一、实验内容

1. 利用卷积神经网络模型进行MINST数据集的分析

二、实验设备

1. 实验设备：台式机/笔记本等不限
2. 平台：Visual C++ / Python等不限

三、实验步骤

实验原理：

图像在计算机中是一堆按顺序排列的数字，数值为0到255。0表示最暗，255表示最亮。

1.导入必要的库

```
import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist
```

① tensorflow：导入TensorFlow库，用于构建和训练深度学习模型。

② layers：从tensorflow.keras导入层，定义不同的神经网络层（如卷积层、池化层、全连接层等）。

③ models：从tensorflow.keras导入models，用于构建神经网络模型。

④ mnist：导入mnist数据集，这是一个常用于手写数字分类任务的数据集。

2. 加载MNIST数据集

```
# 加载 MNIST 数据集
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

① x_train：训练集图片，形状为(60000, 28, 28)，每张图片大小为28x28像素。

② y_train：训练集标签，形状为(60000,)，每个标签是一个0到9的整数，表示手写数字的类别。

③ x_test和y_test是测试集数据，形状分别为(10000, 28, 28)和(10000,)。

3. 数据预处理

```
# 数据预处理
x_train = x_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0
```

① x_train.reshape(-1, 28, 28, 1)：将x_train从(60000, 28, 28)重塑为(60000, 28, 28, 1)，即每张图片为28x28像素，且有1个通道（灰度图像）。 （通常图片应该有三个通道，即红绿蓝）

② .astype("float32")：将数据类型转换为float32，以便进行神经网络训练。

③ / 255.0：将像素值归一化到[0, 1]范围，因为原始的MNIST图片像素值是 0到255之间的整数。(便于计算)

④ 同样地，对测试集数据x_test进行相同的预处理。

4. 使用one-hot编码

```
# 将标签转换为 one-hot 编码
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

使用tf.keras.utils.to_categorical将标签转换为one-hot编码。对于每个标签，生成一个长度为10的向量，其中对应数字的索引为1,其余为0。

5. 构建卷积神经网络模型

```python
# 构建卷积神经网络模型
model = models.Sequential([
    # 第1层卷积 + 池化
    layers.Conv2D(32, (5, 5), activation="relu", padding="same", input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2), padding="same"),

    # 第2层卷积 + 池化
    layers.Conv2D(64, (5, 5), activation="relu", padding="same"),
    layers.MaxPooling2D((2, 2), padding="same"),

    # 展平层
    layers.Flatten(),

    # 全连接层
    layers.Dense(1024, activation="relu"),
    layers.Dropout(0.5),

    # 输出层
    layers.Dense(10, activation="softmax")
])
```

① 第一层卷积层：

```python
    layers.Conv2D(32, (5, 5), activation="relu", padding="same", input_shape=(28, 28, 1)),
```

该层输出32个5x5的过滤器，使用ReLU激活函数，在输入图像的边缘进行填充，使得输出图像的尺寸与输入图像保持一致。

② 第一层池化层：

```python
layers.MaxPooling2D((2, 2), padding="same"),
```

该层的池化窗口大小为2x2，采用最大池化方式取窗口内最大值，边界填充并保持尺寸。

③ 第二层卷积层：

```python
layers.Conv2D(64, (5, 5), activation="relu", padding="same"),
```

与第一层卷积层类似，提取图像的局部特征（如边缘、角点等）

④ 第二层池化层：

```python
layers.MaxPooling2D((2, 2), padding="same"),
```

与第一层池化层类似，对特征图进行降采样，减小数据量，提高计算效率，同时保留最显著的特征。

⑤ Flatten层：

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

将二维图像进行平展，得到一组一维的向量，将卷积层和池化层提取的特

征映射到全连接层输入所需的格式。

⑥ 全连接层：

```
layers.Dense(1024, activation="relu"),
```

输出的神经元个数为1024个，使用Relu激活函数进行激活，将前面提取的特征进行学习，并用于分类任务。

⑦ Dropout层：

```
layers.Dropout(0.5),
```

正则化层，每次训练随机丢弃50%神经元，用于减少过拟合，增强模型的泛化能力。

⑧ 输出层：

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

一共有10个节点，分别代表数字0到9，使用softmax激活函数，将输出转换为概率分布，最终用于多分类任务。每个输出神经元的值表示该类别的概率。

6. 对模型进行编译

```
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

① 优化器：使用Adam优化器，学习率设为1e-4。Adam是一种自适应学习率的优化算法，常用于训练深度学习模型。

② 损失函数：使用categorical_crossentropy，因为这是一个多分类问题（one-hot 编码标签）。

③ 评价指标：使用accuracy来衡量模型的分类准确率。

7. 训练模型

```
history = model.fit(x_train, y_train, batch_size=50, epochs=10, validation_split=0.1,
```

① 训练数据：使用x_train和y_train作为训练数据和标签。

② batch_size=50：每次训练使用50张图片进行梯度更新。

③ epochs=10：训练10个周期（即10次完整的训练集迭代）。

④ validation_split=0.1：将10%的训练数据作为验证集，用于在训练过程中评估模型性能。

8. 测试模型

```
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
```

① 使用model.evaluate()在测试集上评估模型性能，返回损失和准确率。

② verbose=2控制输出的详细程度。

③ 打印出模型在测试集上的损失值和准确率。

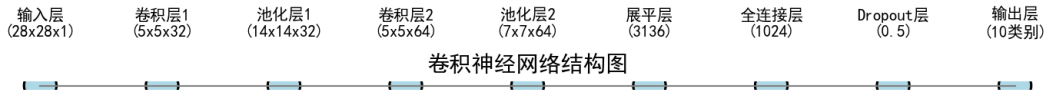
9. 结果可视化

通过添加TensorBoard支持，保存各项数据

```
log_dir = "D:\TrainData"  # 指定日志保存路径
os.makedirs(log_dir, exist_ok=True)
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

10. 扩展说明

对此卷积神经网络做扩展说明，其原理如下图所示：



四、分析说明（包括结果图表分析说明，主要核心代码及解释）

由运行结果截图可得：

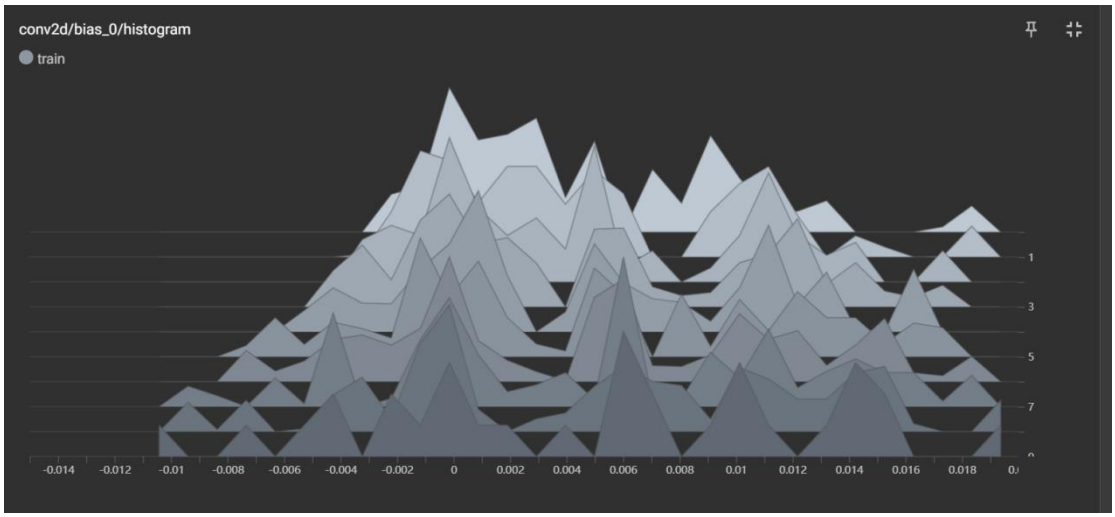


图1 卷积层第0层的偏置项(bias)分布随训练的变化情况

分析：

① 从分布初始可能较为集中在0附近，随着训练进行，偏置分布可能略有左右偏移或者分布范围变窄。
② 初始卷积层捕捉低层次特征（如边缘、线条），其偏置项在微调过程中可能不会发生巨大偏移，而是逐步微调到合适值。
结论：对于低层特征提取层来说，参数（尤其偏置）往往会较为稳定地收敛到特定范围。在训练多个轮次后，偏置的分布可能在一个较小的区域内趋于稳定

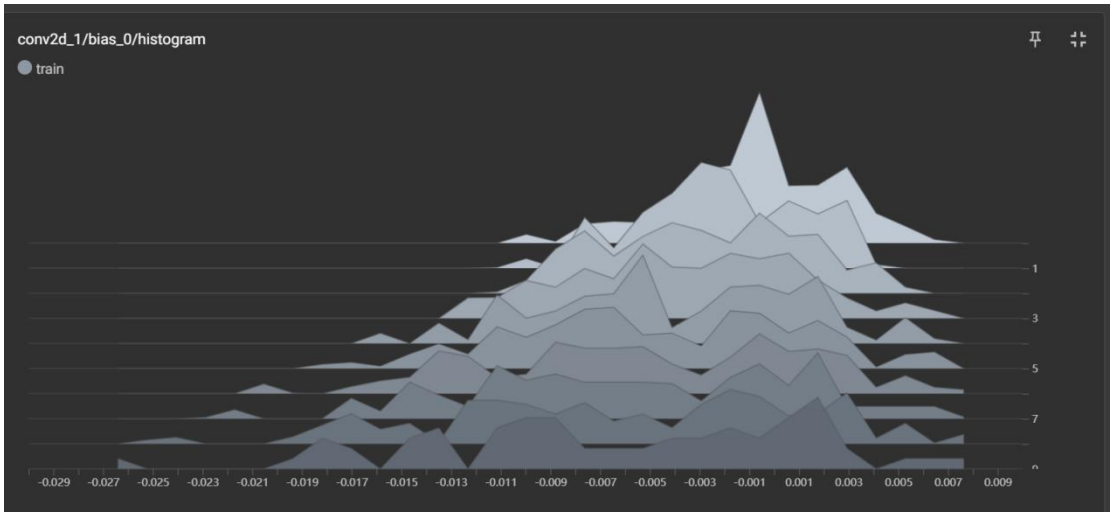


图2 卷积层第1层的偏置项(bias)分布随训练的变化情况

分析：
① 同样的，不过卷积层1的分布更分散、集中速度 更慢，说明这一调整过程比卷积层0更复杂，这一层学习到的特征更加细致或高级。

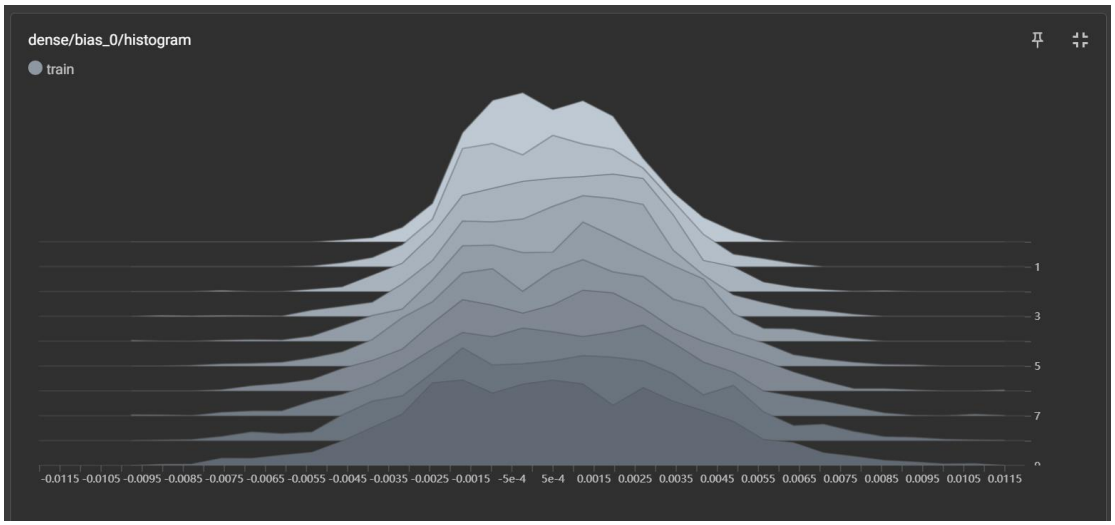


图3 全链接层dense1的偏置项(bias)分布随训练的变化情况

分析：
① 虽然总体变化不大，但还是能看出单峰的峰值在减小，偏置项的对称性和分布的平滑度说明训练稳定，没有出现参数震荡或不稳定现象。

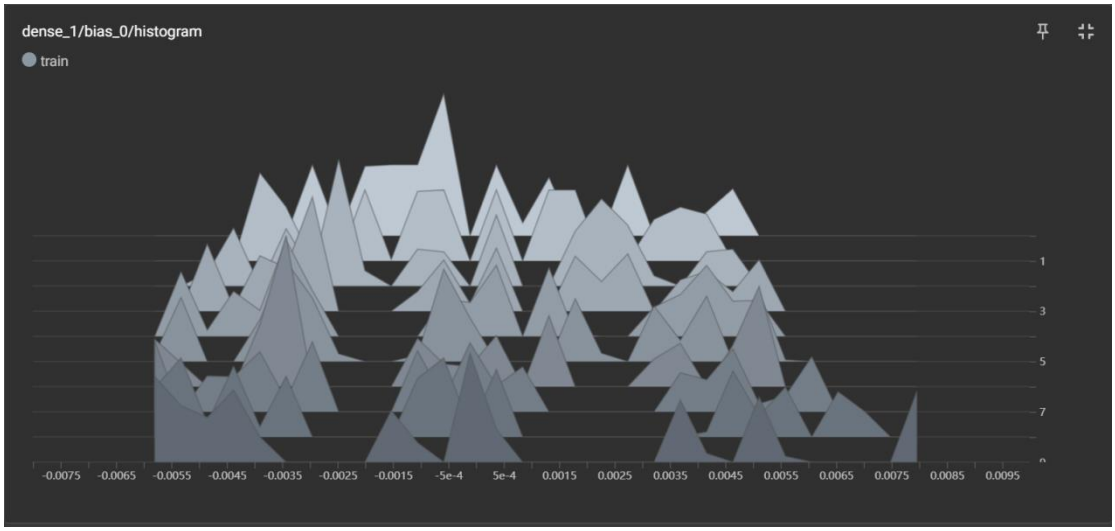


图4 全链接层dense2的偏置项(bias)分布随训练的变化情况

分析：

① 偏置项逐渐收敛，直至出现明显的分离

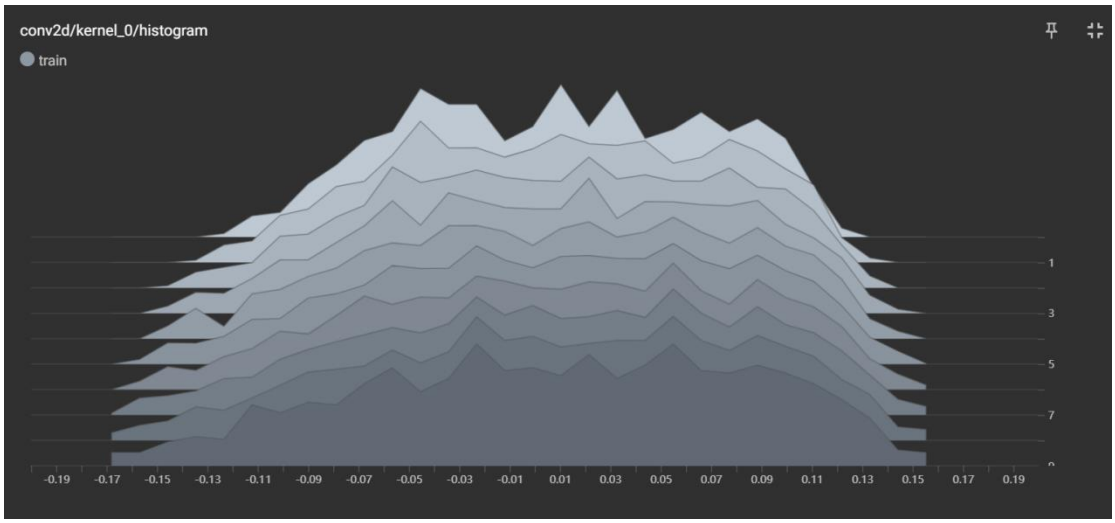


图5 卷积层第0层的卷积核权重(kernel)分布随训练的变化情况

结论：可以看到随着训练轮数的增加，由收敛到分散，最终看着比较均匀。

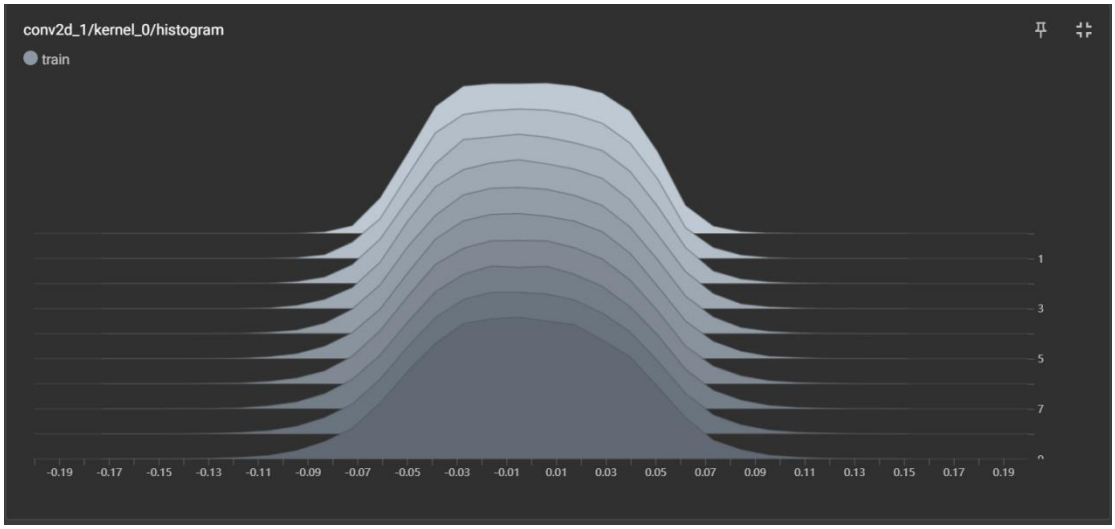


图6 卷积层第1层的卷积核权重(kernel)分布随训练的变化情况

结论：权重分布并未偏向某一侧，也没有异常的权重值。这说明模型训练正常，未出现梯度爆炸或消失的问题

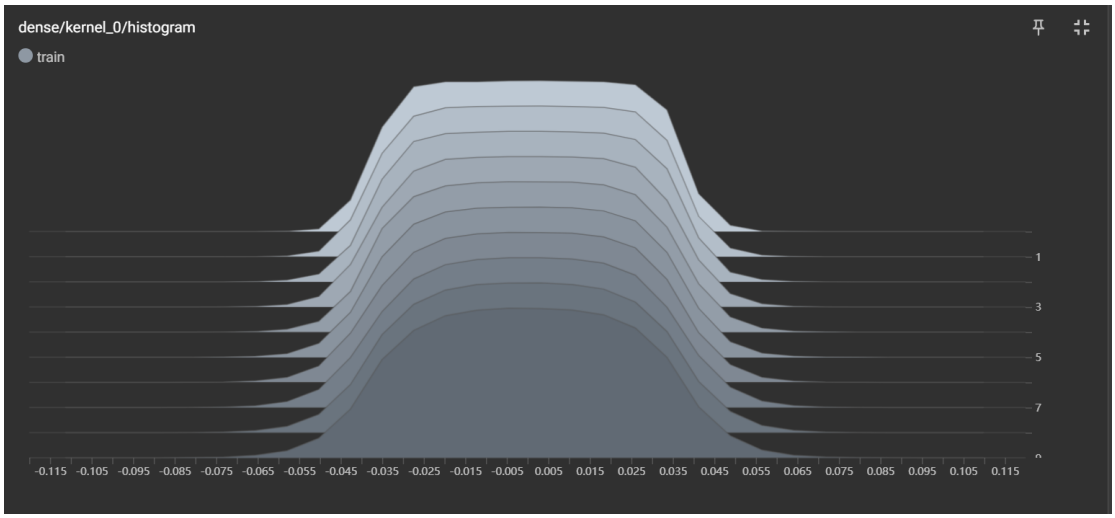


图7 全链接层dense1的卷积核权重(kernel)分布随训练的变化情况

分析：权重分布呈高斯形状，随着训练的进行逐渐收敛，表明网络训练合理且稳定。

结论：没有观察到异常权重或不合理的分布，表明模型训练没有出现不收敛、过拟合或其他异常现象。

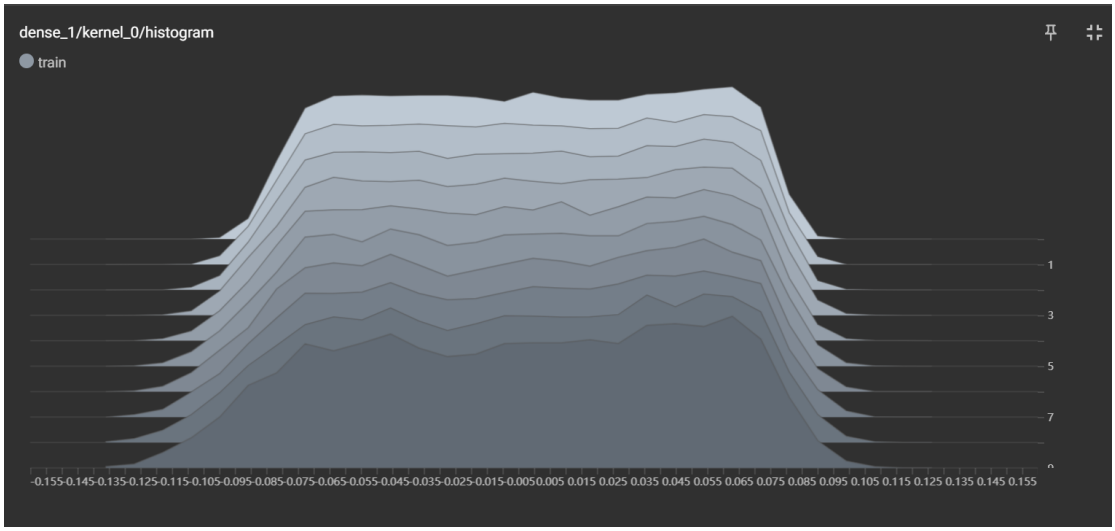


图8 全链接层dense2的卷积核权重(kernel)分布随训练的变化情况

分析：权重分布较为平缓，意味着权重值并未高度集中，可能有助于模型的表达能力。

结论：核权重分布在训练过程中保持对称性且逐渐收敛，说明训练过程稳定，模型优化正常。

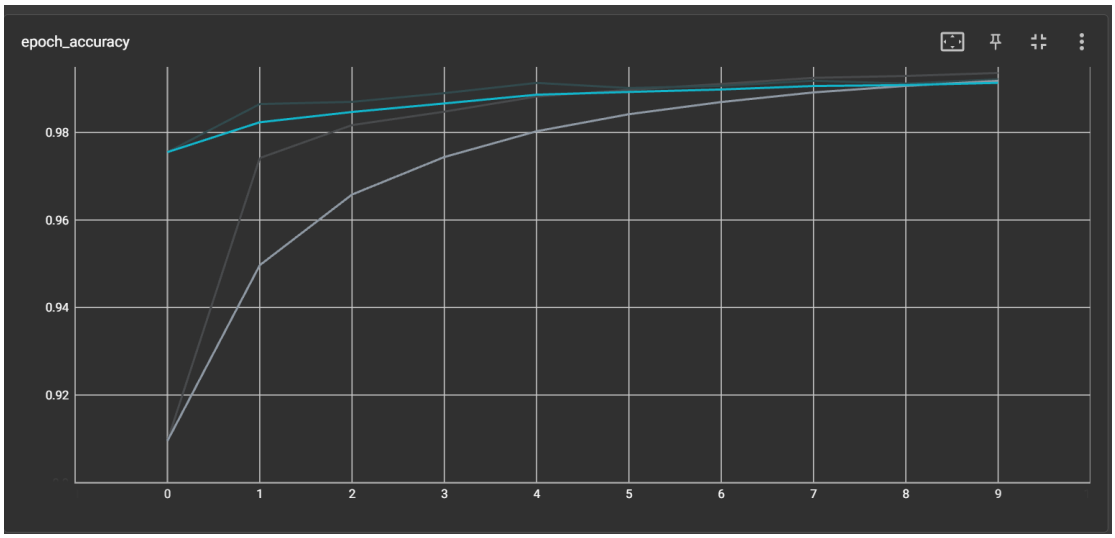


图9 模型的准确率随训练轮次的变化情况

亮蓝色代表平滑处理后的验证集准确率，亮灰色代表平滑处理后的训练集准确率
暗蓝色代表平滑处理前的验证集准确率，暗灰色代表平滑处理前的训练集准确率

分析：训练准确率和验证准确率随着训练轮次的增加而逐渐上升，并在后期趋于平稳，验证集准确率接近训练集准确率，表明模型泛化能力较强，没有明显

的过拟合现象。

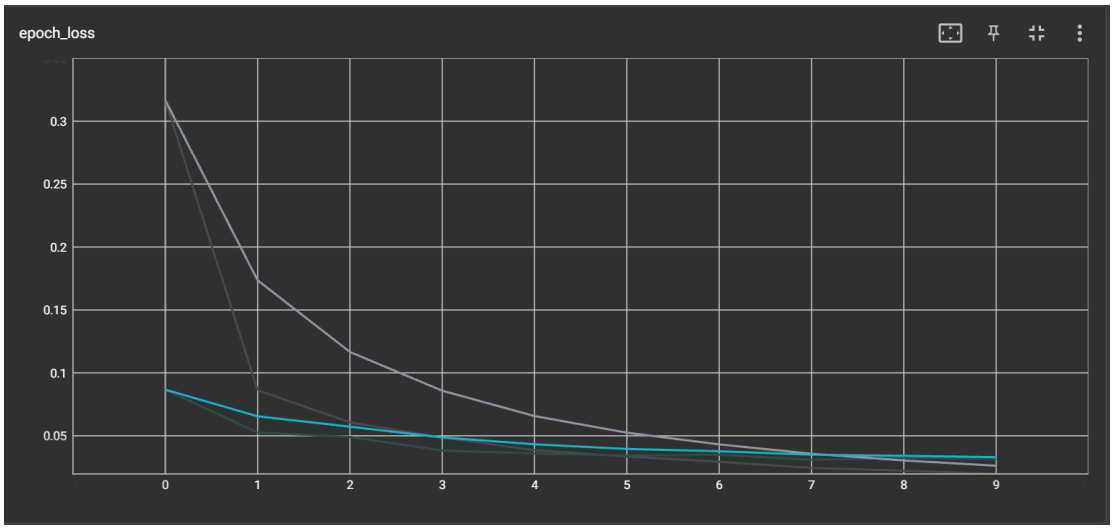


图10 模型的损失值随训练轮次的变化情况

亮蓝色代表平滑处理后的验证集损失值，亮灰色代表平滑处理后的训练集损失值
暗蓝色代表平滑处理前的验证集损失值，暗灰色代表平滑处理前的训练集损失值

分析：随着训练轮次的增加，无论是验证集还是训练集，loss值都在减小，且在后期趋于平稳，验证集损失值接近训练集损失值，表明模型泛化能力较强，没有明显的过拟合现象。

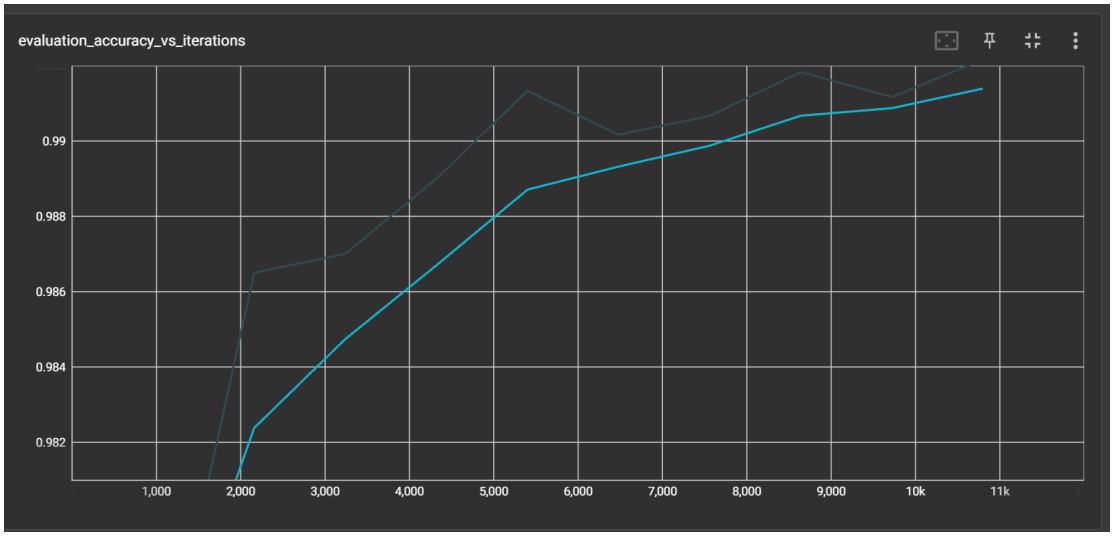


图11 模型的评估准确率随迭代次数的变化情况

（暗蓝色代表平滑处理前，亮蓝色代表平滑处理后）

分析：随着迭代次数的增加，两条线准确率都逐渐上升，最终接近稳定状态。

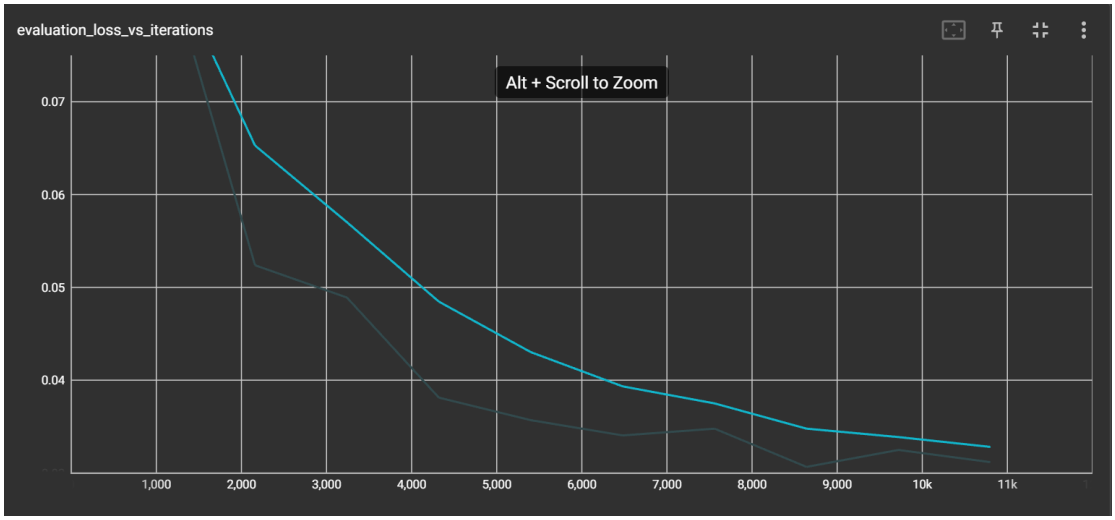


图12 模型的评估损失率随迭代次数的变化情况

（暗蓝色代表平滑处理前，亮蓝色代表平滑处理后）

分析：随着迭代次数的增加，两条线准确率都逐渐下降，最终接近稳定状态。

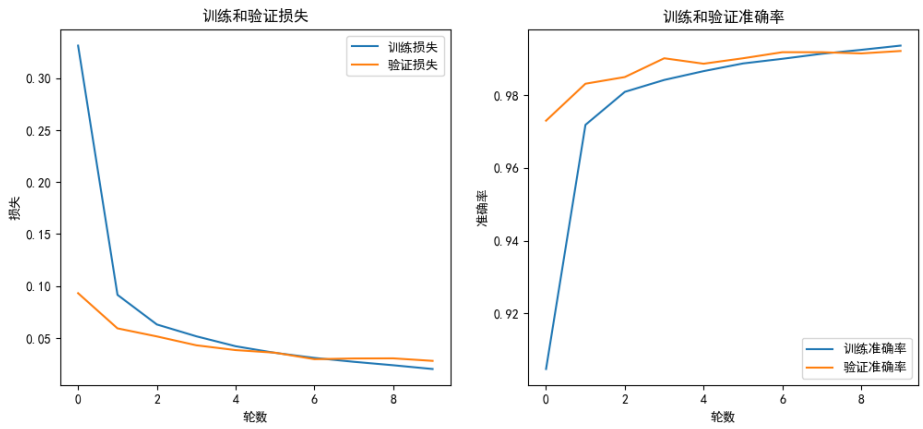
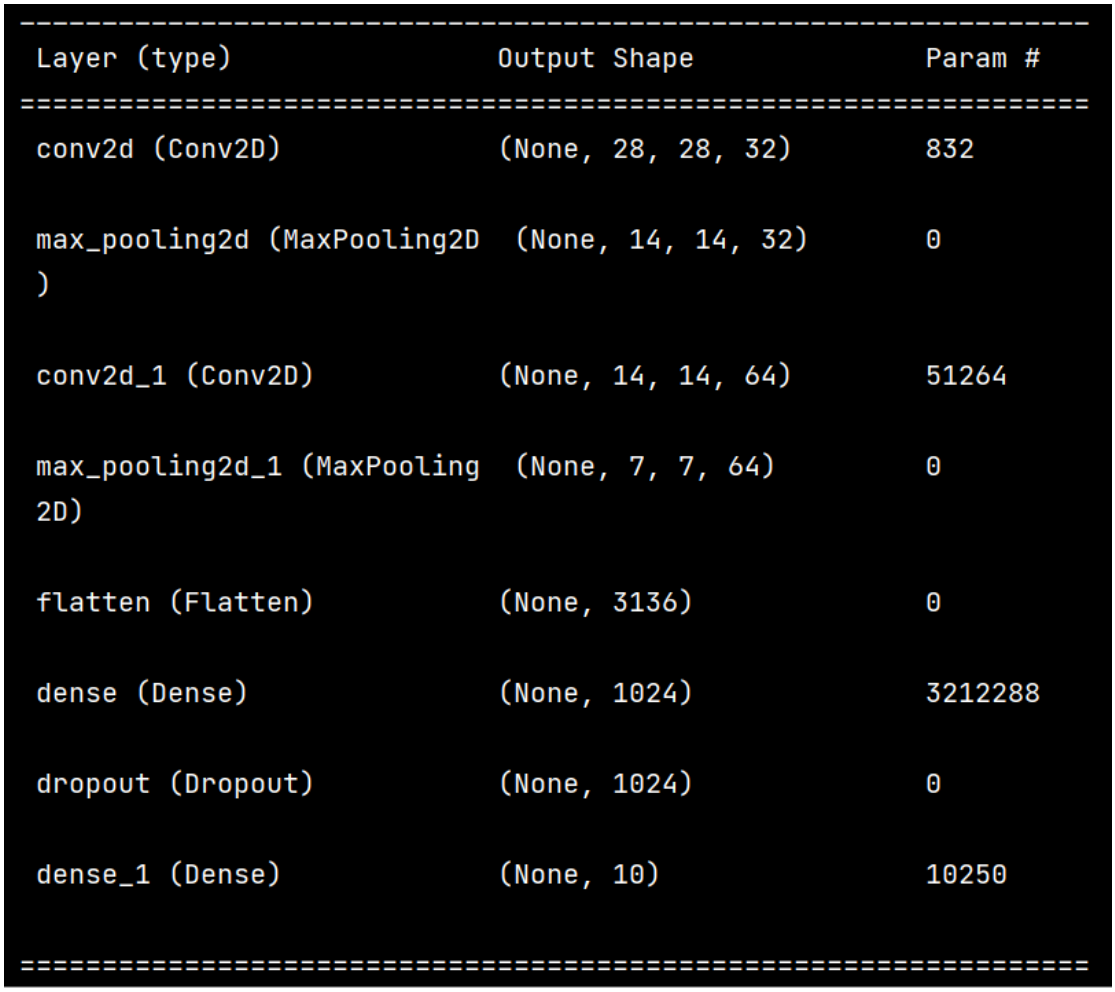
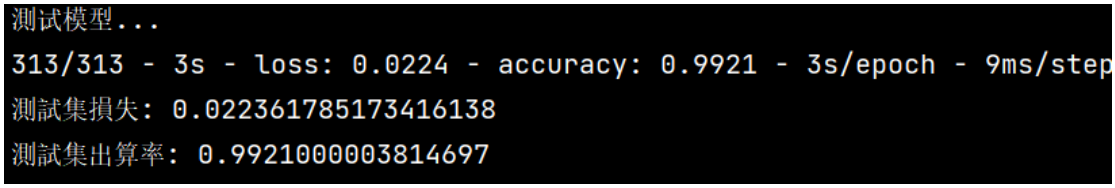


图13 模型的评估损失/准确率随迭代次数的变化情况



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	51264
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 1024)	3212288
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10250

图14 每层的数据类型及其数量



```
测试模型...
313/313 - 3s - loss: 0.0224 - accuracy: 0.9921 - 3s/epoch - 9ms/step
測試集損失: 0.022361785173416138
測試集出算率: 0.9921000003814697
```

图15 测试结果

分析:

① 收敛性：模型的训练和验证损失逐渐下降，准确率逐渐上升，说明模型在训练过程中表现良好，逐步收敛。
② 过拟合现象：验证损失和验证准确率与训练结果接近，表明过拟合问题较小。
③ 训练效果：经过几轮训练后，准确率接近100%，损失接近0，表明模型已训练充分。

五、总结心得

本次实验通过卷积神经网络（CNN）对MNIST手写数字数据集进行了训练与分析。实验重点探讨了不同网络层次结构（卷积层、池化层）及参数设置（激活函数、步长、池化方式等）对模型性能的影响，利用TensorFlow实现了训练过程的可视化分析。

训练过程中，卷积层与全连接层的偏置项与核权重分布逐渐收敛，表现出良好的稳定性。模型参数逐步优化，未出现异常情况，如梯度爆炸或消失。此外，池化与激活函数设置得合理才能有效提取图像特征，提升模型性能。

实验结果显示，模型的准确率与损失值在训练集和验证集上均表现良好，验证准确率接近训练准确率，验证损失值与训练损失值趋于一致，表明模型具有较强的泛化能力，未出现明显的过拟合现象。

通过本次实验，我掌握了CNN模型的设计、参数调优与训练可视化方法，深刻理解了网络各层参数的收敛规律及其在特征提取中的作用。比较不足的是还存在许多待优化问题：如没有与LetNet-5等一些经典的神经网络进行比较，对于激活函数、步长、池化方式、填充方式的搭配还有需要改进的地方，对于如何找出最优的池化参数对于“是否设置的卷积和池化层越多，得到的效果越好呢”这个问题的具体解答还不清楚，只知道dropout层是用来防止过拟合的，之前打数模比赛的时候就粗略地学习过一些模拟退火、随机森林等智能算法，但这次实验学习到的东西很丰富充实，也激发了我对人工智能领域的向往，相信在不断的学习之下，我的理论和实践水平能得到很大的提升。

附录（所有代码）

```
import matplotlib

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

import os



# 添加中文支持

matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体显示中文

matplotlib.rcParams['axes.unicode_minus'] = False    # 正确显示负号
```

```
# 1. 加载 MNIST 数据集
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 2. 数据预处理
x_train = x_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0

# 将标签转换为 one-hot 编码
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# 3. 构建卷积神经网络模型
model = models.Sequential([
    layers.Conv2D(32, (5, 5), activation="relu", padding="same", input_shape=(28,
28, 1)),
    layers.MaxPooling2D((2, 2), padding="same"),

    layers.Conv2D(64, (5, 5), activation="relu", padding="same"),
    layers.MaxPooling2D((2, 2), padding="same"),

    layers.Flatten(),
    layers.Dense(1024, activation="relu"),
```

```
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax")
])


# 打印模型结构
print("\u6a21\u578b\u7ed3\u6784：")
model.summary()

# 4. 编译模型
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
              loss="categorical_crossentropy",
              metrics=["accuracy"])

# 5. 添加 TensorBoard 支持
log_dir = "D:\TrainData"  # 指定日志保存路径
os.makedirs(log_dir, exist_ok=True)
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)

# 6. 训练模型并保存训练过程历史
print("\n\u5f00\u59cb\u8bad\u7ec3\u6a21型...")
```

```
history = model.fit(x_train, y_train, batch_size=50, epochs=10, validation_split=0.1,
                                callbacks=[tensorboard_callback])



# 7. 测试模型

print("\n\u6e2c试\u6a21型...")

test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)

print(f"\u6e2c\u8a66\u96c6\u640d\u5931: {test_loss}")

print(f"\u6e2c\u8a66\u96c6\u51fa\u7b97\u7387: {test_accuracy}")




# 8. 视化训练结果

def plot_training_history(history):

    plt.figure(figsize=(12, 5))



    # 損失图

   plt.subplot(1, 2, 1)

    plt.plot(history.history['loss'], label='训练损失')

    plt.plot(history.history['val_loss'], label='验证损失')

    plt.title('训练和验证损失')

    plt.xlabel('轮数')

    plt.ylabel('损失')

    plt.legend()
```

```
    # 出算率图
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='训练准确率')
    plt.plot(history.history['val_accuracy'], label='验证准确率')
    plt.title('训练和验证准确率')
    plt.xlabel('轮数')
    plt.ylabel('准确率')
    plt.legend()

    plt.show()



plot_training_history(history)


# 9. 运行 TensorBoard 查看训练过程
print("\n运行 TensorBoard 查看结果：\n")
print("tensorboard --D:\TrainData")
```