN，其中包含了ResNet块。网络结构如下：

1. **输入层：** 输入层接受大小为(28, 28, 1)的图像。
2. **卷积层1：** 32个3x3的卷积核，ReLU激活函数，批量归一化。
3. **ResNet块：** 包含两个卷积层，每个卷积层有32个 3x3 或者 5x5 的卷积核，使用了ReLU激活函数，批量归一化。
4. **卷积层2：** 64个3x3的卷积核，ReLU激活函数，批量归一化。
5. **池化层：** 最大池化层，池化窗口大小为2x2。
6. **展平层：** 将卷积层输出的张量展平成一维向量。
7. **全连接层1：** 128个神经元，ReLU激活函数，批量归一化。
8. **输出层：** 10个神经元，使用Softmax激活函数进行多类别分类。

ResNet块

结构说明

ResNet块包含两个卷积层，每个卷积层后跟一个批量归一化层和ReLU激活函数。为了引入残差连接，我在第一个卷积层的输出和第二个卷积层的输出之间添加了一个残差连接。

1. **第一个卷积层：** 使用3x3 或者 5x5 的卷积核，ReLU激活函数，批量归一化。这一层旨在获取输入的特征。
2. **第二个卷积层：** 同样使用3x3 或者 5x5 的卷积核，ReLU激活函数，批量归一化。这一层用来获取第一层的特征。

3. **残差连接：** 如果两个卷积层的输出形状相同，将它们相加。如果不同，我们使用一个额外的1x1卷积层（残差卷积）来调整形状，确保可以相加。

4. **ReLU激活：** 将残差连接的结果通过ReLU激活函数，引入非线性。

目的

引入ResNet块的主要目的是解决梯度消失问题，特别是在训练深度网络时。由于残差连接，即使网络很深，模型也能够有效地学习恒等映射，从而避免了梯度逐渐减小到接近零的问题。

实验结果

本次实验由于本机性能有限，只训练了一个epoch，但是也获得了不错的结果。对于不同网络结构的尝试，主要在于resnet模块的卷积核大小的更改。我分别使用了3x3和5x5的卷积核进行了训练。训练结果和分析如下：

使用 3 * 3 卷积核

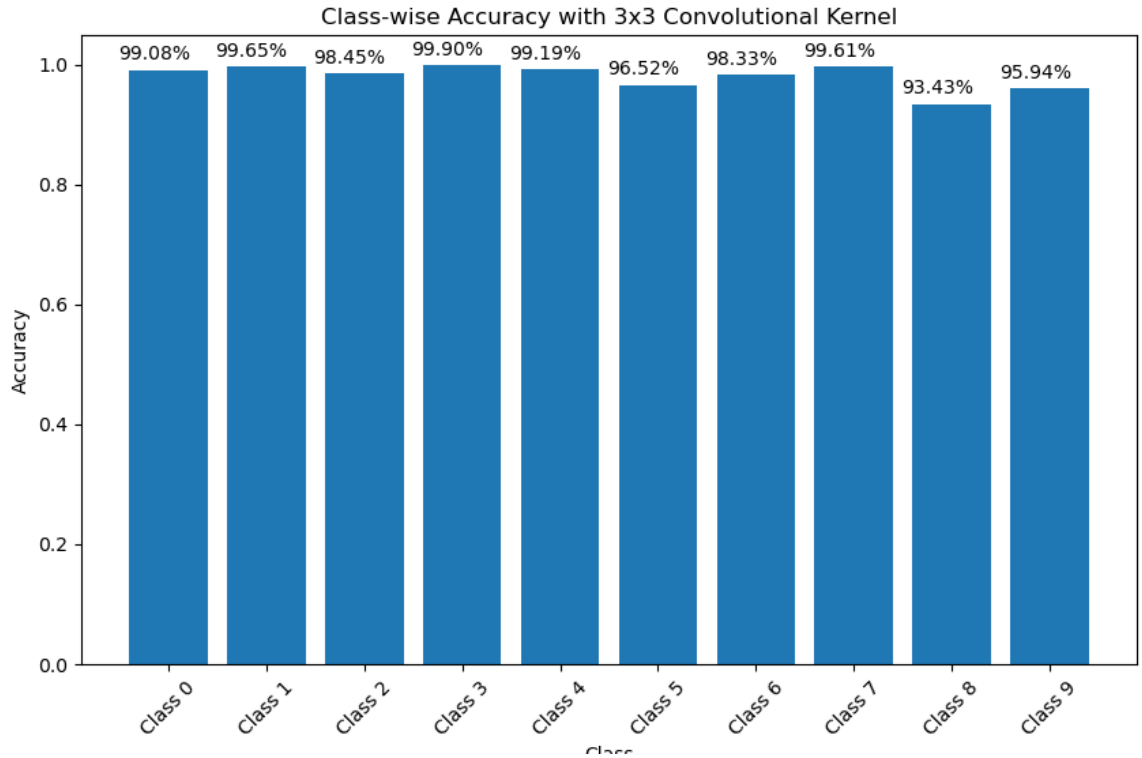
训练结果

- **最终测试结果：**
  - 测试损失：0.0645
  - 测试准确率：98.06%
- **最终训练结果：**
  - 训练损失：0.0552
  - 训练准确率：98.31%

类别准确率

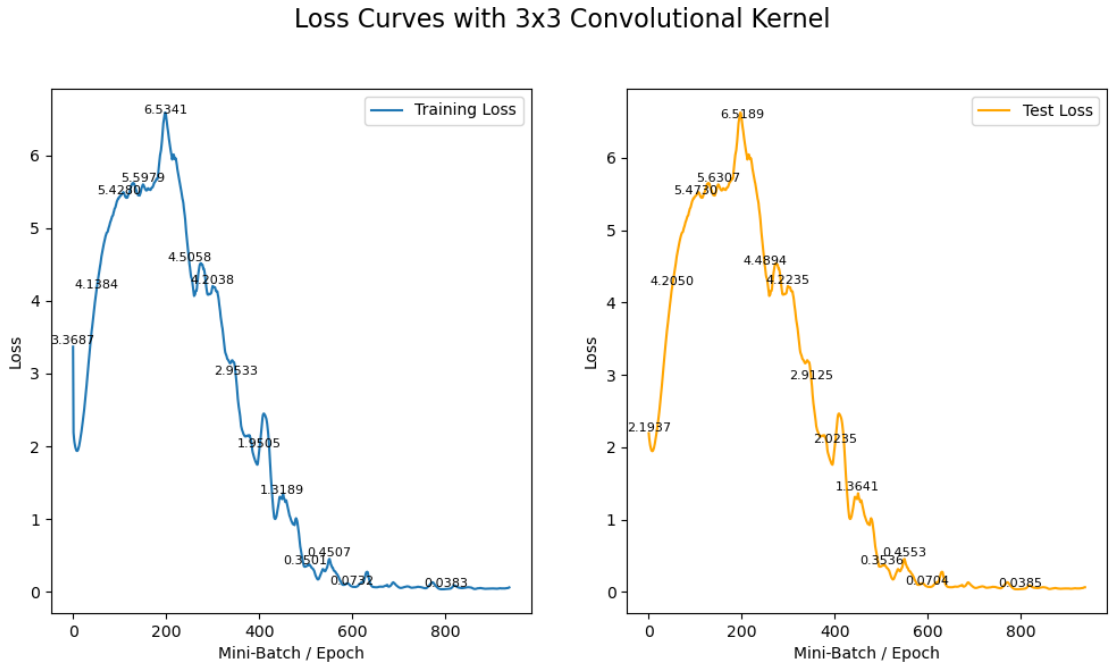
以下是测试集中每个类别的准确率：

```
Class 0: 99.08%
Class 1: 99.65%
Class 2: 98.45%
Class 3: 99.90%
Class 4: 99.19%
Class 5: 96.52%
Class 6: 98.33%
Class 7: 99.61%
Class 8: 93.43%
Class 9: 95.94%
```



通过这些结果可以看出，模型在各个类别上都表现良好，尤其是在数字3的准确率最高，达到了99.90%。整体而言，该模型在MNIST手写数字分类任务上取得了很好的性能。

每一轮mini-batch后的损失



使用 5*5 卷积核

训练结果

- **最终测试结果：**

    - 测试损失：0.0325
    - 测试准确率：98.89%

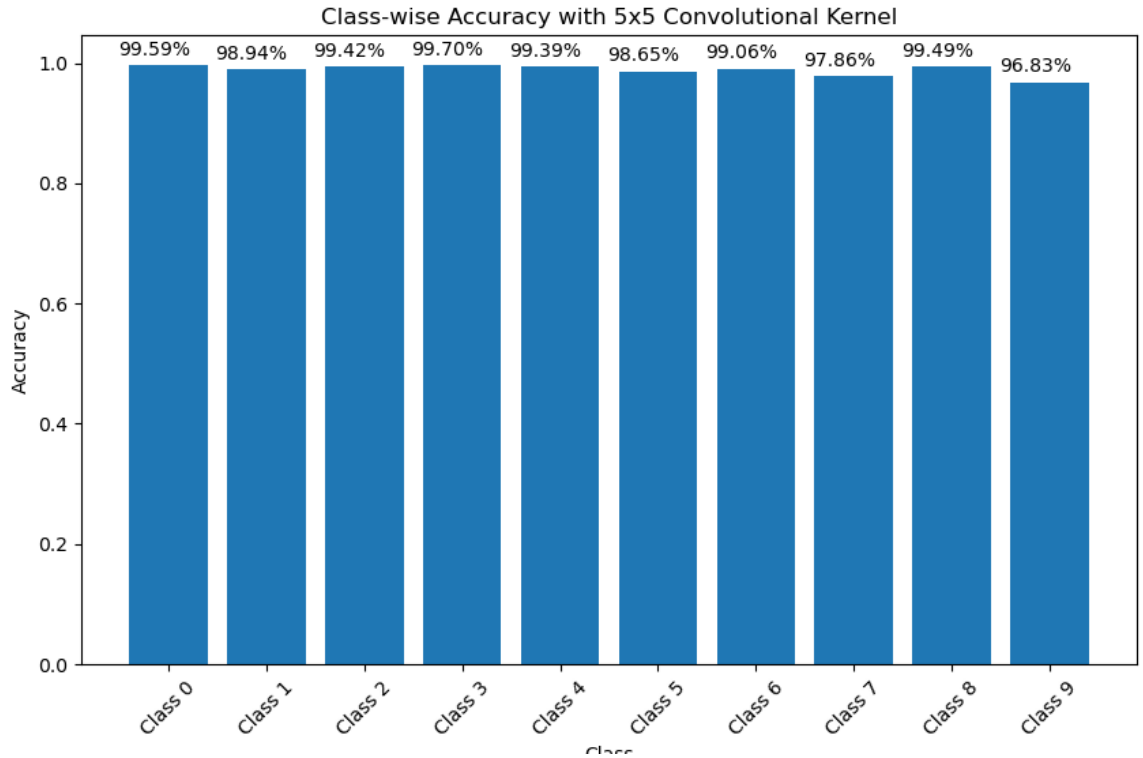
- **最终训练结果：**

    - 训练损失：0.0274
    - 训练准确率：99.13%

类别准确率

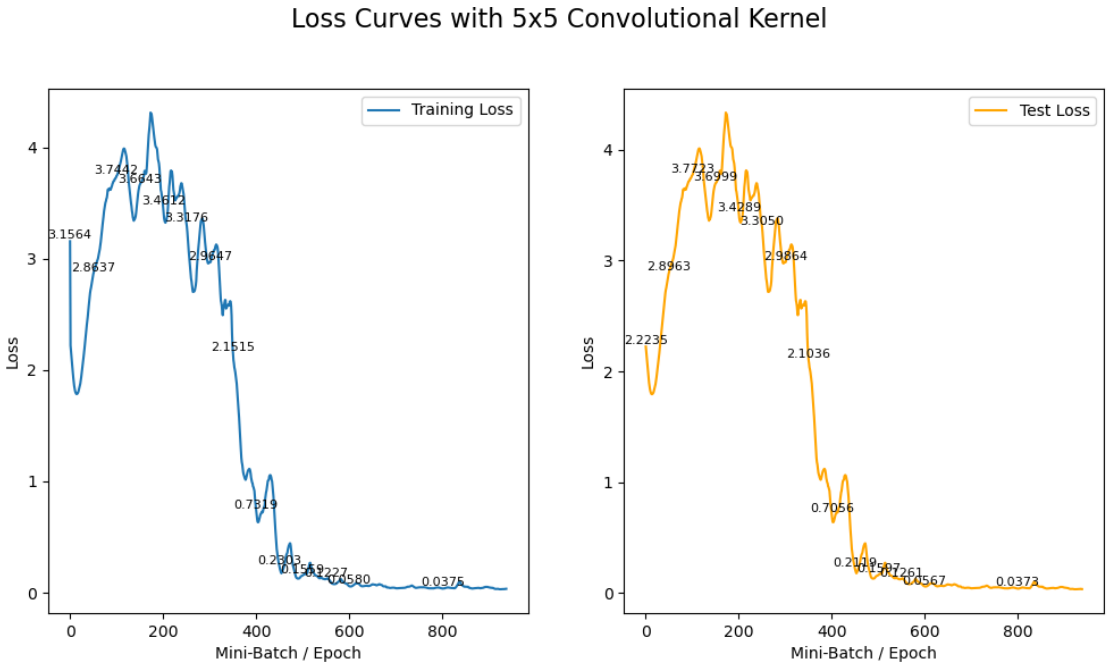
以下是测试集中每个类别的准确率：

```
Class 0: 99.59%
Class 1: 98.94%
Class 2: 99.42%
Class 3: 99.70%
Class 4: 99.39%
Class 5: 98.65%
Class 6: 99.06%
Class 7: 97.86%
Class 8: 99.49%
Class 9: 96.83%
```



通过这些结果可以看出，模型在各个类别上都表现良好

每一轮mini-batch后的损失



关于Loss曲线的分析

不难发现，两种不同网络结构的loss曲线，特点相近：

- **0~50 mini-batch：** 模型刚开始训练，loss 上升。据我分析，这应该是模型正在学习数据中的模式，参数还没有得到充分更新。
- **50~100 mini-batch：** loss 开始下降，模型逐渐学到了数据的特征，性能提升。
- **100~200 mini-batch：** loss 下降相对平稳，模型进一步收敛。这个阶段模型可能已经较好地学到了数据的特征。
- **200~450 mini-batch：** loss 下降的过程中有一些波动，这可能是由于学习率较大或者数据中的噪声引起的。
- **450~600 mini-batch：** loss 下降明显，模型性能进一步提升。此时可以考虑在模型训练的早期引入学习率衰减策略，以更好地控制训练的稳定性。
- **600~900+ mini-batch：** loss 继续下降，最终趋于稳定。在这个阶段，可以观察到 loss 下降的幅度减缓，模型逐渐收敛到最优解。

实验总结

总体来说，一次epoch可以达到这个精度我还是很满意的。至于Resnet框架中的卷积核的大小是3还是5目前来看对精度影响不是很大，倒是对于训练时间影响不小：前者30分钟，后者45分钟，我认为以后设计网络还是得平衡准确度与网络的性能。当然了，我认为最影响性能的是每一轮mini-batch都要计算test loss。我自认为，这会导致在每个 mini-batch 后都进行一次完整的测试集前向传播和损失计算，从而显著增加计算量，导致训练速度变慢。

至于Loss曲线，观察后也不难看出，在mini-batch 200-400附近，loss的下降有一些波动，如果有时间，我还是计划尝试减小学习率来平缓 loss 下降的过程。但是由于硬件限制，一次训练就花了不少时间，我决定在提交本次实验报告之前就不再重新训练了。

```
%matplotlib auto
```

```
Using matplotlib backend: <object object at 0x000002179265F930>
```

3 * 3 卷积核

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import pandas as pd

# 定义ResNet模块
class ResNetBlock(layers.Layer):
    def __init__(self, filters, kernel_size=3, stride=1):
        super(ResNetBlock, self).__init__()
        self.conv1 = layers.Conv2D(filters, kernel_size=kernel_size,
strides=stride, padding='same')
        self.batch_norm1 = layers.BatchNormalization()
        self.relu1 = layers.Activation('relu')

        self.conv2 = layers.Conv2D(filters, kernel_size=kernel_size,
strides=stride, padding='same')
        self.batch_norm2 = layers.BatchNormalization()

        # 1x1 convolutional layer for the residual connection
        if stride > 1:
            self.residual_conv = layers.Conv2D(filters, kernel_size=1,
strides=stride, padding='same')
        else:
            self.residual_conv = None

        self.relu2 = layers.Activation('relu')

    def call(self, inputs, training=False):
        x = self.conv1(inputs)
        x = self.batch_norm1(x, training=training)
        x = self.relu1(x)

        x = self.conv2(x)
        x = self.batch_norm2(x, training=training)

        # Apply residual connection if needed
        if self.residual_conv is not None:
            inputs = self.residual_conv(inputs)

        x = layers.add([inputs, x])
        x = self.relu2(x)
        return x
```

```
# 构建卷积神经网络
def build_cnn_model(input_shape):
    model = models.Sequential()

    # 第一个卷积层
    model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape))
    model.add(layers.BatchNormalization())

    # ResNet模块
    model.add(ResNetBlock(32))

    # 其他卷积层和池化层
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Flatten())

    # 全连接层
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.BatchNormalization())

    # 输出层
    model.add(layers.Dense(10, activation='softmax'))

    return model

# 加载MNIST数据集
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# 数据预处理
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

train_labels = tf.keras.utils.to_categorical(train_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)

# 构建模型
model = build_cnn_model((28, 28, 1))

# 定义一个回调类来获取每个 mini-batch 后的训练损失和测试损失
class LossHistory(tf.keras.callbacks.Callback):
    def on_train_begin(self, logs=None):
        self.train_losses = []
        self.test_losses = []

    def on_batch_end(self, batch, logs=None):
        self.train_losses.append(logs.get('loss'))
        test_loss = self.model.evaluate(test_images, test_labels, verbose=0)[0]
        self.test_losses.append(test_loss)
```

```
    def on_epoch_end(self, epoch, logs=None):
        test_loss = self.model.evaluate(test_images, test_labels, verbose=0)[0]
        self.test_losses.append(test_loss)
        print(f'\nEpoch {epoch + 1}, Test Loss: {test_loss}')

# 编译模型
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])

# 实例化 LossHistory 回调类
loss_history = LossHistory()

# 训练模型，并将 LossHistory 作为回调传递
history = model.fit(train_images, train_labels, epochs=1, batch_size=64,
validation_data=(test_images, test_labels), callbacks=[loss_history])
```

```
model.summary()

# 输出最终测试损失和准确率
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'\nFinal Test Loss: {test_loss}, Test Accuracy: {test_acc}')

# 输出最终训练损失和准确率
train_loss, train_acc = model.evaluate(train_images, train_labels)
print(f'Final Training Loss: {train_loss}, Training Accuracy: {train_acc}')

# 计算测试集中每一类的准确率
predictions = model.predict(test_images)
predicted_labels = tf.argmax(predictions, axis=1)
true_labels = tf.argmax(test_labels, axis=1)

confusion_matrix = tf.math.confusion_matrix(true_labels, predicted_labels,
num_classes=10)
class_accuracy = tf.linalg.diag_part(confusion_matrix) /
tf.reduce_sum(confusion_matrix, axis=1)

class_accuracy_dict = {f'Class {i}': [acc.numpy()] for i, acc in
enumerate(class_accuracy)}

df = pd.DataFrame(class_accuracy_dict)

print("Class Accuracy:")
print(df)

df.to_csv('class_accuracy_3.csv', index=False)

plt.figure(figsize=(10, 6))
```

```
bars = plt.bar(df.columns, df.iloc[0])
plt.xlabel('Class')
plt.ylabel('Accuracy')
plt.title('Class-wise Accuracy with 3x3 Convolutional Kernel')
plt.xticks(rotation=45)


for bar, label in zip(bars, df.iloc[0]):
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.15, bar.get_height() + 0.01,
f'{label:.2%}', ha='center', va='bottom')


plt.savefig('class_accuracy_bar_chart_3.png')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 batch_normalization (Batch  (None, 26, 26, 32)        128
 Normalization)

 res_net_block (ResNetBlock  (None, 26, 26, 32)        18752
 )

 conv2d_3 (Conv2D)           (None, 24, 24, 64)        18496

 batch_normalization_3 (Bat  (None, 24, 24, 64)        256
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 batch_normalization_4 (Bat  (None, 128)               512
 chNormalization)

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1219530 (4.65 MB)
Trainable params: 1218954 (4.65 MB)
Non-trainable params: 576 (2.25 KB)
_________________________________________________________________
313/313 [==============================] - 2s 6ms/step - loss: 0.0645 - accuracy:
0.9806
```

```
Final Test Loss: 0.0645499974489212, Test Accuracy: 0.9805999994277954
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0552 -
accuracy: 0.9831
Final Training Loss: 0.05521545931696892, Training Accuracy: 0.9830999970436096
313/313 [==============================] - 2s 7ms/step
Class Accuracy:
    Class 0   Class 1   Class 2  Class 3   Class 4   Class 5   Class 6  \
0  0.990816  0.996476  0.984496  0.99901  0.991853  0.965247  0.983299

    Class 7   Class 8   Class 9
0  0.996109  0.934292  0.959366
```

```python
plt.figure(figsize=(12, 6))


plt.subplot(1, 2, 1)
plt.plot(loss_history.train_losses, label='Training Loss')
plt.xlabel('Mini-Batch / Epoch')
plt.ylabel('Loss')
plt.legend()

for i, value in enumerate(loss_history.train_losses):
    if i < 600 and i % 50 == 0:
        plt.text(i, value, f'{value:.4f}', ha='center', va='bottom', fontsize=8)
    elif i >= 600 and (i - 600) % 200 == 0:
        plt.text(i, value, f'{value:.4f}', ha='center', va='bottom', fontsize=8)


plt.subplot(1, 2, 2)
plt.plot(loss_history.test_losses, label='Test Loss', color='orange')
plt.xlabel('Mini-Batch / Epoch')
plt.ylabel('Loss')
plt.legend()


for i, value in enumerate(loss_history.test_losses):
    if i < 600 and i % 50 == 0:
        plt.text(i, value, f'{value:.4f}', ha='center', va='bottom', fontsize=8)
    elif i >= 600 and (i - 600) % 200 == 0:
        plt.text(i, value, f'{value:.4f}', ha='center', va='bottom', fontsize=8)


plt.suptitle('Loss Curves with 3x3 Convolutional Kernel', fontsize=16, y=1)


plt.savefig('loss_curves_3.png')
```

5 * 5 卷积核

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import pandas as pd

# 定义ResNet模块
class ResNetBlock(layers.Layer):
    def __init__(self, filters, kernel_size=5, stride=1):
        super(ResNetBlock, self).__init__()
        self.conv1 = layers.Conv2D(filters, kernel_size=kernel_size,
strides=stride, padding='same')
        self.batch_norm1 = layers.BatchNormalization()
        self.relu1 = layers.Activation('relu')

        self.conv2 = layers.Conv2D(filters, kernel_size=kernel_size,
strides=stride, padding='same')
        self.batch_norm2 = layers.BatchNormalization()

        # 1x1 convolutional layer for the residual connection
        if stride > 1:
            self.residual_conv = layers.Conv2D(filters, kernel_size=1,
strides=stride, padding='same')
        else:
            self.residual_conv = None

        self.relu2 = layers.Activation('relu')

    def call(self, inputs, training=False):
        x = self.conv1(inputs)
        x = self.batch_norm1(x, training=training)
        x = self.relu1(x)

        x = self.conv2(x)
        x = self.batch_norm2(x, training=training)

        # Apply residual connection if needed
        if self.residual_conv is not None:
            inputs = self.residual_conv(inputs)

        x = layers.add([inputs, x])
        x = self.relu2(x)
        return x

# 构建卷积神经网络
def build_cnn_model(input_shape):
    model = models.Sequential()

    # 第一个卷积层
    model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape))
```

```
    model.add(layers.BatchNormalization())

    # ResNet模块
    model.add(ResNetBlock(32))

    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Flatten())

    # 全连接层
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.BatchNormalization())

    # 输出层
    model.add(layers.Dense(10, activation='softmax'))

    return model

# 加载MNIST数据集
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# 数据预处理
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

train_labels = tf.keras.utils.to_categorical(train_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)

# 构建模型
model = build_cnn_model((28, 28, 1))

# 定义一个回调类来获取每个 mini-batch 后的训练损失和测试损失
class LossHistory(tf.keras.callbacks.Callback):
    def on_train_begin(self, logs=None):
        self.train_losses = []
        self.test_losses = []

    def on_batch_end(self, batch, logs=None):
        self.train_losses.append(logs.get('loss'))
        test_loss = self.model.evaluate(test_images, test_labels, verbose=0)[0]
        self.test_losses.append(test_loss)

    def on_epoch_end(self, epoch, logs=None):
        test_loss = self.model.evaluate(test_images, test_labels, verbose=0)[0]
        self.test_losses.append(test_loss)
        print(f'\nEpoch {epoch + 1}, Test Loss: {test_loss}')

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

```
# 实例化 LossHistory 回调类
loss_history = LossHistory()

# 训练模型，并将 LossHistory 作为回调传递
history = model.fit(train_images, train_labels, epochs=1, batch_size=64,
validation_data=(test_images, test_labels), callbacks=[loss_history])
```

```
WARNING:tensorflow:From C:\Users\Gavin\anaconda3\Lib\site-
packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy
is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy
instead.

WARNING:tensorflow:From C:\Users\Gavin\anaconda3\Lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Gavin\anaconda3\Lib\site-
packages\keras\src\layers\normalization\batch_normalization.py:979: The name
tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm
instead.

WARNING:tensorflow:From C:\Users\Gavin\anaconda3\Lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Users\Gavin\anaconda3\Lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is
deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Gavin\anaconda3\Lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

  6/938 [..............................] - ETA: 31:34 - loss: 2.0246 - accuracy:
0.5558WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to
the batch time (batch time: 0.0449s vs `on_train_batch_end` time: 2.0407s). Check
your callbacks.
938/938 [==============================] - ETA: 0s - loss: 0.0352 - accuracy:
0.9879
Epoch 1, Test Loss: 0.03254542872309685
938/938 [==============================] - 1972s 2s/step - loss: 0.0325 -
accuracy: 0.9889 - val_loss: 0.0325 - val_accuracy: 0.9889
```

```
%matplotlib auto
```

```
Using matplotlib backend: <object object at 0x00000242CC8FF930>
```

```
model.summary()

# 输出最终测试损失和准确率
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'\nFinal Test Loss: {test_loss}, Test Accuracy: {test_acc}')

# 输出最终训练损失和准确率
train_loss, train_acc = model.evaluate(train_images, train_labels)
print(f'Final Training Loss: {train_loss}, Training Accuracy: {train_acc}')

# 计算测试集中每一类的准确率
predictions = model.predict(test_images)
predicted_labels = tf.argmax(predictions, axis=1)
true_labels = tf.argmax(test_labels, axis=1)

confusion_matrix = tf.math.confusion_matrix(true_labels, predicted_labels,
num_classes=10)
class_accuracy = tf.linalg.diag_part(confusion_matrix) /
tf.reduce_sum(confusion_matrix, axis=1)

class_accuracy_dict = {f'Class {i}': [acc.numpy()] for i, acc in
enumerate(class_accuracy)}

df = pd.DataFrame(class_accuracy_dict)


print("Class Accuracy:")
print(df)

df.to_csv('class_accuracy_5.csv', index=False)

plt.figure(figsize=(10, 6))

bars = plt.bar(df.columns, df.iloc[0])
plt.xlabel('Class')
plt.ylabel('Accuracy')
plt.title('Class-wise Accuracy with 5x5 Convolutional Kernel')
plt.xticks(rotation=45)


for bar, label in zip(bars, df.iloc[0]):
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.15, bar.get_height() + 0.01,
```

```
f'{label:.2%}', ha='center', va='bottom')


plt.savefig('class_accuracy_bar_chart_5.png')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 batch_normalization (Batch  (None, 26, 26, 32)        128
 Normalization)

 res_net_block (ResNetBlock  (None, 26, 26, 32)        51520
 )

 conv2d_3 (Conv2D)           (None, 24, 24, 64)        18496

 batch_normalization_3 (Bat  (None, 24, 24, 64)        256
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 batch_normalization_4 (Bat  (None, 128)               512
 chNormalization)

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1252298 (4.78 MB)
Trainable params: 1251722 (4.77 MB)
Non-trainable params: 576 (2.25 KB)
_________________________________________________________________
313/313 [==============================] - 2s 7ms/step - loss: 0.0325 - accuracy:
0.9889

Final Test Loss: 0.03254542872309685, Test Accuracy: 0.9889000058174133
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0274 -
accuracy: 0.9913
Final Training Loss: 0.027373842895030975, Training Accuracy: 0.9913333058357239
313/313 [==============================] - 2s 7ms/step
Class Accuracy:
    Class 0   Class 1   Class 2  Class 3  Class 4   Class 5   Class 6  \
```

```
0  0.995918  0.989427  0.994186  0.99703  0.99389  0.986547  0.990605

    Class 7   Class 8   Class 9
0  0.978599  0.994867  0.968285
```

```
plt.figure(figsize=(12, 6))


plt.subplot(1, 2, 1)
plt.plot(loss_history.train_losses, label='Training Loss')
plt.xlabel('Mini-Batch / Epoch')
plt.ylabel('Loss')
plt.legend()


for i, value in enumerate(loss_history.train_losses):
    if i < 600 and i % 50 == 0:
        plt.text(i, value, f'{value:.4f}', ha='center', va='bottom', fontsize=8)
    elif i >= 600 and (i - 600) % 200 == 0:
        plt.text(i, value, f'{value:.4f}', ha='center', va='bottom', fontsize=8)


plt.subplot(1, 2, 2)
plt.plot(loss_history.test_losses, label='Test Loss', color='orange')
plt.xlabel('Mini-Batch / Epoch')
plt.ylabel('Loss')
plt.legend()

for i, value in enumerate(loss_history.test_losses):
    if i < 600 and i % 50 == 0:
        plt.text(i, value, f'{value:.4f}', ha='center', va='bottom', fontsize=8)
    elif i >= 600 and (i - 600) % 200 == 0:
        plt.text(i, value, f'{value:.4f}', ha='center', va='bottom', fontsize=8)


plt.suptitle('Loss Curves with 5x5 Convolutional Kernel', fontsize=16, y=1)


plt.savefig('loss_curves_5.png')
```