一、实验目的

- 了解卷积神经网络的基本原理和结构
- 掌握使用TensorFlow搭建和训练卷积神经网络的方法
- 使用卷积神经网络对手写数字图像进行分类
- 分析不同激活函数、dropout技巧、数据量和超参数对模型性能的影响

二、实验内容

- 使用MNIST数据集作为训练和测试数据，该数据集包含60000个训练样本和10000个测试样本，每个样本都是一张28x28像素的灰度手写数字图片
- 使用TensorFlow构建一个卷积神经网络模型，参考Lenet-5的结构，包含两个卷积层和三个全连接层
- 使用交叉熵损失函数和Adam优化器进行模型训练，使用准确率作为评估指标
- 尝试不同的激活函数（sigmoid、tanh、ReLU）和dropout率（0.1~0.5），观察它们对模型性能的影响
- 尝试不同的数据量（10000~60000）、批量大小（32~256）、学习率（0.001~0.1）和训练周期数（5~20），观察它们对模型性能的影响
- 绘制混淆矩阵，分析模型的表现和误差

三、实验环境

- 操作系统：Windows 10
- 编程语言：Python 3.8
- 深度学习框架：TensorFlow 2.4
- 开发工具：Jupyter Notebook

四、实验过程、结果及分析

4.1 原理

Lenet-5是一个由两个卷积层和三个全连接层组成的卷积神经网络模型，它是由Yann LeCun等人于1998年提出的，用于对手写数字图像进行分类 。

Lenet-5的输入是一个32x32像素的灰度图像，经过以下几个步骤：

- 第一个卷积层（C1）：使用6个5x5的卷积核对输入图像进行卷积，得到6个28x28的特征图。卷积操作可以用以下公式表示：

$$C_{i,j}^k = \sum_{m=0}^{4} \sum_{n=0}^{4} W_{m,n}^k X_{i+m,j+n} + b^k$$

其中，$C_{i,j}^k$表示第k个特征图的第(i,j)个像素，$W_{m,n}^k$表示第k个卷积核的第(m,n)个权重，$X_{i+m,j+n}$表示输入图像的第$(i+m,j+n)$个像素，b^k表示第k个偏置项。

- 第一个池化层（S2）：使用2x2的最大池化对每个特征图进行降采样，得到6个14x14的特征图。池化操作可以用以下公式表示：

$$S_{i,j}^k = \max\{C_{2i,2j}^k, C_{2i,2j+1}^k, C_{2i+1,2j}^k, C_{2i+1,2j+1}^k\}$$

其中，$S_{i,j}^k$表示第k个特征图的第(i,j)个像素，$C_{2i,2j}^k$表示第k个特征图的第$(2i,2j)$个像素。

- **第二个卷积层（C3）：使用16组不同的卷积核对每个特征图进行卷积，得到16个10x10的特征图。每组卷积核包含3或4个5x5的卷积核，分别与不同的特征图相连。这样做的目的是为了增加模型的非线性和稀疏性。卷积操作可以用以下公式表示：**

$$C_{i,j}^{k'} = \sum_{k \in G(k')} \sum_{m=0}^{4} \sum_{n=0}^{4} W_{m,n}^{k,k'} S_{i+m,j+n}^{k} + b^{k'}$$

其中，$C_{i,j}^{k'}$表示第k'个特征图的第(i,j)个像素，$W_{m,n}^{k,k'}$表示连接第k个和第k'个特征图的卷积核的第(m,n)个权重，$S_{i+m,j+n}^{k}$表示第k个特征图的第$(i+m,j+n)$个像素，$b^{k'}$表示第k'个偏置项，$G(k')$表示与第k'个特征图相连的特征图编号集合。

- **第二个池化层（S4）：使用2x2的最大池化对每个特征图进行降采样，得到16个5x5的特征图。池化操作可以用以下公式表示：**

$$S_{i,j}^{k'} = \max\{C_{2i,2j}^{k'}, C_{2i,2j+1}^{k'}, C_{2i+1,2j}^{k'}, C_{2i+1,2j+1}^{k'}\}$$

其中，$S_{i,j}^{k'}$表示第k'个特征图的第(i,j)个像素，$C_{2i,2j}^{k'}$表示第k'个特征图的第$(2i,2j)$个像素。

- **第一个全连接层（C5）：将池化层的输出展平为一个400维的向量，然后与一个有120个神经元的全连接层相连，得到一个120维的向量。全连接操作可以用以下公式表示：**

$$F_i = \sum_{j=0}^{399} V_{i,j} S_j + c_i$$

其中，F_i表示第i个神经元的输出，$V_{i,j}$表示第i个和第j个神经元之间的权重，S_j表示第j个输入神经元的值，c_i表示第i个偏置项。

- **第二个全连接层（F6）：将全连接层的输出与一个有84个神经元的全连接层相连，得到一个84维的向量。全连接操作可以用以下公式表示：**

$$F_i' = \sum_{j=0}^{119} W_{i,j} F_j + d_i$$

其中，F_i'表示第i个神经元的输出，$W_{i,j}$表示第i个和第j个神经元之间的权重，F_j表示第j个输入神经元的值，d_i表示第i个偏置项。

- **第三个全连接层（Output）：将全连接层的输出与一个有10个神经元的全连接层相连，得到一个10维的向量。然后使用softmax函数将其转换为分类概率。全连接操作可以用以下公式表示：**

$$O_i = \sum_{j=0}^{83} U_{i,j} F_j' + e_i$$

其中，O_i表示第i个神经元的输出，$U_{i,j}$表示第i个和第j个神经元之间的权重，F_j'表示第j个输入神经元的值，e_i表示第i个偏置项。

softmax函数可以用以下公式表示：

$$P_i = \frac{\exp(O_i)}{\sum_{k=0}^{9} \exp(O_k)}$$

其中，P_i表示第i类的概率。

4.2 数据加载和预处理

首先，我们使用`tensorflow.keras.datasets.mnist.load_data()`函数加载MNIST数据集，并将图像数据转换为浮点数，并归一化到[0,1]范围。为了适应Lenet-5的输入要求，我们将图像数据从28x28扩展为32x32，并增加一个通道维度。我们还将标签数据转换为one-hot编码，方便后续计算交叉熵损失。

```
import tensorflow as tf
from tensorflow import keras
from keras import layers, models, datasets
```

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

# 将图像数据转换为浮点数，并归一化到[0,1]范围
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# 为了适应Lenet-5的输入要求，将图像数据从28x28扩展为32x32，并增加一个通道维度
x_train = tf.pad(x_train, [[0,0],[2,2],[2,2]])
x_test = tf.pad(x_test, [[0,0],[2,2],[2,2]])
x_train = tf.expand_dims(x_train, axis=-1)
x_test = tf.expand_dims(x_test, axis=-1)

# 将标签数据转换为one-hot编码
y_train = tf.one_hot(y_train, depth=10)
y_test = tf.one_hot(y_test, depth=10)
```

4.3 模型构建和编译

接下来，我们使用`tensorflow.keras.models.Sequential()`函数构建一个卷积神经网络模型，参考Lenet-5的结构，包含两个卷积层和三个全连接层。我们使用ReLU作为激活函数，并在全连接层后添加dropout层，以防止过拟合。

```
# 定义Lenet-5模型
model = models.Sequential([
    # 第一个卷积层，使用6个5x5的卷积核，激活函数为relu
    layers.Conv2D(6, kernel_size=5, activation='relu', input_shape=(32, 32, 1)),
    # 第一个池化层，使用2x2的最大池化
    layers.MaxPool2D(pool_size=2),
    # 第二个卷积层，使用16个5x5的卷积核，激活函数为relu
    layers.Conv2D(16, kernel_size=5, activation='relu'),
    # 第二个池化层，使用2x2的最大池化
    layers.MaxPool2D(pool_size=2),
    # 将卷积层的输出展平为一维向量
    layers.Flatten(),
    # 第一个全连接层，有120个神经元，激活函数为relu
    layers.Dense(120, activation='relu'),
    # 在全连接层后添加dropout层，丢弃率为0.5
    layers.Dropout(0.5),
    # 第二个全连接层，有84个神经元，激活函数为relu
    layers.Dense(84, activation='relu'),
    # 在全连接层后添加dropout层，丢弃率为0.5
    layers.Dropout(0.5),
    # 第三个全连接层，有10个神经元，激活函数为softmax，输出分类概率
    layers.Dense(10, activation='softmax')
])
```

然后，我们使用`model.compile()`函数编译模型，使用交叉熵损失函数和Adam优化器，评估指标为准确率。

```
# 编译模型，使用交叉熵损失函数和Adam优化器，评估指标为准确率
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
['accuracy'])
```

4.4 模型训练和评估

接下来，我们使用`model.fit()`函数训练模型，使用128的批量大小和10个周期数，每个周期结束后在测试集上评估模型性能，并记录训练过程中的损失和准确率。我们还使用`model.predict()`函数对测试集进行预测，并计算混淆矩阵，分析模型的表现和性能。

我们训练和评估0.5的dropout率的卷积神经网络模型，激活函数分别为sigmoid，tanh和ReLU。

```
Test loss: 2.3025, Test accuracy: 0.1135
313/313 [==============================] - 1s 2ms/step
Confusion matrix:
 [[   0  980    0    0    0    0    0    0    0    0]
 [   0 1135    0    0    0    0    0    0    0    0]
 [   0 1032    0    0    0    0    0    0    0    0]
 [   0 1010    0    0    0    0    0    0    0    0]
 [   0  982    0    0    0    0    0    0    0    0]
 [   0  892    0    0    0    0    0    0    0    0]
 [   0  958    0    0    0    0    0    0    0    0]
 [   0 1028    0    0    0    0    0    0    0    0]
 [   0  974    0    0    0    0    0    0    0    0]
 [   0 1009    0    0    0    0    0    0    0    0]]
```

sigmoid激活函数

```
Test loss: 0.1107, Test accuracy: 0.9697
313/313 [==============================] - 1s 2ms/step
Confusion matrix:
 [[ 972    0    2    0    0    0    5    1    0    0]
 [   0 1116    6    2    0    1    7    0    3    0]
 [   9    2 1001    3    2    0    0   10    5    0]
 [   0    0    8  969    0   17    0    5    7    4]
 [   0    0    0    0  961    0   14    0    0    7]
 [   7    0    1    2    0  872    5    2    3    0]
 [   8    0    2    0    1    1  945    0    1    0]
 [   1    5   19    3    1    0    0  952    2   45]
 [  10    0    4    1    2    0    7    1  946    3]
 [   6    2    0    2   13    3    2    2   16  963]]
```

tanh激活函数

```
Test loss: 0.0747, Test accuracy: 0.9787
313/313 [==============================] - 1s 2ms/step
Confusion matrix:
 [[ 975    0    1    0    1    1    0    1    1    0]
 [   0 1126    1    2    0    1    3    1    1    0]
 [   2    2 1009    0    2    0    2   10    5    0]
 [   0    0    7  976    0   11    0   10    2    4]
 [   1    1    0    0  964    0    4    0    1   11]
 [   3    0    0    2    0  865   15    1    3    3]
 [   4    3    0    0    1    1  949    0    0    0]
 [   0    2    9    2    0    0    0 1005    2    8]
 [  11    2    7    2    1    3    7    1  933    7]
 [   4    3    0    0    5    5    0    7    0  985]]
```

ReLU激活函数

我们已经比较了不同激活函数和0.5的dropout率的模型在测试集上的准确率，结果如下：

激活函数	测试准确率
sigmoid	0.1135
tanh	0.9697
ReLU	0.9787

可以看出，ReLU激活函数的表现最好，sigmoid激活函数的表现最差。这是因为ReLU激活函数可以解决梯度消失和梯度爆炸的问题，而sigmoid激活函数容易导致神经元饱和2。

接下来，我们将尝试不同的dropout率，从0.1到0.5，并观察它们对模型性能的影响。我们将使用ReLU激活函数和0.01的学习率，保持其他参数不变。

```
# 定义一个列表，存储不同的dropout率
dropout_rates = [0.1, 0.2, 0.3, 0.4, 0.5]
# 定义一个列表，存储不同dropout率对应的测试准确率
test_accs = []
# 对每个dropout率，构建、训练和评估模型
for dropout_rate in dropout_rates:
  # 使用ReLU激活函数和指定的dropout率构建模型
  model = build_model(activation='relu', dropout_rate=dropout_rate)
  # 使用128的批量大小，0.01的学习率和10个周期训练和评估模型
  train_and_evaluate_model(model, batch_size=128, learning_rate=0.01, epochs=10)
  # 记录测试准确率
  test_acc = model.evaluate(x_test, y_test)[1]
  test_accs.append(test_acc)

# 绘制不同dropout率与测试准确率的折线图
plt.plot(dropout_rates, test_accs)
plt.xlabel('Dropout rate')
plt.ylabel('Test accuracy')
plt.show()
```

运行上面的代码，我们可以得到如下结果：

```
Test loss: 0.0577, Test accuracy: 0.9826
313/313 [==============================] - 1s 2ms/step
Confusion matrix:
 [[ 959    1    2    0    0    1   11    1    1    4]
 [   0 1132    2    0    0    0    1    0    0    0]
 [   1    2 1018    8    0    0    0    1    2    0]
 [   0    0    1  997    0    9    0    2    1    0]
 [   1    0    0    0  954    0    5    2    0   20]
 [   1    0    0    6    0  874    4    0    1    6]
 [   0    2    0    1    1    1  952    0    1    0]
 [   0    5   20    3    0    0    0  993    0    7]
 [   7    2    1    3    0    2    0    0  954    5]
 [   0    5    1    0    2    4    0    4    0  993]]
```

```
Test loss: 0.0765, Test accuracy: 0.9762
313/313 [==============================] - 1s 2ms/step
Confusion matrix:
 [[ 973    0    5    0    0    0    0    1    1    0]
 [   0 1116    4    3    1    0    2    3    6    0]
 [   3    0 1022    1    0    0    0    3    3    0]
 [   0    0    3  998    0    1    0    3    3    2]
 [   2    0    1    0  942    0    4    2    3   28]
 [   4    1    0   39    0  835    3    1    9    0]
 [  11    2    2    0    1    3  934    0    5    0]
 [   0    2   20    1    1    0    0  998    1    5]
 [   4    0    1    1    0    1    1    0  964    2]
 [   5    0    2    7    8    1    0    3    3  980]]
```

```
Test loss: 0.1006, Test accuracy: 0.9710
313/313 [==============================] - 1s 2ms/step
Confusion matrix:
 [[ 974    0    0    0    0    0    3    1    1    1]
 [   0 1126    1    0    0    0    1    3    4    0]
 [   2    0 1017    3    0    0    0    9    1    0]
 [   0    0    4  994    0    2    0    5    0    5]
 [   0    3    1    0  938    0    1    2    3   34]
 [   8    2    1   14    0  815    6    1    4   41]
 [  13    2    3    0    0    1  936    0    3    0]
 [   0    3    4    2    1    0    0 1017    0    1]
 [   9    0   38    5    0    0    1    5  897   19]
 [   1    1    0    0    1    0    0    9    1  996]]
```

```
Test loss: 0.0786, Test accuracy: 0.9774
313/313 [==============================] - 1s 2ms/step
Confusion matrix:
 [[ 965    0    1    0    2    1    9    1    1    0]
 [   2 1124    2    3    0    1    1    1    1    0]
 [  15    0  995    7    4    0    0    7    4    0]
 [   2    0    1  995    0    8    0    1    3    0]
 [   0    0    2    0  940    0    1    1    3   35]
 [   3    0    0    4    0  878    4    0    0    3]
 [   0    3    0    0    3    8  942    0    2    0]
 [   0    4   12    3    1    4    0  994    1    9]
 [   3    0    0    3    3    6    2    0  950    7]
 [   1    3    0    1    4    4    0    3    2  991]]
```

```
est loss: 0.0888, Test accuracy: 0.9765
13/313 [==============================] - 1s 2ms/step
onfusion matrix:
[[ 975    0    2    0    0    1    1    1    0    0]
 [   0 1126    2    3    0    0    0    1    3    0]
 [   1    1 1019    2    1    0    0    4    4    0]
 [   0    0    4  992    0    5    0    7    2    0]
 [   2    2    3    0  954    0    7    1    4    9]
 [   3    0    0   12    0  870    3    1    2    1]
 [  14    3    1    0    3    3  932    0    2    0]
 [   1    3   17    2    0    1    0  994    1    9]
 [  10    0    5    2    3    8    2    1  930   13]
 [   2    4    0    3    7    6    0    9    5  973]]
```

从图中可以看出，当dropout率为0.1，模型的测试准确率较高，超过98%。当dropout率较高时（0.4或0.5），模型的测试准确率较低。这说明过高的dropout率会导致模型欠拟合，丢弃了过多的有效信息。因此，我们应该选择一个适中的dropout率，以平衡模型的复杂度和泛化能力。

五、实验总体结论

- **卷积神经网络是一种强大的深度学习模型，可以有效地提取图像的特征，提高分类和识别的准确率。**
- **卷积神经网络的性能受到多个超参数的影响，例如卷积核的大小、个数、步长、填充等，以及池化层的类型、大小、步长等。不同的超参数组合会导致不同的输出维度、计算量和特征表示。**
- **卷积神经网络的设计应该根据具体的任务和数据集进行调整，以达到最优的效果。一般来说，增加卷积层和卷积核的个数可以增加模型的表达能力，但也会增加计算量和过拟合的风险；使用合适的填充和步长可以保持图像的空间信息，避免边缘信息的丢失；使用最大池化层可以降低特征维度，增强模型的鲁棒性，但也会损失一些细节信息。**

六、完整实验代码

```python
# 导入需要的库
import tensorflow as tf
from tensorflow import keras
from keras import layers, models, datasets
import numpy as np
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

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

# 将图像数据转换为浮点数，并归一化到[0,1]范围
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# 为了适应Lenet-5的输入要求，将图像数据从28x28扩展为32x32，并增加一个通道维度
x_train = tf.pad(x_train, [[0,0],[2,2],[2,2]])
x_test = tf.pad(x_test, [[0,0],[2,2],[2,2]])
x_train = tf.expand_dims(x_train, axis=-1)
x_test = tf.expand_dims(x_test, axis=-1)

# 将标签数据转换为one-hot编码
```

```
y_train = tf.one_hot(y_train, depth=10)
y_test = tf.one_hot(y_test, depth=10)

# 定义一个函数，用于构建不同激活函数和dropout率的Lenet-5模型
def build_model(activation='relu', dropout_rate=0.5):
  model = models.Sequential([
    # 第一个卷积层，使用6个5x5的卷积核，激活函数由参数指定
    layers.Conv2D(6, kernel_size=5, activation=activation, input_shape=(32, 32,
1)),
    # 第一个池化层，使用2x2的最大池化
    layers.MaxPool2D(pool_size=2),
    # 第二个卷积层，使用16个5x5的卷积核，激活函数由参数指定
    layers.Conv2D(16, kernel_size=5, activation=activation),
    # 第二个池化层，使用2x2的最大池化
    layers.MaxPool2D(pool_size=2),
    # 将卷积层的输出展平为一维向量
    layers.Flatten(),
    # 第一个全连接层，有120个神经元，激活函数由参数指定
    layers.Dense(120, activation=activation),
    # 在全连接层后添加dropout层，丢弃率由参数指定
    layers.Dropout(dropout_rate),
    # 第二个全连接层，有84个神经元，激活函数由参数指定
    layers.Dense(84, activation=activation),
    # 在全连接层后添加dropout层，丢弃率由参数指定
    layers.Dropout(dropout_rate),
    # 第三个全连接层，有10个神经元，激活函数为softmax，输出分类概率
    layers.Dense(10, activation='softmax')
  ])
  return model

# 定义一个函数，用于训练和评估不同数据量和超参数的模型
def train_and_evaluate_model(model, batch_size=128, learning_rate=0.01,
epochs=10):
  # 编译模型，使用交叉熵损失函数和Adam优化器，评估指标为准确率
  model.compile(loss='categorical_crossentropy',
optimizer=keras.optimizers.Adam(learning_rate=learning_rate), metrics=
['accuracy'])
  # 训练模型，使用指定的批量大小和周期数，每个周期结束后在测试集上评估模型性能，并记录训练过程中
的损失和准确率
  history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_data=(x_test, y_test))
  # 打印模型在测试集上的准确率
  test_loss, test_acc = model.evaluate(x_test, y_test)
  print('Test loss: {:.4f}, Test accuracy: {:.4f}'.format(test_loss, test_acc))
  # 使用模型对测试集进行预测，并计算混淆矩阵
  y_pred = model.predict(x_test)
  y_pred = np.argmax(y_pred, axis=1)
  y_true = np.argmax(y_test, axis=1)
  confusion_matrix = tf.math.confusion_matrix(y_true, y_pred)
  print('Confusion matrix:\n', confusion_matrix.numpy())

# 使用sigmoid激活函数和0.5的dropout率构建模型
model_sigmoid = build_model(activation='sigmoid', dropout_rate=0.5)
# 使用128的批量大小，0.01的学习率和10个周期训练和评估模型
```

```
train_and_evaluate_model(model_sigmoid, batch_size=128, learning_rate=0.01,
epochs=10)

# 使用tanh激活函数和0.5的dropout率构建模型
model_tanh = build_model(activation='tanh', dropout_rate=0.5)
# 使用128的批量大小，0.01的学习率和10个周期训练和评估模型
train_and_evaluate_model(model_tanh, batch_size=128, learning_rate=0.01,
epochs=10)

# 使用ReLU激活函数和0.5的dropout率构建模型
model_relu = build_model(activation='relu', dropout_rate=0.5)
# 使用128的批量大小，0.01的学习率和10个周期训练和评估模型
train_and_evaluate_model(model_relu, batch_size=128, learning_rate=0.01,
epochs=10)

# 定义一个列表，存储不同的dropout率
dropout_rates = [0.1, 0.2, 0.3, 0.4, 0.5]
# 定义一个列表，存储不同dropout率对应的测试准确率
test_accs = []
# 对每个dropout率，构建、训练和评估模型
for dropout_rate in dropout_rates:
  # 使用ReLU激活函数和指定的dropout率构建模型
  model = build_model(activation='relu', dropout_rate=dropout_rate)
  # 使用128的批量大小，0.01的学习率和3个周期训练和评估模型
  train_and_evaluate_model(model, batch_size=128, learning_rate=0.01, epochs=10)
  # 记录测试准确率
  test_acc = model.evaluate(x_test, y_test)[1]
  test_accs.append(test_acc)
# 定义一个列表，存储不同的数据量
data_sizes = [10000, 20000, 30000, 40000, 50000, 60000]

# 定义一个列表，存储不同数据量对应的测试准确率
test_accs = []
# 对每个数据量，构建、训练和评估模型
for data_size in data_sizes:
  # 使用ReLU激活函数和0.5的dropout率构建模型
  model = build_model(activation='relu', dropout_rate=0.5)
  # 使用128的批量大小，0.01的学习率和10个周期训练和评估模型，只使用指定数量的训练数据
  train_and_evaluate_model(model, batch_size=128, learning_rate=0.01, epochs=10,
x_train=x_train[:data_size], y_train=y_train[:data_size])
  # 记录测试准确率
  test_acc = model.evaluate(x_test, y_test)[1]
  test_accs.append(test_acc)
```

七、参考文献

- **周志华著.机器学习，北京：清华大学出版社，2016.1**
- **李航著.统计学习方法，北京：清华大学出版社，2019.5**