实验三报告

所使用的库以及安装过程中出现的问题

```
import argparse
import struct
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
-----------------------------
# 下面是Anaconda的安装步骤
conda create -n myenv python=3.8
conda activate myenv
conda install tensorflow-gpu
# 上面这条命令会自动安装tensorflow及其依赖，可能需要一点时间
# sklearn的安装这里就不写了
```

安装完成后试着`import tensorflow as tf`，但是却出现了一些报错

```
AttributeError: module 'numpy' has no attribute 'object'.
`np.object` was a deprecated alias for the builtin `object`. To avoid this error in
existing code, use `object` by itself. Doing this will not modify any behavior and is
safe.
The aliases was originally deprecated in NumPy 1.20; for more details and guidance see
the original release note at:
    https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

该报错要求numpy的版本在1.20之下，但是按照依赖自动安装的版本是1.24，只能手动`conda install -c conda-forge numpy=1.19`将numpy版本降低到1.19

下面是环境中一些比较重要的包的版本

```
python        3.8
tensorflow    2.3.0
numpy         1.19.5
```

数据预处理以及argparse的调用

```
def load_mnist_images(filename):
    with open(filename, 'rb') as f:
        # 读取文件头
        magic, num_images, rows, cols = struct.unpack('>IIII', f.read(16))
        # 读取图像数据
        images = np.fromfile(f, dtype=np.uint8).reshape(num_images, rows, cols)
    return images

def load_mnist_labels(filename):
    with open(filename, 'rb') as f:
        # 读取文件头
        magic, num_labels = struct.unpack('>II', f.read(8))
        # 读取标签数据
        labels = np.fromfile(f, dtype=np.uint8)
    return labels

# 创建 ArgumentParser 对象
parser = argparse.ArgumentParser(description="Add two numbers")

# 添加命令行参数
parser.add_argument("--model", type=str, required=True, choices=
["lenet", "alexnet", "resnet", "moblienet"])
parser.add_argument("--learning_rate", type=float, required=True)
parser.add_argument("--dropout", type=float, required=True)
parser.add_argument("--epochs", type=int, required=True)

# 解析命令行参数
args = parser.parse_args()
model = args.model

# 加载MNIST训练集图像
all_images = load_mnist_images('train-images.idx3-ubyte')
# 加载MNIST训练集标签
all_labels = load_mnist_labels('train-labels.idx1-ubyte')
# 划分训练集和验证集
train_images, val_images, train_labels, val_labels = train_test_split(
    all_images, all_labels, test_size=0.2, random_state=42)
# 将标签进行独热编码
train_labels_one_hot = tf.keras.utils.to_categorical(train_labels, 10)
val_labels_one_hot = tf.keras.utils.to_categorical(val_labels, 10)
```

模型训练示例，以lenet示例

将train-images.idx3-ubyte中随机80%的数据作为训练集，剩余20%作为验证集，t10k-images.idx3-ubyte作为测试集

```
if model == 'lenet' :
    print('############################')
    print('#    Using LeNet!!!!!!!!   #')
    print('############################')

    # 设置参数
    learning_rate = args.learning_rate  # 设置你想要的学习率
    epoch = args.epochs
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    dropout = args.dropout

    # 构建LeNet模型
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=
(28, 28, 1)),
        tf.keras.layers.AveragePooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(16, kernel_size=(5, 5), activation='relu'),
        tf.keras.layers.AveragePooling2D(pool_size=(2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(120, activation='relu'),
        tf.keras.layers.Dropout(dropout),  # 添加 Dropout 层，参数是 Dropout 的比例
        tf.keras.layers.Dense(84, activation='relu'),
        tf.keras.layers.Dropout(dropout),  # 添加 Dropout 层，参数是 Dropout 的比例
        tf.keras.layers.Dense(10, activation='softmax')
    ])


    # 编译模型，使用指定的优化器
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=
['accuracy'])

    # 数据预处理
    train_images = train_images.reshape(-1, 28, 28, 1).astype('float32') / 255.0
    val_images = val_images.reshape(-1, 28, 28, 1).astype('float32') / 255.0

    # 训练模型
    history = model.fit(train_images, train_labels_one_hot, epochs=epoch,
                        validation_data=(val_images, val_labels_one_hot))

    # 输出最终验证集准确率
    val_loss, val_acc = model.evaluate(val_images, val_labels_one_hot)
    print(f"Final Validation Accuracy of LeNet : {val_acc * 100:.2f}%")

    # 加载MNIST测试集图像
    test_images = load_mnist_images('t10k-images.idx3-ubyte')
    test_images = tf.expand_dims(test_images, axis=-1)

    # 加载MNIST测试集标签
    test_labels = load_mnist_labels('t10k-labels.idx1-ubyte')
    test_labels_one_hot = tf.keras.utils.to_categorical(test_labels, 10)

    # 评估模型在测试集上的性能
    test_loss, test_acc = model.evaluate(test_images, test_labels_one_hot)
    print(f"Test Accuracy of LeNet : {test_acc * 100:.2f}%")
```

其他三个模型的过程都与这个差不多，因此不再放出

结果比较

选择的额外一个架构：mobilenet

参数：

learning_rate: 0.001

epochs: 5

dropout: 0.5

前三个模型的验证集和测试集的准确率都很高，我认为没有什么比较的意义，这个数据集比较简单，对于模型的图像分类来说并不难，选择的mobilenet，MobileNet是一种轻量级的神经网络架构，适用于移动设备。它使用深度可分离卷积来减小模型大小，适合资源受限的环境。

对于模型的性能限制方面，我认为限制模型性能的主要方面是模型的复杂度，更复杂一点的resnet的准确率比前两个更低一点，在所有模型中，resnet我认为是最复杂的，就是因为这个，在训练中应该需要更多的epochs来训练，以达到更高的准确率。

可以发现moblienet的测试集准确率相较于其他模型有点太低了，查看模型的训练过程,可以发现训练集和验证集的准确率都很高，我认为这是训练中模型出现了过拟合现象，导致模型泛化能力变差，将学习率调整至0.0001，其他参数不变，可以发现测试集准确率发生明显上浮，提高至0.93

```
Epoch 1/5
1500/1500 [==============================] - 6s 4ms/step - loss: 0.5926 - accuracy:
0.8331 - val_loss: 0.2442 - val_accuracy: 0.9302
Epoch 2/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2850 - accuracy:
0.9161 - val_loss: 0.1764 - val_accuracy: 0.9479
Epoch 3/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2188 - accuracy:
0.9361 - val_loss: 0.1376 - val_accuracy: 0.9607
Epoch 4/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1767 - accuracy:
0.9479 - val_loss: 0.1146 - val_accuracy: 0.9672
Epoch 5/5
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1499 - accuracy:
0.9569 - val_loss: 0.0961 - val_accuracy: 0.9725
375/375 [==============================] - 1s 2ms/step - loss: 0.0961 - accuracy:
0.9725
```

```
Final Validation Accuracy: 97.25%
313/313 [==============================] - 1s 2ms/step - loss: 24.6326 - accuracy:
0.9367
Test Accuracy: 93.67%
```

遇到的一个问题

首先要讲resnet的训练前，与其他模型不同的是，少了这一步

```
# 数据预处理
train_images = train_images.reshape(-1, 28, 28, 1).astype('float32') / 255.0
val_images = val_images.reshape(-1, 28, 28, 1).astype('float32') / 255.0
```

这是因为如果加上了这一步数据预处理，在模型进行测试集的预测时，模型的准确率会忽高忽低，在10%到70%之间反复横跳，没有一个较为稳定的结果，我是在将代码复制到kaggle平台上进行运行时发现与电脑上跑的不一致，进而发现少复制了一段，才发现这一原因的，暂时没有想到造成这个事件的原因。