华中科技大学

计算机科学与技术学院

《计算机视觉导论》实验报告

基于卷积神经网络的两个数字比较

专　　业:	计算机科学与技术（图灵班）
班　　级:	图灵 2301 班
学　　号:	U202314607
姓　　名:	向恩泽
成　　绩:	
指导教师:	刘　康

完成日期：2025 年 10 月 29 日

目录

I: 任务要求

任务要求：

设计一个卷积神经网络，输入为两张MNIST手写体数字图片，如果两张图片为同一个数字（注意，非同一张图片），输出为1，否则为0。

从MNIST数据集的训练集中选取10%作为本实验的训练图片，从MNIST数据集的测试集中选取10%作为本实验的测试图片。请将该部分图片经过适当处理形成一定数量的用于本次实验的训练集和测试集。

注意事项：

1.深度学习框架任选。

2.实验报告需包含训练集和测试集构成、神经网络架构、每一轮mini-batch训练后的模型在训练集和测试集上的损失、最终的训练集和测试集准确率，以及对应的实验分析。

II: 环境介绍

本次实验使用Python 3.10版本进行开发，并使用Pytorch 2.8 + cu128作为深度学习框架。具体环境信息如下：

环境名	具体信息
CPU	AMD Ryzen 9 9955HX3D 16-Core Processor*
GPU	NVIDIA GeForce RTX 5070ti Laptop 12G
DRAM	32 GB(16GB x2) DDR5 5200MT/s
操作系统	Windows 11 + WSL2 Ubuntu 22.04 LTS
Python 版本	Python 3.10.12
Pytorch 版本	Pytorch 2.8 + cu128

* AMD Ryzen 9 9955HX3D 默认包含两个 CCD（CCD0 和 CCD1），每个 CCD 提供 8 核心（16 线程），总计 16 核心（32 线程）。由于环境设置，CCD1 已被禁用，仅 CCD0 在工作。CCD0 几乎等效于 Ryzen 7 9800X3D，提供 8 核心（16 线程）和 96 MB 的 3D V-Cache。

III: 数据集简介与预处理

MNIST 数据集是一个经典的手写数字识别数据集，包含 60000 张训练图片和 10000 张测试图片，每张图片为 28x28 像素的灰度图像，表示手写的数字（0-9）。本次实验中，我根据实验要求在 MNIST 数据集中，分别于训练集和测试集中各随机选取其中 10% 的数据，构建了本次实验所需的训练集和测试集。共含有 6000 张训练图片（每类数字 600 张）和 1000 张测试图片（每类数字 100 张）。

当然，由于图片原始是 $[0, 255]$ 的灰度值范围，因此我对图片进行了归一化处理，将像素值缩放到 $[0, 1]$ 范围内，以便于神经网络的训练。具体的预处理可以参考以下代码：

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,)),
])
```

选择出原始的训练图片和测试图片之后，为了匹配本次实验任务，我选择以下逻辑构建训练集，而测试集同理：

1. 正样本：从每个类别数字中，进行**完全的**两两匹配，每一对都会构成一个正样本对，标签为二维向量 $[0, 1]$ 表示不同的概率为 0，相同的概率为 1；这样构成的正样本对理论上有 $\frac{600 \times 599}{2} \times 10 = 1797000$ 对（具体实验可能因为奇怪取整稍有偏差）。

```
# for label == 1
for i in range(10):
    buc_size = len(selected_data[i])
    for j in range(buc_size):
        for k in range(j + 1, buc_size):
            img1 = selected_data[i][j]
            img2 = selected_data[i][k]
            label = torch.tensor([0, 1], device=device, dtype=dtype)

            self.data.append((img1, img2))
            self.labels.append(label)
            label_counter_1 += 1
```

2. 负样本：如果按照正样本进行完全组合构造负样本，会导致负样本数量远大于正样本，从而导致数据集严重不平衡，影响模型训练效果。因此，我使用类似下采样的思想，在所有负样本对中随机选取与正样本数相等的负样本对。具体选择方式为：随机选择两个不同类别的数字，然后从各自类别中随机选择一张图片，构成一个负样本对，标签为二维向量 $[1, 0]$ 表示不同的概率为 1，相同的概率为 0；

```
random.seed(seed)
for _ in range(label_counter_1):
    label_i, label_j = random.sample(range(10), 2)
    assert label_i != label_j, "label_i and label_j must be different"
    buc_size_i = len(selected_data[label_i])
    buc_size_j = len(selected_data[label_j])
    idx_i = random.randint(0, buc_size_i - 1)
    idx_j = random.randint(0, buc_size_j - 1)

    img1 = selected_data[label_i][idx_i]
    img2 = selected_data[label_j][idx_j]
    label = torch.tensor([1, 0], device=device, dtype=dtype)

    self.data.append((img1, img2))
    self.labels.append(label)
```

IV: 模型设计

由于本次实验任务与上一次不同，这一次的目的是区分两张图片是否为同一个数字，因此我设计了一种基于孪生网络（Siamese Network）结构的深度神经网络，用于对输入样本对之间的特征相似度进行建模与判别。模型整体由三个主要模块组成：Embedding 模块、ResidualBlock 残差模块以及 Siamese 主干网络，最终由一个分类头部输出两类判别结果。具体的网络结构可以在提交文件的 `./models.py` 中查看代码实现。需要说明的是，残差模块并不是独立的网络层，而是作为一个组件存在于 Siamese 主干网络中。

网络的具体实例化可以在 `./main.py` 中查看 `model = Net(...)`，实例化时可以指定网络的嵌入通道维度以及在孪生网络中的隐藏通道维度。接下来具体说明各个模块的设计思路：

IV.1: Embedding 模块

```python
class Embedding(nn.Module):
    def __init__(self, in_channels: int=1, out_channels: int=8):
        super().__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.model = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.Dropout(0.2),
        )
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        ...  #More code details in ./model.py
```

Embedding 模块的作用是将原始输入的单通道归一化后的图像映射到一个较高通道维度的特征空间，以提取底层特征。该实验中，其主要由一个卷积层、一个批归一化层和一个 Dropout 层组成。卷积层用于提取输入图片的初步特征，批归一化层用于加速训练过程并提高模型的稳定性，Dropout 层用于防止过拟合现象。

IV.2: Siamese 主干网络

IV.2.a: ResidualBlock 模块

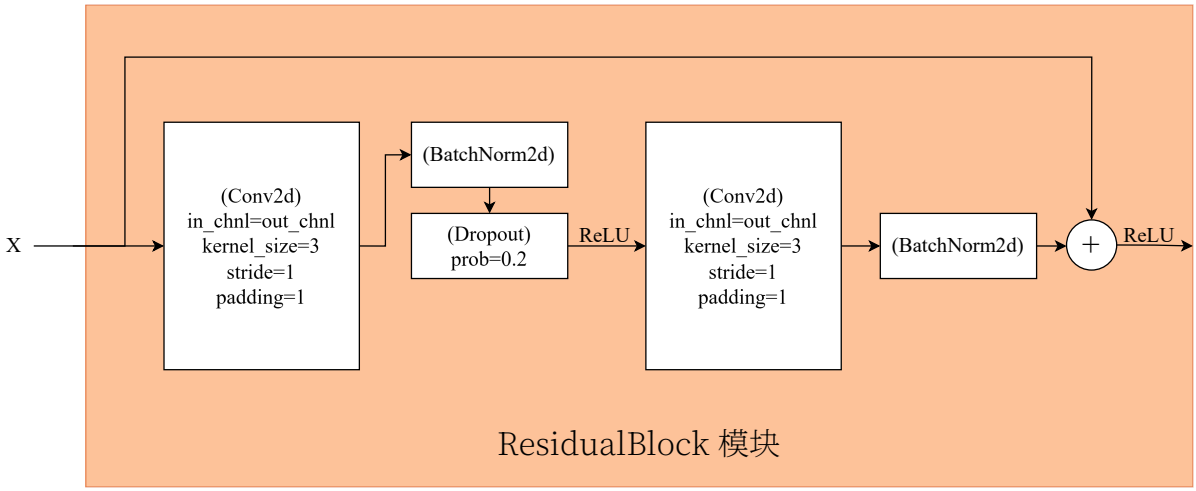

图 1　ResidualBlock 模块结构示意图

该模块使用 ResNet 中提出的残差连接思想，旨在缓解深层网络训练中的梯度消失问题。具体来说，ResidualBlock 模块包含两个卷积层，每个卷积层后面跟随一个批归一化

层。输入通过两个卷积层后，会与输入直接相加形成残差连接，然后再经过一个 ReLU 激活函数输出。这样设计可以使得网络更容易学习恒等映射，从而提升训练效果。

IV.2.b: Siamese 特征提取网络（Extractor）结构

特征提取网络主要负责提取输入图片对的深层特征。其结构由多个 ResidualBlock 模块堆叠而成，每个模块后面跟随一个最大池化层，用于降低特征图的空间维度。通过多层残差模块的堆叠，网络能够学习到更加复杂和抽象的特征表示。最后通过 Flatten 层以及一个全连接层，将提取到的特征映射到一个固定维度的嵌入空间中。具体网络结构如下：

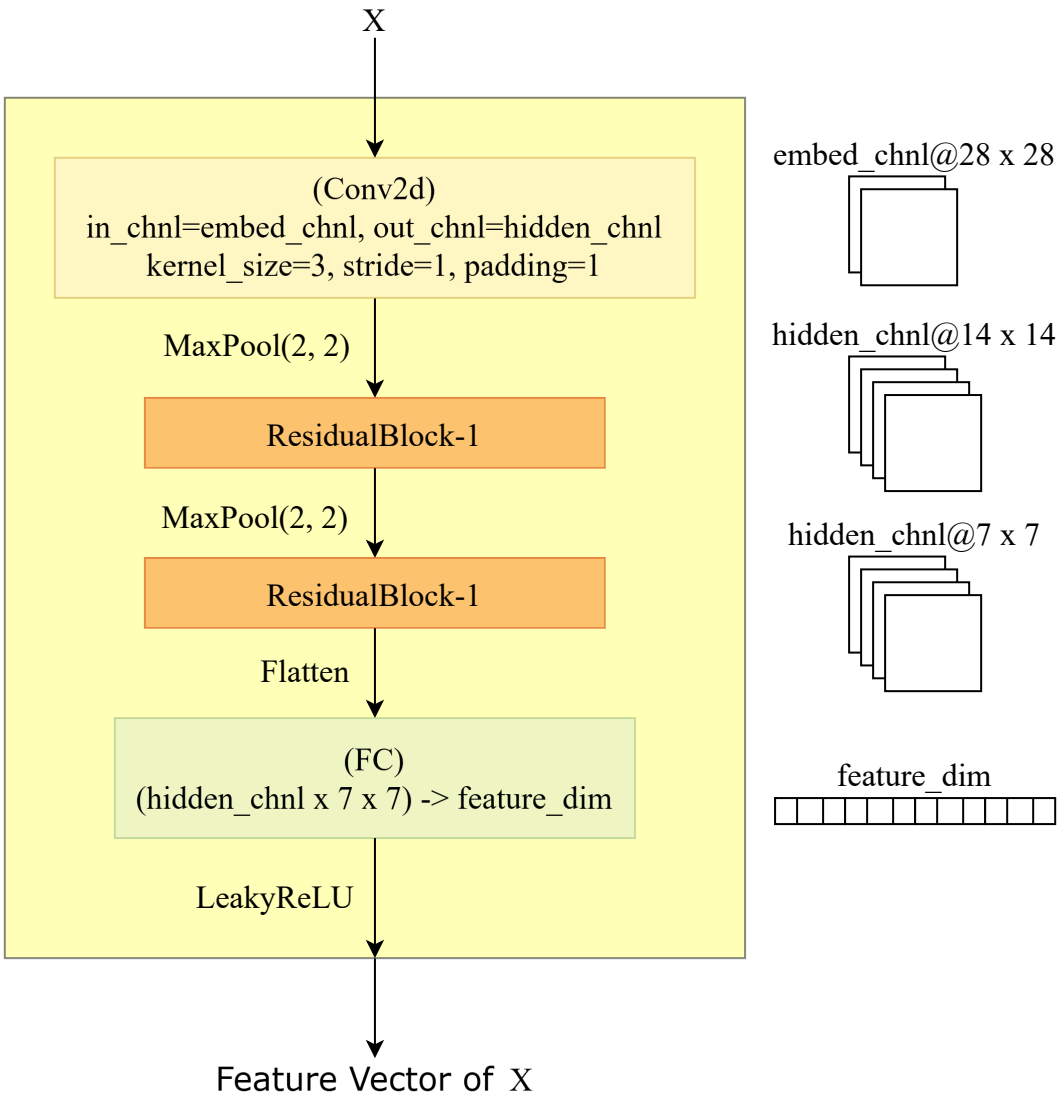

图 2　Siamese 特征提取网络结构示意图

IV.2.c: Siamese 主干网络

Siamese 主干网络由两个共享权重的特征提取网络组成，分别处理输入的两张图片。通过共享特征提取器权重，确保两个分支提取到的特征具有相同的表示能力。提取到的

两个嵌入向量随后通过一系列拼接、全连接层来获得二者的“距离”。最终得到一个二维向量，表示输入图片对属于不同类别和相同类别的权重。具体网络结构如下：

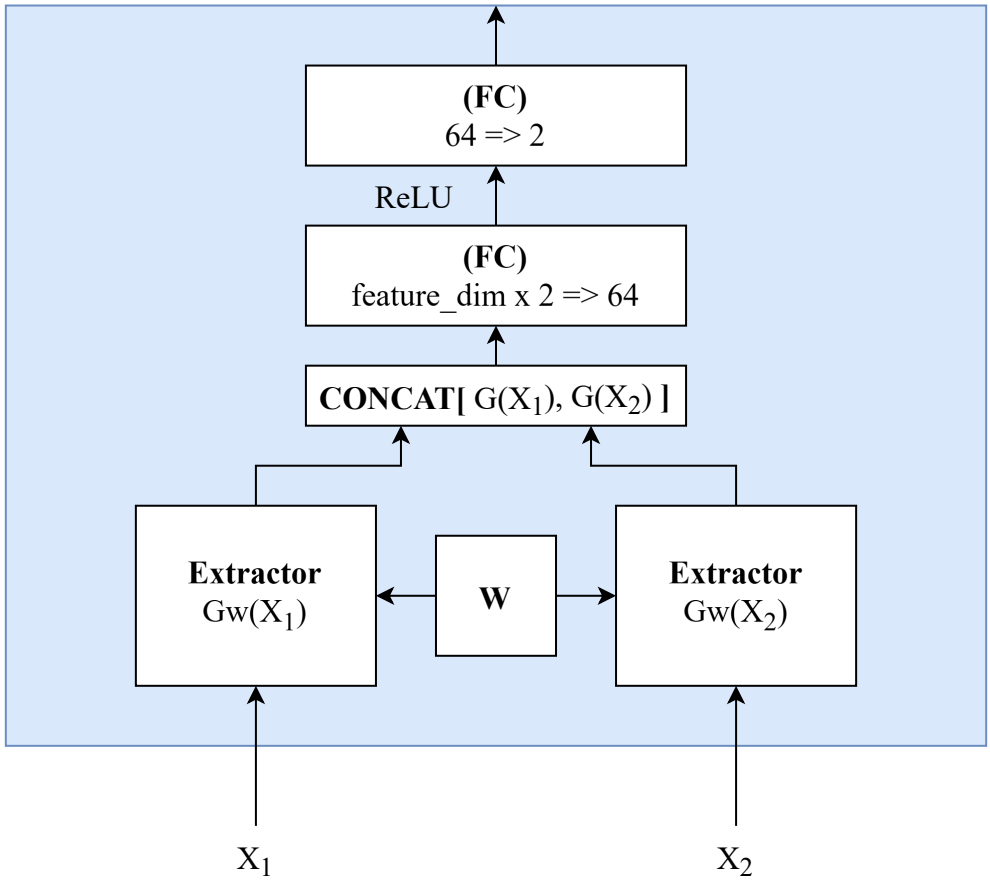

图 3　Siamese 主干网络结构示意图

IV.3: Head 分类头部

由于经过孪生网络已经得到了一个二维向量表示输入图片对的类别权重，因此我在模型最后添加了一个 Softmax 层，将输出转换为概率分布形式，便于后续的交叉熵损失计算与类别判别。

V: 实验分析

V.1: 数据分析

注意：由于数据分析要求以 mini-batch 为单位进行，因此后续分析“一次迭代”均指一个 mini-batch 的训练过程。

由于本次实验数据量巨大（多达 3600000 组图片对），而本地设置 batch_size=1024，因此可能有约 3500 个 mini-batch，如果每一个 mini-batch 都进行 validate，模型在 train

和 eval 之间频繁切换会极大影响训练效率。因此，我选择每隔 10 个 mini-batch 进行一次 validate，从而在保证一定数据分析粒度的同时，提升训练效率。使用 tensorboard 进行可视化后，得到以下结果：

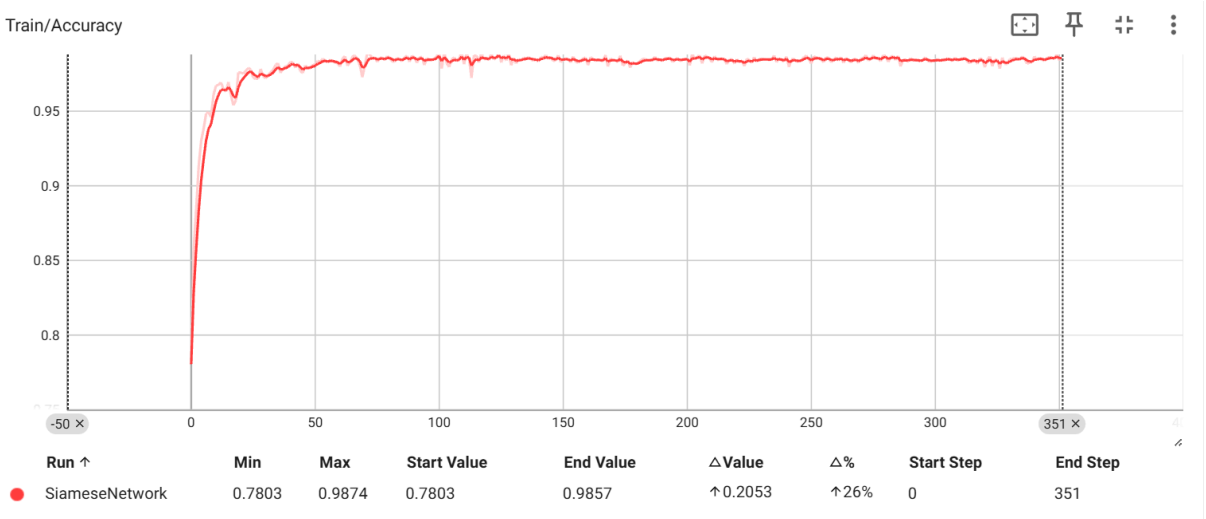

图 4　Accuracy/mini-batch Iteration 变化图（smooth=0.6）

从 Accuracy/mini-batch Iteration 变化图中可以看出，该模型的表现十分优异，准确率提升十分显著，在第九个 mini-batch Iteration 之后就可以达到 95% 的准确率，并在后续的训练过程中持续提升，最终在测试集上能够达到 98.57% 的准确率。

Loss/mini-batch Iteration 变化图其实与 Accuracy/mini-batch Iteration 变化图呈现出类似的趋势：

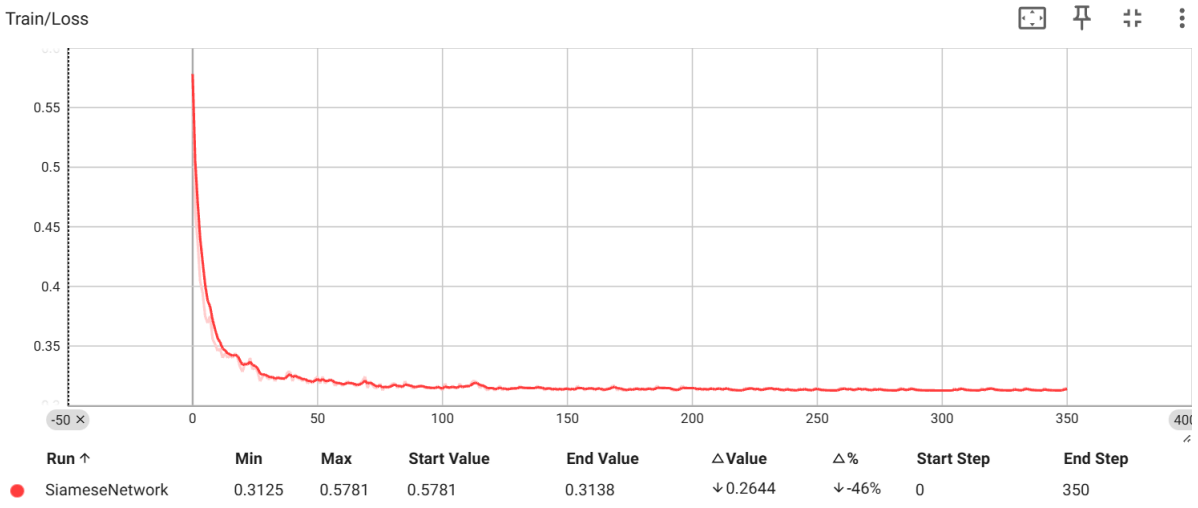

图 5　Loss/mini-batch Iteration 变化图（smooth=0.75）

Loss 的变化与上述 Accuracy 变化趋势类似，模型的收敛速度十分迅速，在整个训练进程的 10% 左右模型便几乎收敛，并且过拟合现象并不明显，Loss 曲线较为平滑，说明模型的泛化能力较强。

V.2: 实验结论

孪生网络结构在处理图像对相似性判别任务中表现出色。通过残差连接和多层特征提取，模型能够有效学习到输入图片对的深层特征表示，从而实现高准确率的分类。实验结果表明，该模型在训练集和测试集上均表现出良好的性能，验证了其设计思路的有效性。

VI: 启动 / Startup

VI.1: 环境要求

由于本次实验数据量较大，如果需要在本地运行实验代码，请确保硬件环境满足以下要求：

- GPU：建议使用 NVIDIA GPU，显存至少 8GB（经过计算，如果 select_scale=0.1，那么使用 float32 的数据量大概在 5GB，使用 bfloat16 半精度则需要约 2.5GB）；如果 GPU 显存不足，可以考虑将数据集保存在 CPU 内存中，此时要求 CPU 内存充足，建议至少 32GB RAM。

VI.2: 依赖安装

代码所需依赖保存在 `./requirements.txt` 文件中，可以通过以下命令进行安装：

```
pip install -r ./requirements.txt
```

本实验使用 python venv 虚拟环境进行环境隔离与开发，建议在虚拟环境中安装依赖。

VI.3: 启动脚本

实验代码的启动通过命令行参数进行配置，主要参数包括：

- `--select_scale` : 选择 MNIST 数据集的比例，默认为 0.1（即 10%）；
- `--embed_channels` : Embedding 模块的输出通道数，默认为 8；
- `--hidden_channels` : Siamese 主干网络中的隐藏通道数，默认为 32；
- `--learning_rate` : 学习率，默认为 0.001；

- `--batch_size`：每个 mini-batch 的样本数量，默认为 1024；
- `--num_epoch`：训练轮数，默认为 10；
- `--log_path`：Tensorboard 日志保存目录，默认为 `./notebook/logs`。

本地启动脚本 `launch.sh` 示例：

```
LOG_DIR="./notebook/logs/SiameseNetwork"
# clear logs folder
rm -r $LOG_DIR
mkdir -p $LOG_DIR

# run main.py with specified arguments
python main.py \
    --select_scale=0.1 \
    --embed_channels=8 \
    --hidden_channels=32 \
    --learning_rate=1e-3 \
    --batch_size=1024 \
    --num_epoch=1 \
    --log_path=$LOG_DIR
```

然后运行 `./launch.sh` 即可一键开始训练。

VI.4: Tensorboard 可视化

训练代码使用 Tensorboard 进行训练过程的可视化监控。

如果需要使用 Tensorboard 进行可视化，可以在命令行中运行以下命令：

```
tensorboard --logdir=/path/to/tensorboard/logs --port=6006
```

然后在浏览器中打开 `http://localhost:6006` 即可查看训练过程中的各种指标变化情况，例如 Loss/Train 和 Accuracy/Train 等。

VII: 实验代码与数据

实验代码和实验报告均可以在以下 GitHub 仓库中找到：Arextre/CVLabs/lab2.

具体的实验数据通过 Tensorboard 保存在**提交文件** `./notebook/logs` 目录下，如果需要验证检查，可以使用

```
tensorboard --logdir=./logs --port=6006
```

注意：Github 仓库中没有 tensorboard 日志，该日志仅在提交文件中存在（缓存日志记录被设置 .gitignore 忽略了，没有上传至 Github 仓库）

附录：提交文件说明

提交的文件结构如下：

```
向恩泽-U202314607-实验报告二
├── launch.sh           # 启动脚本实例
├── notebook            # 运行实验的缓存目录
│   ├── logs            # Tensorboard 日志文件夹
│   │   └── ......
├── main.py             # 程序入口源代码
├── report              # 实验报告源代码
│   ├── img
│   │   └── ......
│   └── report.typ
├── requirements.txt    # 依赖列表
├── utils.py            # 工具函数源代码
├── models.py           # 模型定义源代码
└── 实验报告.pdf        # 实验报告 pdf 版本
```

实验所用代码位于 main.py, utils.py 和 models.py 中。实验报告源代码位于 report/report.typ 中，实验日志文件保存在 ./notebook/logs/ 目录下。