清华大学电子工程系

媒体与认知 课堂 2

2022-2023 学年春季学期

作业 2

卜丞科 2023 年 4 月 23 日

理论部分

1 单选题（15 分）

1.1 D

1.2 B

1.3 C

1.4 D

1.5 A

2 计算题（15 分）

2.1 **已知某卷积层的输入为 X(该批量中样本数目为 1，输入样本通道数为 1)，采用一个卷积核 W，即卷积输出通道数为 1，卷积核尺寸为 2×2，卷积的步长为 1，无边界延拓，偏置量为 b：**

$$X=\begin{bmatrix} -0.5 & 0.2 & 0.3 \\ 0.6 & -0.4 & 0.1 \\ 0.4 & 0.5 & -0.2 \end{bmatrix}, W=\begin{bmatrix} -0.2 & 0.1 \\ 0.4 & -0.3 \end{bmatrix}, b=0.05$$

2.1.1 请计算卷积层的输出 Y。

$$
\begin{aligned}
Y &= X * W + b \\
&= \begin{bmatrix} 0.48 & -0.2 \\ -0.15 & 0.35 \end{bmatrix} + b \\
&= \begin{bmatrix} 0.53 & -0.15 \\ -0.1 & 0.4 \end{bmatrix}
\end{aligned}
$$

2.1.2 若训练过程中的目标函数为 L，且已知 $\frac{\partial L}{\partial Y} = \begin{bmatrix} 0.1 & -0.2 \\ 0.2 & 0.3 \end{bmatrix}$，请计算 $\frac{\partial L}{\partial X}$。

注：本题的计算方式不限，但需要提供计算过程以及各步骤的结果。

2.1.2

设 $Y' = \begin{bmatrix} 0.53 \\ -0.15 \\ -0.1 \\ 0.4 \end{bmatrix}$ $X' = \begin{bmatrix} -0.5 & 0.2 & 0.6 & -0.4 \\ 0.2 & 0.3 & -0.4 & 0.1 \\ 0.6 & -0.4 & 0.4 & 0.5 \\ 0.4 & 0.1 & 0.5 & -0.2 \end{bmatrix}$ $W' = \begin{bmatrix} -0.2 \\ 0.1 \\ 0.4 \\ -0.3 \end{bmatrix}$

$\therefore Y' = X'W' + b$

$\therefore \frac{\partial L}{\partial Y'} = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.2 \\ 0.3 \end{bmatrix}$

$\therefore \frac{\partial L}{\partial X'} = W \cdot \frac{\partial L}{\partial Y'} = \begin{bmatrix} -0.02 & 0.04 & -0.04 & -0.06 \\ 0.01 & -0.02 & 0.02 & 0.03 \\ 0.04 & -0.08 & 0.08 & 0.12 \\ -0.03 & 0.06 & 0.06 & -0.09 \end{bmatrix}$

$\therefore \frac{\partial L}{\partial X} = \begin{bmatrix} -0.02 & 0.05 & -0.02 \\ 0 & -0.15 & 0.09 \\ 0.08 & 0.06 & -0.09 \end{bmatrix}$

编程部分

3 编程作业报告

3.1 代码补全

3.1.1 datasets.py

```python
data_transforms = [
    transforms.Resize(image_size),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
]

# You should insert some data augmentation techniques to `data_transforms` when `augment` is True
# for the training dataset.
# Consider what is an appropriate data augmentation technique for traffic sign classification.
if mode == 'train' and augment:
    data_transforms.append([transforms.RandomApply([transforms.RandomRotation((-90, 90))], p=0.5)])

# Else, the `data_transforms` should be left unchanged
# <<< TODO 1.1
# Use `transforms.Compose` to compose the list of transforms into a single transform
data_transforms = transforms.Compose(data_transforms)

# >>> TODO 1.2: Define the dataset.
# You should build the path to the selected dataset according to the `mode` parameter,
# and use the `ImageFolder` class from `torchvision.datasets` to load the datasets.
# Docs: https://pytorch.org/vision/stable/generated/torchvision.datasets.ImageFolder.html
# The `ImageFolder` class takes in the path to the dataset and the transform to apply to the images.
# The `ImageFolder` class will automatically load the images and labels for you.
dataset = ImageFolder(data_root, transform=data_transforms)
# <<< TODO 1.2

# >>> TODO 1.3: Define the data loader.
# You should set the `shuffle` parameter to `True` when `mode=='train'`, and `False` otherwise.
loader = DataLoader(dataset, shuffle = (mode=='train'), batch_size = batch_size, num_workers = num_workers)
# <<< TODO 1.3

return loader
```

3.1.2 networks.py

```
    self.conv1 = nn.Sequential(nn.Conv2d(in_channels, 32, 5, stride=1, padding=2), bn2d(32), nn.ReLU())
    self.conv2 = nn.Sequential(nn.Conv2d(32, 64, 5, stride=2, padding=2), bn2d(64), nn.ReLU())
    self.pool1 = nn.MaxPool2d(2, stride=2, padding=0)
    self.conv3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding=1), bn2d(64), nn.ReLU())
    self.conv4 = nn.Sequential(nn.Conv2d(64, 128, 3, stride=1, padding=1), bn2d(128), nn.ReLU())
    self.pool2 = nn.MaxPool2d(2, stride=2, padding=0)
    self.conv5 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding=1), bn2d(128), nn.ReLU())
    self.dropout = nn.Dropout(p = dropout_prob)
    # <<< TODO 2.1

    # >>> TODO 2.2: complete a sub-network with two linear layers by using nn.Sequential function
    # Hint: note that the size of input images is (3, 32, 32) by default, what is the size of
    # the output of the convolution layers?
    # Network structure:
    #          out_channels
    # linear       256
    # activation
    # batchnorm
    # dropout(p), where p is input parameter of dropout ratio
    # linear    num_class
    self.fc_net = nn.Sequential(
        nn.Linear(4*4*128, 256),
        nn.ReLU(),
        bn1d(256),
        self.dropout,
        nn.Linear(256, 26)

    )
    # <<< TODO 2.2
```

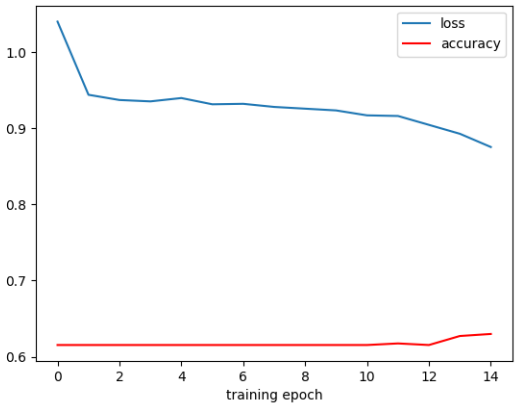
```
# Step 1: apply spatial transformer network if applicable
x0 = self.stn(x)

# >>> TODO 2.3: forward process
# Step 2: forward process for the convolutional layers, apply residual connection in conv3 and conv5
x1 = self.conv1(x0)
x2 = self.conv2(x1)
x3 = self.pool1(x2)
x4 = self.conv3(x3) + x3
x5 = self.conv4(x4)
x6 = self.pool1(x5)
x7 = self.conv5(x6) + x6
x7 = self.dropout(x7)

# Step 3: use `Tensor.view()` to flatten the tensor to match the size of the input of the
# fully connected layers.
x7 = x7.view(-1, 4*4*128)
# Step 4: forward process for the linear layers
out = self.fc_net(x7)
# <<< TODO 2.3

return out
```

```python
# >>> TODO 3.1: Build your localization net
# Step 1: Build a convolutional network to extract features from input images.
# Hint: Combine convolutional layers, batch normalization layers and ReLU activation functions to build
# this network.
# Suggested structure: 3 layers of down-sampling convolution (e.g. each layer reduces the feature map
# size by half), double the number of channels in each layer, use BN and ReLU.
self.localization_conv = nn.Sequential(
        nn.Conv2d(in_channels, in_channels*2, 3, stride=2),
        nn.BatchNorm2d(in_channels*2),
        nn.ReLU(),
        nn.Conv2d(in_channels*2, in_channels*4, 3, stride=2),
        nn.BatchNorm2d(in_channels*4),
        nn.ReLU(),
        nn.Conv2d(in_channels*4, in_channels*8, 3, stride=2),
        nn.BatchNorm2d(in_channels*8),
        nn.ReLU(),
)

# Step 2: Build a fully connected network to predict the parameters of affine transformation from
# the extracted features.
# Hint: Combine linear layers and ReLU activation functions to build this network.
# Suggested structure: 2 linear layers with one BN and ReLU.
self.localization_fc = nn.Sequential(
    nn.Linear(3*3*8*in_channels, 64),
    nn.ReLU(),
    nn.BatchNorm1d(64),
    nn.Linear(64, 6)
)
# <<< TODO 3.1

# >>> TODO 3.2: Initialize the weight/bias of the last linear layer of the fully connected network
# Hint: The STN should generate the identity transformation by default before training.
# How to initialize the weight/bias of the last linear layer of the fully connected network to
# achieve this goal?
self.localization_fc[-1].weight.data.zero_()
self.localization_fc[-1].bias.data.copy_(torch.tensor([1, 0, 0, 0, 1, 0], dtype=torch.float))
# <<< TODO 3.2
```

3.2 实验评价

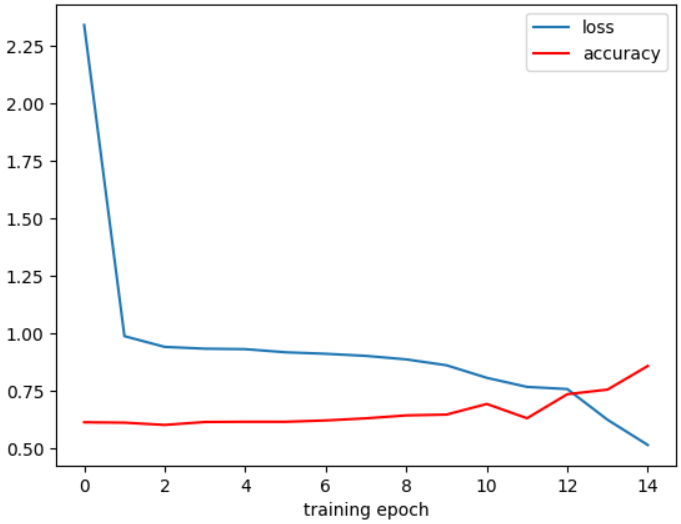
3.2.1 探究 batch normalization 和 dropout 的作用

使用默认配置（不启用 BN 和 dropout），训练 baseline 模型：

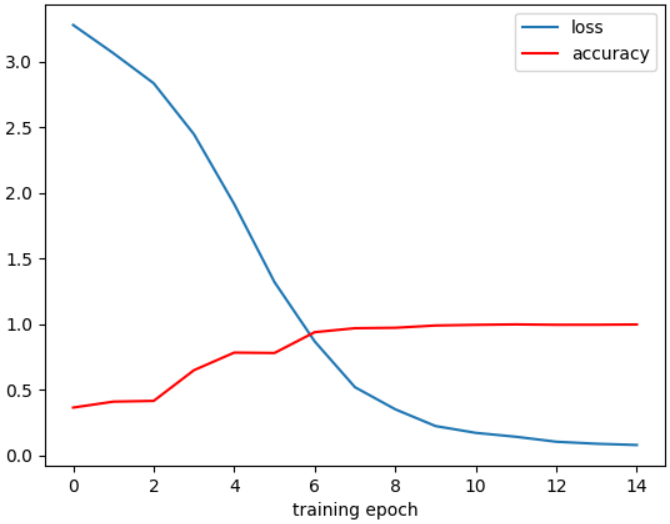


可以发现此时训练效果比较差，loss 值下降非常慢，虽然第一次训练之后准确率就超过了百分之 60 但是之后几乎没有提高。

启用 batch normalization：



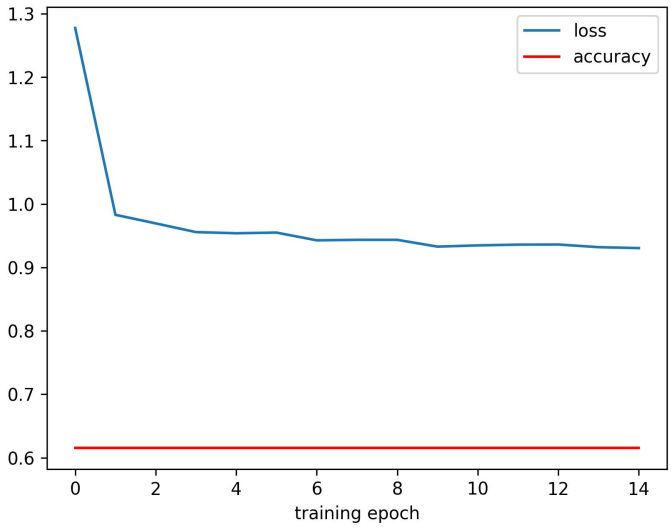
可以发现此时训练效果有所改进，loss 值在一开始下降迅速，准确率逐步上升提高至百分之 85.6。但是观察到准确率在中途有所降低，猜测降低学习率可能有有所好转，于是把学习率从 1e-3 调整为 1e-4。



调整以后结果非常惊人，准确率达到了离谱的百分之 99.6，进行测试集测试，结果也非常好。

```
(test) E:\桌面\媒体与认知\HW2\src>python test.py --ckpt_path checkpoints/bn
[Info] loading checkpoint from checkpoints/bn\ckpt_epoch_15.pth ...
accuracy on the test set: 0.996
```

启用 dropout 并设置概率为 0.3：

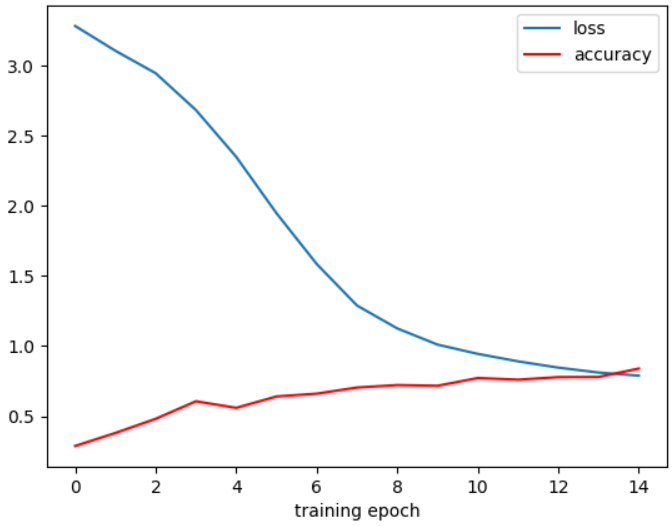


效果并不是很好，修改了学习率依然没有变化，可能是因为 dropout 的主要用途是防止过拟合，但是在一定程度上也会影响学习，所以加入 dropout 以后学习之后准确率并不理想。

3.2.2 探究数据增广的作用

我选择了将图片随机旋转一个角度，因为交通图标不是上下对称或左右对称，使用对称方法增广并不符合图片特性，但是图片有的会偏移一个角度，可以选择旋转一个角度来进行数据增广。

随机顺时针或逆时针旋转 90 度，下图已经是已知最好的一个模型，但是效果并不是很好。

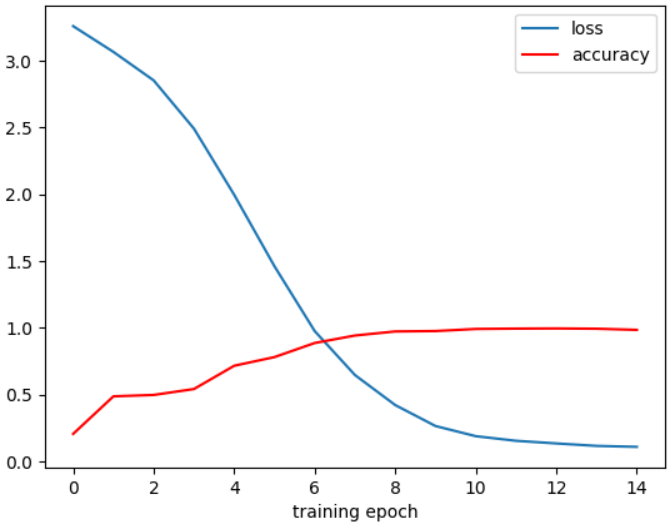



```
(test) E:\桌面\媒体与认知\HW2\src>python test.py --ckpt_path checkpoints/bn_aug
[Info] loading checkpoint from checkpoints/bn_aug\ckpt_epoch_15.pth ...
accuracy on the test set: 0.840
```

观察了数据集中的图像，但是大部分图片都比较正，只是有时候视角在侧面，所以左右旋转在这个数据集上可能效果确实不是很好。

3.2.3 探究空间变换网络 (STN) 的作用

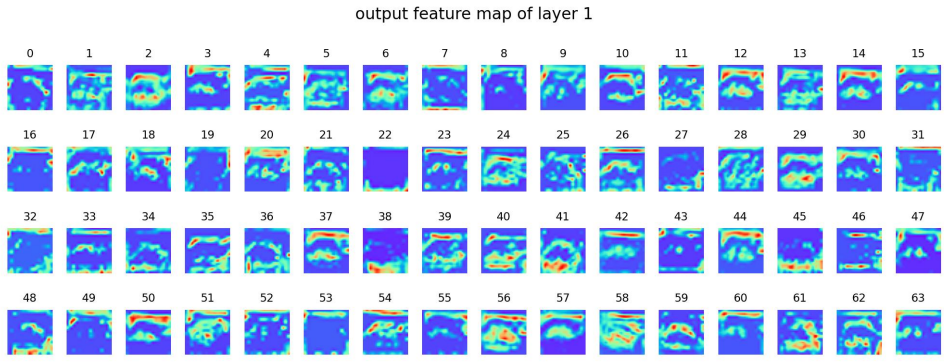
使用空间变换网络以后，效果确实很好，正确率超过百分之 98




```
(test) E:\桌面\媒体与认知\HW2\src>python test.py --ckpt_path checkpoints/stn
[Info] loading checkpoint from checkpoints/stn\ckpt_epoch_15.pth ...
accuracy on the test set: 0.986
```

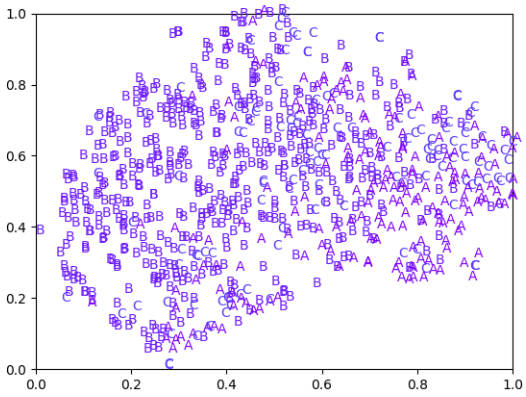
3.2.4 可视化

可视化第 10 张图像第 1 层卷积层的输出特征图:



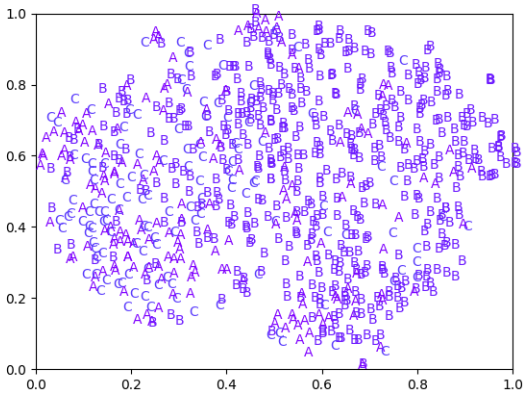
观察到每个图片都有一部分符合原图片的特征，可能意味着这张图片学习了这个图片这方面的特性。

t-SNE 可视化最后一层隐藏层的输出特征：



A 大约在图片的右方偏下一点，B 大约在图片的左边，C 大约在图片的右边偏上一点，总体上出现了一定的聚集。

可视化 STN 学习到的变换：



A 大约在图片的上下和左边，B 大约在图片的右边，C 大约在图片的左边，总体上出现了一定的聚集。