研一周报

周报时间：2024.9.2-2.24.9.6

一、当前任务

学习卷积神经网络

二、本周工作

1.通过视频学习CNN的基本结构：卷积、池化、全连接；学习典型的网络结构：AlexNet、VGG、GoogleNet、ResNet。

2.通过Colab平台进行代码练习。

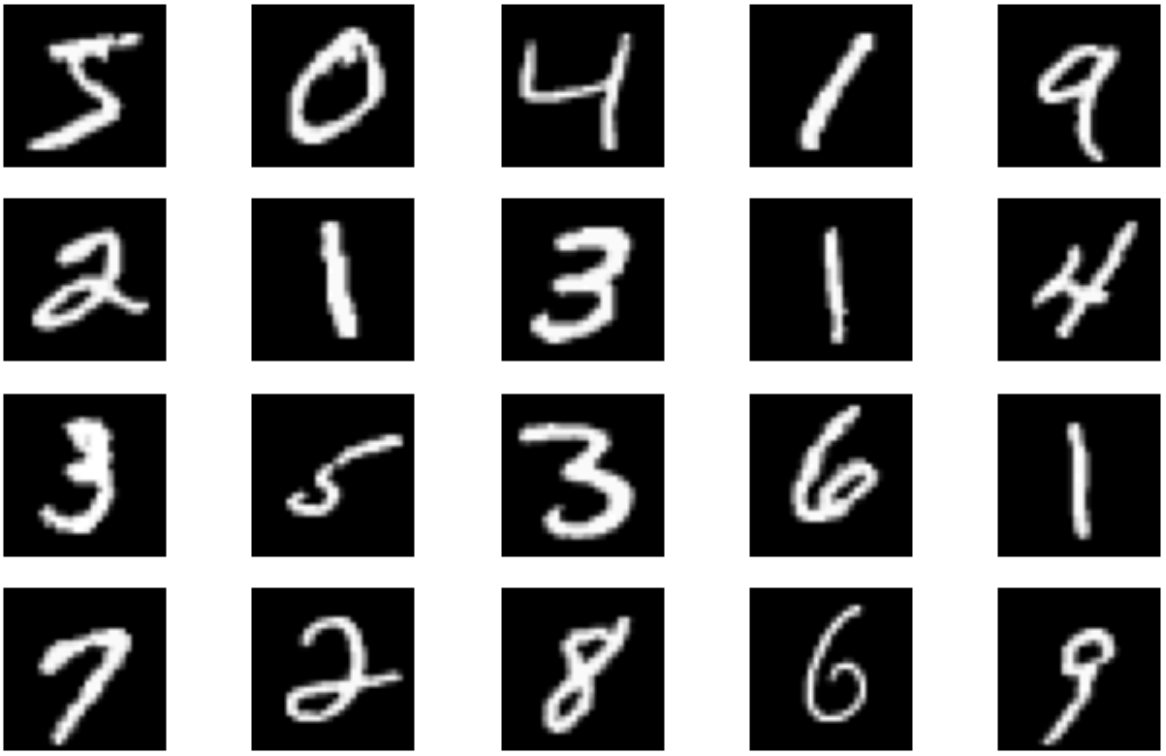
2.1MNIST数据集分类

```
# 数据集加载
input_size = 28*28
output_size = 10

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=True, download=True,
        transform=transforms.Compose(
            [transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081,))])),
            batch_size=64, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=False, transform=transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081,))])),
            batch_size=1000, shuffle=True)
```

```
plt.figure(figsize=(8, 5))
for i in range(20):
    plt.subplot(4, 5, i + 1)
    image,_ = train_loader.dataset.__getitem__(i)
    plt.imshow(image.squeeze().numpy(),'gray')
    plt.axis('off');
```



```
# 仅使用全连接层和激活函数的模型
class FC2Layer(nn.Module):
    def __init__(self, input_size, n_hidden, output_size):
        super(FC2Layer, self).__init__()
        self.input_size = input_size
        self.network = nn.Sequential(
            nn.Linear(input_size, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, n_hidden),
            nn.ReLU(),
            nn.Linear(n_hidden, output_size),
            nn.LogSoftmax(dim=1)
        )
    def forward(self, x):
        x = x.view(-1, self.input_size)
        return self.network(x)

# 使用CNN的卷积神经网络模型
class CNN(nn.Module):
    def __init__(self, input_size, n_feature, output_size):
        super(CNN, self).__init__()
        self.n_feature = n_feature
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=n_feature,
kernel_size=5)
        self.conv2 = nn.Conv2d(n_feature, n_feature, kernel_size=5)
        self.fc1 = nn.Linear(n_feature*4*4, 50)
        self.fc2 = nn.Linear(50, 10)
    def forward(self, x, verbose=False):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2)
        x = self.conv2(x)
        x = F.relu(x)
```

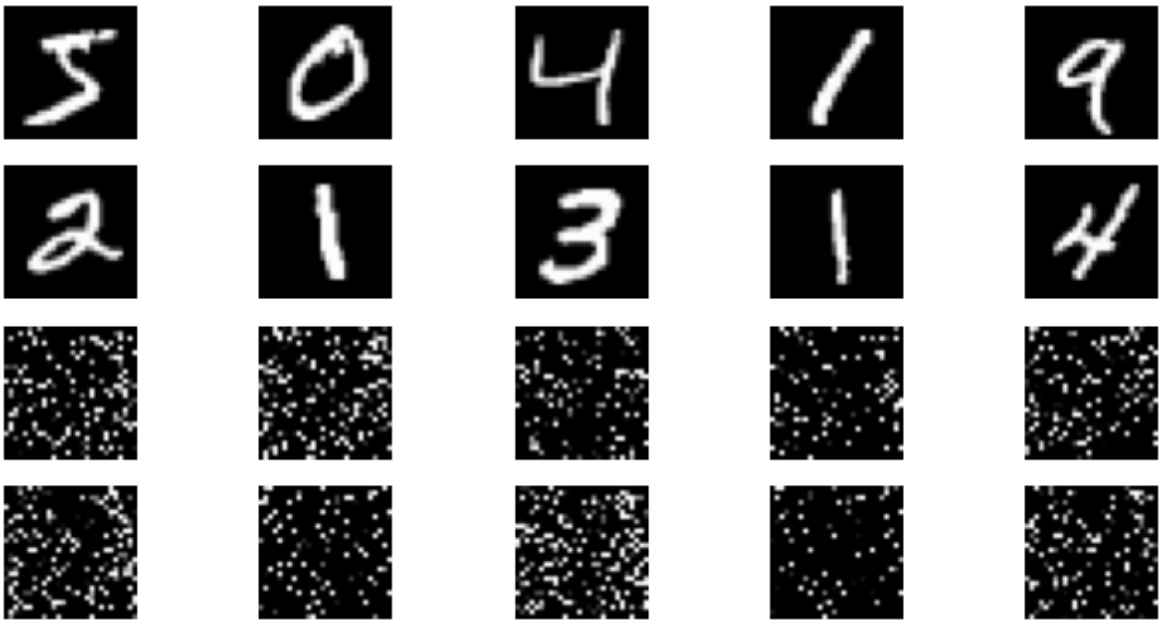
```
        x = F.max_pool2d(x, kernel_size=2)
        x = x.view(-1, self.n_feature*4*4)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.log_softmax(x, dim=1)
        return x
```

经过训练后对两个模型得到的准确率如下，可以看到卷积神经网络的准确率更高。

Test set: Average loss: 0.4438, Accuracy: 8711/10000 (87%)

Test set: Average loss: 0.1506, Accuracy: 9537/10000 (95%)

```
# torch.randperm 函数，给定参数n，返回一个从0到n-1的随机整数排列
perm = torch.randperm(784)
#打乱图像的像素分布
plt.figure(figsize=(8, 4))
for i in range(10):
    image, _ = train_loader.dataset.__getitem__(i)
    # permute pixels
    image_perm = image.view(-1, 28*28).clone()
    image_perm = image_perm[:, perm]
    image_perm = image_perm.view(-1, 1, 28, 28)
    plt.subplot(4, 5, i + 1)
    plt.imshow(image.squeeze().numpy(), 'gray')
    plt.axis('off')
    plt.subplot(4, 5, i + 11)
    plt.imshow(image_perm.squeeze().numpy(), 'gray')
    plt.axis('off')
```



```
# 对每个 batch 里的数据，打乱像素顺序的函数
def perm_pixel(data, perm):
    # 转化为二维矩阵
    data_new = data.view(-1, 28*28)
    # 打乱像素顺序
    data_new = data_new[:, perm]
    # 恢复为原来4维的 tensor
    data_new = data_new.view(-1, 1, 28, 28)
    return data_new
```

对于对像素随机打乱后的图像，依然使用两个模型进行训练并进行准确率评估

Test set: Average loss: 0.4396, Accuracy: 8660/10000 (87%)

Test set: Average loss: 0.5921, Accuracy: 8159/10000 (82%)

可以看到，像素打乱后，对全连接网络性能影响不大，但是卷积神经网络的性能大幅度下降了，这是因为卷积神经网络会利用像素的局部关系，而打乱后，局部关系无法利用。

2.2 使用CNN处理CIFAR10 数据集分类

```
def imshow(img):
    plt.figure(figsize=(8,8))
    img = img / 2 + 0.5     # 转换到 [0,1] 之间
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# 得到一组图像
images, labels = next(iter(trainloader))
# 展示图像
imshow(torchvision.utils.make_grid(images))
# 展示第一行图像的标签
for j in range(8):
    print(classes[labels[j]])
```

car
deer
dog
plane
dog
bird
cat
deer

```
# 定义网络
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
```

```
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)



# 得到一组图像
images, labels = next(iter(testloader))
# 展示图像
imshow(torchvision.utils.make_grid(images))
# 展示图像的标签
for j in range(8):
    print(classes[labels[j]])
```


```
cat
ship
ship
plane
frog
frog
car
frog
```

```
outputs = net(images.to(device))
_, predicted = torch.max(outputs, 1)
```

```
# 展示预测的结果
for j in range(8):
    print(classes[predicted[j]])
```

```
cat
plane
ship
plane
cat
frog
car
frog
```

```
# 测试训练出的模型性能
correct = 0
total = 0

for data in testloader:
    images, labels = data
    images, labels = images.to(device), labels.to(device)
    outputs = net(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()
print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct
/ total))
```

Accuracy of the network on the 10000 test images: 63 %

2.3使用 VGG16 对 CIFAR10 分类

```
# VGG16 模型的定义
class VGG(nn.Module):
    def __init__(self):
        super(VGG, self).__init__()
        self.cfg = [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512,
'M']
        self.features = self._make_layers(self.cfg)
        self.classifier = nn.Linear(512, 10)

    def forward(self, x):
        out = self.features(x)
        out = out.view(out.size(0), -1)
        out = self.classifier(out)
        return out

    def _make_layers(self, cfg):
        layers = []
        in_channels = 3
        for x in cfg:
            if x == 'M':
                layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                           nn.BatchNorm2d(x),
                           nn.ReLU(inplace=True)]
                in_channels = x
        layers += [nn.AvgPool2d(kernel_size=1, stride=1)]
        return nn.Sequential(*layers)
```

```
#在性能进行评估经过训练后，对模型
correct = 0
total = 0

for data in testloader:
    images, labels = data
    images, labels = images.to(device), labels.to(device)
```

```
    outputs = net(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %.2f %%' % (
    100 * correct / total))
```

Accuracy of the network on the 10000 test images: 83.39 %

可以看到，一个简化版的VGG16网络可以将识别的正确率从63%提升到了83%，正确率大幅度提高。

3.问题思考：

1. dataloader 里面 shuffle 取不同值有什么区别?

在shuffle = True时，DataLoader会将数据集打乱再生成小批次数据，常用于训练阶段，有助于减少模型的过拟合，提高模型的泛化能力；防止模型对数据的顺序产生依赖，尤其是数据顺序存在潜在结构时。

在shuffle = False时，数据按照原有顺序读取，不会打乱，常用用于测试集和验证集，可以按照原始顺序以便分析结果。在某些数据顺序重要的任务，比如时间序列预测中使用。

2. transform 里，取了不同值，这个有什么区别?

transforms.Normalize(mean, std)用于对图像的每个通道进行标准化处理。不同的mean和std值会影响图像数据的分布，从而对模型训练产生影响。mean时每个通道的均值，用于中心化图像数据，通常根据RGB图像的每个通道计算独立的均值。std是每个通道的标准差，用于调整像素值的尺度。

3. epoch 和 batch 的区别?

epoch是整个训练数据集在模型中训练的次数。batch是每轮遍历中，将数据集分批次传入模型，batch是每一批次的数据数量。

4. 1x1的卷积和 FC 有什么区别？主要起什么作用?

1×1的卷积针对每个输入位置的多个通道做一次线性组合，相当于改变特征图的通道数，而不会改变特征图的空间维度。FC层每个神经元与上一层的所有神经元相连，接受一个一维数据输入，并输出一个一维数据。

1×1的卷积核可以用来减少通道数，从而减少计算量；与ReLU等激活函数结合，引入非线性；通过对不同通道进行加权求和，能够在不同通道之间融合信息。FC可以将特征组合起来，用于分类任务等，通常将特征映射到类的概率分布（如Softmax输出）。

5. residual leanring 为什么能够提升准确率?

Residual learning可以缓解梯度消失问题；避免模型退化；更容易优化；提高准确率。

6. 代码练习二里，网络和1989年 Lecun 提出的 LeNet 有什么区别?

代码练习二中输入通道是彩色图像（3个通道，一般为RGB图像），LeNet输入通道仅有1个通道，为灰度图；代码练习二中使用的最大池化，LeNet使用平均池化；代码练习二中激活函数使用ReLU函数，LeNet使用Sigmoid或tanh激活函数。

7. 代码练习二里，卷积以后feature map 尺寸会变小，如何应用 Residual Learning?

Residual Learning在shortcut中采用1x1的卷积,设置步长为2；1x1在卷积的时候设置输出维度为残差连接相同的维度，上述操作也被称为线性变换，进而对大小不同的feature map进行调整。

8. 有什么方法可以进一步提升准确率?

使用特征更明显、分类更合理的数据集，对数据预处理；选择性能最优的神经网络结构；损失函数、激活函数。

下周计划

1.阅读论文和观看视频学习ResNet网络和ResNeXt网络。

2.使用ResNet网络实现猫狗分类。