

电子科技大学

计算机专业类课程

实验报告

课程名称：人工智能综合实验 II

学 院：计算机科学与工程学院

学院专业：计算机科学与技术

学 号：2021060904008

学生姓名：周杰锋

指导教师：高连丽，宋井宽

日 期：2023 年 10 月 19 日

电子科技大学

实验报告

实验一

一、实验室名称:

电子科技大学清水河校区主楼 A2-413

二、实验项目名称:

人工智能实验 II-1：环境熟悉及深度学习初探

三、实验原理:

- 1) 线性回归的基本理论，线性回归模型是利用称为线性回归方程的最小平方函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。
- 2) softmax 回归的基本理论，softmax 回归模型是线性回归模型在多分类问题上的推广，在多分类问题中，类标签 y 可以取两个以上的值。
- 3) 多层感知器的基本理论，多层感知器模型是在线性回归或者 softmax 回归的基础上添加多层的操作，并有层与层之间有激活函数。
- 4) 对比分析在深度学习领域是使用不同参数、模型结构等选择出最优模型，达到分析比较的目的。可视化分析是人工操作将数据进行关联分析，并做出完整的分析图表。

四、实验目的:

- 1) 学习基本的 Pytorch 的基本数据操作
- 2) 深度学习模型搭建的基本流程
- 3) 利用 Pytorch 实现线性回归、Softmax 及多层感知器回归模型
- 4) 掌握实验结果的分析能力，包括两个方面：对比分析与可视化分析

五、实验内容:

- 1) 问题描述在本次试验中，我们首先需要配置好相关的深度学习环境，然后利用 Pytorch 实现线性回归、Softmax 及多层感知器回归模型，并对比分析

三种模型的性能差异，最后利用可视化工具对实验结果进行可视化分析。

2) 算法分析与概要设计

- a) 输入：训练数据集、测试数据集
- b) 输出：测试集的预测结果、模型性能的对比分析、实验结果的可视化分析
- c) 算法描述：本次实验中我们需要实现层感知器模型。在实现该模型时，我们需要对模型的输入输出进行定义，然后利用 Pytorch 搭建模型，最后利用 Pytorch 相关函数进行训练与测试。

3) 核心算法的详细设计与实现

- a) 多层感知器模型的详细设计与实现
 - i. 输入：训练数据集、测试数据集
 - ii. 输出：测试集的预测结果
 - iii. 算法描述：利用 Pytorch 搭建一个多层感知器模型，然后利用 Pytorch 相关函数进行训练与测试。

六、实验器材（设备、元器件）：

- 1) 硬件平台：CPU: AMD Ryzen 7 5800U 1.90 GHz; GPU: NVIDIA GeForce RTX 3050 LAPTOP; 内存: 16GB@4266MHz
- 2) 开发环境: Ubuntu 20.04.2 LTS, Python 3.9.17, Pytorch 2.0.1+cu117
- 3) 测试环境: 同上

七、实验步骤:

（一）网络框架结构搭建介绍

- 1) 多层感知器模型的搭建
 - a) 输入：训练数据集、测试数据集
 - b) 输出：测试集的预测结果
 - c) 算法描述：利用 Pytorch 搭建一个多层感知器模型，然后利用 Pytorch 相关函数进行训练与测试。

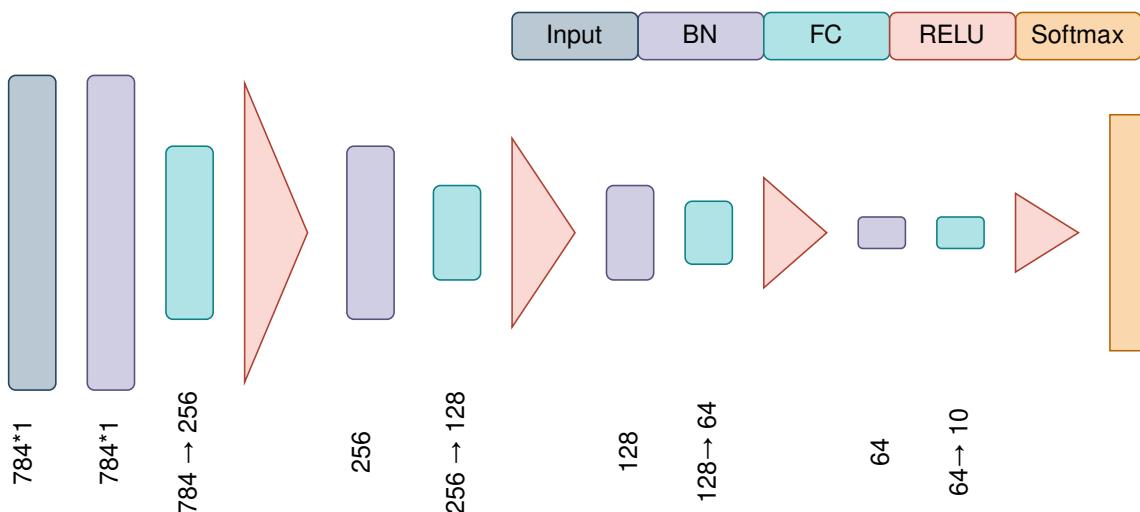


图 1: 多层感知器模型架构

感知器网络架构

如图所示，该模型总共有 5 层，其中输入层为 784 维，输出层为 10 维，中间两层分别为 256 维和 128 维，激活函数为 ReLU 函数，最后一层为 Softmax 函数。每层之间都有 BatchNorm 层，用于对层的输入进行归一化处理，防止梯度消失或梯度爆炸。最后一层为 Softmax 函数，用于将输出转换为概率分布。

模型超参数设置

表 1: 模型超参数设置

超参数	数值
epoch	20
batch size	256
learning rate	0.005

(二) 核心代码介绍

网络结构代码

代码 1: MLP 网络定义

```

1 class LinearNet(nn.Module):
2     def __init__(self, input_nums, output_nums):
3         super(LinearNet, self).__init__()
4         self.batch_norm0 = nn.BatchNorm1d(input_nums)
5         self.fc0 = nn.Linear(input_nums, 256)

```

```

6     self.relu0 = nn.ReLU()
7     self.batch_norm1 = nn.BatchNorm1d(256)
8     self.fc1 = nn.Linear(256, 128)
9     self.relu1 = nn.ReLU()
10    self.batch_norm2 = nn.BatchNorm1d(128)
11    self.fc2 = nn.Linear(128, 64)
12    self.relu2 = nn.ReLU()
13    self.batch_norm3 = nn.BatchNorm1d(64)
14    self.fc3 = nn.Linear(64, output_nums)
15    self.relu3 = nn.ReLU()
16    self.softmax = nn.Softmax(dim=1)
17
18  def forward(self, x):
19      x = self.batch_norm0(x)
20      x = self.fc0(x)
21      x = self.relu0(x)
22      x = self.batch_norm1(x)
23      x = self.fc1(x)
24      x = self.relu1(x)
25      x = self.batch_norm2(x)
26      x = self.fc2(x)
27      x = self.relu2(x)
28      x = self.batch_norm3(x)
29      x = self.fc3(x)
30      x = self.relu3(x)
31      x = self.softmax(x)
32  return x

```

如代码所示，该模型通过调用 Pytorch 中预定义的层来完成 MLP 的搭建。

训练设计代码

代码 2: MLP 网络训练代码（部分）

```

1  for epoch in range(epocbes):
2      model.train()
3      losses = []
4      correct_predictions = 0
5      for batch_idx, (data, target) in enumerate(train_loader):
6          data = data.to(device)
7          target = target.to(device)
8          optimizer.zero_grad()
9          # flatten the input tensor
10         data = data.view(data.size(0), -1)
11         # forward
12         output = model(data)
13         # loss
14         loss = loss_fn(output, target)
15         losses.append(loss.item())
16         # backward
17         loss.backward()
18         # update weights
19         optimizer.step()
20         # calculate accuracy
21         _, predicted = torch.max(output.data, 1)
22         correct_predictions += (predicted == target).sum().item()
23         train_loss = np.mean(losses)
24         train_accuracy = 100. * correct_predictions / len(train_loader.dataset)
25         writer.add_scalar('Train Loss', train_loss, epoch)
26         writer.add_scalar('Train Accuracy', train_accuracy, epoch)
27         print(f"Train Loss: {train_loss:.4f}, Train Acc: {train_accuracy:.2f}")

```

在 MLP 网络训练过程中，先使用 ‘data_loader’ 加载训练集和测试集，ensor 格式，并将数据传入 GPU 中进行加速。在训练过程中，首先将模型的梯度清零，然

后调用模型的‘forward’函数得到预测结果，接着计算损失值，之后进行反向传播，更新模型参数。最后计算训练集的准确率，并将训练过程中的损失值和准确率写入Tensorboard中。

测试设计代码

代码 3: MLP 网络测试代码（部分）

```
1 model.eval()
2 test_loss = 0
3 correct_predictions = 0
4 with torch.no_grad():
5     for data, target in test_loader:
6         data = data.to(device)
7         target = target.to(device)
8         data = data.view(data.size(0), -1)
9         output = model(data)
10        test_loss += loss_fn(output, target).item()
11        _, predicted = torch.max(output.data, 1)
12        correct_predictions += (predicted == target).sum().item()
13
14    test_loss /= len(test_loader.dataset)
15    test_accuracy = 100. * correct_predictions / len(test_loader.dataset)
16    writer.add_scalar('Test Loss', test_loss, epoch)
17    writer.add_scalar('Test Accuracy', test_accuracy, epoch)
18    print(f"Test Loss: {test_loss:.4f}, Test Acc: {test_accuracy:.2f}")
```

在MLP网络测试过程中，首先将模型转换为测试模式，然后对测试集进行预测，并计算损失值和准确率，最后将测试过程中的损失值和准确率写入Tensorboard中。

（三）训练执行流程介绍

- 1) 环境配置
 - a) 配置 Pytorch 环境
- 2) 数据准备
 - a) 下载 MNIST 数据集
 - b) 利用 Pytorch 的‘DataLoader’ 加载数据集
- 3) 模型搭建
- 4) 模型训练
- 5) 模型测试
- 6) TensorBoard 结果分析

八、实验数据及结果分析：

实验数据

在本次实验中，我们探究了四种不同的参数设置对模型性能的影响，分别为：不同的优化器、不同的损失函数、不同的批次训练大小、不同的迭代次数。最终的实验结果如表??_resultstab:MLP_results

表 2: 实验结果对比

参数类型	参数设置	训练集 ACC	测试集 ACC
Epoches	10	91.58	87.96
	20	93.72	88.54
	40	95.73	89.29
Loss function	CrossEntropyLoss	91.58	87.96
	NLLLoss	86.28	84.89
	MSELoss	87.75	85.34
Batch_size	64	92.02	87.61
	128	93.02	88.92
	256	95.72	89.16
Optim	SGD	95.72	89.16
	Adam	91.58	87.96
	Adadelta	88.64	86.49

从实验结果中可以看出：

- 1) 当迭代次数增加时，模型的性能也随之增加，但是增加的幅度逐渐减小，当迭代次数达到 40 次时，模型的性能已经达到了一个较高的水平。
- 2) 当损失函数为交叉熵损失函数时，模型的性能最好，当损失函数为负对数似然损失函数时，模型的性能最差。
- 3) 当批次训练大小为 256 时，模型的性能最好，当批次训练大小为 64 时，模型的性能最差。
- 4) 当优化器为 SGD 时，模型的性能最好，当优化器为 Adadelta 时，模型的性能最差。

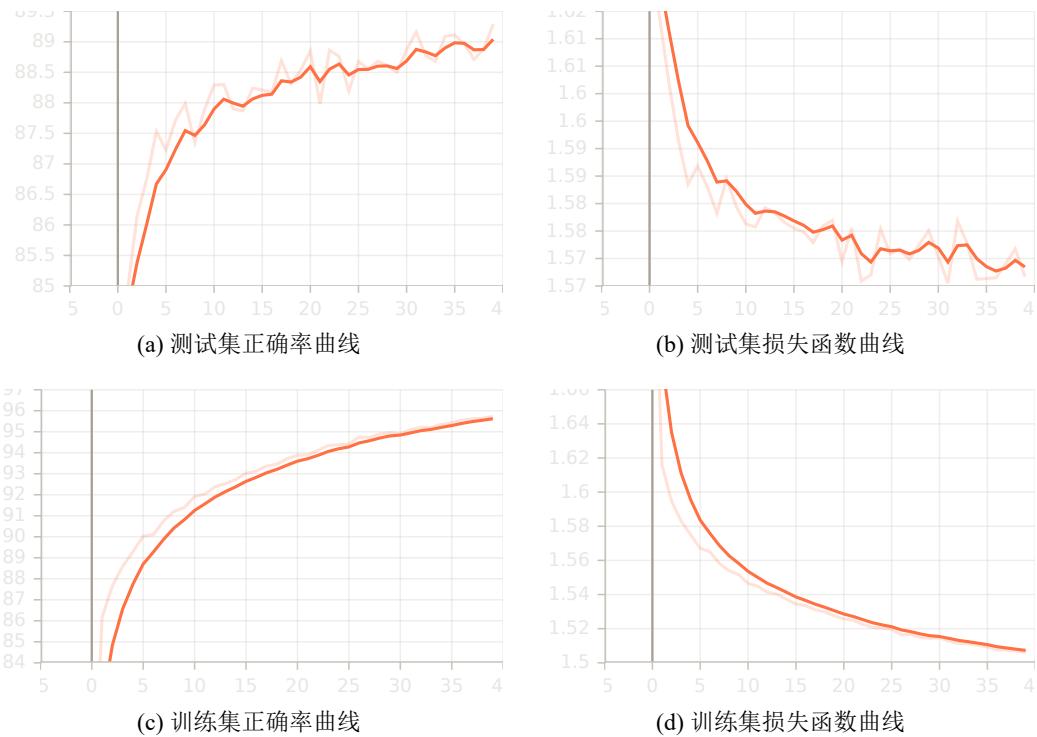


图 2: MLP 网络训练过程中的正确率和损失函数曲线

（一）结果分析

该结果为 MLP 网络训练过程中的正确率和损失函数曲线，确定的参数为：epoches=40, batch size=256, loss function=CrossEntropyLoss, optim=SGD, learning rate=0.005。

从图中可以看出，随着迭代次数的增加，训练集的正确率和测试集的正确率都在增加，但是增加的幅度逐渐减小，当迭代次数达到 40 次时，模型的性能已经达到了一个较高的水平。

九、总结及心得体会：

在本次实验中，我们首先配置好相关的深度学习环境，然后利用 Pytorch 实现线性回归、Softmax 及多层感知器回归模型，并对比分析三种模型的性能差异，最后利用可视化工具对实验结果进行可视化分析。

在本次实验中，我学习了 Pytorch 的基本数据操作，深度学习模型搭建的基本流程，以及如何利用 Pytorch 实现线性回归、Softmax 及多层感知器回归模型。同时，我也学会了如何利用 TensorBoard 对实验结果进行可视化分析。

十、对本实验过程及方法、手段的改进建议及展望:

- 1) 能否提供 Tex 版本的实验报告模板，方便使用 Tex 的同学进行实验报告的编写。
- 2) 在课堂演示中，可以增加对 Tensorboard 的使用方法的介绍。
- 3) 在探究不同参数设置对实验结果的影响中，能否减少探究个数，以减少实验时间。

报告评分：

指导教师签字：

电子科技大学

实验报告

实验二

一、实验室名称:

电子科技大学清水河校区主楼 A2-413

二、实验项目名称:

人工智能实验 II-2：基于卷积神经网络图像分类算法实现

三、实验原理:

卷积网络在本质上是一种输入到输出的映射，它能够学习大量的输入与输出之间的映射关系，它主要被用来识别位移、缩放及其他形式扭曲不变性的二维图像。由于 CNN 的特征检测层通过训练数据进行学习，所以在使用 CNN 时，避免了显式的特征抽取，而隐式地从训练数据中进行学习；再者由于同一特征映射面上的神经元权值相同，所以网络可以并行学习，这也是卷积网络相对于神经元彼此相连网络的一大优势。卷积神经网络以其局部权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性，其布局更接近于实际的生物神经网络，权值共享降低了网络的复杂性，特别是多维输入向量的图像可以直接输入网络这一特点避免了特征提取和分类过程中数据重建的复杂度。

四、实验目的:

通过 Pytorch 框架搭建 LeNet 网络，实现在 CIFAR-10 数据集上的图像分类，并以此掌握：

- 1) 卷积神经网络中卷积层、池化层、全连接层的使用场景和作用以及它们参数的具体含义。
- 2) 了解 Pytorch 框架下模型的训练流程：训练、测试以及评估方法，并亲自代码实践。
- 3) 利用 Pytorch 实现线性回归、Softmax 及多层感知器回归模型。
- 4) 实验结果的分析能力，包括两个方面：量化分析与可视化分析。

五、实验内容：

- 1) 问题描述本次实验中需要实现 ResNet，最后利用可视化工具对实验结果进行可视化分析。
- 2) 算法分析与概要设计
 - a) 输入：训练数据集、测试数据集
 - b) 输出：测试集的预测结果、模型性能的对比分析、实验结果的可视化分析
 - c) 算法描述：搭建对应的 Data loader，构建 ResNet，对网络进行训练，最后对实验结果进行可视化分析。
- 3) 核心算法的详细设计与实现
 - a) Residual_Block 实现
 - i. 输入：输入维度、输出维度、stride
 - ii. 输出：对应的 Residual_Block 实例
 - iii. 算法描述：使用 Class 定义一个 Residual_Block 类，然后在类中定义 forward 函数，最后在 forward 函数中实现 Residual_Block 的功能。
 - b) ResNet 实现
 - i. 输入：输入维度、输出维度、残差块数量、stride
 - ii. 输出：对应的 ResNet 实例
 - iii. 算法描述：使用 Class 定义一个 ResNet 类，然后在类中定义 forward 函数，最后在 forward 函数中实现 ResNet 的功能。

六、实验器材（设备、元器件）：

- 1) 硬件平台：CPU: AMD Ryzen 7 5800U 1.90 GHz; GPU: NVIDIA GeForce RTX 3050 LAPTOP; 内存: 16GB@4266MHz
- 2) 开发环境：Ubuntu 20.04.2 LTS, Python 3.9.17, Pytorch 2.0.1+cu117
- 3) 测试环境：同上

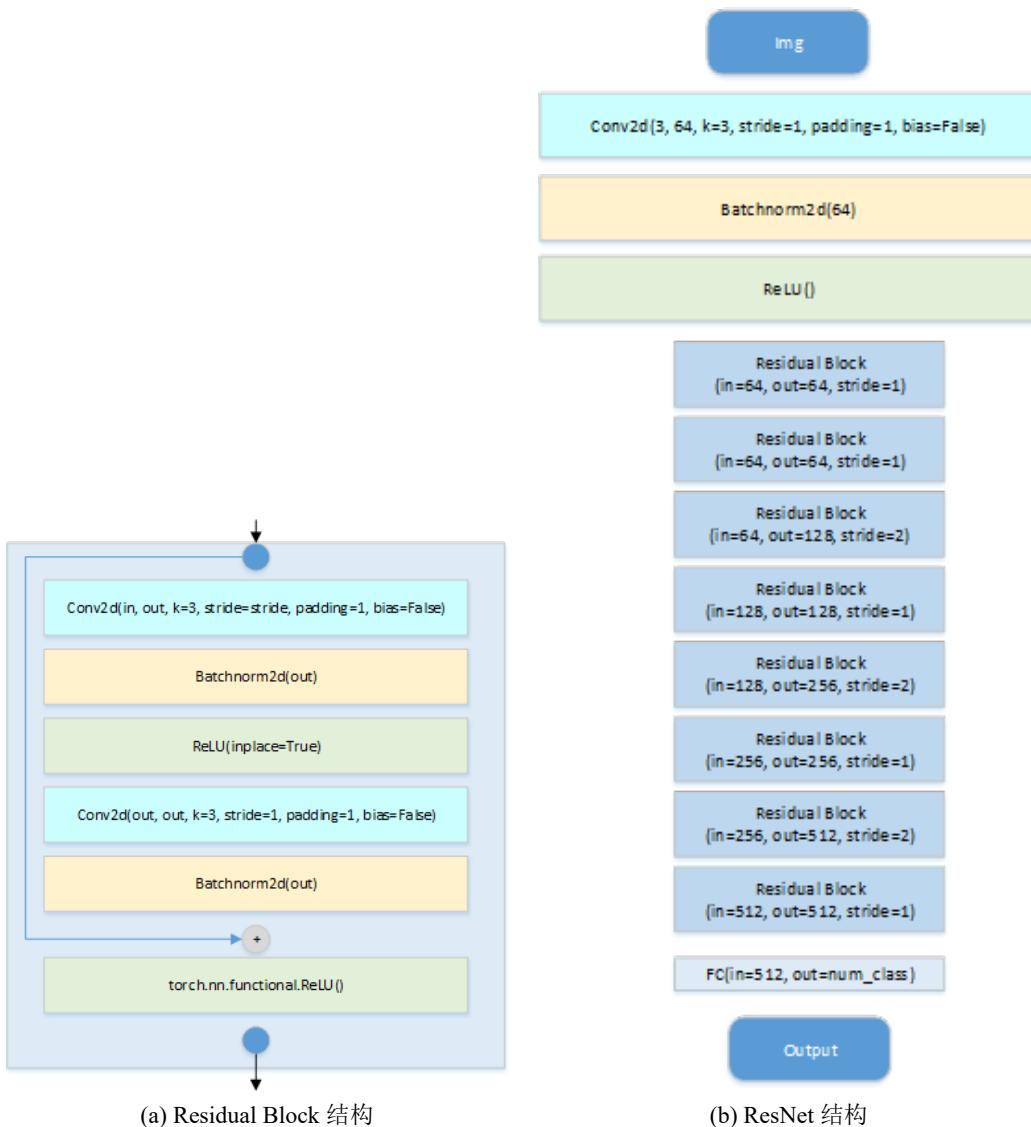


图 3: ResNet18 网络结构

七、实验步骤：

网络框架结构搭建介绍

如上图所示，本次实验中复现了 ResNet18 的网络结构，包括 Residual Block，以及整体的网络结构。

(一) 核心代码介绍

Residual Block 代码

代码 4: Residual Block 代码

```
1 class Residual_Block(nn.Module):
2     def __init__(self, input_channels, num_channels,
3                  use_1x1conv=False, strides=1):
4         super().__init__()
5         self.conv1 = nn.Conv2d(input_channels, num_channels,
6                               kernel_size=3, padding=1, stride=strides)
7         self.conv2 = nn.Conv2d(num_channels, num_channels,
8                               kernel_size=3, padding=1)
9         if use_1x1conv:
10            self.conv3 = nn.Conv2d(input_channels, num_channels,
11                               kernel_size=1, stride=strides)
12        else:
13            self.conv3 = None
14        self.bn1 = nn.BatchNorm2d(num_channels)
15        self.bn2 = nn.BatchNorm2d(num_channels)
16
17    def forward(self, X):
18        Y = F.relu(self.bn1(self.conv1(X)))
19        Y = self.bn2(self.conv2(Y))
20        if self.conv3:
21            X = self.conv3(X)
22        return F.relu(Y + X)
```

ResNet 网络结构代码

代码 5: ResNet 网络结构代码（部分）

```
1 def resnet_block(input_channels, num_channels, num_residuals, first_block=
2     False):
3     blk = []
4     for i in range(num_residuals):
5         if i == 0 and not first_block:
6             blk.append(Residual_Block(input_channels, num_channels,
7                           use_1x1conv=True, strides=2))
8         else:
9             blk.append(Residual_Block(num_channels, num_channels))
10    return nn.Sequential(*blk)
11
12 net = nn.Sequential(
13     nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
14     nn.BatchNorm2d(64),
15     nn.ReLU(),
16     nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
17     resnet_block(64, 64, 2, first_block=True),
18     resnet_block(64, 128, 2),
19     resnet_block(128, 256, 2),
20     resnet_block(256, 512, 2),
21     nn.AdaptiveAvgPool2d((1, 1)),
```

```
20 |         nn.Flatten(),
21 |         nn.Linear(512, 10)
22 |     )
```

训练执行流程介绍

- 1) 配置好相关环境，导入相关库
- 2) 加载数据集
 - a) 下载 CIFAR10 数据集
 - b) 利用 Pytorch 的 ‘DataLoader’ 加载数据集并配置好 Data Loader
- 3) 模型搭建
 - a) 定义 Residual Block 类
 - b) 定义 ResNet Block 类
 - c) 实例化 ResNet
- 4) 模型训练
 - a) 定义优化器以及损失函数
 - b) 加载上次训练的模型
 - c) 训练模型
 - d) 写入数据到 Tensorboard, 保存模型
- 5) 模型可视化分析
- 6) TensorBoard 结果分析

模型超参数设置

表 3: 模型超参数设置

超参数	数值
epoch	10
image size	28
batch size	64
learning rate	0.001
optimizer	Adam
loss function	CrossEntropyLoss

八、实验数据及结果分析:

实验数据

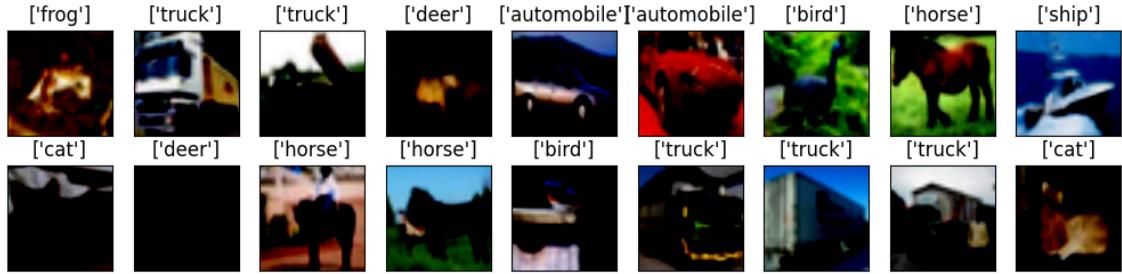


图 4: CIFAR10 数据集图片样本

本次选用的 CIFAR10 数据集包含 50000 张训练图片和 10000 张测试图片，每张图片的大小为 32×32 ，共有 10 个类别，分别为：airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck。

结果分析

受限于本地 GPU 性能，本次训练的模型只进行了 30 个 epoch 的训练（训练分为两次，由于 step 设置导致两次数据没有续上），最终的训练结果如图所示。

从图中可以看到，随着训练次数的增加，训练集的损失函数逐渐减小，训练集的正确率逐渐增加，测试集的损失函数先减少后增加，测试集的正确率逐渐增加，但是增加的幅度逐渐减小，当训练次数达到 30 次时，模型的性能已经达到了一个较高的水平，但是相比于论文中的结果，还有一定的差距。

本次实验中测试集的损失函数突然增加，可以认为该模型在训练后期出现了过拟合的现象，可以通过增加训练集的大小或者增加正则化项来解决该问题。

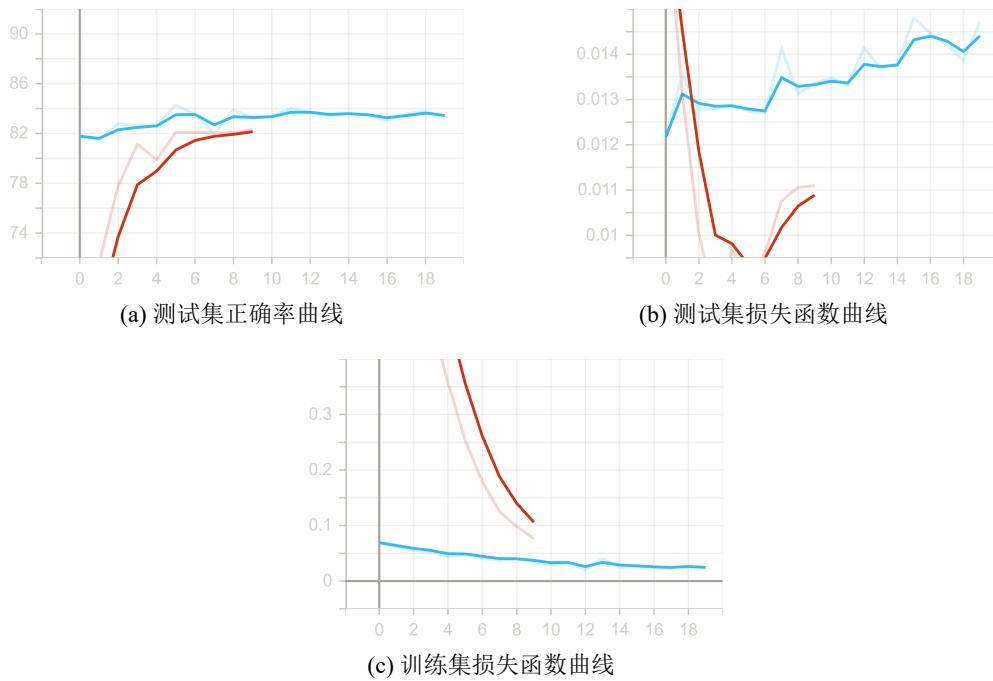
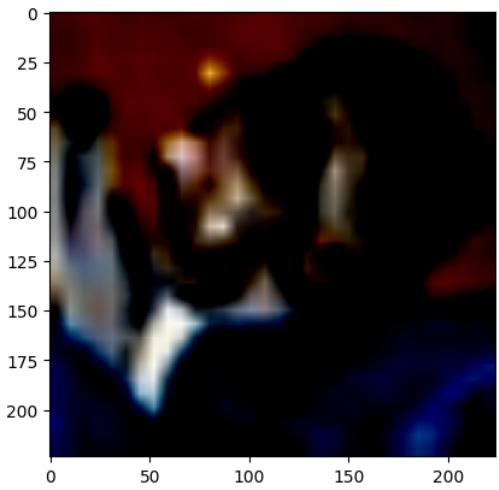


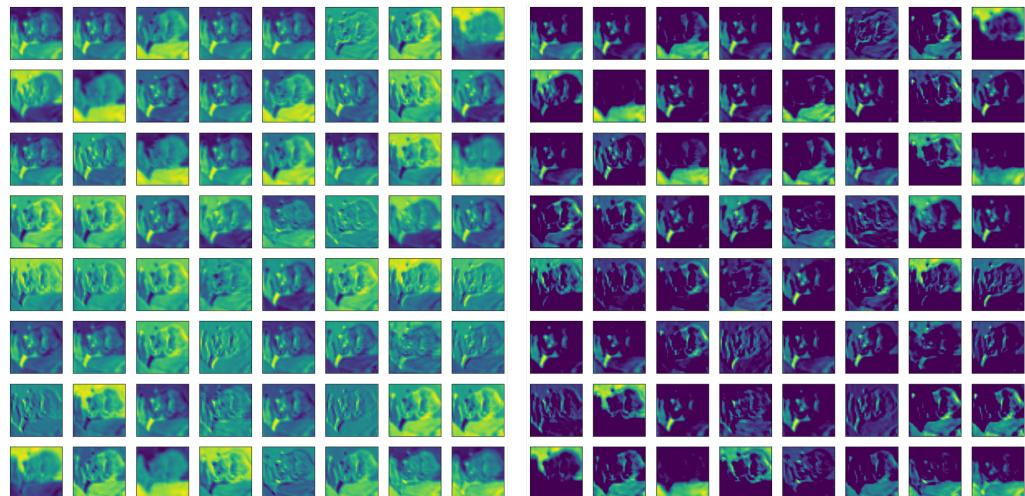
图 5: ResNet18 训练过程中的正确率和损失函数曲线

受限于篇幅限制，本次特征图可视化分析只展示前面四层的情况，从图中可以看出随着层数的增加，网络开始逐渐关注到图片的细节与整体，最后一层的特征图已经可以看出图片的大致轮廓。

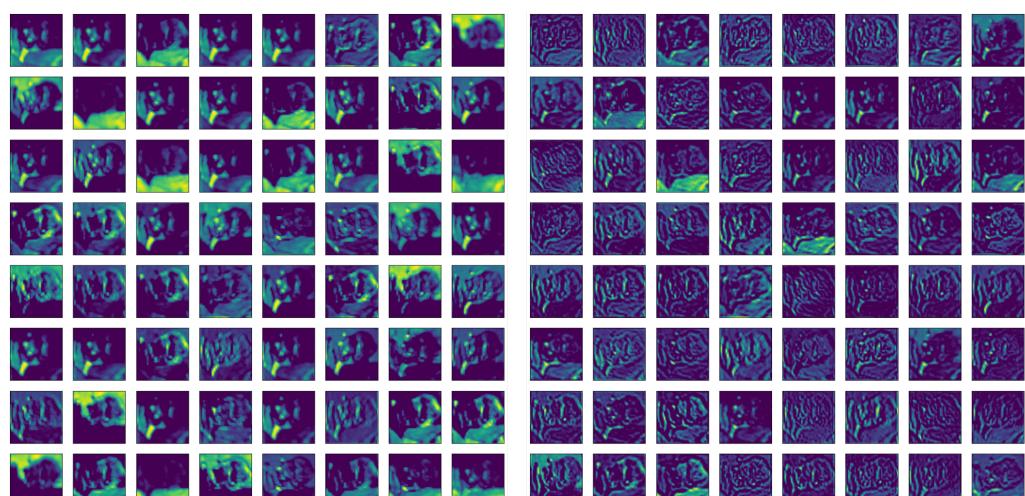
但是在笔记本中的更深层可视化，发现其中大部分的特征图都是全黑的，这可能是由于训练次数不够导致的，如果训练次数足够，可能会出现更多的特征图；也有可能是由于网络结构的问题，导致网络在训练过程中出现了梯度消失的现象，导致后面的特征图全为黑。



(a) 所选用的测试图片



(b) ResNet 第一层与第二层特征图可视化



(c) ResNet 第三层与第四层特征图可视化

图 6: ResNet18 训练过程中前面四层的特征图可视化

九、 实验结论：

本次实验复现了 ResNet18，最后利用可视化工具对实验结果进行可视化分析。通过对特征图的分析，可以加深对 ResNet 的理解。

十、 总结及心得体会：

在本次实验中，我们首先配置好相关的深度学习环境，然后利用 Pytorch 实现 ResNet18，最后利用可视化工具对实验结果进行可视化分析。

在本次实验中，我学习了 Pytorch 的基本数据操作，深度学习模型搭建的基本流程，以及如何利用 Pytorch 实现 ResNet18。同时，我也学会了如何利用 TensorBoard 对实验结果进行可视化分析。

十一、 对本实验过程及方法、手段的改进建议及展望：

本次实验（特别是扩展实验）对于实验的要求说明不够明确，扩展实验中提到复现 ResNet18，但是其它要求是否与原实验相同并未说明（由于 ResNet18 参数、层数原因，对其层进行特征图分析十分不方便）。

不少人在上该课程前已经接触过深度学习，在动手实践时往往倾向于选择扩展实验，但是本次 ResNet18 参数过大，训练时间过长，导致很多同学无法按照预期开展实验。所以能否将本次扩展实验的要求进行一定程度修改，比如将残差连接的层数减少，或者将 ResNet18 改为 ResNet10，将残差连接应用到经典 CNN 网络中等，这样可以减少训练时间，提高实验效率。

报告评分：

指导教师签字：

电子科技大学

实验报告

实验三

一、实验室名称:

电子科技大学清水河校区主楼 A2-413

二、实验项目名称:

人工智能实验 II-3：基于 RNN 的文本翻译

三、实验原理:

RNN 作为一种递归式处理序列问题的模型，在机器翻译、文本自动摘要和语音识别中有着成功的应用。巧妙的网络结构设计使得 RNN 可以捕捉语言中的长距离依赖关系，例如性别一致性和语法结构，而不必事先知道它们，也不需要跨语言进行 1:1 映射。

Seq2seq 是一类特殊的 RNN，它遵循了 Encoder-Decoder 的设计结构，两个部分均由 RNN 构成；Encoder 将源语句转换为表示语义的向量，然后这个向量通过 Decoder 可以产生对应的翻译结果。

四、实验目的:

通过 PyTorch 框架，搭建神经网络，首先实现基本的 RNN 模型，以及 RNN 的变体—GRU。使用 RNN/GRU 搭建 seq2seq 模型，包括 Encoder 和 Decoder 两个部分以进行文本翻译任务。在此过程中需掌握：

- 1) 基于 PyTorch 文本预处理(词表构建，词嵌入)的实现；
- 2) RNN 和 GRU 的基本框架的代码实现；
- 3) Seq2seq 的基本结构和代码实现。

扩展掌握：

- 1) 手动实现 GRU 模型
- 2) 【选做】基于 RNN 的图像描述生成 (image caption) 基本原理以及具体的实现方式。

五、实验内容：

基于 Pytorch 实现 seq2seq 文本翻译实验，主要实验内容具体包括四个部分：

- 1) 文本翻译模型的实现
- 2) 分析模型在对应数据集上的准确率
- 3) 分析使用 RNN 和 GRU 作为 seq2seq 模型的架构时分别对最终结果(准确率，损失函数收敛程度)有什么影响，并比较 RNN 和 GRU 的优缺点
- 4) 可视化、结果实验分析

六、实验器材（设备、元器件）：

- 1) 硬件平台：CPU: AMD Ryzen 7 5800U 1.90 GHz; GPU: NVIDIA GeForce RTX 3050 LAPTOP; 内存: 16GB@4266MHz
- 2) 开发环境: Ubuntu 20.04.2 LTS, Python 3.9.17, Pytorch 2.0.1+cu117
- 3) 测试环境: 同上

七、实验步骤：

问题描述

本次实验中需要构建一个基于 GRU 的 seq2seq 模型，用于实现文本翻译任务。需要解决的问题有：

- 1) 如何构建词表
- 2) 如何构建数据集
- 3) 如何构建模型
- 4) 如何训练模型
- 5) 如何评估模型
- 6) 如何可视化分析

算法的概要设计与分析

词表和数据集构建

- 1) 输入：训练数据集、测试数据集
- 2) 输出：原语言数据集、目标语言数据集、对应的配对键值对
- 3) 算法描述：输入源语言和目标语言，读取相关数据，构建对应的数据集。

模型搭建

- 1) 输入：词表大小、词向量维度、GRU 隐藏层大小、GRU 层数、GRU 层 dropout 概率
- 2) 输出：对应的 seq2seq 模型
- 3) 算法描述：使用 Class 定义一个 seq2seq 类，然后在类中定义 forward 函数，最后在 forward 函数中实现 seq2seq 的功能。

模型超参数设置

表 4: 模型超参数设置

超参数	数值
epoch	55000
batch size	64(10)
learning rate	0.01
optimizer	SGD
loss function	NLLLoss

模型训练与评估

- 1) 输入：训练数据集、测试数据集、seq2seq 模型、优化器、损失函数、训练次数
- 2) 输出：训练集的损失函数曲线、测试集的损失函数曲线、训练集的正确率曲线、测试集的正确率曲线
- 3) 算法描述：使用 Pytorch 的 DataLoader 加载数据集，然后使用 Pytorch 的优化器和损失函数对模型进行训练，最后使用 Tensorboard 对训练结果进行可视化分析。

核心算法的详细设计与实现

词表和数据集构建代码

代码 6: 词表代码

```
1 class Lang:  
2     def __init__(self, name):  
3         self.word2index = {"<SOS>": 0, "<EOS>": 1}  
4         self.index2word = {0: "<SOS>", 1: "<EOS>"}  
5         self.n_words = 2
```

```

6     self.name = name
7
8     def addSentence(self, sentence):
9         for word in sentence.split(' '):
10            self.addWord(word)
11
12    def addWord(self, word):
13        if word not in self.word2index:
14            self.word2index[word] = self.n_words
15            self.index2word[self.n_words] = word
16            self.n_words += 1
17        else:
18            pass

```

该代码首先定义‘SOS’和‘EOS’两个特殊的 token，然后定义了一个 Lang 类，该类中包含了词表的相关信息，包括词表大小、词表中每个词的索引、每个索引对应的词。最后定义了两个函数，分别用于向词表中添加句子和单词。

代码 7: 数据集构建代码

```

1  def prepareData(lang1, lang2, reverse=False):
2      input_lang, output_lang, pairs = readLangs(lang1, lang2, reverse)
3      print("Read %d sentence pairs" % len(pairs))
4      pairs = filterPairs(pairs)
5      print("Trimmed to %d sentence pairs" % len(pairs))
6      print("Counting words...")
7      for pair in pairs:
8          input_lang.addSentence(pair[0])
9          output_lang.addSentence(pair[1])
10     print("Counted words:")
11     print(input_lang.name, input_lang.n_words)
12     print(output_lang.name, output_lang.n_words)
13     return input_lang, output_lang, pairs

```

该代码首先定义了一个函数，用于读取数据集，然后定义了一个函数，用于过滤数据集，最后定义了一个函数，用于构建词表。

GRU 模型代码

代码 8: GRU 模型代码

```

1  class originalGRU(nn.Module):
2      def __init__(self, input_size=512, hidden_size=512):
3          super(originalGRU, self).__init__()
4          self.sigmoid = nn.Sigmoid()
5          self.tanh = nn.Tanh()
6          self.Wr = nn.Linear(input_size, hidden_size)
7          self.Wz = nn.Linear(input_size, hidden_size)
8          self.W = nn.Linear(input_size, hidden_size)
9          self.Wy = nn.Linear(hidden_size, input_size)
10
11     def forward(self, input, hidden):
12         r = self.sigmoid(self.Wr(input) + hidden)
13         z = self.sigmoid(self.Wz(input) + hidden)
14         h_hat = self.tanh(self.W(input) + r * hidden)
15         h = (1 - z) * h_hat + z * hidden
16         output = self.Wy(h)
17         return output, h
18
19     def initHidden(self):
20         return torch.zeros(1, 1, self.hidden_size, device=device)

```

该代码首先定义了一个 originalGRU 类，该类继承自 nn.Module 类，然后在类中定义了 forward 函数，最后在 forward 函数中实现了 GRU 的功能。

EncoderGRU 模型代码

代码 9: EncoderGRU 模型代码

```
1 class EncoderGRU(nn.Module):
2     def __init__(self, input_size, hidden_size, n_layers=1):
3         super(EncoderGRU, self).__init__()
4         self.hidden_size = hidden_size
5         self.n_layers = n_layers
6
7         self.embedding = nn.Embedding(input_size, hidden_size)
8         self.gru = originalGRU(hidden_size, hidden_size)
9
10    def forward(self, input, hidden):
11        embedded = self.embedding(input).view(1, 1, -1) # S=1 x B x N
12        output = embedded
13        for i in range(self.n_layers):
14            output, hidden = self.gru(output, hidden)
15        return output, hidden
16
17    def initHidden(self):
18        return torch.zeros(self.n_layers, 1, self.hidden_size, device=device)
```

该代码首先定义了一个 EncoderGRU 类，该类继承自 nn.Module 类，然后在类中定义了 forward 函数，最后在 forward 函数中实现了 EncoderGRU 的功能，同理 DecoderGRU 也类似定义。

八、实验数据及结果分析:

实验数据

```
input_lang, output_lang, pairs = prepareData('eng', 'fra', True)
print(random.choice(pairs))
✓ 13s

Reading lines...
Read 39365 sentence pairs
Trimmed to 2515 sentence pairs
Counting words...
Counted words:
fra 2465
eng 1860
['je ne suis pas sure de ce dont il s agissait mais ca a fait le bruit d une detonation .', 'i
```

图 7: eng-fra.txt 文件转换成数据集后的情况 (max-length=64)

本次读取的 eng-fra.txt 文件包含了 39365 条数据，其中每条数据包含了一句英文和一句法语，每句话的长度不一，最后构建出来的法语词表为 2465，英文词表为 1860。

结果分析

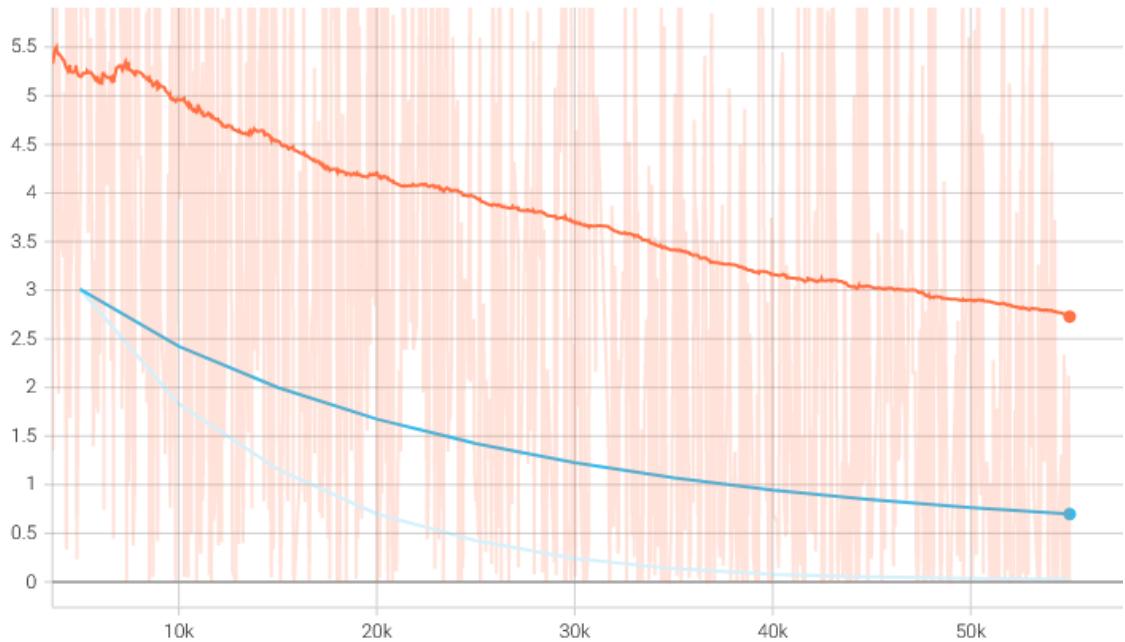


图 8: 训练过程中不同大小数据集下的损失函数曲线 (蓝色曲线:max-length=64, 橙色:max-length=10)

从图中可以看出，通过调整‘max-length’构建不同大小的数据集，然后其它设置保持不变情况下，训练过程中损失函数的收敛程度与最终的损失函数值都有所不同，但是总体上来说，损失函数的收敛程度与最终的损失函数值都与数据集的大小成正比，即数据集越大，损失函数的收敛程度越好，最终的损失函数值越小。

不同数据集训练出来的不同模型最终体现在翻译效果上。从图中可以看出，数据集越大，模型的翻译效果越好，但是数据集越大，模型的训练时间越长，所以需要在翻译效果和训练时间之间进行权衡。

```

> elle n a pas le moral aujourd hui .
= she is in low spirits today .
< she is today in her today . <EOS>

> tu es fatigued et moi aussi .
= you are tired and so am i .
< he s head at everything . <EOS>

> c est de notre faute a tous .
= we re all to blame for that .
< he s a book . <EOS>

> je le fais malgre toi .
= i m doing it in spite of you .
< i m doing it in spite of you . <EOS>

> elle est habituee a vivre seule .
= she is used to living alone .
< she is her husband in her husband . <EOS>

> il est plutot difficile a contenter .
= he is rather hard to please .
< he is rather hard to please . <EOS>

> vous n etes pas rationnelle .
= you re not being rational .
< you re not being rational . <EOS>

> je ne vous vends pas ma voiture .
= i m not selling you my car .
< i m not selling you my car . <EOS>

> son travail lui plait .
= he is pleased with his work .
< he is not what we . <EOS>

> c est un vieil ami a moi .
= he is an old friend of mine .
< he s head at work . <EOS>

```

(a) Max_length=10

```

1 > elle porte un manteau ample .
2 = she s wearing a loose coat .
3 < she s wearing a loose coat . <EOS>
4
5 > je me rejoins de voir que vous etudiez avec davantage d application que vous ne le fa
6 = i m glad to see that you re studying harder than you used to .
7 < i m glad to see that you re studying harder than you used to . <EOS>
8
9 > je me rends au supermarche pour effectuer quelques emplettes .
10 = i m going to the supermarket to do some shopping .
11 < i m going to the supermarket to do some shopping . <EOS>
12
13 > nous sommes confrontes a un probleme bien plus grave .
14 = we re facing a much bigger problem than that .
15 < we re facing a much bigger problem than that . <EOS>
16
17 > je nourris mon bebe au sein .
18 = i m breast feeding my baby .
19 < i m breast feeding my baby . <EOS>
20
21 > tu es la femme la plus belle au monde .
22 = you re the most beautiful woman in the whole world .
23 < you re the most beautiful woman in the world . <EOS>
24
25 > je creve d envie de la revoir .
26 = i am dying to see her again .
27 < i am dying to see her again . <EOS>
28
29 > il t attend chez nous .
30 = he s waiting for you at home .
31 < he s waiting for you at home . <EOS>
32
33 > j attends avec impatience d aller chasser avec mon pere .
34 = i m looking forward to going hunting with my father . <EOS>
35 < i m looking forward to going hunting with my father . <EOS>
36
37 > elle est tres receptive a la suggestion hypnotique .
38 = she s very susceptible to hypnotic suggestion .
39 < she s very susceptible to hypnotic suggestion . <EOS>
40

```

(b) Max_length=64

图 9: 训练过程中不同大小数据集下的测试结果情况

九、实验结论：

本次实验使用了 GRU 实现 Seq2seq 模型，在 eng-fra 互译任务中进行测试，取得了预期之中的结果。

十、总结及心得体会：

在本次实验中，我们首先配置好相关的深度学习环境，然后利用 Pytorch 实现了 GRU，最后利用 GRU 实现了 Seq2seq 模型，用于实现文本翻译任务。

在本次实验中，我学习了 Pytorch 的基本数据操作，深度学习模型搭建的基本流程，以及如何利用 Pytorch 实现 GRU。同时，我也学会了如何利用 TensorBoard 对实验结果进行可视化分析。

十一、对本实验过程及方法、手段的改进建议及展望：

本次实验（特别是扩展实验）对于实验的要求说明不够明确，扩展实验中提到构建 GRU，但是基础实验要求里却又提到将 RNN 和 GRU 进行对比；

实验结果分析中，要求量化数据集大小对于损失函数的影响以及对最终文本翻译的性能影响过于模糊。首先如何量化，其次如何评估性能，最后如何分析数据集大小对于性能的影响，这些都没有明确的说明。

再者，对于本次 GRU 来说，每次在本地 GPU 上进行测试都需要花费较多时间，重复测试的过程较为繁琐，能否在实验报告中提供一些测试结果，方便同学们进行对比分析。

提供的笔记本代码内容过于紊乱，缺乏条理，难以阅读理解。能否对其进行一定的整理，方便同学们进行学习。

报告评分：

指导教师签字：

电子科技大学

实验报告

实验四

一、实验室名称:

电子科技大学清水河校区主楼 A2-413

二、实验项目名称:

人工智能实验 II-4：基于自动编码器的手写数字生成

三、实验原理:

自动编码器是无监督学习方法中的一种结构。它通过编码器提取输入数据的隐含特征，再通过解码器根据隐含特征重构输入，实现了一种自监督学习的方法。其中，编码器和解码器通常为一个非线性映射函数，隐含特征往往比输入数据更加紧凑。因此，自动编码器能够实现数据降维、压缩和数据隐含分布的投影。随着深度神经网络的广泛应用，人们尝试使用神经网络搭建编码器和解码器，并取得了广泛成功。本次实验基于神经网络搭建自动编码器，以探究其在隐变量映射和图像降维与还原任务中体现出的各种特性。

四、实验目的:

通过使用 PyTorch 深度学习框架，搭建神经网络，实现自动编码器以进行手写数字生成等任务，并在此过程中掌握：

- 1) PyTorch 中全连接网络、卷积网络等的实现方法；
- 2) 自动编码器的基本框架和设计思路；
- 3) 探究自动编码器在图像降维和还原中体现的具体功能。

扩展掌握：

- 1) 自动编码器的隐变量可视化分析；
- 2) 变分自编码器的实现方式；
- 3) 对抗生成网络的实现方式。

五、实验内容：

本次实验为基于 Pytorch 实现自动编码器用于手写数字生成等任务，主要实验内容具体包括四个部分：

- 1) 文本翻译模型的实现
- 2) 分析模型在对应数据集上的准确率
- 3) 分析使用 RNN 和 GRU 作为 seq2seq 模型的架构时分别对最终结果(准确率，损失函数收敛程度)有什么影响，并比较 RNN 和 GRU 的优缺点
- 4) 可视化、结果实验分析

六、实验器材（设备、元器件）：

- 1) 硬件平台：CPU: AMD Ryzen 7 5800U 1.90 GHz; GPU: NVIDIA GeForce RTX 3050 LAPTOP; 内存: 16GB@4266MHz
- 2) 开发环境: Ubuntu 20.04.2 LTS, Python 3.9.17, Pytorch 2.0.1+cu117
- 3) 测试环境: 同上

七、实验步骤：

问题描述

本次实验中需要构建生成模型，用于实现手写数字生成等任务。需要解决的问题有：

- 1) 如何构建数据集
- 2) 如何构建模型
- 3) 如何训练模型
- 4) 如何评估模型
- 5) 如何可视化分析

算法的概要设计与分析

数据集构建

- 1) 输入：训练数据集、测试数据集的对应文件目录
- 2) 输出：对应的 Dataloader
- 3) 算法描述：输入数据集文件地址，读取相关数据，构建对应的数据集，并返回相应的 data loader.

模型搭建

- 自编码器模型

- 1) 输入：输入数据维度、隐变量维度、编码器层数、解码器层数
- 2) 输出：对应的自编码器模型

- VAE 模型

- 1) 输入：输入数据维度、隐变量维度、编码器层数、解码器层数
- 2) 输出：对应的 VAE 模型

- WGAN 模型

- 1) 输入：输入数据维度、隐变量维度、编码器层数、解码器层数
- 2) 输出：对应的 WGAN 模型

模型训练与评估

- 1) 输入：模型、data loader、优化器、损失函数、训练 epoch
- 2) 输出：训练集的损失函数曲线、测试集的损失函数曲线、训练集的正确率曲线、测试集的正确率曲线等
- 3) 算法描述：使用 Pytorch 的 DataLoader 加载数据集，然后使用 Pytorch 的优化器和损失函数对模型进行训练，最后使用 Tensorboard 对训练结果进行可视化分析。

模型超参数设置

表 5: 自编解码器模型超参数设置

超参数	数值
epoch	20
batch size	64
learning rate	5e-3
optimizer	Adam
loss function	MSELoss

核心算法的详细设计与实现

自编解码器网络设计

表 6: VAE 模型超参数设置

超参数	数值
epoch	40
batch size	64
learning rate	0.01
optimizer	Adam
loss function	MSE LOSS + KLD

表 7: WGAN 模型超参数设置

超参数	数值
epoch	100
batch size	64
learning rate	0.002
optimizer	Adam

代码 10: 自编解码器构建代码

```

1 class Autoencoder(nn.Module):
2     def __init__(self):
3         super(Autoencoder, self).__init__()
4         # 编码器
5         self.encoder = nn.Sequential(
6             nn.Conv2d(3, 16, 3, stride=2, padding=1), # Update input
7             # channel to 3
8             nn.ReLU(),
9             nn.Conv2d(16, 32, 3, stride=2, padding=1),
10            nn.ReLU(),
11            nn.Conv2d(32, 64, 7),
12            # nn.ReLU(),
13            # nn.Conv2d(64, 128, 3, stride=2, padding=1)
14        )
15        # 解码器
16        self.decoder = nn.Sequential(
17            # nn.ConvTranspose2d(128, 64, 3, stride=2, padding=1,
18            # output_padding=1),
19            # nn.ReLU(),
20            nn.ConvTranspose2d(64, 32, 7),
21            nn.ReLU(),
22            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1,
23            # output_padding=1),
24            nn.ReLU(),
25            nn.ConvTranspose2d(16, 3, 3, stride=2, padding=1,
26            # output_padding=1),
27            nn.Sigmoid()
28        )
29
30     def forward(self, x):
31         x = self.encoder(x)
32         x = self.decoder(x)
33         return x

```

代码 11: 自编解码器网络结构

```

1 Test Input Shape: torch.Size([1, 3, 28, 28])
2 Test Output Shape: torch.Size([1, 3, 28, 28])
3 -----
4 Layer (type)          Output Shape         Param #
5 =====
6 Conv2d-1            [-1, 16, 14, 14]      448
7 ReLU-2              [-1, 16, 14, 14]      0
8 Conv2d-3            [-1, 32, 7, 7]       4,640
9 ReLU-4              [-1, 32, 7, 7]       0
10 Conv2d-5           [-1, 64, 1, 1]      100,416
11 ConvTranspose2d-6   [-1, 32, 7, 7]      100,384
12 ReLU-7              [-1, 32, 7, 7]       0
13 ConvTranspose2d-8   [-1, 16, 14, 14]      4,624
14 ReLU-9              [-1, 16, 14, 14]      0
15 ConvTranspose2d-10  [-1, 3, 28, 28]      435
16 Sigmoid-11          [-1, 3, 28, 28]      0
17 -----
18 Total params: 210,947
19 Trainable params: 210,947
20 Non-trainable params: 0
21 -----
22 Input size (MB): 0.01
23 Forward/backward pass size (MB): 0.18
24 Params size (MB): 0.80
25 Estimated Total Size (MB): 0.99
26 -----

```

该网络结构首先使用了三个卷积层，然后使用了三个反卷积层，最后使用了一个 Sigmoid 层。

VAE 模型代码

代码 12: VAE 模型代码

```
1 class VAE(nn.Module):
2     def __init__(self):
3         super(VAE, self).__init__()
4         self._hidden_dim = 128
5         self._latent_dim = 128
6         # 编码器
7         self.encoder = nn.Sequential(
8             nn.Conv2d(3, 16, 3, stride=2, padding=1),    # Update input channel
9                 to 3
10                # [1,3,28,28] -> [1,16,14,14]
11                nn.BatchNorm2d(16),
12                nn.ReLU(),
13                nn.Conv2d(16, 32, 3, stride=2, padding=1),
14                # [1,16,14,14] -> [1,32,7,7]
15                nn.BatchNorm2d(32),
16                nn.ReLU(),
17                nn.Conv2d(32, 64, 7, stride=2, padding=1),
18                # [1,32,7,7] -> [1,64,2,2]
19                nn.BatchNorm2d(64),
20                nn.ReLU(),
21                nn.Conv2d(64, self._hidden_dim, 3, stride=2, padding=1),
22                nn.BatchNorm2d(self._hidden_dim),
23                # [1,64,2,2] -> [1,128,1,1]
24            )
25         # 解码器
26         self.decoder = nn.Sequential(
27             nn.ConvTranspose2d(self._hidden_dim, 64, 3, stride=2, padding=1,
28                 output_padding=1),
29             nn.BatchNorm2d(64),
30             nn.ConvTranspose2d(64, 32, 7, stride=2, padding=1),
```

```

29         nn.BatchNorm2d(32),
30         nn.ReLU(),
31         nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding
32             =1),
33         nn.BatchNorm2d(16),
34         nn.ReLU(),
35         nn.ConvTranspose2d(16, 3, 3, stride=2, padding=1, output_padding
36             =1),
37         nn.Sigmoid()
38     )
39
# 均值
40     self.fc_mu = nn.Linear(self._hidden_dim, self._latent_dim)
# 方差
41     self.fc_var = nn.Linear(self._hidden_dim, self._latent_dim)
42
43     def encode(self, x):
44         x = self.encoder(x)
45         x = torch.flatten(x, start_dim=1)
46         mu = self.fc_mu(x)
47         log_var = self.fc_var(x)
48         return mu, log_var
49         # [1, _latent_dim], [1, _latent_dim]
50
51     def decode(self, z):
52         z = z.view(-1, self._hidden_dim, 1, 1) # Fix the input size of the
53             decoder
54         x = self.decoder(z)
55         return x
56
57     def reparameterize(self, mu, log_var):
58         std = torch.exp(0.5 * log_var)
59         eps = torch.randn_like(std)
60         z = mu + eps * std
61         return z
62         # [1, _latent_dim]
63
64     def forward(self, x):
65         mu, log_var = self.encode(x)
66         z = self.reparameterize(mu, log_var)
67         x = self.decode(z)
68         return x, mu, log_var
69
70     def sample(self, num_samples=1):
71         z = torch.randn(num_samples, self._latent_dim).to(DEVICE)
72         samples = self.decode(z)
73         return samples

```

代码 13: VAE 模型网络结构

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 14, 14]	448
BatchNorm2d-2	[-1, 16, 14, 14]	32
ReLU-3	[-1, 16, 14, 14]	0
Conv2d-4	[-1, 32, 7, 7]	4,640
BatchNorm2d-5	[-1, 32, 7, 7]	64
ReLU-6	[-1, 32, 7, 7]	0
Conv2d-7	[-1, 64, 2, 2]	100,416
BatchNorm2d-8	[-1, 64, 2, 2]	128
ReLU-9	[-1, 64, 2, 2]	0
Conv2d-10	[-1, 128, 1, 1]	73,856
BatchNorm2d-11	[-1, 128, 1, 1]	256
Linear-12	[-1, 128]	16,512
Linear-13	[-1, 128]	16,512
ConvTranspose2d-14	[-1, 64, 2, 2]	73,792

```

19      BatchNorm2d-15           [-1, 64, 2, 2]          128
20      ConvTranspose2d-16        [-1, 32, 7, 7]         100,384
21      BatchNorm2d-17           [-1, 32, 7, 7]          64
22      ReLU-18                 [-1, 32, 7, 7]          0
23      ConvTranspose2d-19        [-1, 16, 14, 14]        4,624
24      BatchNorm2d-20           [-1, 16, 14, 14]        32
25      ReLU-21                 [-1, 16, 14, 14]        0
26      ConvTranspose2d-22        [-1, 3, 28, 28]         435
27      Sigmoid-23              [-1, 3, 28, 28]         0
28 =====
29 Total params: 392,323
30 Trainable params: 392,323
31 Non-trainable params: 0
32 -----
33 Input size (MB): 0.01
34 Forward/backward pass size (MB): 0.26
35 Params size (MB): 1.50
36 Estimated Total Size (MB): 1.77
37 -----
38 Test Input Shape: torch.Size([128, 3, 28, 28])
39 Test Output Shape: torch.Size([128, 3, 28, 28])
40 Test Mu Shape: torch.Size([128, 128])
41 Test Log Var Shape: torch.Size([128, 128])

```

该代码首先定义了一个 VAE 类，该类继承自 nn.Module 类，然后在类中定义了 forward 函数，最后在 forward 函数中实现了 VAE 的功能。

WGAN 模型

代码 14: WGAN 模型-生成器部分

```

1 -----
2      Layer (type)           Output Shape        Param #
3 =====
4
5      ConvTranspose2d-1       [-1, 256, 3, 3]     230,656
6      BatchNorm2d-2          [-1, 256, 3, 3]     512
7      ReLU-3                 [-1, 256, 3, 3]     0
8      ConvTranspose2d-4        [-1, 128, 6, 6]    524,416
9      BatchNorm2d-5          [-1, 128, 6, 6]    256
10     ReLU-6                 [-1, 128, 6, 6]     0
11     ConvTranspose2d-7        [-1, 64, 13, 13]   73,792
12     BatchNorm2d-8          [-1, 64, 13, 13]   128
13     ReLU-9                 [-1, 64, 13, 13]     0
14     ConvTranspose2d-10       [-1, 1, 28, 28]    1,025
15     Tanh-11                [-1, 1, 28, 28]     0
16 =====
17 Total params: 830,785
18 Trainable params: 830,785
19 Non-trainable params: 0
20 -----
21 Input size (MB): 0.00
22 Forward/backward pass size (MB): 0.42
23 Params size (MB): 3.17
24 Estimated Total Size (MB): 3.59
25 -----

```

该模型首先使用了三个反卷积层，然后使用了一个 Tanh 层，作为一个简单的生成器。

代码 15: WGAN 模型-评分器部分

```

1 |      Layer (type)          Output Shape       Param #
2 | -----
3 | -----
4 |       Conv2d-1            [-1, 64, 13, 13]     1,088
5 |       BatchNorm2d-2        [-1, 64, 13, 13]     128
6 |       LeakyReLU-3         [-1, 64, 13, 13]      0
7 |       Conv2d-4            [-1, 128, 5, 5]    131,200
8 |       BatchNorm2d-5        [-1, 128, 5, 5]    256
9 |       LeakyReLU-6         [-1, 128, 5, 5]      0
10 |      Conv2d-7             [-1, 1, 1, 1]       2,049
11 | -----
12 | Total params: 134,721
13 | Trainable params: 134,721
14 | Non-trainable params: 0
15 | -----
16 | Input size (MB): 0.00
17 | Forward/backward pass size (MB): 0.32
18 | Params size (MB): 0.51
19 | Estimated Total Size (MB): 0.84
20 |

```

该模型首先使用了两个卷积层，然后使用了一个 LeakyReLU 层，作为一个简单的评分器。

最后定义好损失函数和优化器，然后进行训练。

八、实验数据及结果分析：

实验数据

本次实验使用 CIFAR-10、MNIST、Fashion-MNIST 作为自己的数据集，构建好对应的 data loader 后，对部分图片进行可视化展示如图所示。

结果分析

Autoencoder 模型

从图中可以看出，随着迭代次数的增加，测试集和训练集的损失函数都在不断的减小，说明模型在不断的收敛。

从图中可以看出，模型生成的图片与输入图片相比，有一定的相似性，但是模型生成的图片与输入图片相比，有一定的模糊性，说明模型还有一定的提升空间。

VAE 模型

从图中可以看出，随着迭代次数的增加，测试集和训练集的损失函数都在不断的减小，说明模型在不断的收敛。

从图中可以看出，模型生成的图片与输入图片相比，有一定的相似性，但是模型生成的图片与输入图片相比，有一定的模糊性，说明模型还有一定的提升空间。此外，VAE 模型通过随机正态分布生成的图片也具有良好的质量。

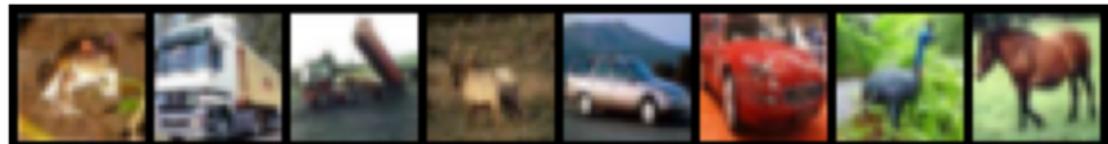


图 10: 本次实验所使用的三个数据集: CIFAR-10、MNIST、Fashion-MNIST

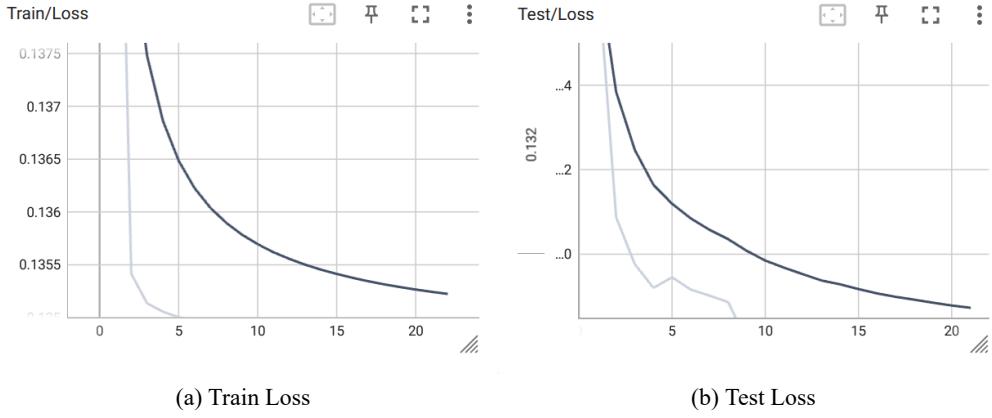


图 11: Auto Encoder 模型在 CIFAR10 训练集中的损失函数曲线

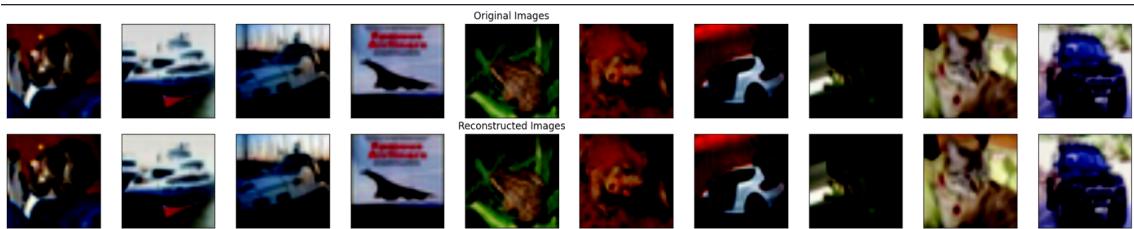


图 12: autoencoder 模型生成的图片 (上: 输入; 下: 输出)

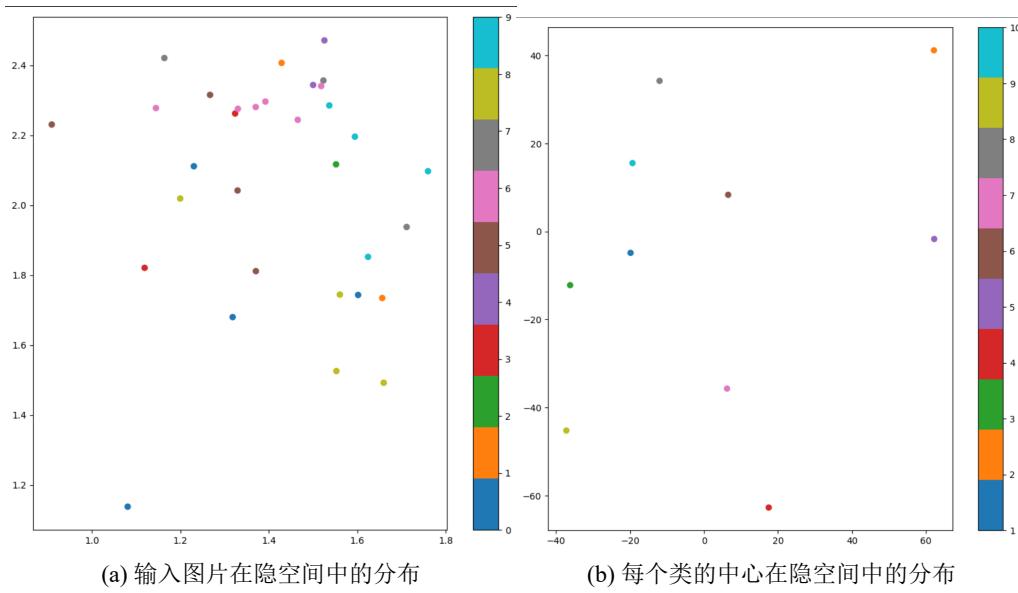


图 13: Auto Encoder 模型对于 CIFAR10 不同类别图片在隐空间中的分布

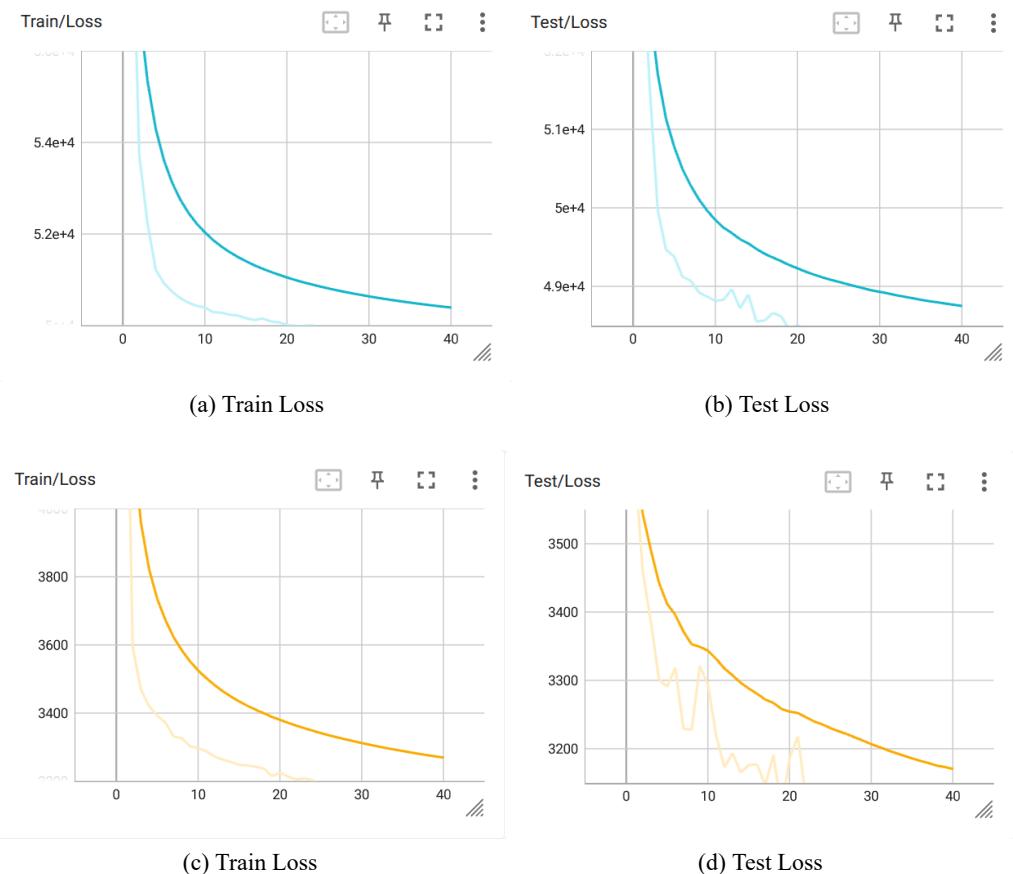
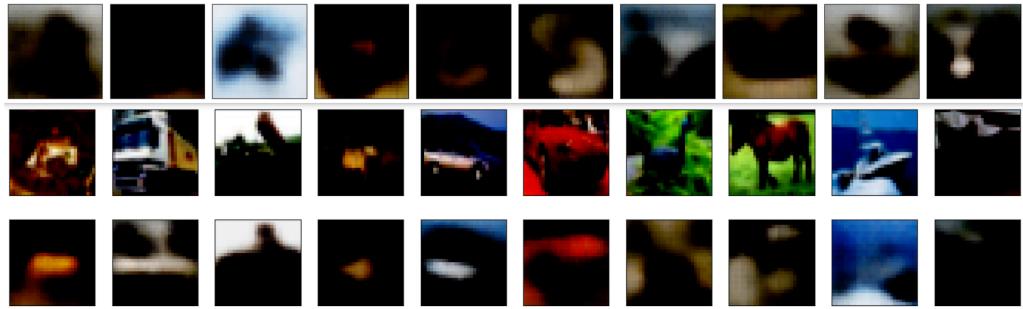
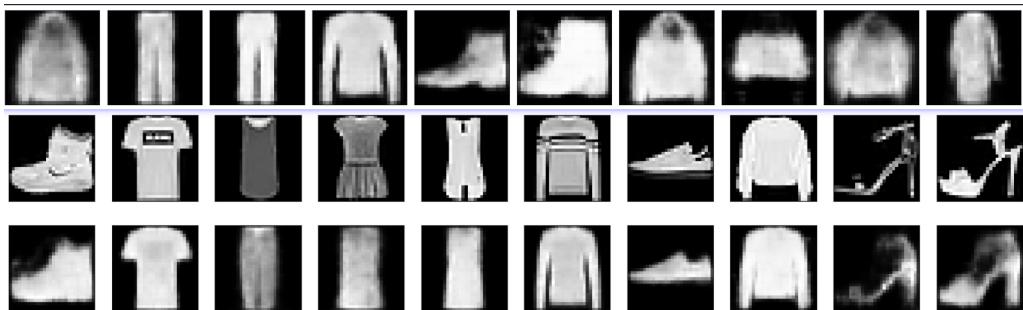


图 14: VAE 模型在 CIFAR10 (上) 以及 FASHION-MNIST (下) 数据集上的训练过程



(a) VAE 模型生成的 CIFAR10 数据集图片



(b) VAE 模型生成的 FASHION-MNIST 数据集图片

图 15: VAE 模型生成的图片（上：随机正态分布生成的图片；中：输入的图片；下：对应输出的图片）

WGAN 模型

从图中可以看出，随着迭代次数的增加，WGAN 模型在 MNIST 数据集上一方面生成器的损失函数不断增大；另一方面判断器的损失函数在不断减少，说明模型在不断的收敛。但是在 FASHION-MNIST 数据集上，WGAN 模型在训练过程中，生成器的损失函数不断减小，判断器的损失函数也在不断减少，说明模型没有收敛。这一现象也从实际生成的效果中可以展示出来。

从图中可以看出，模型生成的图片与输入图片相比，训练初期的模型生成的图片质量较好，模型训练到了后期出现过拟合甚至崩溃的现象，说明模型还有一定的提升空间。

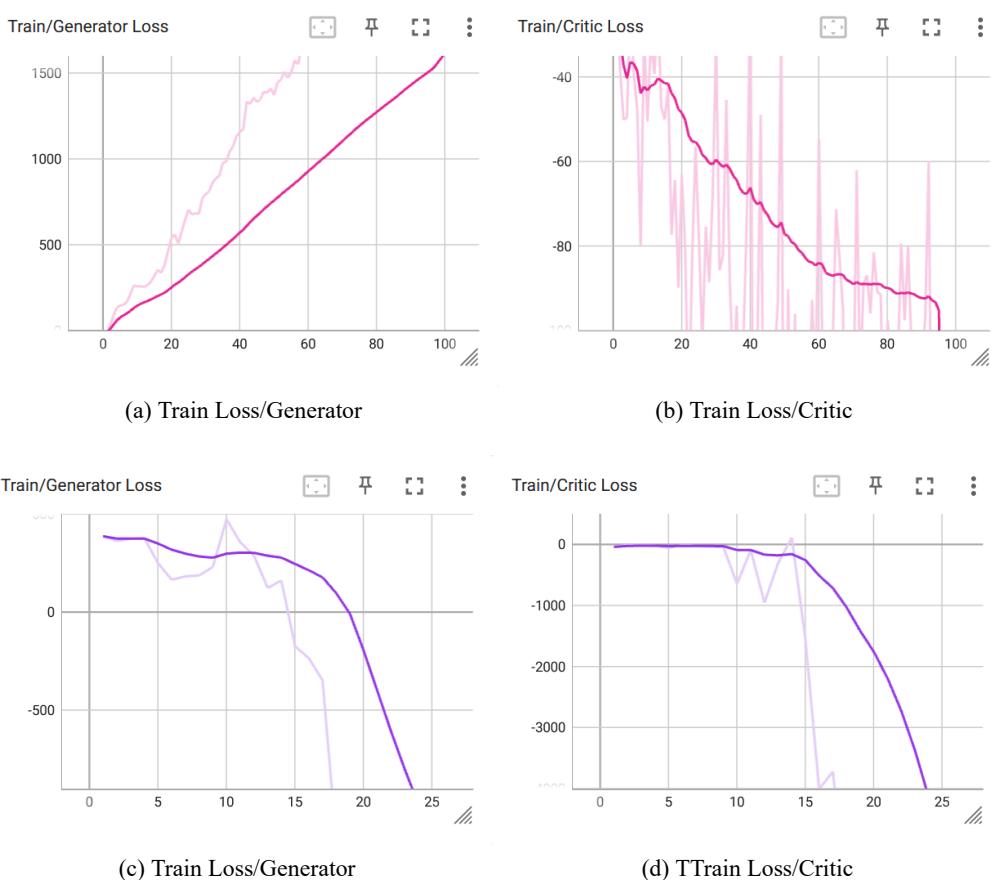
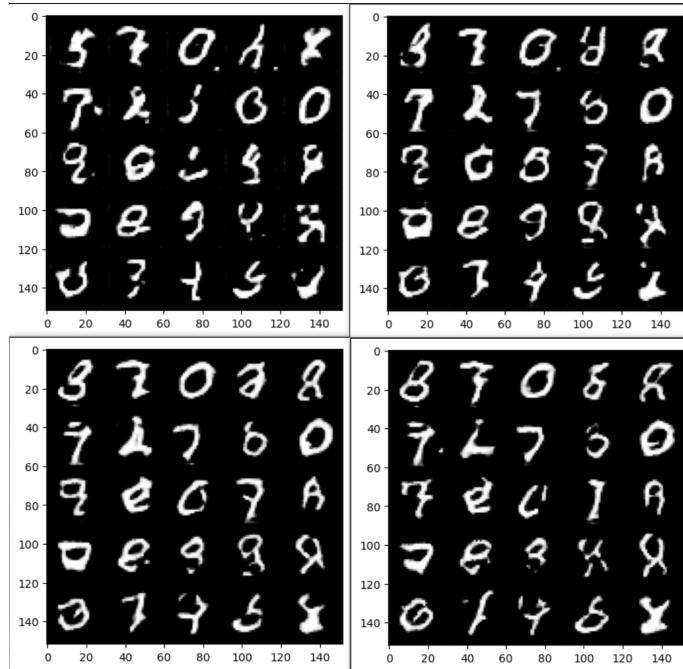
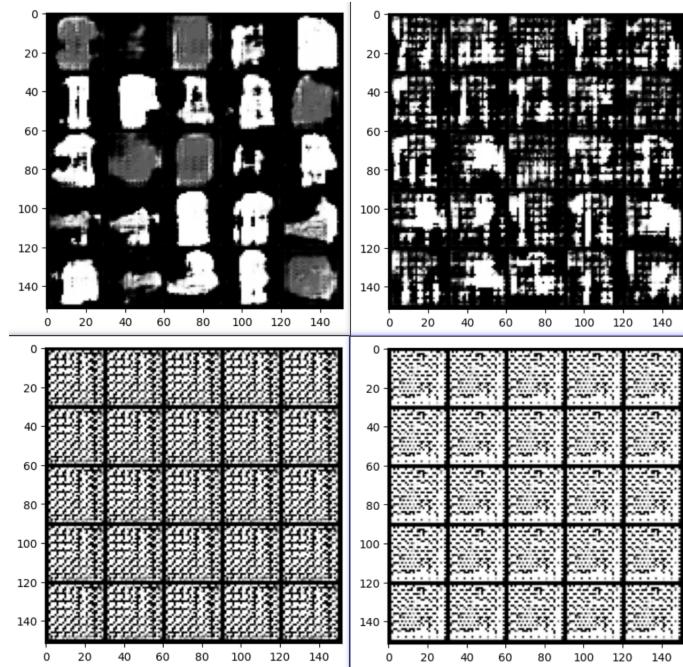


图 16: WGAN 模型在 MNIST (上) 以及 FASHION-MNIST (下) 数据集上的训练过程



(a) WGAN 模型生成的 MNIST 数据集图片 (每 10 个 epoch 测试一次)



(b) WGAN 模型生成的 FASHION-MNIST 数据集图片 (每 5 个 epoch 测试一次)

图 17: WGAN 模型生成的图片 (从左上到右下依次为训练过程中的测试效果)

九、实验结论：

本次实验在 CIFAR10、MNIST、FASHION-MNIST 这三个数据集中测试了三种类型的视觉生成模型：Auto Encoder、VAE、WGAN。

从实验结果中可以看出，VAE 模型在生成图片的质量上要优于 Auto Encoder 模型，但是 WGAN 模型在生成图片的质量上要差于 Auto Encoder 模型。这一现象可能是因为 WGAN 模型在训练过程中出现了过拟合的现象，导致模型的泛化能力下降，从而导致模型生成的图片质量下降。

十、总结及心得体会：

在本次实验中，上手并学习了经典的视觉生成模型，掌握了相关的算法以及模型复现。

在本次实验中，我对于 Pytorch 的使用更加熟练，对于 Pytorch 的相关 API 也有了更加深入的理解。

在本次实验中，我对于视觉生成模型有了更加深入的理解，对于视觉生成模型的优缺点也有了更加深入的理解。

十一、对本实验过程及方法、手段的改进建议及展望：

本次实验中对于数据集的选取比较简单与模糊，实验题目中说使用 MNIST，但是实际作业需要做 CIFAR10。建议要么指定一个数据集，要么直接让同学自由选择数据集。

本次实验使用的 ipynb 文件缺乏条理，适合教学但是不适合个人部署学习，且强制要求使用该笔记本开展实验缺乏灵活性。每个人代码风格都不一样，建议让同学们选择使用模板或者自行完成实验。

报告评分：

指导教师签字：