華中科技大學

课 程 实 验 报 告

课程名称： 计算机视觉

实验名称： 实验三

院　　系：计算机科学与技术学院

专业班级：本硕博 2301 班

学　　号：U202315763

姓　　名：王家乐

2025 年 10 月 16 日

1. 任务要求

对实验二构建的分类神经网络进行权重剪枝实现模型压缩。做法如下：对最后一层卷积层，依据输出特征图的神经元激活的排序，进行依次剪枝。若最后一层卷积层的权重大小为$D \times 3 \times 3 \times P$，输出特征图大小为$M \times N \times P$，在测试数据集上对$P$个输出特征图的神经元激活（test_$dataset_size \times M \times N$）求平均并进行排序。按激活水平由低到高，对前 K 个神经元权重进行剪枝，$K = 1\ to\ P - 1$。剪枝后的卷积层权重大小为$D \times 3 \times 3 \times (P - K)$，测试此时神经网络分类准确率。提示：可将待剪枝的神经元权重、偏置设为 0，即相当于神经元剪枝而不用改变网络架构。

2. 实验设计

2.1 实验环境

项	参数
操作系统	Windows11
CPU	12th Gen Intel(R) Core(TM) i7-12700H
RAM	16GB
编程语言	Python3.10.18
平台	VScode & Jupyter Notebook

2.2 相关依赖

依赖项	版本
torch	2.8.0
pandas	2.3.1
numpy	2.2.6
matplotlib	3.10.5
torchvision	0.21.0
scikit-learn	1.7.2

2.3 配置信息

写一个 Cfg 类来管理本实验的相关配置，包括：实验二的数据集路径、batch_size、实验二训练出的模型保存路径及模型名称等。

```
class Cfg:
    data_root = os.path.abspath(os.path.join(os.getcwd(), "../exp-2/data"))
    train_fraction = 0.10          # use 10% of the train split
    test_fraction  = 0.10          # use 10% of the test split
    train_num_pairs = 10000        # number of training pairs
    test_num_pairs  = 2000         # number of test pairs
    batch_size = 64
    num_workers = 2
    save_dir = os.path.abspath(os.path.join(os.getcwd(), "../exp-2/output"))
    model_name = 'siamese_mnist_same_digit.pth'
```

2.4 数据集加载和处理（同实验二）

使用 torchvision.datasets.MNIST 加载数据集，按题目要求选取训练集 10%与测试集 10%。标准 MNIST 规模为 60000/10000，因此本实验实际使用 6000 张训练图片与 1000 张测试图片。使用 transforms 将像素值归一化并进行标准化。

```
# Transforms 预处理
transform = transforms.Compose([
    transforms.ToTensor(), # 归一化[0,1], shape (1,28,28)
    transforms.Normalize((0.1307,), (0.3081,)),
])

# 加载 MNIST 数据集
train_dataset = datasets.MNIST(Cfg.data_root, train=True,  download=True,
     transform=transform)
test_dataset  = datasets.MNIST(Cfg.data_root, train=False, download=True,
     transform=transform)

# 抽取 10%的数据
def sample_dataset(dataset, sample_ratio=0.1):
    indices = list(range(len(dataset)))
    sampled_indices = random.sample(indices, int(len(dataset) * sample_ratio))
    return Subset(dataset, sampled_indices)
train_subset = sample_dataset(train_dataset, Cfg.train_fraction)
test_subset = sample_dataset(test_dataset, Cfg.test_fraction)
```

继承 Dataset 类构建 MNISTPairsDataset 类，根据 Cfg 中的 train_num_pairs 和 test_num_pairs 从采样后的训练集和测试集随机生成比例为 1:1 的正负样本（正样本为相同数字的配对、负样本为不同数字的配对），并为正样本打上 Label=1 的标签，为负样本打上 Label=0 的标签。进而创建供 pytorch 使用的数据加载器。

```
# 创建配对数据集
class MNISTPairsDataset(Dataset):
    def __init__(self, subset, num_pairs):
        self.subset = subset
        self.num_pairs = num_pairs
        self.labels = [label for _, label in subset]
        self.images = [image for image, _ in subset]
        # 生成相同数字的索引映射
        label_to_indices = {}
        for idx, label in enumerate(self.labels):
            if label not in label_to_indices:
                label_to_indices[label] = []
            label_to_indices[label].append(idx)
        # 创建配对
        self.pairs = []
        self.pair_labels = []
        for _ in range(num_pairs // 2):
            # 正样本对
            label = random.choice(list(label_to_indices.keys()))
            if len(label_to_indices[label]) >= 2:
                idx1, idx2 = random.sample(label_to_indices[label], 2)
                self.pairs.append((idx1, idx2))
                self.pair_labels.append(1)
            # 负样本对
            label1, label2 = random.sample(list(label_to_indices.keys()), 2)
            idx1 = random.choice(label_to_indices[label1])
            idx2 = random.choice(label_to_indices[label2])
            self.pairs.append((idx1, idx2))
            self.pair_labels.append(0)

    def __len__(self):
        return len(self.pairs)

    def __getitem__(self, idx):
        idx1, idx2 = self.pairs[idx]
        x1 = self.images[idx1]
        x2 = self.images[idx2]
```

```
        y = self.pair_labels[idx]
        return x1, x2, torch.tensor(y, dtype=torch.float32)

# 创建训练和测试配对数据集
train_pairs_dataset = MNISTPairsDataset(train_subset, num_pairs=Cfg.train_num_pairs)
test_pairs_dataset = MNISTPairsDataset(test_subset, num_pairs=Cfg.test_num_pairs)

# 数据加载器
train_loader = DataLoader(train_pairs_dataset, batch_size=Cfg.batch_size,
      shuffle=True, num_workers=Cfg.num_workers)
test_loader = DataLoader(test_pairs_dataset, batch_size=Cfg.batch_size,
      shuffle=False, num_workers=Cfg.num_workers)
```

2.5 孪生卷积神经网络（同实验二）

网络架构的核心采用共享权重的 Siamese 卷积网络，包含以下部分：

1) ConvEncoder：两层卷积(ReLU 激活)与 2×2 最大池化提取 28×28 灰度图的表征；随后为 Flatten 与全连接层映射到嵌入空间。

2) 特征融合：对两个嵌入向量 z_1、z_2计算$|z_1-z_2|$与 $z_1 \odot z_2$并连接。

3) 判别头：构建全连接神经网络输出二分类的 logit 并使用 Sigmoid 激活输出[0,1]之间的概率，大于 0.5 认为是正样本，否则认为是负样本。

```
class ConvEncoder(nn.Module):
    def __init__(self, emb_dim=64):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, padding=1),   # 1x28x28 -> 16x28x28
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),                              # -> 16x14x14
            nn.Conv2d(16, 32, kernel_size=3, padding=1),  # -> 32x14x14
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),                              # -> 32x7x7
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32*7*7, 128),
            nn.ReLU(inplace=True),
            nn.Dropout(0.2),
            nn.Linear(128, emb_dim),
```

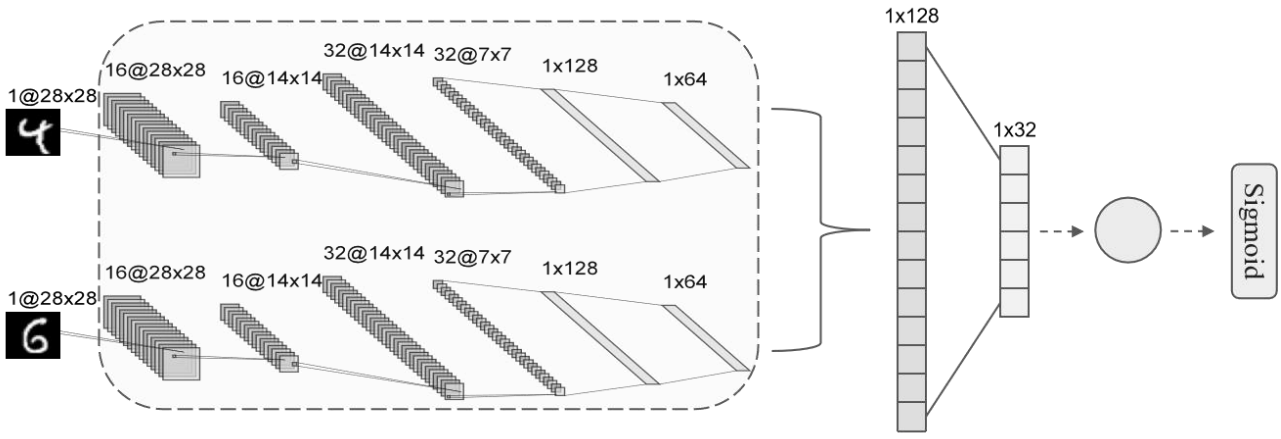
```
        )

    def forward(self, x):
        x = self.features(x)
        x = self.fc(x)
        return x

class SiameseSameDigit(nn.Module):
    def __init__(self, emb_dim=64):
        super().__init__()
        self.encoder = ConvEncoder(emb_dim=emb_dim)
        # 使用绝对差和元素级乘积组合特征
        self.head = nn.Sequential(
            nn.Linear(emb_dim*2, 32),
            nn.ReLU(inplace=True),
            nn.Linear(32, 1),  # logit
            nn.Sigmoid()  # 输出概率
        )

    def forward(self, x1, x2):
        z1 = self.encoder(x1)
        z2 = self.encoder(x2)
        feat = torch.cat([torch.abs(z1 - z2), z1 * z2], dim=1)
        logit = self.head(feat)
        return logit.squeeze(1)  # (B,)
```

该 Siamese 卷积网络结构如下图所示：



remark:虚线部分实际为同一网络，二者共享权重，这里为了画图方便将其分开

接下来加载实验二保存的模型用作该实验的测试。

```
model = torch.load(os.path.join(Cfg.save_dir, Cfg.model_name), map_location=device,
    weights_only=False)
model = model.to(device)
```

2.6 平均输出特征图

根据网络结构易知：最后一层卷积层的输入通道为 16，输出通道为 32，卷积核尺寸为 3×3，padding 为 1，步长为 1。因此共有 32 个大小为 16×3×3 的卷积核，最终输出图像的大小为 32×14×14。使用网络遍历一遍测试数据集，计算所有数据集在最后一层卷积层上输出的特征图的平均值，最后再计算特征图在每个通道上的平均特征值便于后续剪枝。

```
@torch.no_grad()
def get_avg_feature_maps_over_test(model, test_loader):
    model.eval()
    acts_sum = None
    n_imgs = 0

    def hook(module, inp, out):
        nonlocal acts_sum, n_imgs
        # out: [N, C, H, W]
        o = out.detach().to("cpu")
        batch_sum = o.sum(dim=0)     # [C, H, W]
        acts_sum = batch_sum if acts_sum is None else (acts_sum + batch_sum)
        n_imgs += o.shape[0]

    conv = None
    for m in model.modules():
        if isinstance(m, nn.Conv2d):
            conv = m
    assert conv is not None

    handle = conv.register_forward_hook(hook)
    # 跑一遍测试集，正常前向（Siamese 会对 x1/x2 两次触发 hook）
    for x1, x2, y in test_loader:
        x1, x2 = x1.to(device), x2.to(device)
        _ = model(x1, x2)   # 只为触发 hook
    handle.remove()

    avg_maps = acts_sum / float(n_imgs)          # [C, H, W]
    channel_scores = avg_maps.mean(dim=(1,2))   # [C]
    return avg_maps, channel_scores

avg_maps, channel_scores = get_avg_feature_maps_over_test(model, test_loader)
```

再将平均特征图可视化，共有 32 个 14×14 的特征图，将其像素值归一化。

```
def plot_avg_feature_maps(avg_maps, cols=8):
    C, H, W = avg_maps.shape
    # 归一化到[0,1]
    def norm01(x):
        x = x - x.min()
        den = x.max() + 1e-8
        return (x / den)

    rows = int(np.ceil(C / cols))
    fig, axes = plt.subplots(rows, cols, figsize=(cols * 1.2, rows * 1.2))

    for i in range(C):
        ax = axes[i // cols, i % cols]
        im = ax.imshow(norm01(avg_maps[i].numpy()), cmap='gray')
        ax.axis('off')
        ax.set_title(str(i + 1), fontsize=8)

    cbar = fig.colorbar(im, ax=axes, orientation='vertical', fraction=0.02, pad=0.04)
    cbar.set_label('Feature Intensity', fontsize=10)
    plt.show()

plot_avg_feature_maps(avg_maps, cols=8)
```

2.7 剪枝

将特征图在每个通道上的平均特征值由小到大排序，根据需要剪枝的神经元个数 K，得到平均特征最小的 K 个卷积核的索引。再将原模型拷贝，并将相应索引位置处的卷积核的权重和偏置均设置为 0，以达到剪枝的效果，测试剪枝后的模型在测试集上的准确率。

```
def prune_last_conv_by_K(model, channel_scores, K):
    pruned = copy.deepcopy(model)
    conv = None
    for m in pruned.modules():
        if isinstance(m, nn.Conv2d):
            conv = m
    assert conv is not None
    idx_sorted = torch.argsort(channel_scores)     # [C] 升序
    idx_prune = idx_sorted[:K].tolist() if K>0 else []
```

```
    with torch.no_grad():
        if len(idx_prune) > 0:
            conv.weight[idx_prune, ...] = 0.0
            if conv.bias is not None:
                conv.bias[idx_prune] = 0.0
    return pruned

@torch.no_grad()
def evaluate(model, loader):
    model.eval()
    total_correct = 0
    total = 0
    for x1, x2, y in loader:
        x1, x2, y = x1.to(device), x2.to(device), y.to(device)
        outputs = model(x1, x2)
        preds = (outputs >= 0.5).float()
        total_correct += (preds == y.view_as(preds)).sum().item()
        total += y.size(0)
    return total_correct/total

K_list = list(range(1,32))
acc_list = []
for K in K_list:
    pruned_model = prune_last_conv_by_K(model, channel_scores, K)
    acc = evaluate(pruned_model, test_loader)
    acc_list.append(acc)
    print(f"K={K:2d} | Test Acc={acc:.4f}")
```

2.8 Accuaracy-K 图像

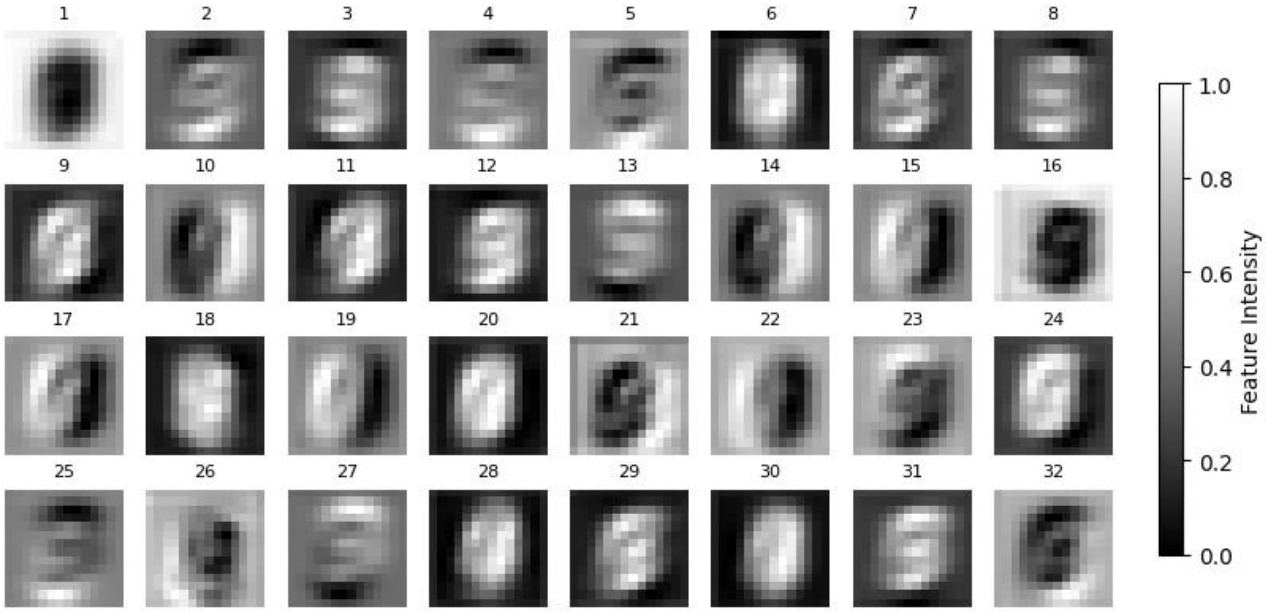
根据 acc_list 绘制 Accuracy 随剪枝个数 K 的变化曲线。

```
def pltot_accuracy_vs_K(x, y):
    plt.figure(figsize=(10, 5))
    plt.plot(x, y, color='#1f77b4', marker='o', label='Test Accuracy')
    plt.xlabel("K (Pruned Channels)", fontsize=14, fontweight='bold')
    plt.ylabel("Test Accuracy", fontsize=14, fontweight='bold')
    plt.title("Accuracy vs K (Pruning Last Conv2d)", fontsize=16, fontweight='bold')
    plt.grid(True, linestyle='--', linewidth=0.6, alpha=0.7)
    plt.legend(fontsize=12, loc='best', frameon=False)
    plt.tight_layout()
    plt.show()

pltot_accuracy_vs_K(K_list, acc_list)
```

3. 实验结果与分析

3.1 平均特征图

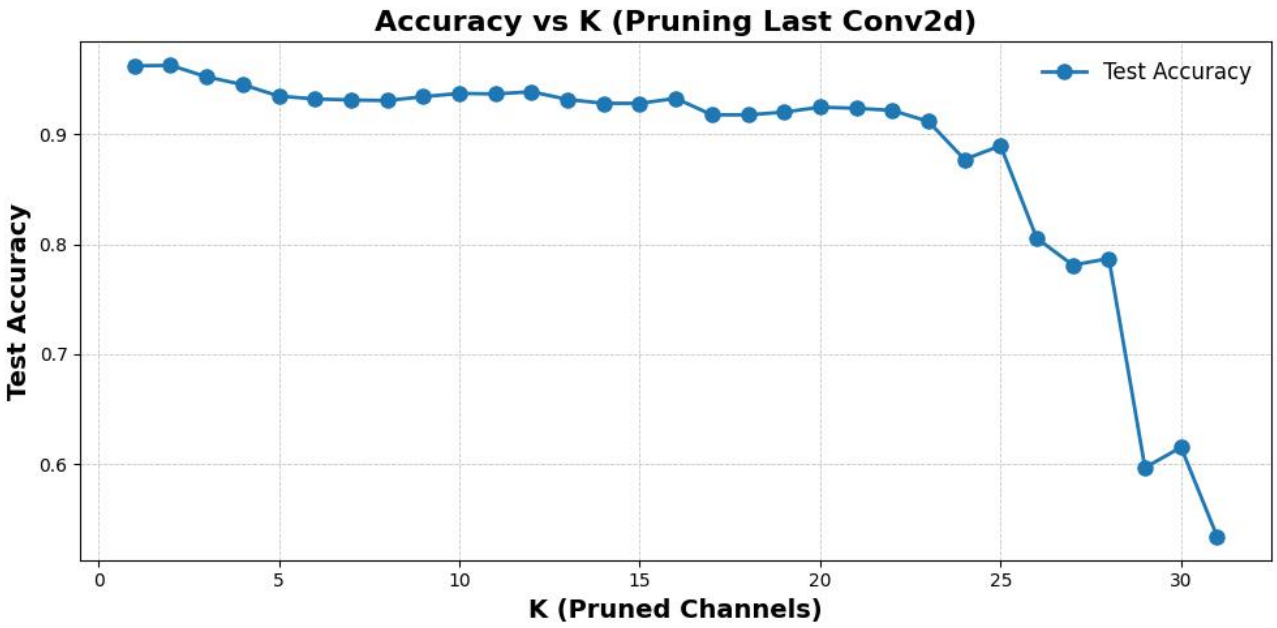


3.2 Accuaracy 与 K 的关系

实验中 K 从 1 逐渐增大到 31，测试数据集的 Accuarcy 随 K 的变化如下：

```
K= 1 | Test Acc=0.9620   K= 2 | Test Acc=0.9625
K= 3 | Test Acc=0.9520   K= 4 | Test Acc=0.9450
K= 5 | Test Acc=0.9345   K= 6 | Test Acc=0.9320
K= 7 | Test Acc=0.9310   K= 8 | Test Acc=0.9305
K= 9 | Test Acc=0.9340   K=10 | Test Acc=0.9370
K=11 | Test Acc=0.9365   K=12 | Test Acc=0.9385
K=13 | Test Acc=0.9315   K=14 | Test Acc=0.9280
K=15 | Test Acc=0.9280   K=16 | Test Acc=0.9325
K=17 | Test Acc=0.9175   K=18 | Test Acc=0.9175
K=19 | Test Acc=0.9200   K=20 | Test Acc=0.9245
K=21 | Test Acc=0.9235   K=22 | Test Acc=0.9215
K=23 | Test Acc=0.9115   K=24 | Test Acc=0.8770
K=25 | Test Acc=0.8895   K=26 | Test Acc=0.8055
K=27 | Test Acc=0.7810   K=28 | Test Acc=0.7870
K=29 | Test Acc=0.5970   K=30 | Test Acc=0.6155
                K=31 | Test Acc=0.5345
```

画出相应变化曲线如下图所示：



实验结果表明，Siamese 网络在 MNIST 相似性比较任务中存在显著冗余性，通过基于通道激活排序的剪枝方法，在剪除多达 50%的通道（K=16）时模型准确率仅下降约 3%，显示出模型压缩的巨大潜力；然而当剪枝通道过多后性能开始急剧恶化，说明模型对轻微剪枝具有很强的鲁棒性，该方法能有效识别并去除冗余参数而不损害模型核心功能，为资源受限环境下的模型部署提供了重要依据。

4. 总结与体会

4.1 总结

本次实验基于剪枝算法对实验二中构建的分类神经网络进行了模型压缩，重点针对最后一层卷积层进行了权重剪枝。通过计算测试数据集上各输出特征图的平均激活值，并按激活水平从低到高依次剪枝前 K 个神经元，实现了对模型结构的有效简化。实验过程中，观察到随着剪枝比例的增加，模型分类准确率呈现出先保持稳定后逐渐下降的趋势，说明在适度剪枝范围内，模型仍能保持良好的性能，同时也验证了剪枝在减少参数数量、降低计算开销方面的有效性。

4.2 体会

本次实验，我深刻体会到模型压缩在深度学习实际应用中的重要性。剪枝不仅是一种有效的模型优化手段，也让我对神经网络中各部分权重的重要性有了更直观的理解。实验中，激活值较低的神经元往往对最终输出的贡献较小，适当剪除这些部分可以在几乎不影响模型性能的前提下显著减小模型体积。此外，这次实践也增强了我在网络调试和性能分析方面的能力，为今后在实际场景中的应用打下了良好基础。