



# 研究生《深度学习》课程

## 实验报告

实验名称: 卷积神经网络

姓 名: 唐麒

学 号: 21120299

上课类型: 专业课

日 期: 2022 年 11 月 3 日

# 一、实验内容

本实验报告包括的实验有分别手写和 torch.nn 实现二维卷积，并在至少一个数据集（两个数据集分别对应了分类任务和回归任务）上进行实验，从训练时间、预测精度、Loss 变化等角度分析实验结果，并在此基础上对不同的超参数进行对比分析（例如卷积层数、卷积核大小、batchsize、lr 等）。除此之外，还可以使用实验 2 中的前馈神经网络模型来进行实验，并将实验结果与卷积模型结果进行对比分析。更进一步，使用 torch.nn 实现空洞卷积并在至少一个数据集上进行实验，从训练时间、预测精度、Loss 变化等角度分析实验结果。类似地，还会将空洞卷积模型的实验结果与卷积模型的结果进行分析比对，以及不同超参数对实验结果的影响。最后，实现经典模型 AlexNet 和 ResNet 并在至少一个数据集进行试验分析。通过本次实验，学生将对卷积神经网络的原理及实现、空洞卷积的设计以及经典卷积神经网络的设计和实现等有较为清晰的认识。

## 1.1 二维卷积实验

- (1) 手写二维卷积的实现，并在至少一个数据集上进行实验，至少迭代一个 batchsize，表明程序的正确性；
- (2) 使用 torch.nn 实现二维卷积，并在至少一个数据集上进行实验，从训练时间、预测精度、Loss 变化等角度分析实验结果；
- (3) 不同超参数的对比分析（包括卷积层数、卷积核大小、batchsize、lr 等）选其中至少 1-2 个进行分析；
- (4) 使用 PyTorch 实现经典模型 AlexNet 并在至少一个数据集进行试验分析；
- (5) 使用实验 2 中的前馈神经网络模型来进行实验，并将实验结果与卷积模型结果进行对比分析。

实验一具体内容为手写和 torch.nn 实现二维卷积，对实验结果进行分析，并探究不同超参数对模型性能的影响。

实验一涉及的内容主要为在了解卷积神经网络原理的基础上进行实现和在不同的数据集上训练优化，该实验可以帮助学生更好地了解和熟悉卷积神经网络的原理及训练技巧。

实验涉及的算法主要包括卷积神经网络的搭建和优化。

## 1.2 空洞卷积实验

- (1) 使用 `torch.nn` 实现空洞卷积, 要求 `dilation` 满足 HDC 条件(如 1,2,5)且要 堆叠多层并在至少一个数据集上进行实验, 从训练时间、预测精度、Loss 变化等角度分析实验结果;
- (2) 将空洞卷积模型的实验结果与卷积模型的结果进行分析比对, 训练时间、 预测精度、Loss 变化等角度分析;
- (3) 不同超参数的对比分析 (包括卷积层数、卷积核大小、不同 `dilation` 的选择, `batchsize`、`lr` 等) 选其中至少 1-2 个进行分析;

实验二具体内容为使用 `torch.nn` 实现空洞卷积, 对实验结果进行分析, 并探究不同超参数对模型性能的影响。

实验二涉及的内容主要为在了解空洞卷积原理的基础上利用 `torch.nn` 实现和在不同数据集上训练优化, 该实验可以帮助学生更好地了解和熟悉空洞卷积的原理以及 `torch.nn` 的使用。

实验涉及的算法主要包括空洞卷积在卷积神经网络中的设计和使用。

## 1.3 残差网络实验

- (1) 实现给定结构的残差网络, 在至少一个数据集上进行实验, 从训练时间、预测精度、Loss 变化等角度分析实验结果;
- (2) 将残差网络与空洞卷积相结合, 在至少一个数据集上进行实验, 从训练时间、预测精度、Loss 变化等角度分析实验结果 (选做);

实验三具体内容为实现给定结构的残差网络并对实验结果进行分析。

实验三涉及的内容主要为在了解残差网络原理的基础上利用 `torch.nn` 实现和在不同数据集上训练优化, 该实验可以帮助学生更好地了解和熟悉残差网络的原理等。

实验涉及的算法主要包括残差网络的搭建和优化。

## 二、实验设计

本次实验所包含的内容皆为指定内容，涉及部分数据加载、模型及其损失函数和优化器的实现，故此部分省略，在报告的后续内容中进行介绍。

## 三、实验环境及实验数据集

实验所用的开发环境如表 1 所示。

表 1 实验环境

开发环境	版本
操作系统	macOS Big Sur 11.5.1
开发工具	Jupyter Notebook
Python	3.8.2
PyTorch	1.8.0 (CPU)

本次实验所涉及的数据集包括：车辆分类数据和照片去雾数据。车辆分类数据和照片去雾数据分别对应了分类问题和回归问题。

车辆分类数据共 1357 张车辆图片，包含汽车、客车和货车三类。在划分训练集与测试集时，每个类别的后 20% 作为测试集。由于图片的大小不一，将图片拉伸到相同大小（32 × 32）。

表 2 车辆分类数据

	汽车	客车	货车	合计
训练集	623	174	288	1085
测试集	156	44	72	272
合计	779	218	360	1357

去雾数据共 520 张图片，包含带雾图片和去雾图片。在划分训练集与测试集时，后 20% 作为测试集。由于图片的大小不一，将图片拉伸到相同大小（32 × 32）。

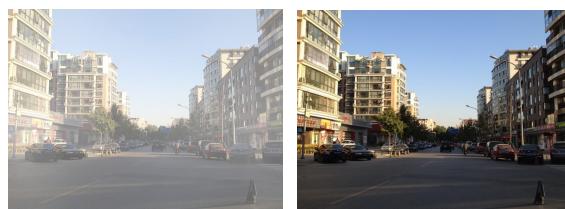


图 1 去雾数据（部分）

## 四、实验过程

本小节将对实验的具体内容进行详细说明。

首先先对两个数据集划分训练集和测试集的代码进行介绍和展示。

(1) 车辆分类数据。

```
1 def is_image_file(filename):
2     return any(filename.endswith(extension) for extension in [".png", ".jpg", ".jpeg"])
3
4 image_dirs = [r'./车辆分类数据集/bus', r'./车辆分类数据集/car', r'./车辆分类数据集
5 /truck']
6 train_list = []
7 test_list = []
8
9 for image_dir in image_dirs:
10     image_filenames = [join(image_dir, x) for x in listdir(image_dir) if is_image_file(x)]
11     list_len = len(image_filenames)
12     for id, image_filename in enumerate(image_filenames):
13         if (id + 1) / list_len <= 0.8:
14             train_list.append(image_filename)
15         else:
16             test_list.append(image_filename)
17
18 for image in train_list:
19     img = Image.open(image)
20     filename = image.split('/')[-1]
21     if 'bus' in filename:
22         prefix = 'bus/'
23     elif 'car' in filename:
24         prefix = 'car/'
25     else:
26         prefix = 'truck/'
27     img.save(r'./车辆分类数据集/train/' + prefix + filename)
28 for image in test_list:
29     img = Image.open(image)
30     filename = image.split('/')[-1]
31     if 'bus' in filename:
32         prefix = 'bus/'
```

```

33     elif 'car' in filename:
34         prefix = 'car/'
35     else:
36         prefix = 'truck/'
37     img.save(r'./车辆分类数据集/test/'+prefix+filename)
38
39 path_train = r'./车辆分类数据集/train/' # 路径
40 path_test = r'./车辆分类数据集/test/' # 路径
41 train = torchvision.datasets.ImageFolder(path_train, transform=transforms.Compose(
41 [transforms.Resize((32, 32)), transforms.CenterCrop(32), transforms.ToTensor()]))
43 test =torchvision.datasets.ImageFolder(path_test, transform=transforms.Compose(
44 [transforms.Resize((32, 32)), transforms.CenterCrop(32), transforms.ToTensor()]))

```

## (2) 去雾数据。

```

1 def is_image_file(filename):
2     return any(filename.endswith(extension) for extension in [".png", ".jpg", ".jpeg"])
3
4 image_dirs = [r'./去雾数据集/去雾图片/',r'./去雾数据集/有雾图片/']
5 train_list = []
6 test_list = []
7 for image_dir in image_dirs:
8     image_filenames = [join(image_dir, x) for x in listdir(image_dir) if is_image_file(x)]
9     list_len = len(image_filenames)
10    for id, image_filename in enumerate(image_filenames):
11        if (id + 1) / list_len <= 0.8:
12            train_list.append(image_filename)
13        else:
14            test_list.append(image_filename)
15
16    for image in train_list:
17        img = Image.open(image)
18        filename = image.split('/')[-1]
19        subdirectory = image.split('/')[-2]
20        if '去雾图片' in subdirectory:
21            prefix = 'gt/'
22        else:
23            prefix = 'input/'
24        img.save(r'./去雾数据集/train/'+prefix+filename)

```

```

25   for image in test_list:
26     img = Image.open(image)
27     filename = image.split('/')[-1]
28     subdirectory = image.split('/')[-2]
29     if '去雾图片' in subdirectory:
30       prefix = 'gt/'
31     else:
32       prefix = 'input/'
33     img.save(r'./去雾数据集/test/' + prefix + filename)
34
35 def load_img(filepath):
36   img = Image.open(filepath)
37   return img
38
39 path_train = r'./去雾数据集/train/' # 路径
40 path_test = r'./去雾数据集/test/' # 路径
41 class DatasetFromFolder(data.Dataset):
42   def __init__(self, image_dir, gt_dir, input_transform=None, target_transform=None):
43     super(DatasetFromFolder, self).__init__()
44     self.image_filenames = [join(image_dir, x) for x in listdir(image_dir) if
45     is_image_file(x)]
46     self.gt_dir = gt_dir
47     self.input_transform = input_transform
48     self.target_transform = target_transform
49   def __getitem__(self, index):
50     input = load_img(self.image_filenames[index])
51     _, file = os.path.split(self.image_filenames[index]) # dir,filename
52     target = load_img(os.path.join(self.gt_dir, os.path.splitext(file)[0] + '.jpg'))
53     if self.target_transform:
54       target = self.target_transform(target)
55     if self.input_transform:
56       input = self.input_transform(input)
57     return input, target
58   def __len__(self):
59     return len(self.image_filenames)
60
61   def input_transform():
62

```

```

63     return transforms.Compose([transforms.Resize((32, 32)), transforms.CenterCrop(32),
64     transforms.ToTensor()])
65
66 def target_transform():
67     return transforms.Compose([transforms.Resize((32, 32)), transforms.CenterCrop(32),
68     transforms.ToTensor()])
69
70 train = DatasetFromFolder(path_train + 'input/', path_train + 'gt/',
71     input_transform=input_transform(),
72     target_transform=target_transform())
73 test = DatasetFromFolder(path_test + 'input/', path_test + 'gt/',
74     input_transform=input_transform(),
75     target_transform=target_transform())

```

为了利用图表对训练损失曲线和测试损失曲线进行展示，通过将损失值和预测准确率存储来进行曲线图像的绘制。

```

1 def Draw_Loss_Curve():
2     plt.figure(figsize=(15, 9))
3     plt.xlabel("epoch")
4     plt.ylabel("loss")
5     plt.title("Loss Function")
6     x_loss = np.linspace(0, len(train_loss_list), len(train_loss_list))
7     plt.plot(x_loss, train_loss_list, label=u'Train Loss', linewidth=1.5)
8     plt.plot(x_loss, test_loss_list, label=u'Test Loss', linewidth=1.5)
9     plt.legend()
10    plt.show()
11
12 def Draw_Acc_Curve():
13    plt.figure(figsize=(15, 9))
14    plt.xlabel("epoch")
15    plt.ylabel("acc")
16    plt.title("Accuracy")
17    x_acc = np.linspace(0, len(train_acc_list), len(train_acc_list))
18    plt.plot(x_acc, train_acc_list, label="train_acc", linewidth=1.5)
19    plt.plot(x_acc, test_acc_list, label="test_acc", linewidth=1.5)
20    plt.legend()
21    plt.show()

```

对于图像去雾任务，采用了 SSIM 作为评价模型性能的指标。SSIM (structural similarity) 结构相似性，是一种全参考的图像质量评价指标，它分别从亮度、对比度、结构三方面度量图像相似性。SSIM 值越大，表示图像失真越小。

```
1 def cal_ssime(img_pre,img_gt):
2     img_gt_u = np.mean(img_gt)
3     img_pre_u = np.mean(img_pre)
4     img_gt_var = np.var(img_gt)
5     img_pre_var = np.var(img_pre)
6     img_gt_std = np.sqrt(img_gt_var)
7     img_pre_std = np.sqrt(img_pre_var)
8     c1 = np.square(0.01 * 7)
9     c2 = np.square(0.03 * 7)
10    ssim_0 = (2 * img_gt_u * img_pre_u + c1) * (2 * img_gt_std * img_pre_std + c2)
11    denom = (img_gt_u ** 2 + img_pre_u ** 2 + c1) * (img_gt_var + img_pre_var + c2)
12    ssim = ssim_0 / denom
13    return ssim
```

在下文的介绍中，将重点介绍与实验探究目的相关的代码，数据加载、模型训练等基本骨架代码将不再进行详细展示。

## 4.1 二维卷积实验

手动实现二维卷积。

```
1 def corr2d(X, K):
2     batch_size,H,W = X.shape
3     k_h, k_w = K.shape
4     Y = torch.zeros((batch_size,H-k_h+1, W-k_w+1)).to(device)
5     for i in range(Y.shape[1]):
6         for j in range(Y.shape[2]):
7             Y[:,i,j] = (X[:,i:i+k_h, j:j+k_w] * K).sum()
8     return Y
9
10    def corr2d_multi_in(X, K):
11        res = corr2d(X[:,0, :, :], K[0, :, :])
12        for i in range(1, X.shape[1]):
```

```

13         res += corr2d(X[:,i, :, :], K[i, :, :])
14     return res
15
16 def corr2d_multi_in_out(X, K):
17     return torch.stack([corr2d_multi_in(X, k) for k in K],dim=1)
18
19 class MyConv2D(nn.Module):
20     def __init__(self,in_channels,out_channels,kernel_size):
21         super(MyConv2D,self).__init__()
22         if isinstance(kernel_size,int):
23             kernel_size = (kernel_size,kernel_size)
24         self.weight = nn.Parameter(torch.randn((out_channels,in_channels)+kernel_size))
25         self.bias = nn.Parameter(torch.randn(out_channels,1,1))
26     def forward(self,x):
27         return corr2d_multi_in_out(x,self.weight)+self.bias
28
29 class MyConvModule(nn.Module):
30     def __init__(self):
31         super(MyConvModule,self).__init__()
32         self.conv = nn.Sequential(
33             MyConv2D(in_channels=3,out_channels=32,kernel_size=3),
34             nn.BatchNorm2d(32),
35             nn.ReLU(inplace=True)
36         )
37         self.fc = nn.Linear(32,num_classes)
38     def forward(self,X):
39         out = self.conv(X)
40         out = F.avg_pool2d(out,30)
41         out = out.squeeze()
42         out = self.fc(out)
43         return out

```

torch.nn 实现二维卷积。

```

1 class ConvModule(nn.Module):
2     def __init__(self):
3         super(DilatedConvModule,self).__init__()
4         self.conv = nn.Sequential(

```

```

5      nn.Conv2d(in_channels=3,
6          out_channels=32,kernel_size=3,stride=1,padding=0,dilation=0),
7      nn.BatchNorm2d(32),
8      nn.ReLU(inplace=True),
9      nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1,
10         padding=0,dilation=0),
11     nn.BatchNorm2d(64),
12     nn.ReLU(inplace=True),
13     nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1,
14         padding=0,dilation=0),
15     nn.BatchNorm2d(128),
16     nn.ReLU(inplace=True)
17   )
18   self.fc = nn.Linear(128,num_classes)
19 def forward(self,X):
20   out = self.conv(X)
21   out = F.avg_pool2d(out,16)
22   out = out.squeeze()
23   out = self.fc(out)
24   return out

```

torch.nn 实现 AlexNet。

```

1  class ConvModule(nn.Module):
2    def __init__(self):
3      super(ConvModule,self).__init__()
4      self.conv = nn.Sequential(
5        nn.Conv2d(in_channels=3, out_channels=96,kernel_size=11,stride=4,padding=2),
6        nn.BatchNorm2d(96),
7        nn.ReLU(inplace=True),
8        nn.MaxPool2d(kernel_size=3,stride=2),
9        nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5, stride=1,
10           padding=2),
11        nn.BatchNorm2d(256),
12        nn.ReLU(inplace=True),
13        nn.MaxPool2d(kernel_size=3, stride=2),
14        nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3, stride=1,
15           padding=1),

```

```

16         nn.BatchNorm2d(384),
17         nn.ReLU(inplace=True),
18         nn.Conv2d(in_channels=384, out_channels=384, kernel_size=3, stride=1,
19                    padding=1),
20         nn.BatchNorm2d(384),
21         nn.ReLU(inplace=True),
22         nn.Conv2d(in_channels=384, out_channels=256, kernel_size=3, stride=1,
23                    padding=1),
24         nn.BatchNorm2d(256),
25         nn.ReLU(inplace=True),
26         nn.MaxPool2d(kernel_size=3, stride=2)
27     )
28     self.avgpool=nn.AdaptiveAvgPool2d((6,6))
29     self.fc = nn.Sequential(
30         nn.Dropout(),
31         nn.Linear(256 * 6 * 6, 4096),
32         nn.ReLU(inplace=True),
33         nn.Dropout(),
34         nn.Linear(4096, 4096),
35         nn.ReLU(inplace=True),
36         nn.Linear(4096, num_classes)
37     )
38     def forward(self,X):
39         out = self.conv(X)
40         out = self.avgpool(out)
41         out = out.view(out.size(0),-1)
42         out = self.fc(out)
43         return out

```

## 4.2 空洞卷积实验

空洞卷积模型与卷积神经网络模型的区别在于卷积层的膨胀率不为 0。

```

1  class DilatedConvModule(nn.Module):
2      def __init__(self):
3          super(DilatedConvModule,self).__init__()
4          self.conv = nn.Sequential(
5

```

```

6      nn.Conv2d(in_channels=3,
7          out_channels=32,kernel_size=3,stride=1,padding=0,dilation=1),
8      nn.BatchNorm2d(32),
9      nn.ReLU(inplace=True),
10     nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1,
11             padding=0,dilation=2),
12     nn.BatchNorm2d(64),
13     nn.ReLU(inplace=True),
14     nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1,
15             padding=0,dilation=5),
16     nn.BatchNorm2d(128),
17     nn.ReLU(inplace=True)
18 )
19     self.fc = nn.Linear(128,num_classes)
20
21     def forward(self,X):
22         out = self.conv(X)
23         out = F.avg_pool2d(out,16)
24         out = out.squeeze()
25         out = self.fc(out)
26
27         return out

```

$$M_3 = r_3 = 5$$

$$\begin{aligned} M_2 &= \max\{M_3 - 2r_2, M_3 - 2(M_3 - r_2), r_2\} \\ &= \max\{-1, 1, 3\} = 3 \leq k = 3 \end{aligned}$$

故调整 dilation rate [1, 3, 5] with 3 × 3 kernel。

### 4.3 残差网络实验

```

1   class ResidualBlock(nn.Module):
2       def __init__(self,inchannel,outchannel,stride=1):
3           super(ResidualBlock,self).__init__()
4           self.left = nn.Sequential(
5               nn.Conv2d(inchannel,outchannel,kernel_size=3,stride=stride,padding=1,bias=False),
6               nn.BatchNorm2d(outchannel),
7               nn.ReLU(inplace=True),
8               nn.Conv2d(outchannel,outchannel,kernel_size=3,stride=1,padding=1,bias=False),
9               nn.BatchNorm2d(outchannel)
10

```

```

11     )
12     self.shortcut = nn.Sequential()
13     if stride != 1 or inchannel != outchannel:
14         self.shortcut = nn.Sequential(
15             nn.Conv2d(inchannel, outchannel, kernel_size=1, stride=stride, bias=False),
16             nn.BatchNorm2d(outchannel)
17         )
18     def forward(self,x):
19         out = self.left(x)
20         out += self.shortcut(x)
21         out = F.relu(out)
22         return out
23
24 class ResNet(nn.Module):
25     def __init__(self, ResidualBlock, num_classes=3):
26         super(ResNet, self).__init__()
27         self.inchannel = 64
28         self.conv1 = nn.Sequential(
29             nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
30             nn.BatchNorm2d(64),
31             nn.ReLU()
32         )
33         self.layer1 = self.make_layer(ResidualBlock, 64, 2, stride=1)
34         self.layer2 = self.make_layer(ResidualBlock, 128, 2, stride=2)
35         self.layer3 = self.make_layer(ResidualBlock, 256, 2, stride=2)
36         self.layer4 = self.make_layer(ResidualBlock, 512, 2, stride=2)
37         self.fc = nn.Linear(512, num_classes)
38     def make_layer(self, block, channels, num_blocks, stride):
39         strides = [stride] + [1] * (num_blocks - 1) #strides=[1,1]
40         layers = []
41         for stride in strides:
42             layers.append(block(self.inchannel, channels, stride))
43             self.inchannel = channels
44         return nn.Sequential(*layers)
45     def forward(self, x):
46         out = self.conv1(x)
47         out = self.layer1(out)
48         out = self.layer2(out)

```

```

49         out = self.layer3(out)
50         out = self.layer4(out)
51         out = F.avg_pool2d(out, 4)
52         out = out.view(out.size(0), -1)
53         out = self.fc(out)
54
55     return out
56
57 def ResNet18():
58     return ResNet(ResidualBlock)

```

## 五、实验结果

除明确说明外，下文所陈述的训练集与测试集均指在汽车分类数据上的划分。

### 5.1 二维卷积实验

#### (1) 手写二维卷积

在该实验中，模型训练了 15 个 epoch，模型在训练集和测试集上的 loss 曲线如图 2 所示，均呈下降趋势。

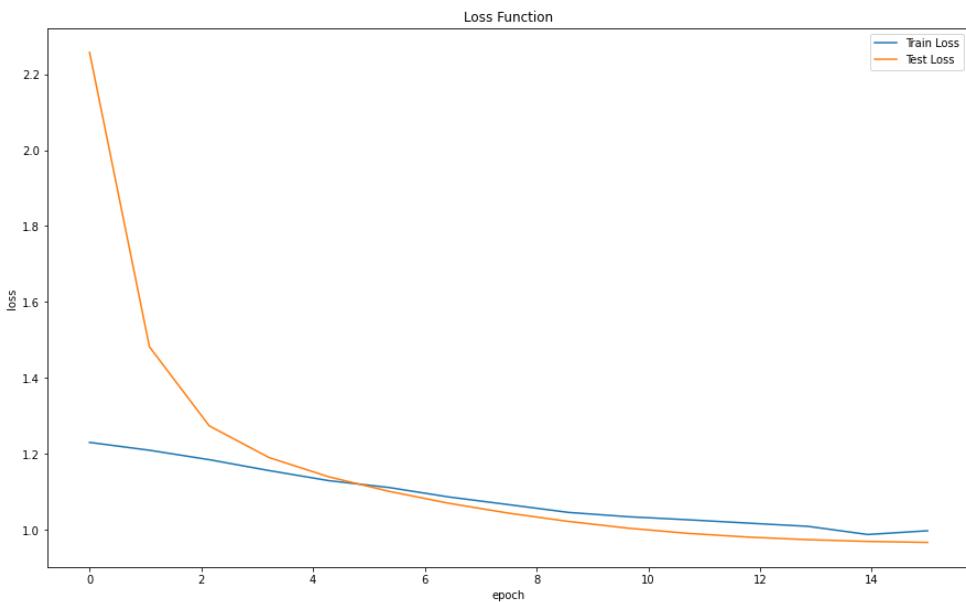


图 2 手动实现卷积神经网络损失函数曲线图

模型在训练集和测试集上的预测准确率曲线如图 3 所示，随着训练的迭代，均呈上升趋势。

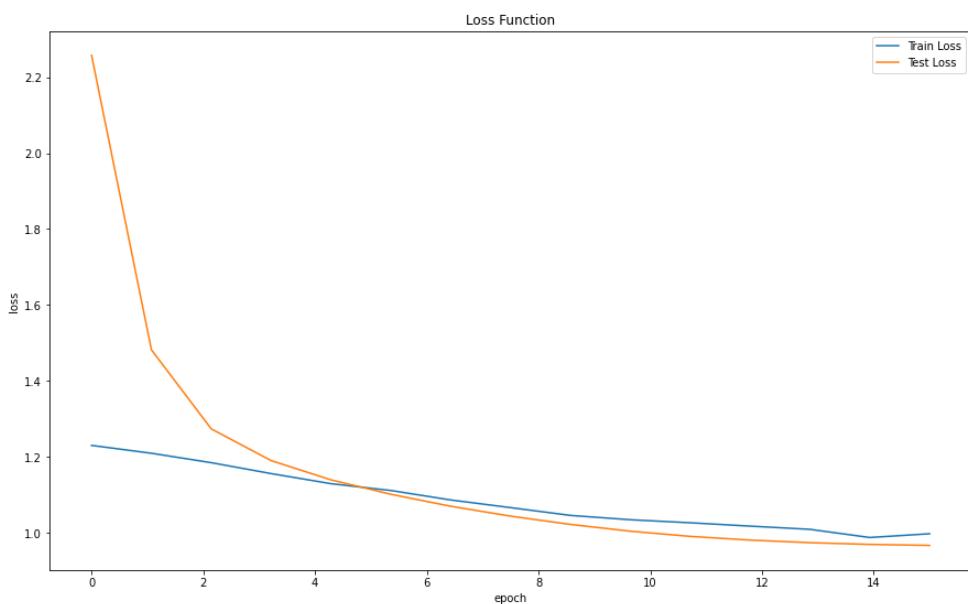


图 3 手动实现卷积神经网络预测准确率曲线图

训练 15 个 epoch 在测试集上的分类准确率最高为 0.5735, 共用时约 983.04 秒。

## (2) torch.nn 实现二维卷积

在该实验中, 模型训练了 100 个 epoch, 模型在训练集和测试集上的 loss 曲线如图 4 所示, 均呈下降趋势。

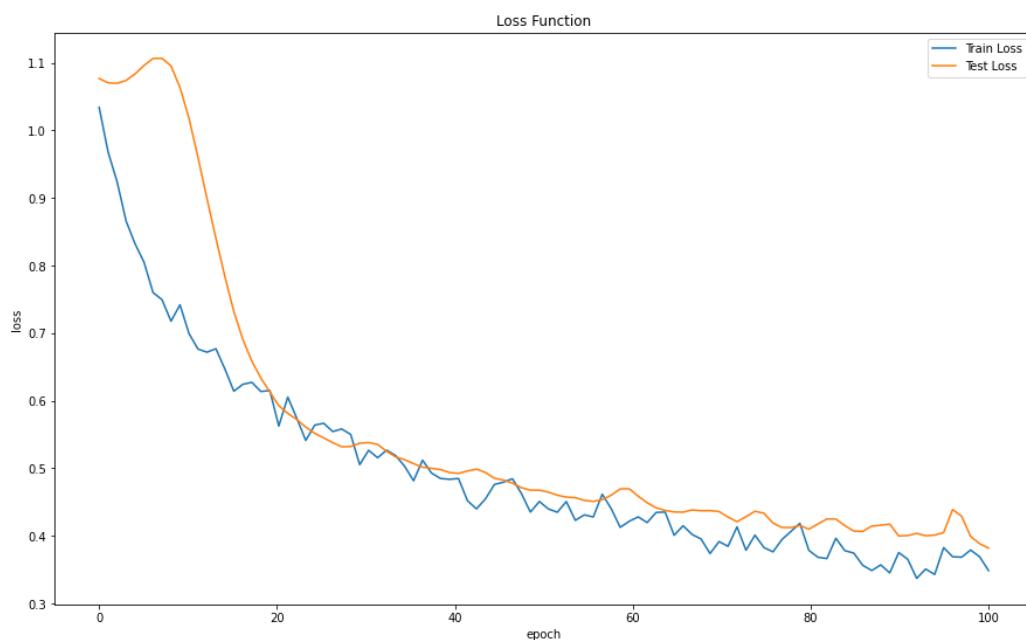


图 4 torch.nn 实现卷积神经网络损失函数曲线图

模型在训练集和测试集上的预测准确率曲线如图 5 所示，随着训练的迭代，均呈上升趋势。

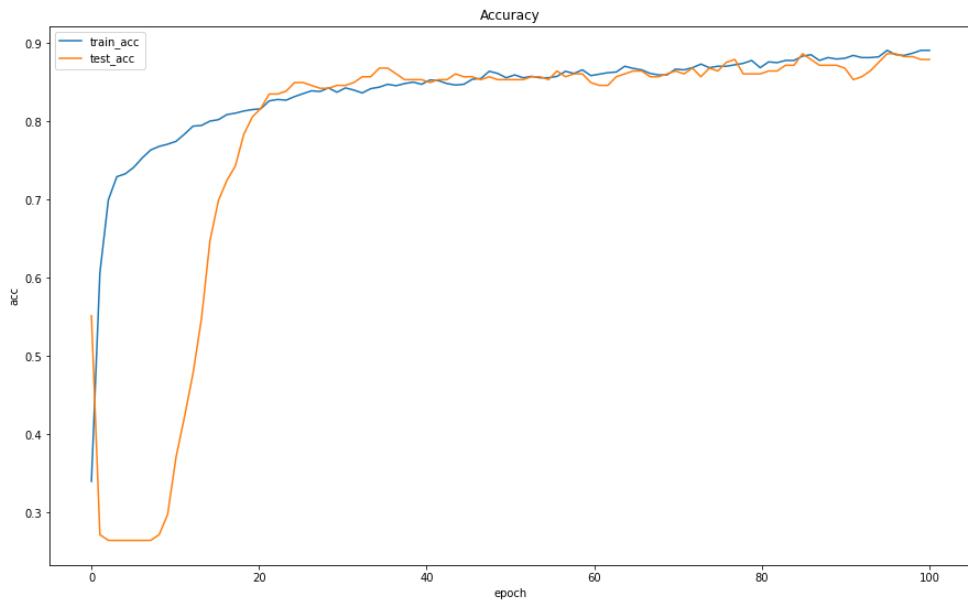


图 5 torch.nn 实现卷积神经网络预测准确率曲线图

torch.nn 实现的卷积神经网络在测试集上的分类准确率最高为 0.8786，与手动实现相比，训练速度更快，100 个 epoch 所需时间约为 784.70 秒。

除了在汽车分类数据上进行训练和测试外，还将 torch.nn 实现的卷积神经网络在去雾数据上进行训练与测试。

在训练参数不变的前提下，模型在训练集和测试集上的 loss 曲线如图 6 所示，均呈下降趋势。

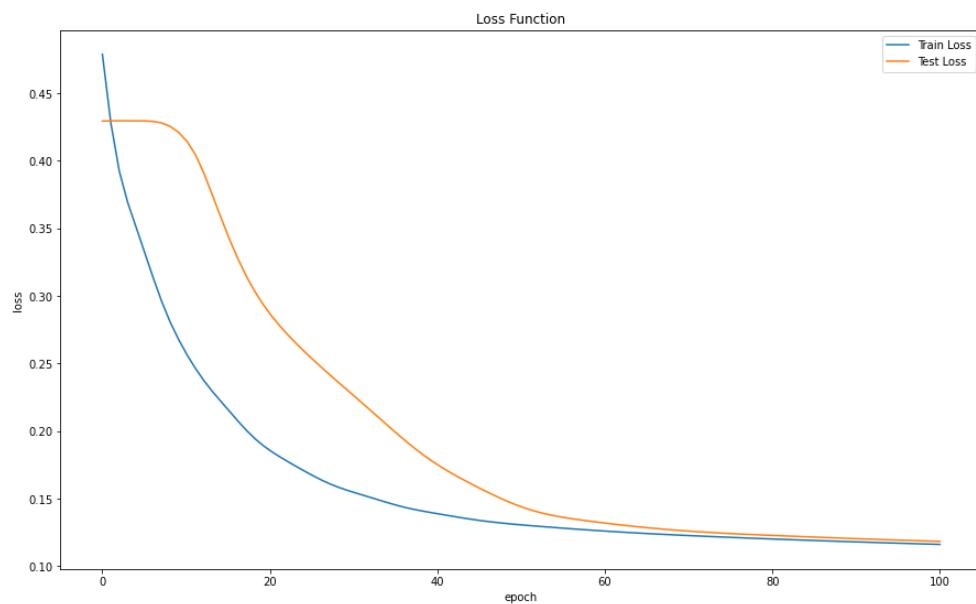


图 6 torch.nn 实现卷积神经网络损失函数曲线图

模型在训练集和测试集上的 SSIM 曲线如图 7 所示，随着训练的迭代，均呈上升趋势（SSIM 值越大，表示图像失真越小），最高为 0.9693。

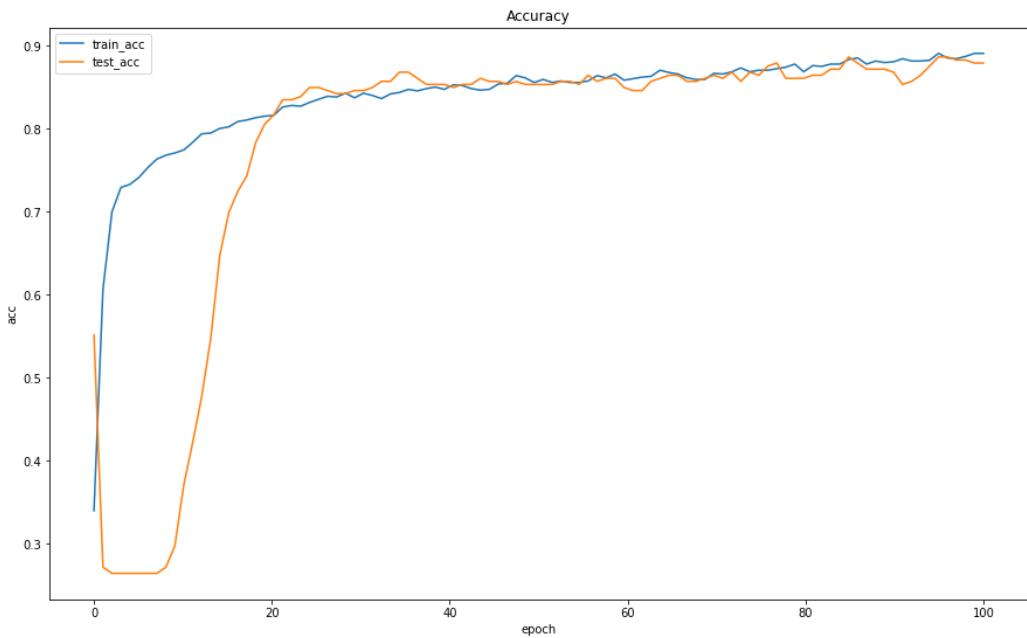


图 7 torch.nn 实现卷积神经网络 SSIM 曲线图

训练 100 个 epoch 共用时约 845.04 秒。

### (3) 不同超参数的对比分析

#### 1) 卷积层数

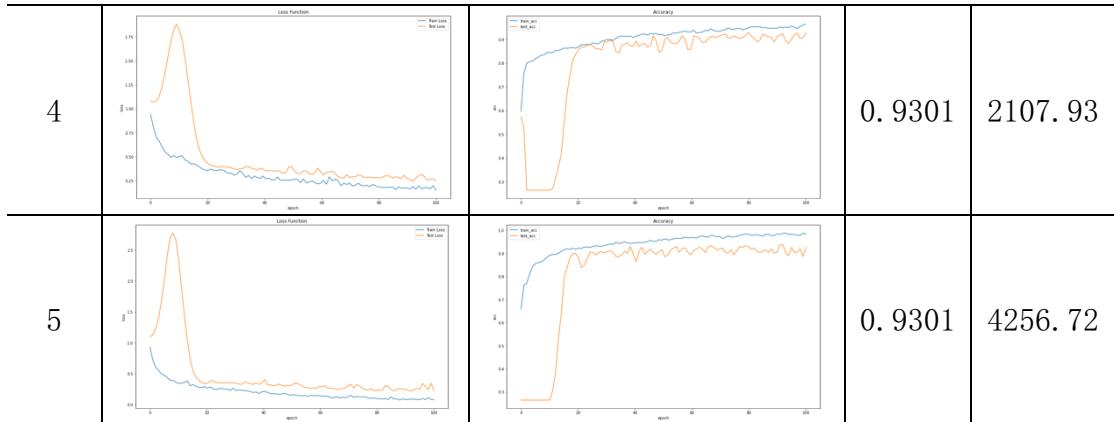
可以看到，适当模型层数的增加，可以提升分类的准确率，但同时也会造成计算开销的增加，从而导致训练时间的增长。此外，模型层数的增加提升了模型的复杂度，可以更好地拟合训练集的特征，但这种特征表示能力并不一定会在测试集上的性能带来增益。

表 3 卷积神经网络不同卷积层数结果对比

卷积层数	损失函数曲线图	预测准确率曲线图	Acc	Times(s)
3			0.8786	777.10

表 4 卷积神经网络不同卷积层数结果对比

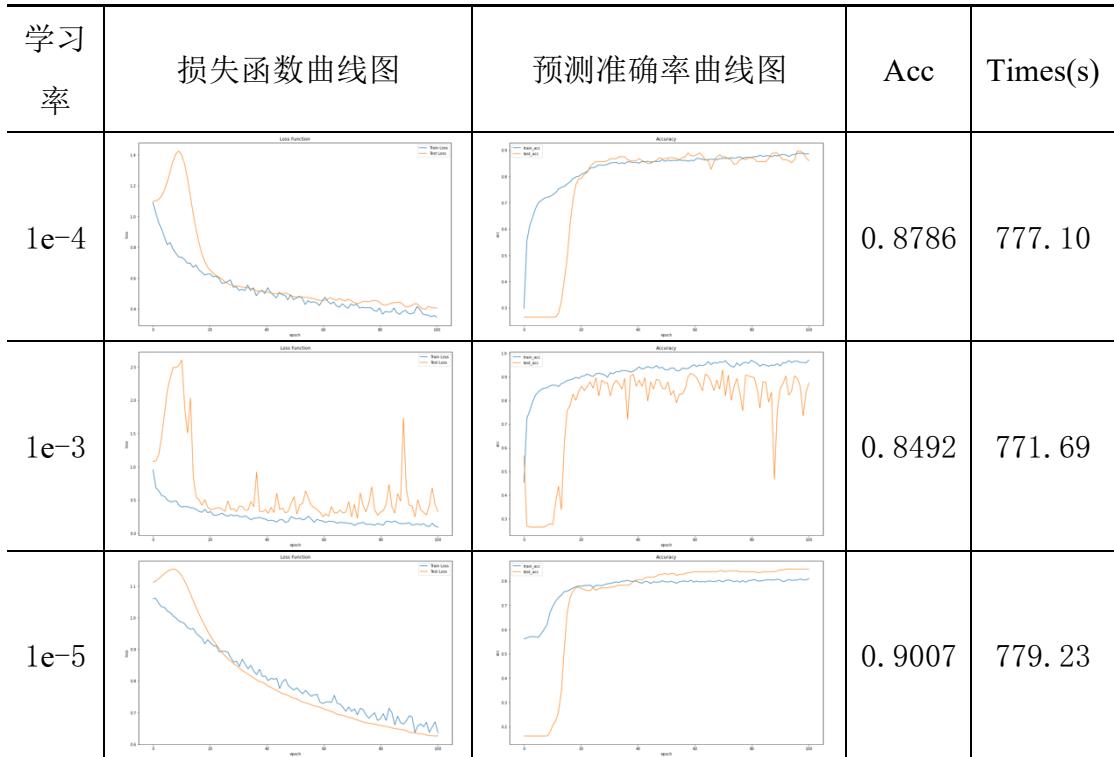
(续)



## 2) 学习率

可以看到,过大的学习率会导致损失震荡,难以收敛,对训练时间影响不大。

表 5 卷积神经网络不同学习率结果对比



## (4) PyTorch 实现经典模型 AlexNet

在该实验中,模型训练了 10 个 epoch,模型在训练集和测试集上的 loss 曲线如图 8 所示,均呈下降趋势。

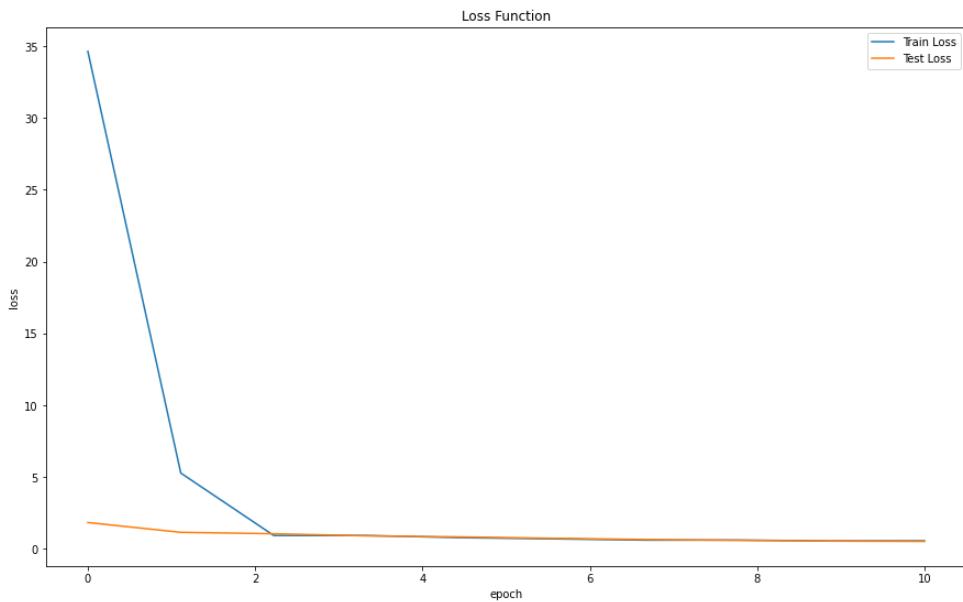


图 8 PyTorch 实现经典模型 AlexNet 损失函数曲线图

模型在训练集和测试集上的预测准确率曲线如图 9 所示，随着训练的迭代，均呈上升趋势。

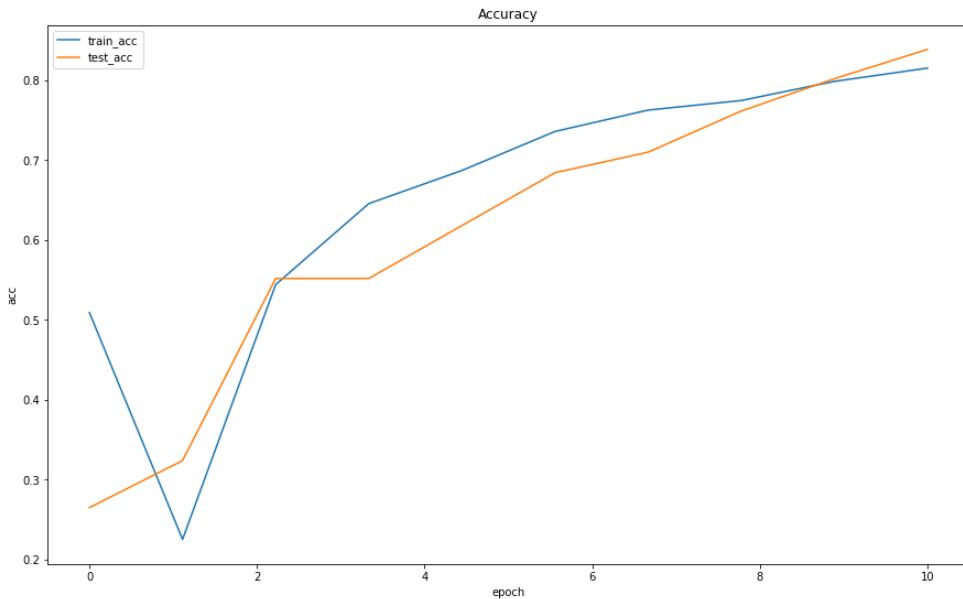


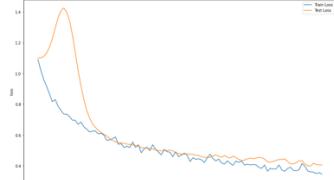
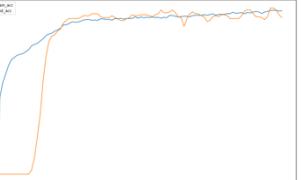
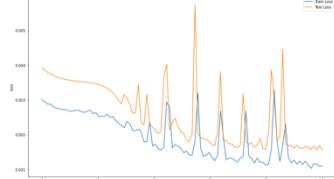
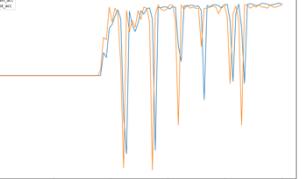
图 9 PyTorch 实现经典模型 AlexNet 预测准确率曲线图

训练 10 个 epoch 在测试集上的分类准确率最高为 0.8382, 共用时约 834.75 秒。

### (5) 与前馈神经网络模型对比

在保证前馈神经网络层数等其他超参数相同的情况下，对卷积神经网络和前馈神经网络的结果进行对比。

表 6 前馈神经网络与卷积神经网络结果对比

网络	损失函数曲线图	预测准确率曲线图	Acc	Times(s)
卷积 神经 网络			0.8786	777.10
前馈 神经 网络			0.8051	71.65

可以看到，相较于前馈神经网络而言，卷积神经网络的性能更高，但训练时间也相对更长。

## 5.2 空洞卷积实验

### (1) torch.nn 实现空洞卷积

在该实验中，模型训练了 100 个 epoch，模型在训练集和测试集上的 loss 曲线如图 10 所示，均呈下降趋势。

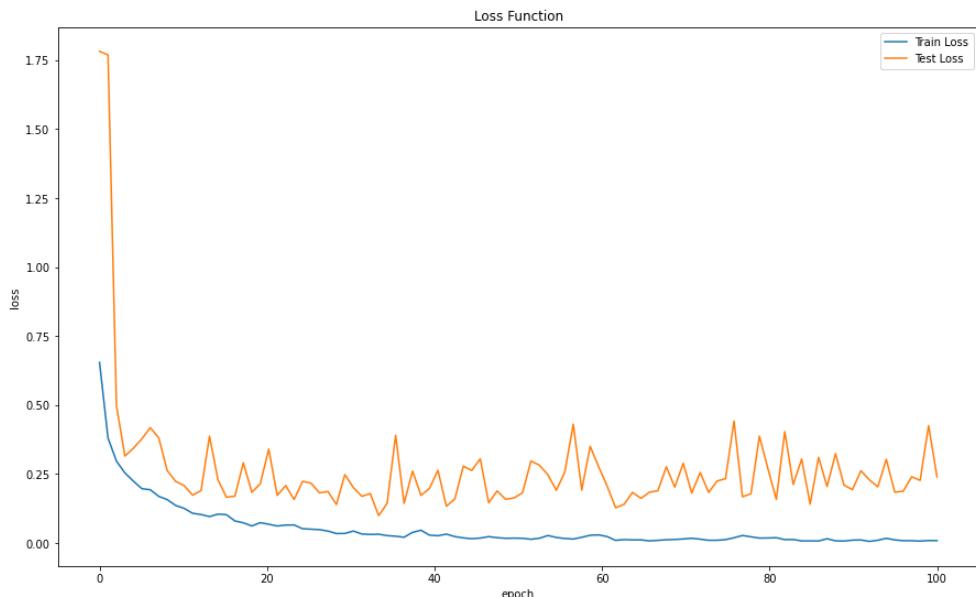


图 10 利用 torch.nn 实现空洞卷积损失函数曲线图

模型在训练集和测试集上的预测准确率曲线如图 9 所示，随着训练的迭代，均呈上升趋势。

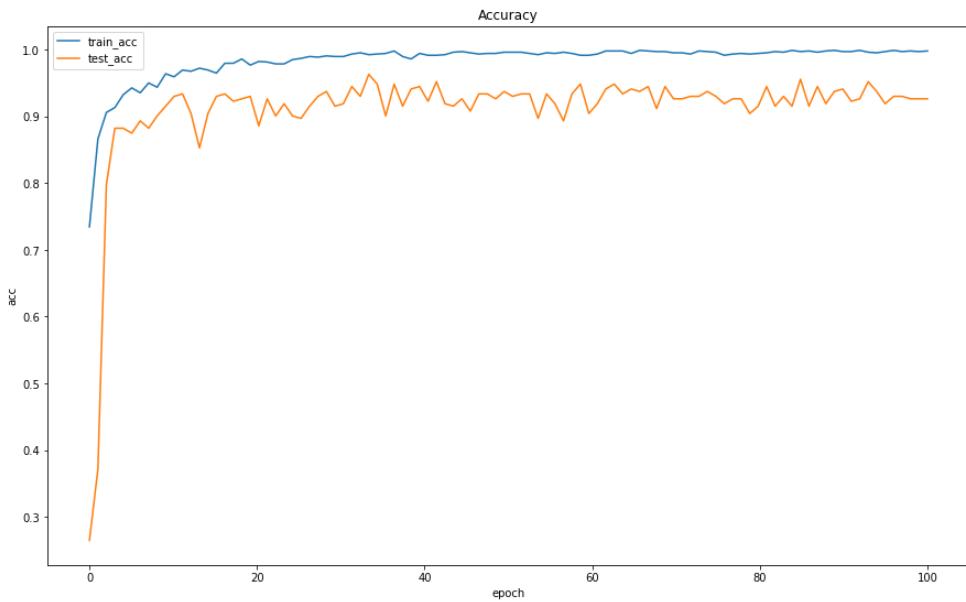


图 11 利用 torch.nn 实现空洞卷积预测准确率曲线图

训练 100 个 epoch 在测试集上的分类准确率最高为 0.9264，共用时约 448.64 秒。

## (2) 空洞卷积模型与卷积模型对比

表 7 空洞卷积模型与卷积模型结果对比

网络	损失函数曲线图	预测准确率曲线图	Acc	Times(s)
卷积 神经 网络			0.8786	777.10
空洞 卷积 模型			0.9264	448.63

可以看到，空洞卷积模型相对于卷积卷积神经网络而言具有更高的性能和更短的训练时间。

### (3) 空洞卷积模型超参数对比

1) 将卷积核大小调整为 4, 其他保持不变

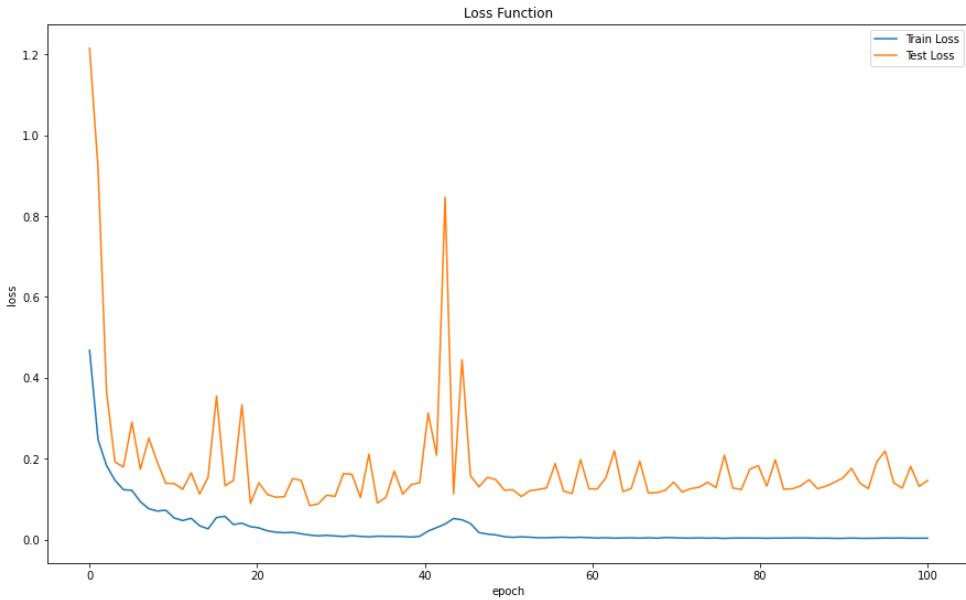


图 12 调整空洞模型卷积核大小损失函数曲线图

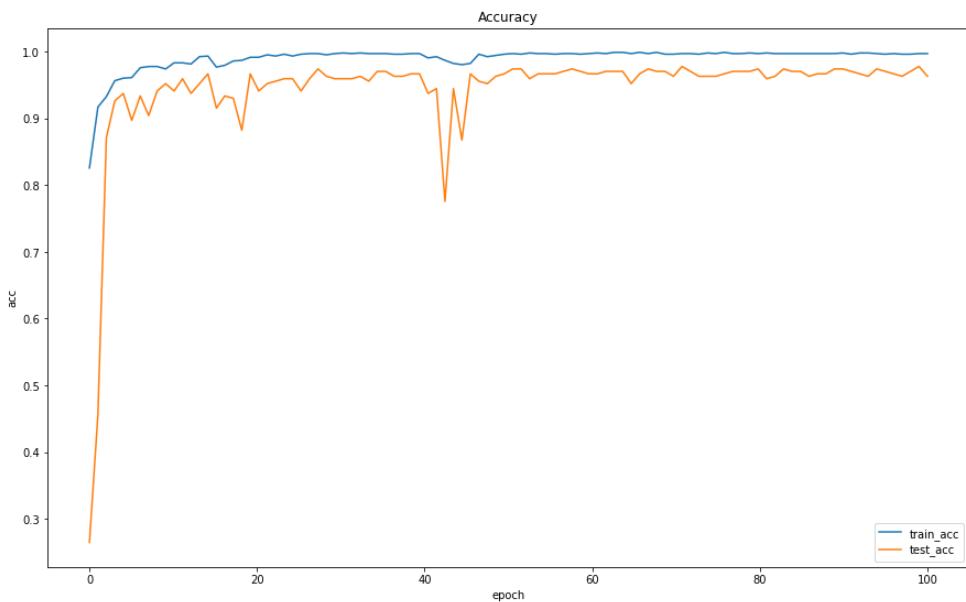


图 13 调整空洞模型卷积核大小预测准确率曲线图

训练 100 个 epoch 在测试集上的分类准确率最高为 0.9742(性能得到提升),  
共用时约 374.11 秒 (训练时间减少)。

2) 将膨胀率调整为 1、3、5，其他保持不变

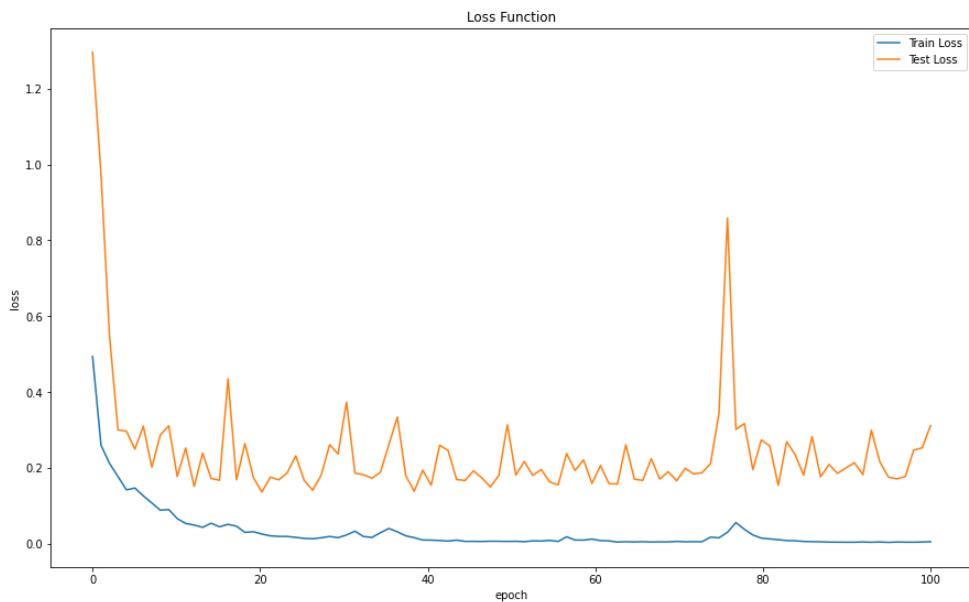


图 14 调整空洞模型膨胀率为 1、3、5 损失函数曲线图

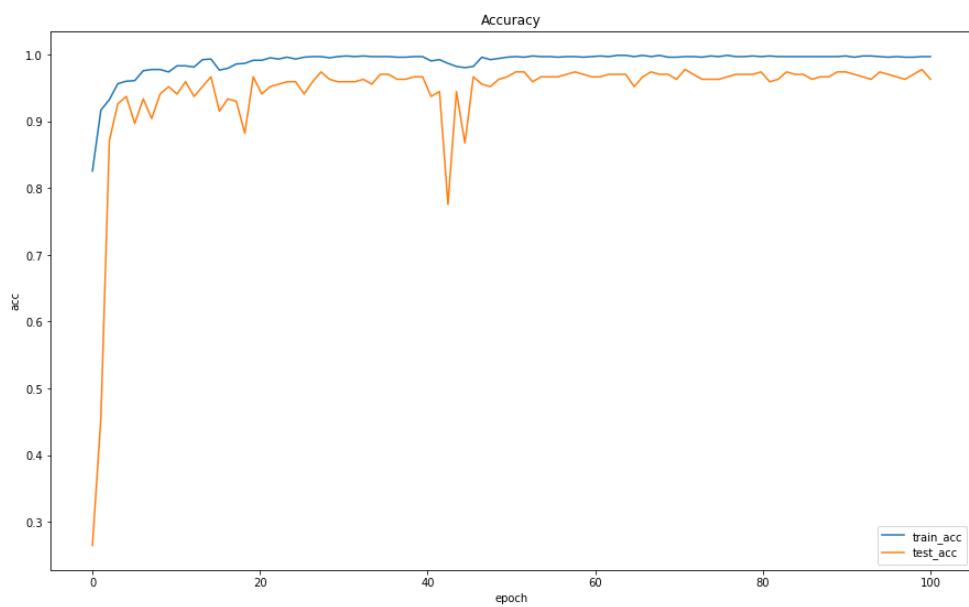


图 15 调整空洞模型膨胀率为 1、3、5 预测准确率曲线图

训练 100 个 epoch 在测试集上的分类准确率最高为 0.9586(性能得到提升)，  
共用时约 384.98 秒 (训练时间减少)。

3) 将学习率调整为  $1e-4$ , 其他保持不变

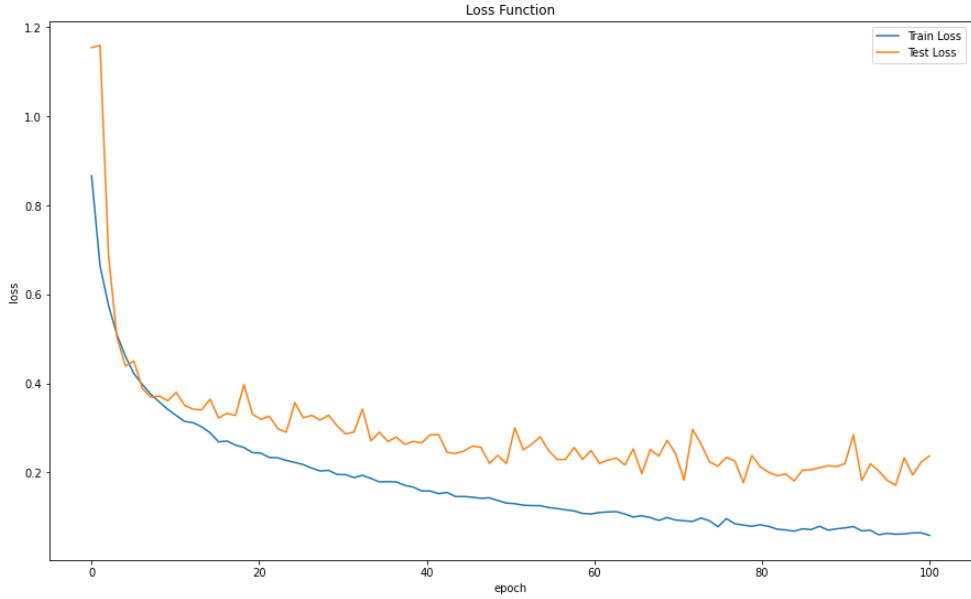


图 16 调整空洞模型学习率为  $1e-4$  损失函数曲线图

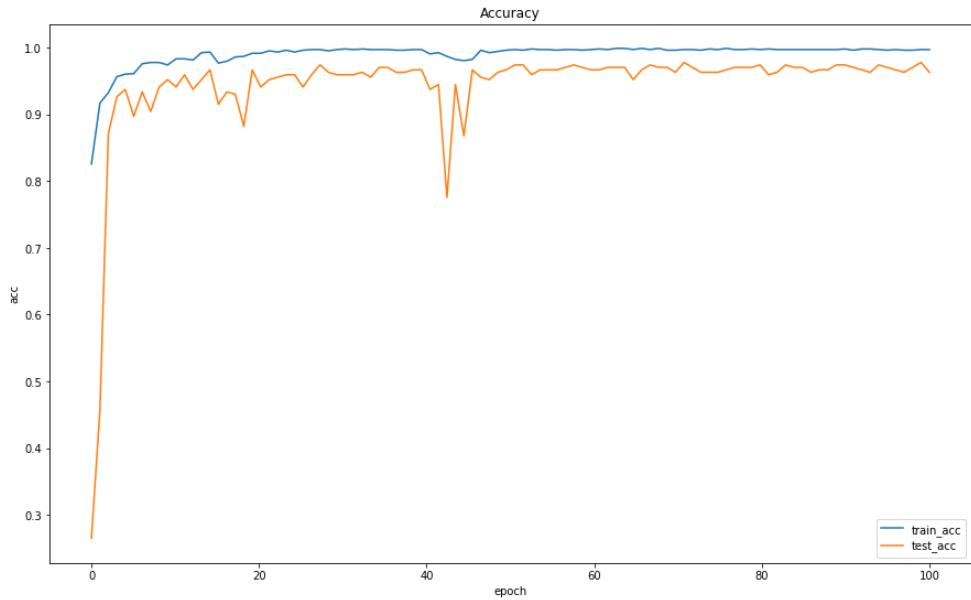


图 17 调整空洞模型学习率为  $1e-4$  预测准确率曲线图

训练 100 个 epoch 在测试集上的分类准确率最高为 0.9375(性能得到提升),  
共用时约 452.43 秒 (训练时间变化较小)。

### 5.3 残差网络实验

在该实验中, 模型训练了 25 个 epoch, 模型在训练集和测试集上的 loss 曲线如图 18 所示, 均呈下降趋势。

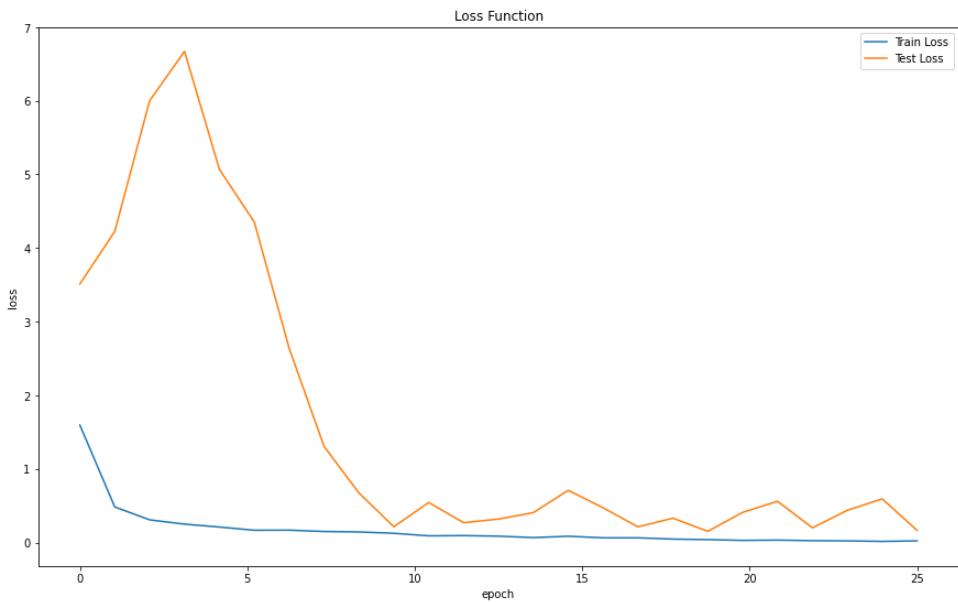


图 18 给定结构的残差网络损失函数曲线图

模型在训练集和测试集上的预测准确率曲线如图 19 所示，随着训练的迭代，均呈上升趋势。

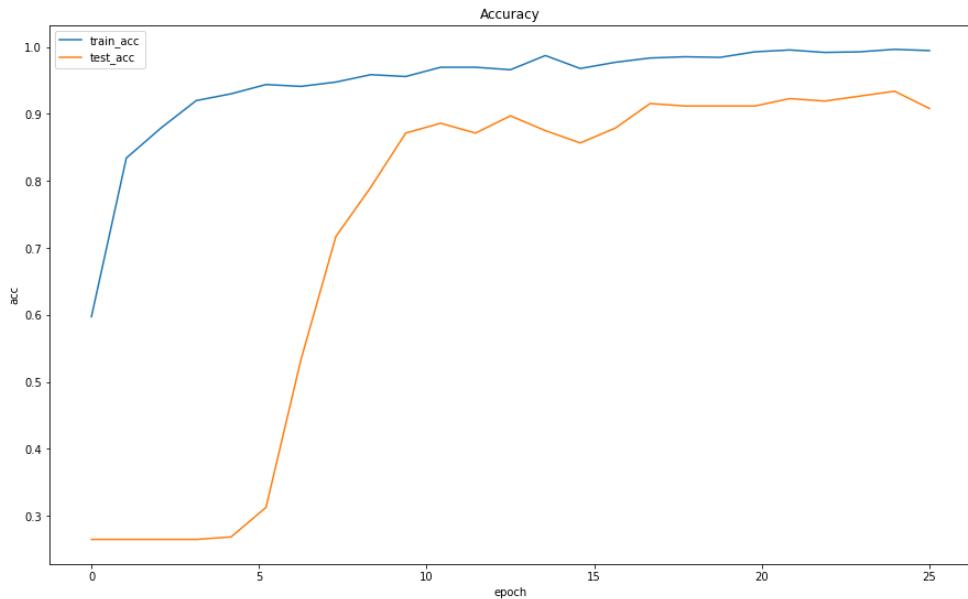


图 19 给定结构的残差网络预测准确率曲线图

残差网络训练 25 个 epoch 准确率最高为 0.9338，所需时间约为 1348.13 秒。

除了在汽车分类数据上进行训练和测试外，还将实现的给定结构的残差网络在去雾数据上进行训练与测试。

在训练参数不变的前提下，模型在训练集和测试集上的 loss 曲线如图 20 所示，均呈下降趋势。

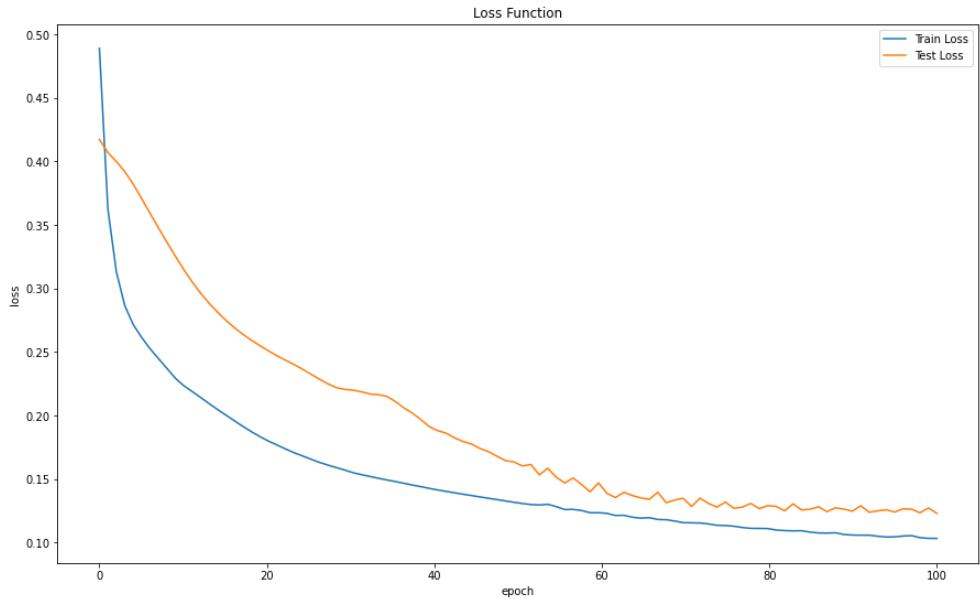


图 20 给定结构的残差网络损失函数曲线图

模型在训练集和测试集上的 SSIM 曲线如图 21 所示，随着训练的迭代，均呈上升趋势（SSIM 值越大，表示图像失真越小）。

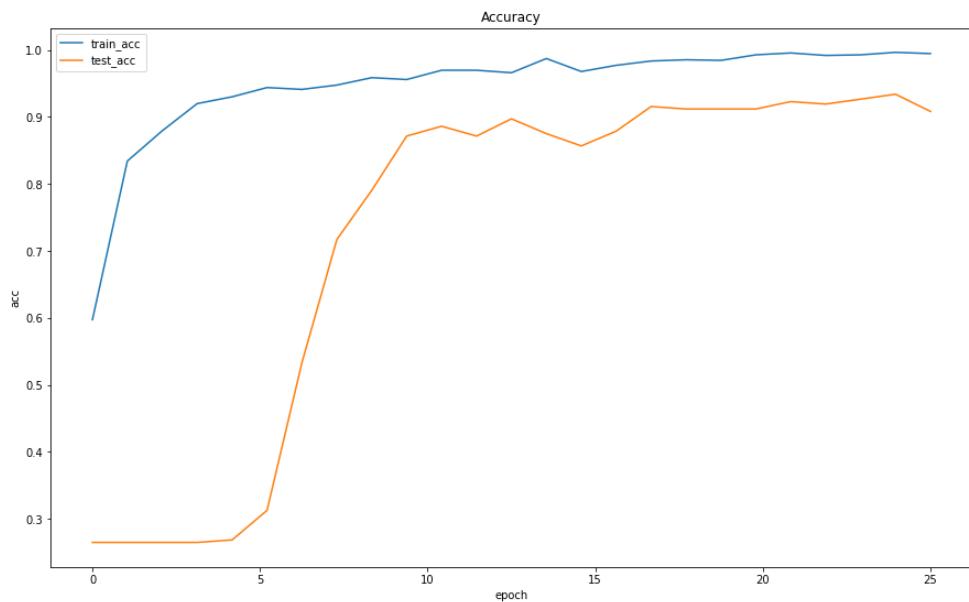


图 21 给定结构的残差网络 SSIM 曲线图

实现的给定结构的残差网络训练 25 个 epoch 在测试集上的 SSIM 最高为 0.9893，所需时间约为 4260.32 秒。

## 六、实验心得体会

通过本次实验，对卷积神经网络的原理和实现有了更扎实的理解和掌握，以及对经典模型 AlexNet 和 ResNet 的结构也有了清晰的认识，解决了我在之前自学过程中的迷惑。除此以外，按照课堂讲授内容对空洞卷积的设计与实现进行具体实验，也充分理解了空洞卷积的设计方法和作用机理，为以后网络设计中空洞卷积的使用做了铺垫。

而在实验报告撰写的过程中，可以对实验的目的、问题和解决方案进行明确的梳理，进一步在编码实验的基础上获得了提升，做到了理论和实践的齐头并进。

## 七、参考文献

- [1] 北京交通大学《深度学习》课程组，实验 3 卷积神经网络实验，北京交通大学《深度学习》课件

## 八、附录

在第五部分的实验结果中，虽然将 `torch.nn` 实现的卷积神经网络和残差网络在去雾数据上进行了训练与测试，但由于计算资源的限制以及数据集中图片大小不一致的问题，在加载数据时统一将图片的大小拉伸到  $32 \times 32$ ，这为图片的质量造成了严重损坏。为了对预测结果进行可视化展示，故借助 Kaggle 提供的计算资源，并将图片的大小拉伸到  $256 \times 256$  进行训练和测试，实验结果如下。

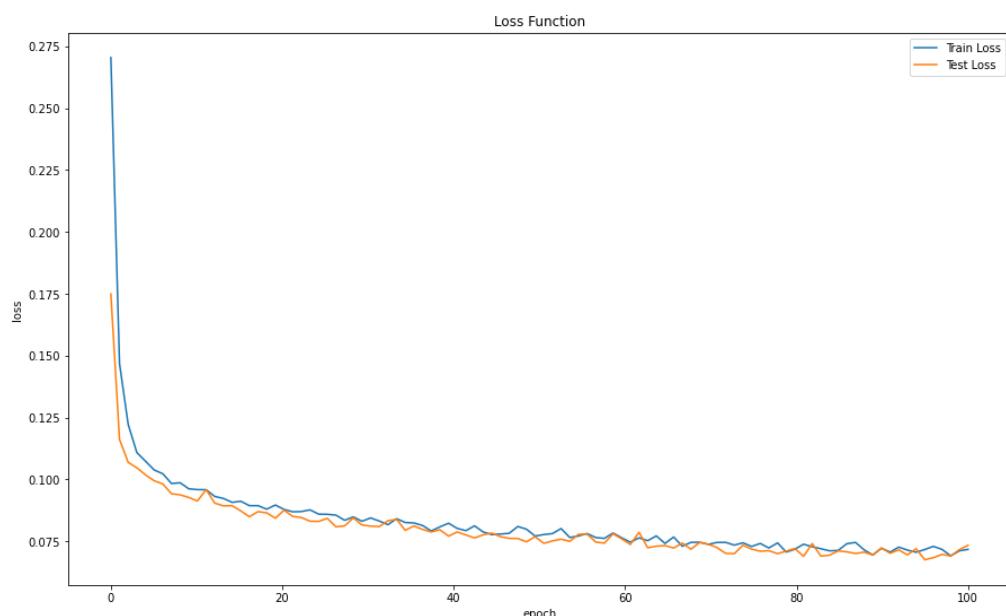


图 22 `torch.nn` 实现卷积神经网络损失函数曲线图

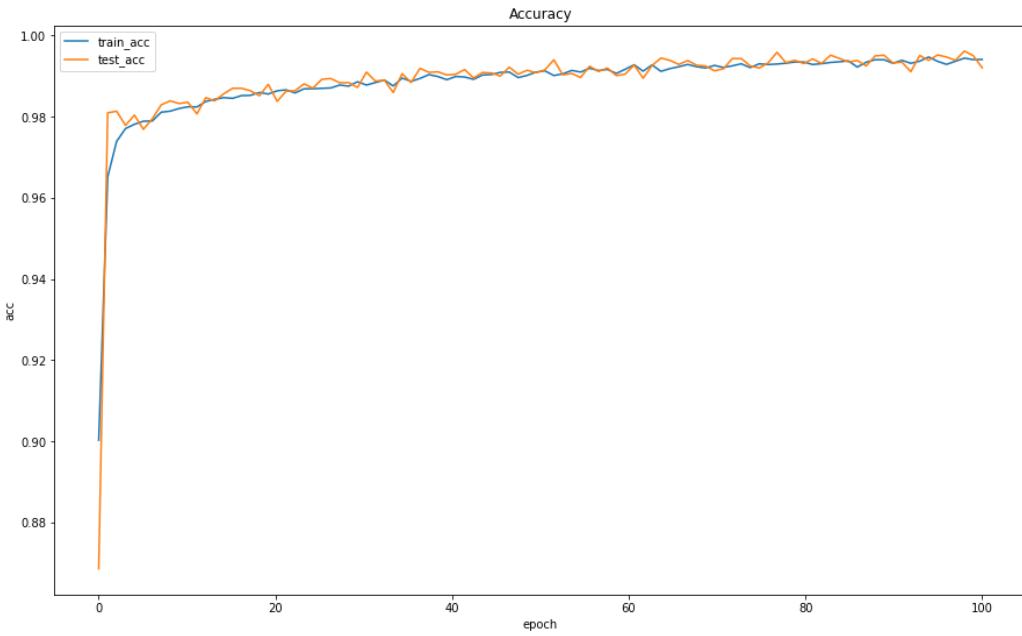


图 23 torch.nn 实现卷积神经网络 SSIM 曲线图

在其他参数保持不变的情况下，去雾数据集图像大小拉伸至  $256 \times 256$ ，  
torch.nn 实现卷积神经网络经过 100 个 epoch 的迭代后，SSIM 最高可达 0.9994。

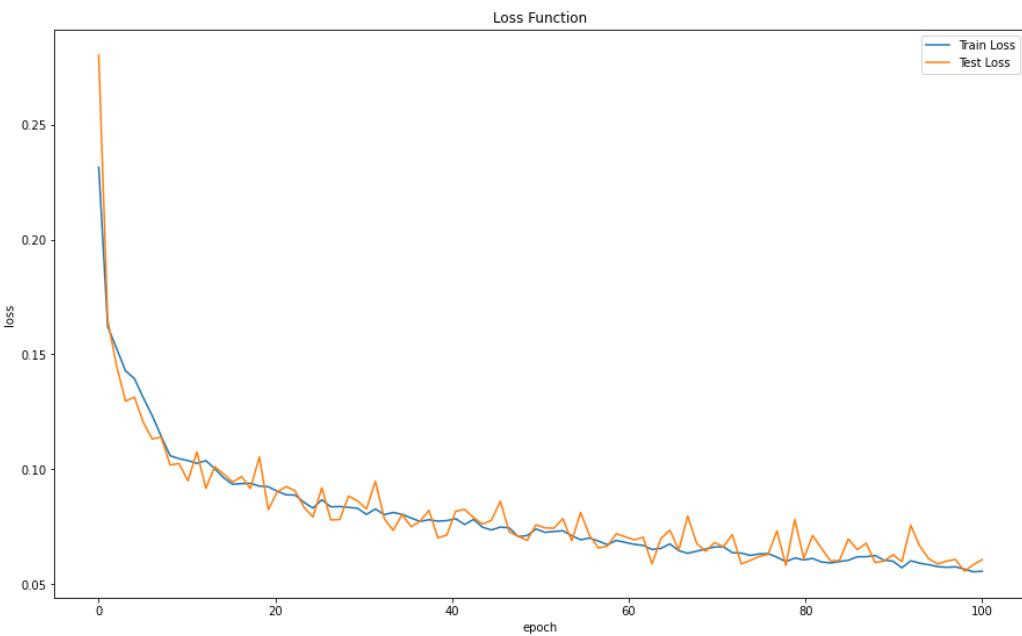


图 24 给定结构的残差网络损失函数曲线图

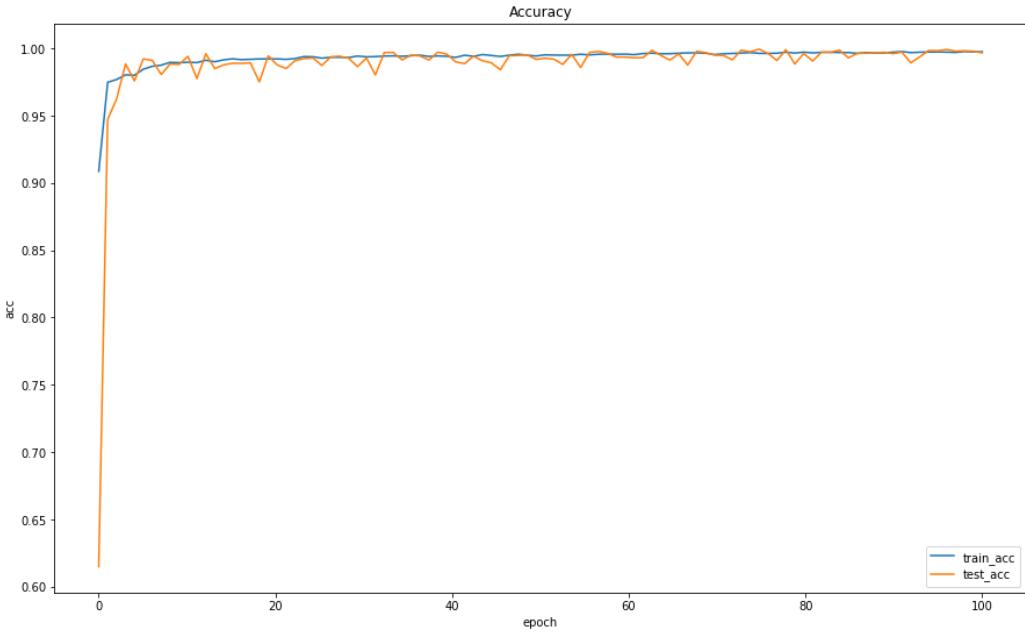


图 25 给定结构的残差网络预测准确率曲线图

在其他参数保持不变的情况下，去雾数据集图像大小拉伸至  $256 \times 256$ ，给定结构的残差网络经过 100 个 epoch 的迭代后，SSIM 最高可达 0.9994。

预测结果如下（部分）。从实验结果来看，残差网络能取得更好的可视化结果，但由于对图像进行了下采样，实验结果仍存在一定的失真。

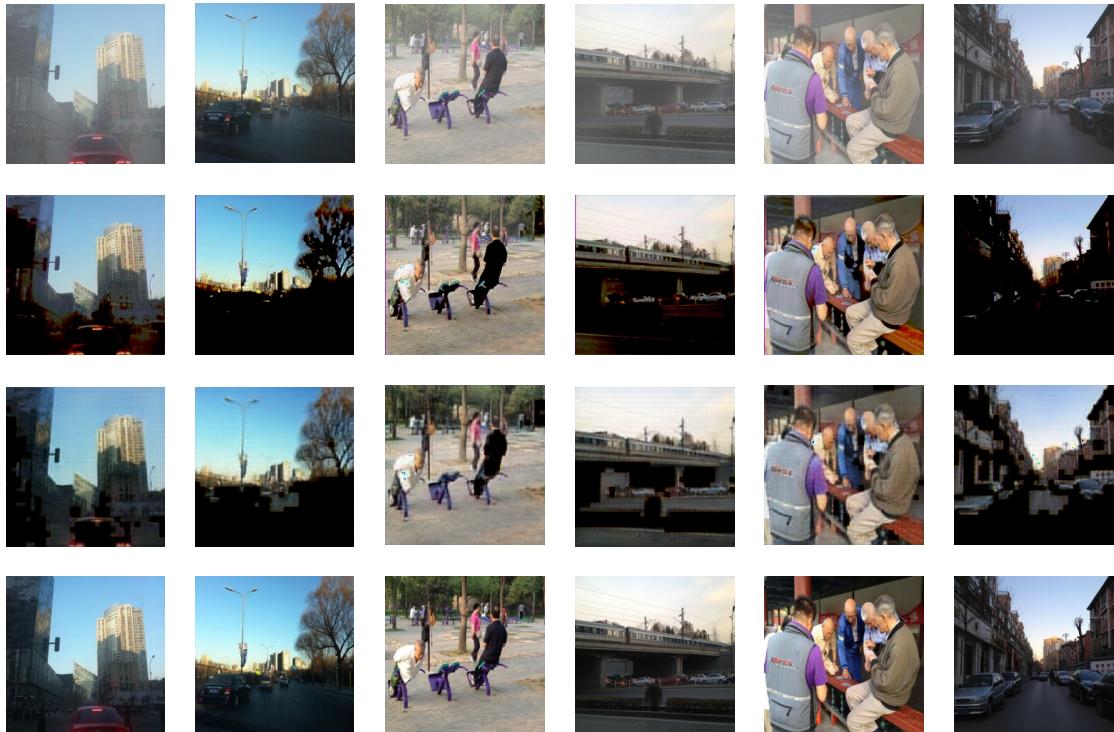


图 26 torch.nn 实现的卷积神经网络和残差网络去雾结果（部分）实验图。第一行：原始数据集图像；第二行：torch.nn 实现的卷积神经网络去雾结果；第三行：残差网络去雾结果；第四行：ground truth