清华大学电子工程系

媒体与认知 课堂 2

2021-2022 学年春季学期

作业 3

孙一 2022 年 4 月 12 日

理论部分

1 单选题（15 分）

1.1 D

1.2 B

1.3 C

1.4 C

1.5 B

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。

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。

答案见图 1

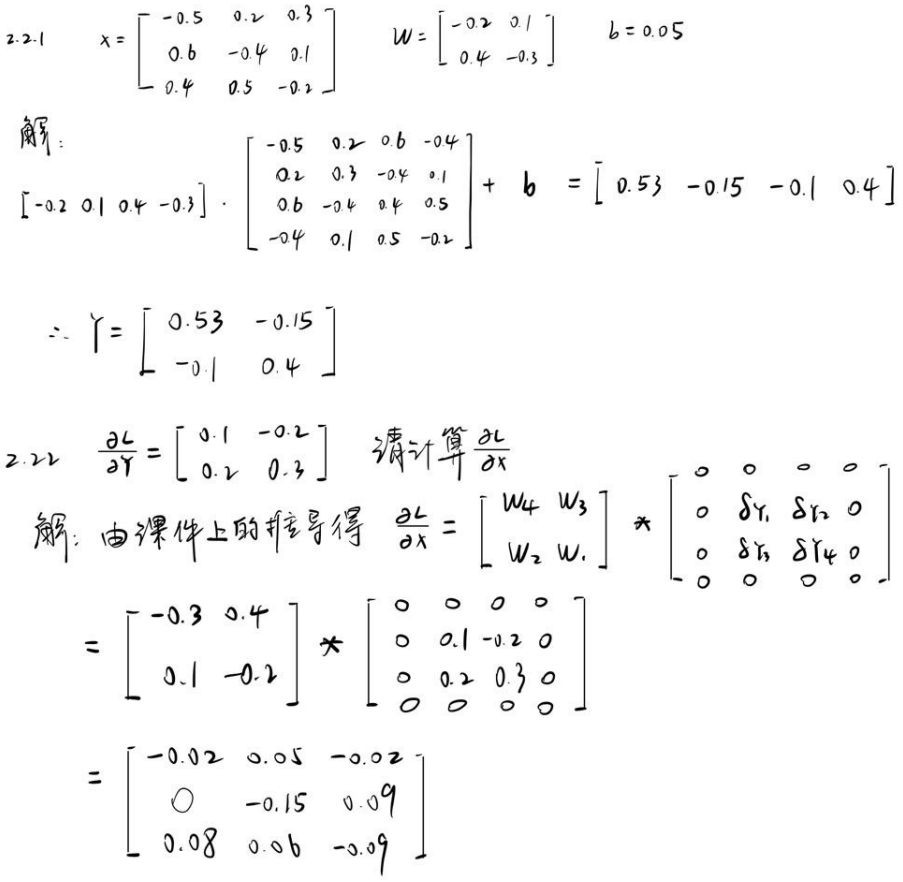

图 1: 计算题

编程部分

3 编程作业报告

3.1 目的

使用卷积神经网络完成识别英文字符图像（非线性分类）任务，在本任务中，我们将使用卷积神经网络识别英文字符图像，即输入一张图像，模型输出识别结果。请注意图像样本包括大小写英文字母，识别时不区分大小写形式，即大写字母“A”和小写字母“a”都对应于一个类别，输出时转为大写字母“A”显示。

3.2 环境配置

本次实验中我采用的各 package 版本如下：

```
python==3.8.8 numpy==1.19.5
pytorch==1.9.0 torchvision==0.10.0
opencv==4.0.1 matplotlib==3.5.0
```

3.3 一些定义

字母	含义
b	batch_size
o	out_channels
c	in_channels*kernel_h*kernel_w
n	out_h*out_w

3.4 完成卷积层输出特征图尺寸计算——conv.py 文件

```
# TODO 1.1: calculate the height and width of output feature maps via
# input_h/input_w, kernel_h/kernel_w, stride_h/stride_w and padding_h/padding_w.
out_h = (input_h - kernel_h + 2*padding_h)//stride_h + 1    # 注意用双斜杠代表下取整
out_w = (input_w - kernel_w + 2*padding_w)//stride_w + 1
# end TODO 1.1
```

3.5 完成卷积层的前向计算——network.py 文件

```
# TODO 1.2: calculate the results of 2D convolution via img2col and matrix multiplica
# step 1: adopt torch.nn.functional.unfold function to
# transfer images into columns. This step is also called img2col.
# The output size of unfold function is  (b,c,n)
x_cols = nn.functional.unfold(input, kernel_size=kernel_size,
padding= padding, stride = stride)

# step 2: reshape weights and calculate W * x based on the
# unfolded x_cols, weight_re via torch.matmul()
# reshape the size of weights from [o, in_channels,kernel_h,kernel_w] to (o,c)
weight_re = weight.reshape(out_channels, -1)
# Hint: the size of x_cols is [b,c,n]
# the size of output is [b,o,n]
```

```
output = torch.matmul(weight_re, x_cols)
# 用了广播机制，weight_re 最前面加了代表 batch_size 的新维度 b
# 关于 torch.matmul 的广播机制可见
# https://pytorch.org/docs/stable/generated/torch.matmul.html#torch.matmul

# step 3: reshape bias and calculate the final result of 2D convolution W * x + b
# the size of output is (b,o,n)
# reshape the size of bias from [o] to [1, o, 1]
# 因为这里自动广播会出问题
bias_re =  bias.reshape(1, out_channels ,1)
output = output + bias_re
# Hint: we expect that the size of result is [b, o, out_h, out_w]
output = output.reshape(batch_size, out_channels, out_h, out_w)
# end TODO 1.2
```

3.6 完成卷积层的反向计算——train.py 文件

```
# TODO 1.3: calculate dL/dW, dL/db and dL/dx.
# step 1: reshape output_grad (dL/dy) before matrix multiplication
# reshape the size of dL/dy from [b, o, out_h, out_w] to [b, o, n]
output_grad_re = output_grad.reshape(batch_size, out_channels, -1)

# step 2: calculate dL/dW based on the unfolded x_cols and
# output_grad_re via torch.matmul().
# Hint: the size of x_cols is (b,c,n)
# the size of output_grad_re is (b,o,n)
# the size of dL/dW (i.e. W_grad) is the same as W_re which is (b,o,c).sum(dim=0)
W_grad = torch.matmul(output_grad_re, x_cols.transpose(1,2)).sum(0)
# Hint: we expect that the size of W_grad is (o, in_channels, kernel_h, kernel_w)

# step 3: calculate dL/db based on output_grad via torch.sum().
# the size of output_grad_re is [b, o, n]
# the size of dL/db (i.e. b_grad) is [out_channels,]
b_grad = output_grad_re.sum(-1).sum(0)

# step 4: calculate dL/d(x_cols) based on output_grad and weight_re
# via torch.matmul() and obtain dL/dx via torch.nn.functional.fold
```

```
# Hint: the size of output_grad_re is (b,o,n)
# reshape the size of weight from [o, in_channels, kernel_h, kernel_w] to (o,c)
weight_re = weight.reshape(out_channels, -1)
# the size of dL/d(x_cols) is (b,c,n)
x_cols_grad = torch.matmul(weight_re.transpose(0,1), output_grad_re)
# Use torch.nn.functional.fold function to transfer the size of x_cols_grad from
# [b c, o] to [b, in_channels, input_h, input_w].
# Note that kernel_size, stride and padding are Tensor here, use tuple(x.numpy()) to
# transfer them into tuple as parameters for fold function.
x_grad = nn.functional.fold(x_cols_grad, tuple(input_size.numpy()), kernel_size =
tuple(kernel_size.numpy()), padding =
tuple(padding.numpy()), stride=tuple(stride.numpy()))
# fold 函数的第二个参数应该是 [input_h, input_w]
# end TODO 1.3
```

3.7 完成卷积层参数的初始化——train.py 文件

```
# TODO 1.4: initialize weights and bias of the 2D convolution layer and
# set W and b trainable parameters
self.W = Parameter(torch.randn(self.out_channels, self.in_channels,
self.kernel_size[0], self.kernel_size[1]))
self.b = Parameter(torch.zeros(self.out_channels))
# End TODO 1.4
```

3.8 完成卷积神经网络中卷积层和池化层的定义——network.py

```
# TODO 2.1: complete a multilayer convolutional neural network with nn.Sequential.
# For convolution layers, please use Conv2d you have completed in conv.py
# input image with size [batch_size, in_channels, img_h, img_w]
# Network structure:
#        kernel_size  stride  padding  out_channels
# conv       5          1        2          32
# batchnorm
# relu
self.conv1 = nn.Sequential(Conv2d(in_channels, 32, 5, 1, 2), bn2d(32), nn.ReLU())
# conv       5          1        2          64
```

```
# batchnorm
# relu
self.conv2 = nn.Sequential(Conv2d(32, 64, 5, 1, 2), bn2d(64), nn.ReLU())
# maxpool    2          2        0
self.pool1 = MaxPool2d(2, 2, 0)
# conv       3          1        1          64
# batchnorm
# relu
self.conv3 = nn.Sequential(Conv2d(64, 64, 3, 1, 1), bn2d(64), nn.ReLU())
# conv       3          1        1          128
# batchnorm
# relu
self.conv4 = nn.Sequential(Conv2d(64, 128, 3, 1, 1), bn2d(128), nn.ReLU())
# maxpool    2          2        0
self.pool2 = MaxPool2d(2, 2, 0)
# conv       3          1        1          128
# batchnorm
# relu
# dropout(p), where p is input parameter of dropout ratio
self.conv5=nn.Sequential(Conv2d(128,128,3,1,1),bn2d(128), nn.ReLU(),nn.Dropout(p))
# end TODO 2.1
```

3.9 完成卷积神经网络中线性层的定义——network.py 文件

```
# TODO 2.2: complete a sub-network with two linear layers by using nn.Sequential.
# Hint: note that the size of input images is (1, 32, 32) by default.
# TODO 2.1 中各层输出图像的 [h,w] 分别为 [32,32],[32,32],[16,16],[16,16],[16,16],
# [8,8], [8,8], 所以输入线性层数据形状为 [batch_size, out_channels = 128, h=8, w=8]
# Network structure:
# linear       (in_features, out_features=256)
# batchnorm 1D (out_features=256)
# activation, i.e. nn.ReLu()
# dropout(p), where p is input parameter of dropout ratio
# linear    (in_features_output_layer, num_class)
self.fc_net = nn.Sequential(
    nn.Linear(128*8*8, 256),
    bn1d(256),
```

```
    nn.ReLU(),
    nn.Dropout(p),
    nn.Linear(256, num_class))
# end TODO 2.2
```

3.10 完成卷积神经网络的计算——network.py 文件

```
# TODO 2.3: forward process
# step 1: forward process for convolutional layers,
# apply residual connection in conv3 and conv5
x1 = self.conv1(x)
x2 = self.conv2(x1)
p1 = self.pool1(x2)
x3 = self.conv3(p1) + p1
x4 = self.conv4(x3)
p2 = self.pool2(x4)
x5 = self.conv5(p2) +p2
# step 2: using Tensor.view() to flatten the tensor so as to match the
# size of input of fully connected layers.
out = x5.view(batch_size, -1)
# step 3: forward process for linear layers
out = self.fc_net(out)
# end TODO 2.3
```

3.11 使用 ImageFolder 构建数据集——train.py 文件

```
# TODO 3.1: use ImageFolder to construct a dataset for image classification task
dataset = ImageFolder(data_path, transform)
# end TODO 3.1
```

4 训练/测试/可视化

4.1 训练及测试

4.1.1 使用默认配置的命令

loss 曲线见图 2，test 准确率为：0.855。

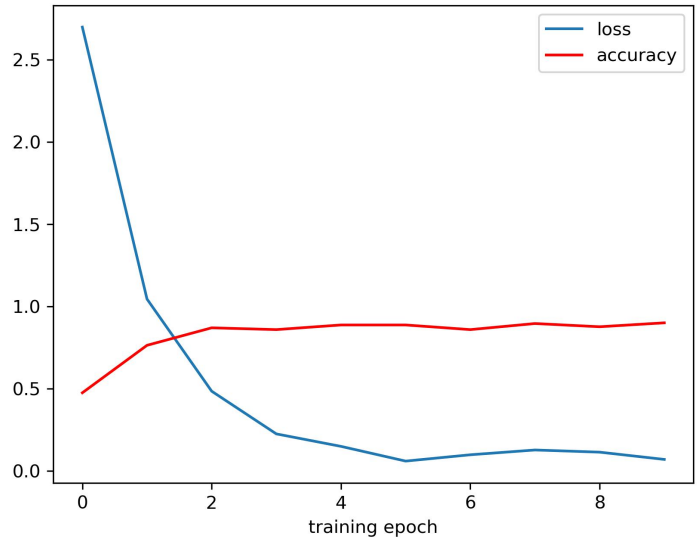

图 2: 默认配置命令

4.1.2 启用 batch normalization 后

loss 曲线见图 3，test 准确率为：0.920。

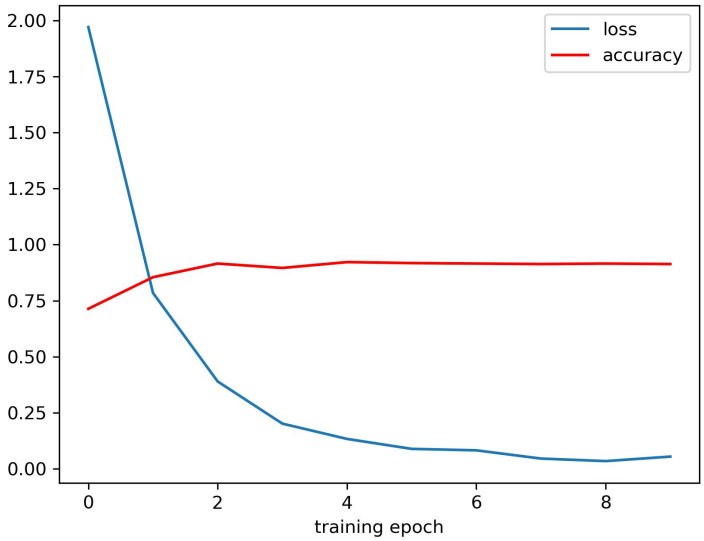

图 3: 启用 batch normalization 后

4.1.3 改变 dropout 概率为 0.3

loss 曲线见图 4，test 准确率为：0.894。

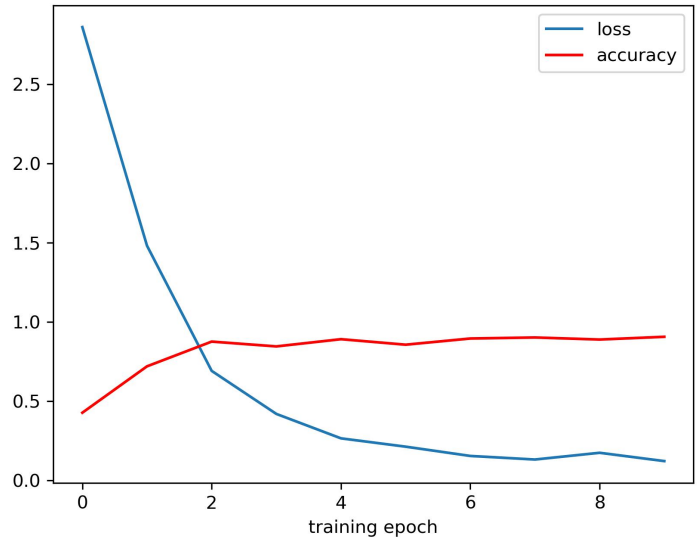

图 4: 改变 dropout 概率为 0.3

可见，启用 batch normalization 后的 test 准确率最高，模型效果最好。

4.2 可视化

4.2.1 可视化第 0 层卷积层的卷积核

从图 5中可以看出，输出的 32 组卷积核分别具有不同的“纹理”，用以提取输入图像不同方位的信息。

4.2.2 可视化第 100 张图像第 0 层卷积层的输出特征图

从图 6中可以看出，输出的 32 张特征图分别是输入图像经过上面的滤波器后的结果，字母 A 的不同特征被提取了出来。

4.2.3 可视化第 1 层卷积层的卷积核

从图 7中可以看出，输出的 64 组卷积核分别具有不同的“纹理”，用以提取上一层卷积层输出结果不同方位的信息，而且纹理的细节更加丰富。

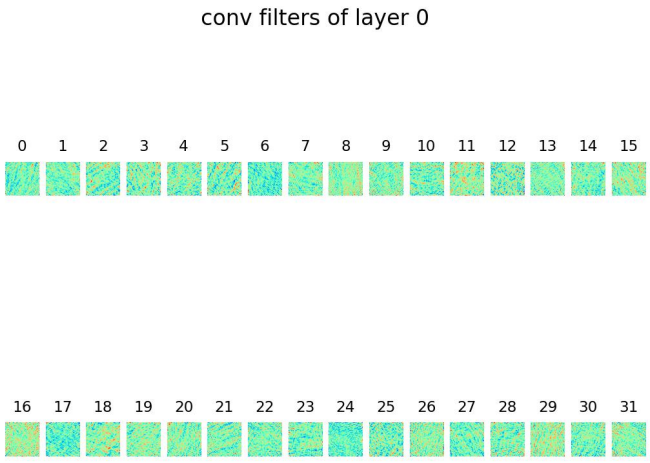

图 5: 第 0 层卷积层的卷积核

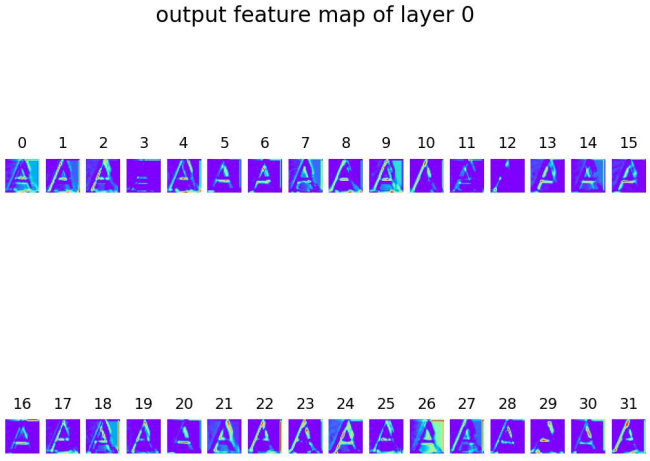

图 6: 第 100 张图像第 0 层卷积层的输出特征图

4.2.4 可视化第 100 张图像第 1 层卷积层的输出特征图

从图 8中可以看出，输出的 64 张特征图分别是经过上面的滤波器后的结果，字母 A 的更多特征被提取了出来。

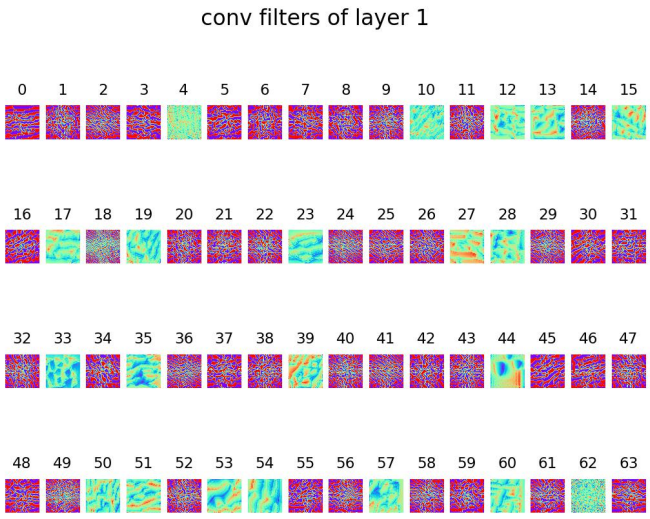

图 7: 第 1 层卷积层的卷积核

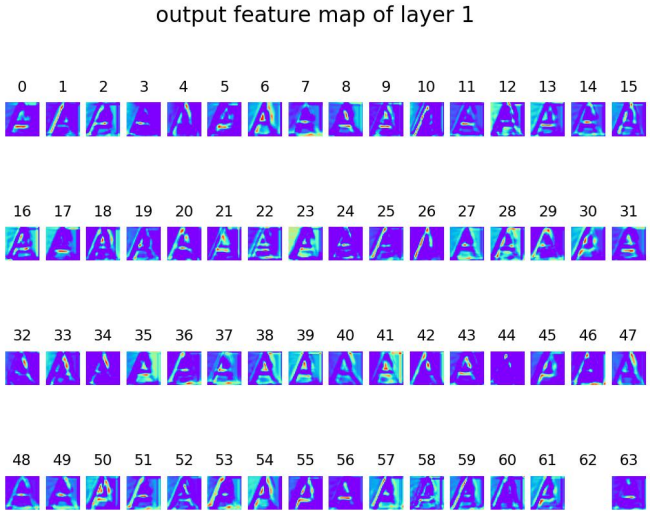

图 8: 第 100 张图像第 1 层卷积层的输出特征图

4.2.5 t-SNE 显示分类结果

用 t-SNE 将高维特征降维到二维空间后，可以发现字母特征呈现聚类分布。图 9显示的是图像经过卷积层、线性层、bn1d 和 ReLu 后的降维特征，

图 10是在此基础上又经过 dropout 和最后一层 linear 输出的特征图，可以看出分类效果比较明显的“主瓣”有 10 个。

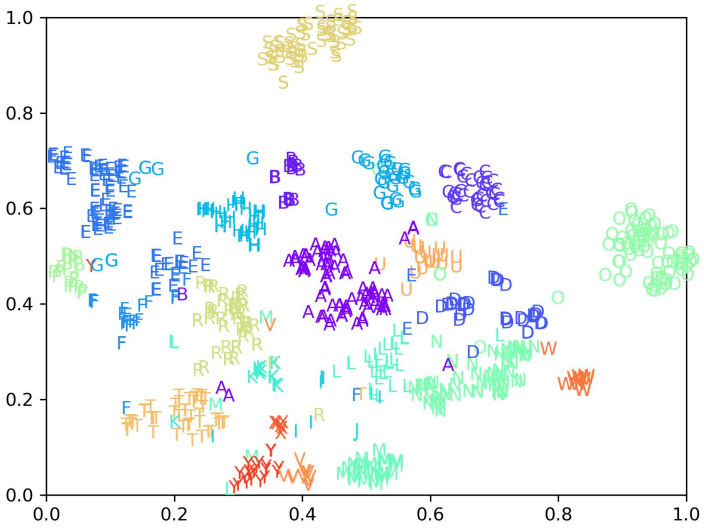

图 9: t-SNE 显示分类结果 0

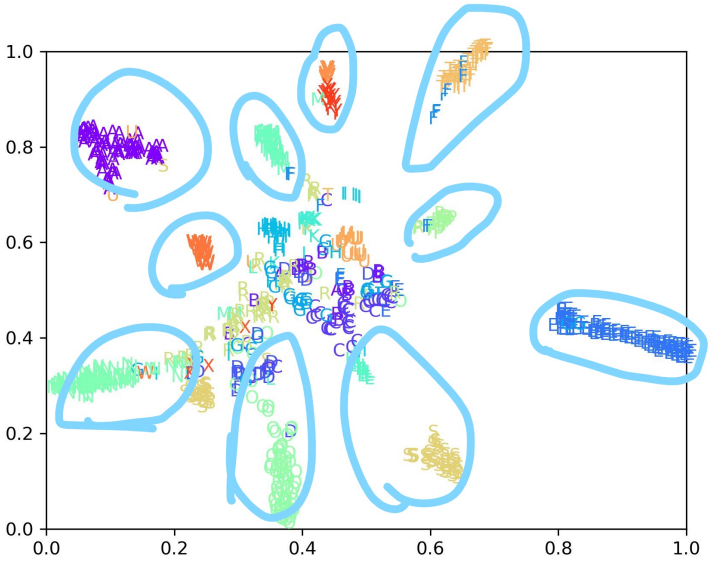

图 10: t-SNE 显示分类结果 1

4.3 熵计算

选择效果最好的一个模型: 启用 batch normalization 后的模型。先计算图像 1390 的图像熵、特征熵以及模型输出预测概率的信息熵:

```
Recognition result: B (confidence = 1.00)
Entropy of input image = 3.86
Entropy of features = 2.79, 2.68, 3.74, 1.91, 3.16
Entropy of prediction = 0.04
```

然后计算整个测试集以上熵的平均值:

```
Entropy of input test images = 4.11
Entropy of features = 3.06, 3.02, 4.05, 2.56, 3.48
Entropy of random guess = 3.26
Entropy of symbols in text labels = 2.93
Entropy of using trained model = 0.47
```

可以看出不管是单张图片还是整个测试集，经过卷积层后的特征图的信息熵都要小于输入图片的信息熵，而且输出概率分布的信息熵远小于原始图片的信息熵、随机猜测以及按照标签出现概率计算得到的信息熵，说明经过卷积层、线性层，图像的不确定性减小，我们有更高的把握判断这张图片属于哪个字母。

5 总结

1. 因为卷积的前向计算和反向传播过程中都要注意 X、weight 以及 output 的变形、反变形，同时考虑 tensor 的形状是否能自动广播，如果不能（bias 的情况），需要手动添加/删减维度。比如执行下面的代码:

```
import torch
a = torch.tensor([1,2,3])
b = torch.zeros(3,3,3)
print(a+b)
print(b+a.reshape(1,3,1))

tensor([[[1., 2., 3.],
         [1., 2., 3.],
         [1., 2., 3.]],
```

```
        [[1., 2., 3.],
         [1., 2., 3.],
         [1., 2., 3.]],

        [[1., 2., 3.],
         [1., 2., 3.],
         [1., 2., 3.]]])
    tensor([[[1., 1., 1.],
         [2., 2., 2.],
         [3., 3., 3.]],

        [[1., 1., 1.],
         [2., 2., 2.],
         [3., 3., 3.]],

        [[1., 1., 1.],
         [2., 2., 2.],
         [3., 3., 3.]]])
```

显然结果不同，但第二种结果正确。输出数组的形状是 [b, o, n]，每一组滤波器的 bias 应该是相同的，不同 out_channels 上的 bias 不同。

2. 由于采用了卷积层，所以和 hw2 相比，本实验的 test 预测准确率提高了至少 5 个百分点，说明引入卷积层可以提高模型抽取图片特征的能力。

3. 另外本实验中通过不同文件名区分训练和可视化结果，为比较和撰写报告带来了极大的便利，值得我们在今后的学习、科研中加以使用。