

**模式识别与机器学习**

**实验报告**

**班级：10052001**

**姓名：郑名哲**

**学号：2020302891**

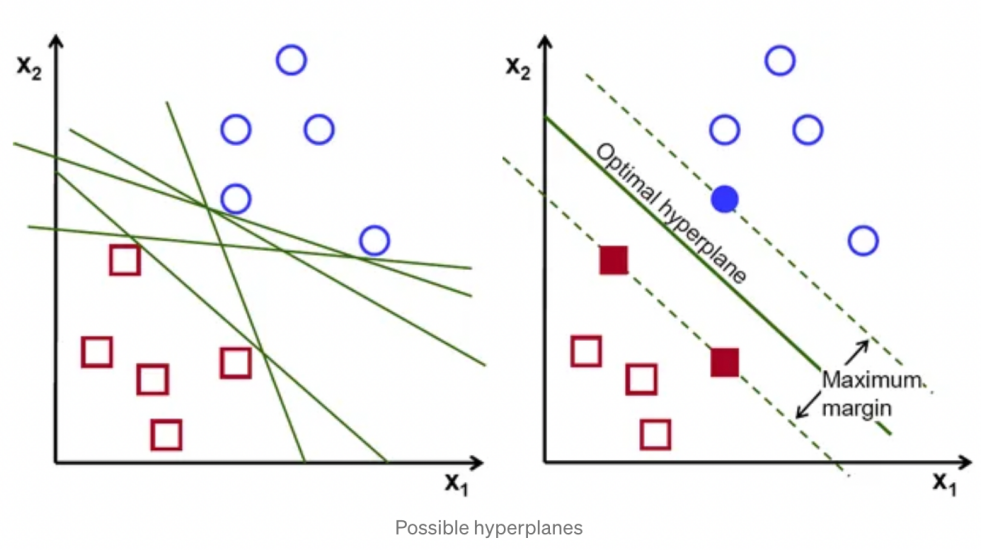
1. **实验目的**

分别使用SVM和CNN，完成对200类别omniglot数据集的分类

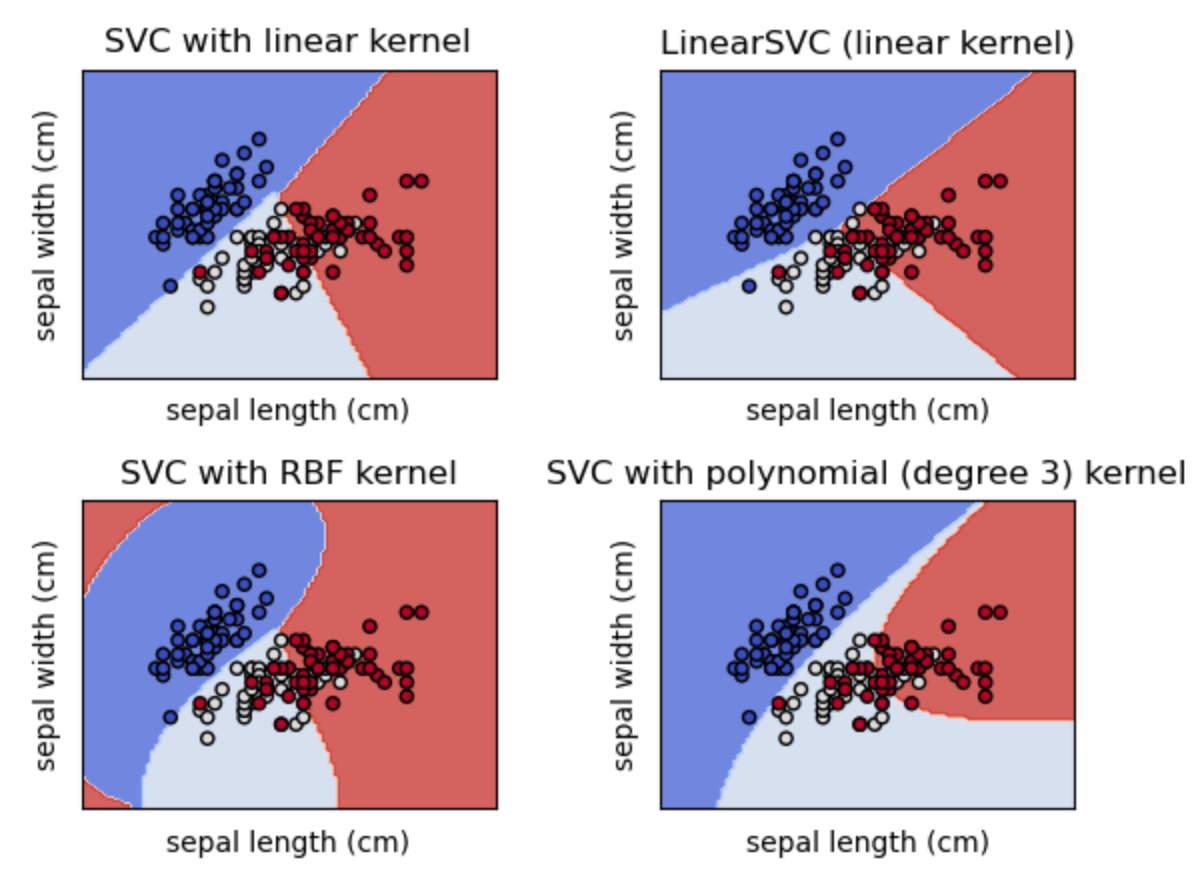
1. **实验原理**

**2.1 SVM**

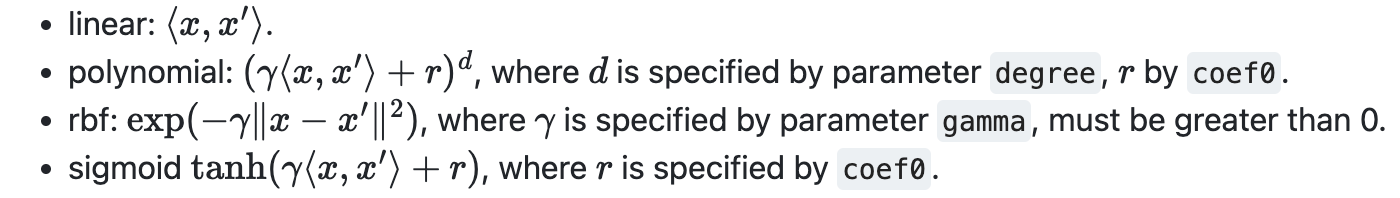
Support vector machines (SVMs)是一组用于分类、回归和异常值检测的监督学习方法[1]。支持向量机算法的目标是在 N 维空间（N —特征数）中找到一个超平面，该超平面可以明确地对数据点进行分类[2]。它的主要思想是找到一个最大化边际（margin）的超平面，以最大程度地将不同类别的数据点分开。边际是指超平面和最靠近它的数据点之间的距离。这些最靠近超平面的数据点被称为支持向量（support vectors）



面对多分类问题，主要使用[SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html" \l "sklearn.svm.SVC" \o "sklearn.svm.SVC), [NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \o "sklearn.svm.NuSVC) and [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \o "sklearn.svm.LinearSVC)



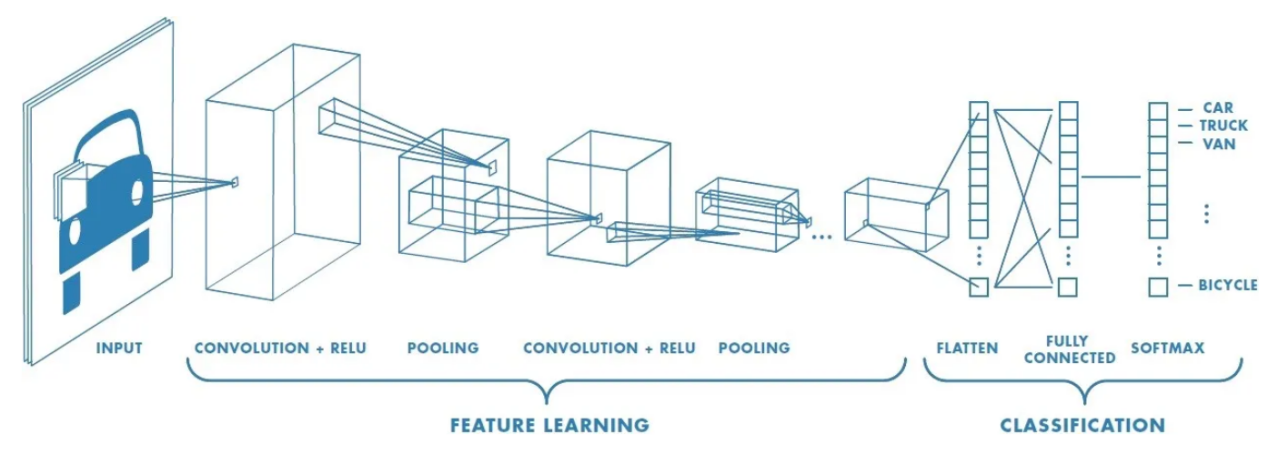
LinearSVC 基于线性kernel的SVM的另一种更快速的实现，而NuSVC和SVM类似，但使用不同的参数集以及 kernel function



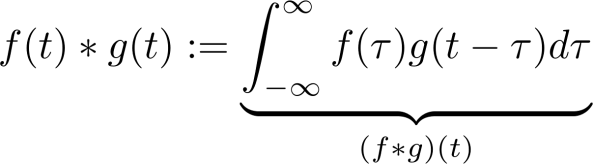
\*代码实现中通过配置SVC的kernel参数实现各种SVMs的实现，e.g. *Appendix. X1*

**2.2 CNN**

Convolutional Neural Network (ConvNet/CNN)是一种[前馈神经网络](https://zh.wikipedia.org/wiki/%E5%89%8D%E9%A6%88%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C" \o "前馈神经网络)，由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。与其他深度学习结构相比，卷积神经网络在图像和[语音识别](https://zh.wikipedia.org/wiki/%E8%AF%AD%E9%9F%B3%E8%AF%86%E5%88%AB" \o "语音识别)方面能够给出更好的结果。这一模型也可以使用[反向传播算法](https://zh.wikipedia.org/wiki/%E5%8F%8D%E5%90%91%E4%BC%A0%E6%92%AD%E7%AE%97%E6%B3%95" \o "反向传播算法)进行训练。相比较其他深度、前馈神经网络，卷积神经网络需要考量的参数更少，使之成为一种颇具吸引力的深度学习结构[4]

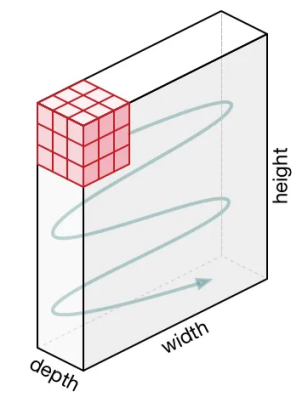
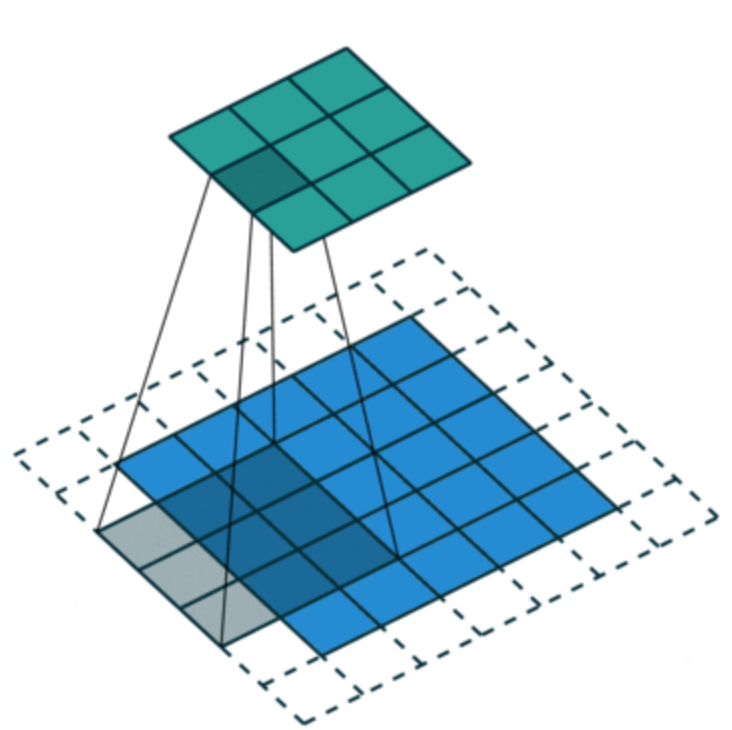


**卷积操作的数学定义为：**

****

其中通常情况下，wpsoffice

**在常见的面向图像的2D CNN使用的二维卷积中，就是卷积核的平移遍历**

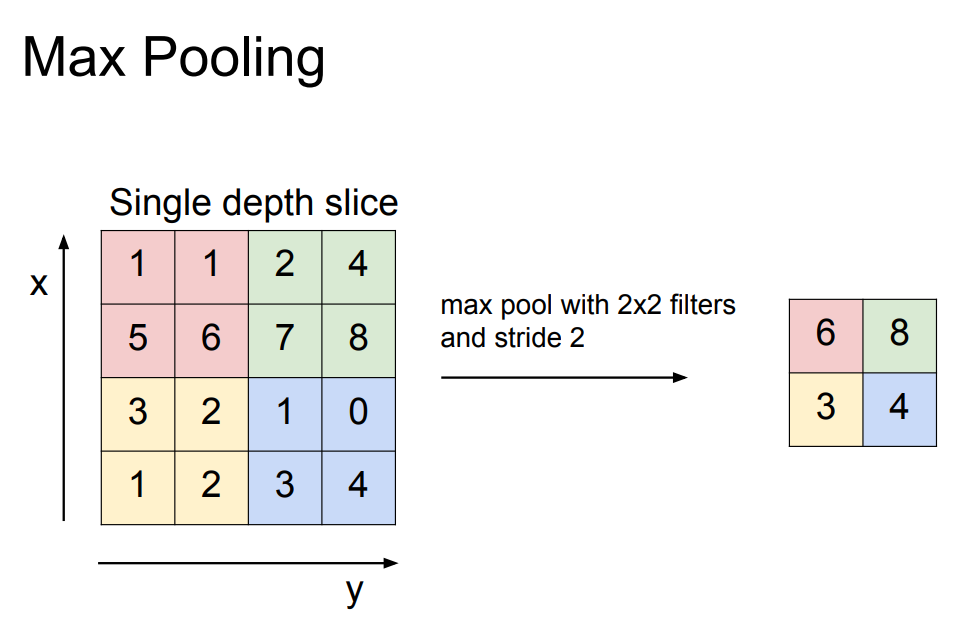
此时可以通过设置卷积层的参数来控制输出层的shape

假设

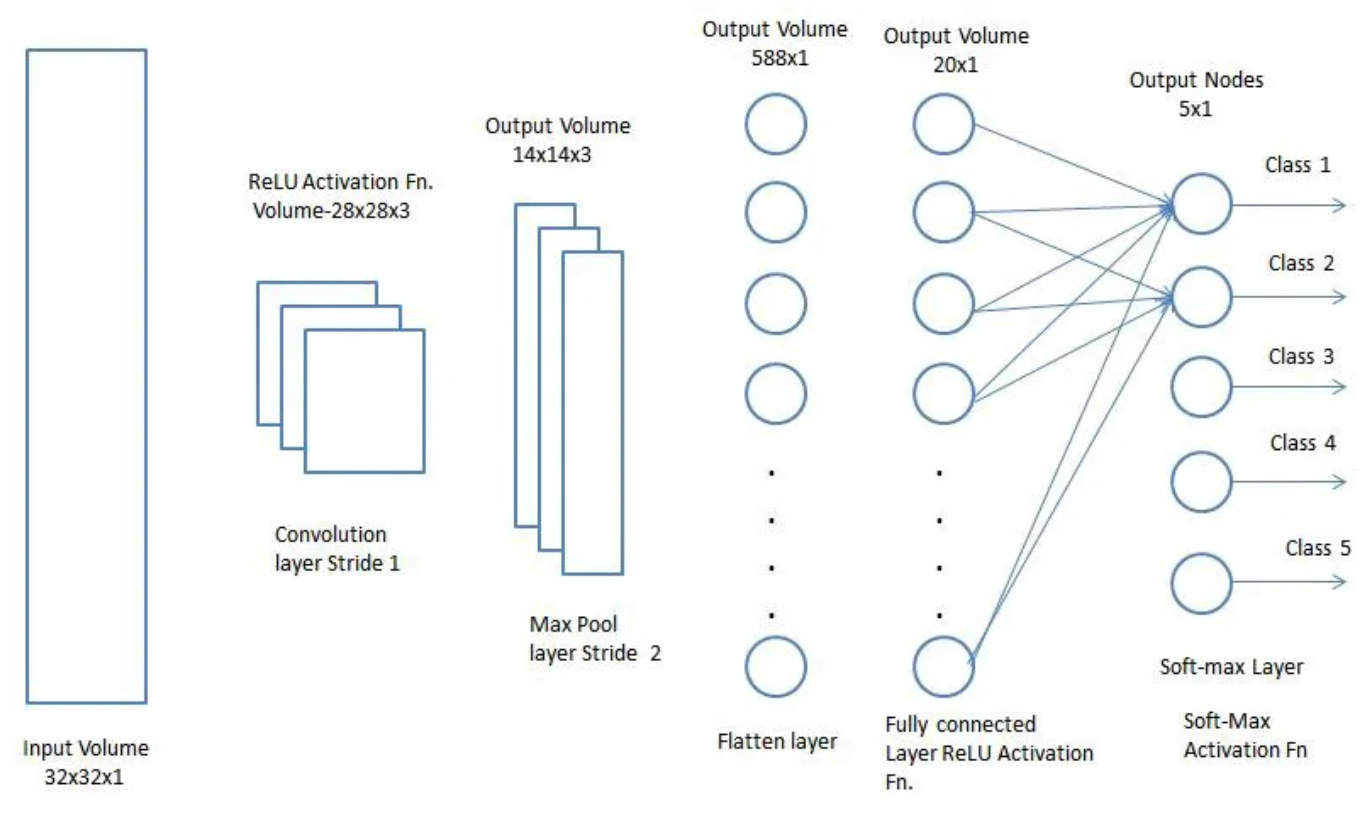
* 输入图片大小 W×W
* Filter大小 F×F
* 步长 S
* padding的像素数 P

于是我们可以根据**N = (W − F + 2P )/S+1**得到输出图片大小为N×N，在必要时刻通过设置特定的stride和padding，可以得到我们想要大小的output feature map（比如同维卷积）

池化，是CNN中常用的减小feature map的方法



而全连接（FC）通常在临近输出层的位置，将高维的feature map拉平到输出结果的同一维度上，也便于后续的softmax等操作



1. **实验步骤和程序流程**

**3.1 数据预处理**

**基本按照提供的baseline完成**，即首先初始化train和valid两个Dataset类

train\_dataset = Dataset(train\_val\_dataset\_path, input\_shape, epoch\_length=Epoch, is\_train=True, random\_seed=random\_seed)

val\_dataset = Dataset(train\_val\_dataset\_path, input\_shape, epoch\_length=Epoch, is\_train=False, random\_seed=random\_seed)

初始化的方法定义在Dataset类的\_\_getitem\_\_方法内



同时，在原Dataset的基础上加入了random seed的固定，使用与外部torch相同的random seed，减少后续横向对比各种model性能时的随机因素影响

self.random\_seed = random\_seed

np.random.seed(random\_seed)

torch.manual\_seed(random\_seed)

然后，使用torch标准库的DataLoader

from torch.utils.data import DataLoader

就可以把dataset装载为DataLoader类，便于后续使用pytorch常规处理流获取并处理数据了e.g. 使用enumerate(train\_gen)获取每一个batch的数据

**3.2 SVM**

**首先初始化模型**

model = SVC(C=1, kernel='poly', degree=3, gamma='scale', coef0=0.0,

shrinking=True, probability=False, tol=1e-4, cache\_size=200,

class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape='ovo', random\_state=500)

**接下来如果需要CV调参，可以通过GridSearchCV来通过交叉检验的方式进行调参（这里使用10折交叉）**

gsearch = GridSearchCV(model, param\_grid=parameters, scoring='accuracy', cv=10)

gsearch.fit(train\_images, train\_label)

**待检验的参数集例如，每一组键值对代表参数以及待检验的候选值（可以只有一个，即直接代入该值参与其他参数的检验，最后直接返回）**

parameters = {

"C": [1],

"kernel": ["poly"],

"gamma": ["scale", "auto"],

"decision\_function\_shape": ['ovo'],

"tol": [1e-4, 1e-3],

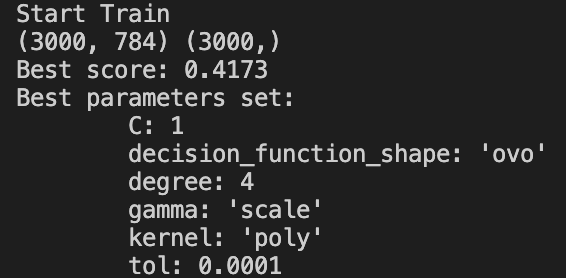
"degree": [2,3,4],

}

**那么CV调参后会得到你提供的所有候选参数中表现最好的那一组参数**

best\_parameters = gsearch.best\_estimator\_.get\_params()

**（这里我基于accuracy导向，那么返回acc最高的那一组参数）**



**调参后将best params写入model初始化的地方即可**

**不论是否需要CV调参，使用模型预测前都需要fit一下**

model.fit(train\_images, train\_label)

**然后就可以测一组train acc了**

train\_pred = model.predict(train\_images)

accuracy = accuracy\_score(train\_label, train\_pred)

print(f"iteration {iteration}: train acc: {accuracy}")

最后，使用**[pickle](https://docs.python.org/3/library/pickle.html)**在训练完毕后保存权重，在测试时也同理导入预训练权重

测试阶段，用valid\_loader来测试模型性能

val\_images, val\_label = batch[0], batch[1]

val\_pred = clf.predict(val\_images)

通过accuracy\_score计算测试集准确率

accuracy = accuracy\_score(val\_label, val\_pred)

report = classification\_report(val\_label, val\_pred)

print("Test Accuracy:", accuracy)

print("Classification report:", report)

**3.3 CNN train&test&inference pipeline**

**首先初始化模型**

model = Baseline2()

如果需要导入预训练权重，则导入类别以外的权重（baseline拓展实验存在（过）导入resnet预训练权重的需求）

if pretrained\_model\_path != '':

print('Load weights {}.'.format(pretrained\_model\_path))

# pretrained\_dict = torch.load(pretrained\_model\_path, map\_location= device)

# model.load\_state\_dict(pretrained\_dict)

layers\_False = ["fc.weight", "fc.bias"]

pretrained\_dict = torch.load(pretrained\_model\_path, map\_location= device)

# 删除有关分类类别的权重

for k in list(pretrained\_dict.keys()):

if k in layers\_False:

del pretrained\_dict[k]

print(model.load\_state\_dict(pretrained\_dict, strict=False))

接下来面向训练，需要通过model.train()让model进入train状态，同时面向单机多卡条件可以使用DP完成分布式训练（\*更好的选择是DDP或者Accelerate，不过本次实验不需要多卡的算力无需做此修改）

model\_train = model.train()

if Cuda:

Generator\_train = torch.nn.DataParallel(model)

cudnn.benchmark = True

Generator\_train = Generator\_train.cuda()

接下来定义优化器optimizer以及学习率调整scheduler

opt\_model = torch.optim.AdamW(model.parameters(), lr=Init\_lr, betas=(0.9, 0.999), eps=1e-08, weight\_decay=0.01, amsgrad=False)

scheduler = CosineAnnealingLR(opt\_model, T\_max=5, eta\_min=0)

基于3.1的方法获取预处理的训练集和测试集后，开始Epoch轮训练，在单个epoch中，通过tqdm打印进度条，使用交叉熵作为损失函数，让模型根据输入的数据预测出每个label的概率向量，计算loss；通过argmax获得最大概率值对应的label作为预测结果，计算acc

prob\_tensor = model\_train(images)

# import pdb; pdb.set\_trace()

class\_index = torch.argmax(prob\_tensor, dim=1)

acc = acc + (label == class\_index).sum().item()

loss\_value = criterion(prob\_tensor, label)

然后将opt的梯度清零，loss梯度反传，更新模型参数，最后scheduler据此更新学习率

opt\_model.zero\_grad()

loss\_value.backward()

opt\_model.step()

scheduler.step()

test阶段不反传梯度不更新参数，model.train()换成model.eval()，其余与train同理，得到test acc和抽样的val loss point

此时保存所有train loss/acc&test loss/acc，用于训练完毕后的整体画图

绘制loss曲线

def draw\_loss(train\_counter, train\_losses, test\_counter, test\_losses):

# draw loss curve

fig = plt.figure()

plt.plot(train\_counter, train\_losses, color='blue')

print(f"test\_counter = {len(test\_counter)}, test\_losses = {len(test\_losses)}")

plt.scatter(test\_counter, test\_losses, color='red')

plt.legend(['Train Loss', 'Valid Loss'], loc='upper right')

plt.xlabel('number of training examples seen')

plt.ylabel('negative log likelihood loss')

plt.savefig(f"./figure/resnet50\_loss.png")

plt.show()

绘制acc曲线

def draw\_acc(total\_epochs, train\_acc, test\_acc):

# draw acc curve

fig = plt.figure()

plt.plot(total\_epochs,train\_acc, color='red')

plt.plot(total\_epochs, test\_acc, color='green')

plt.legend(['Train Acc', 'Valid Acc'], loc='upper right')

plt.xlabel('number of training examples seen')

plt.ylabel('negative log likelihood acc')

plt.savefig(f"./figure/resnet50\_acc.png")

plt.show()

inference阶段，则获取testloader后

def get\_testloader():

train\_val\_dataset\_path = 'dataset/NewDataset.mat'

input\_shape = [28, 28]

random\_seed = 3407

batch\_size = 128

num\_workers = 0

test\_dataset = Dataset(train\_val\_dataset\_path, input\_shape, epoch\_length=1, is\_train=False, random\_seed=random\_seed)

test\_loader = DataLoader(test\_dataset, shuffle=True, batch\_size=batch\_size, num\_workers=num\_workers,

pin\_memory=True, drop\_last=True, collate\_fn=dataset\_collate, sampler=None)

return test\_loader

使用预训练模型，抽一组test data，可视化展现预测结果

def process():

model = Baseline2()

network\_state\_dict = torch.load('./checkpoints/baseline2\_ep100.pth')

model.load\_state\_dict(network\_state\_dict)

test\_loader = get\_testloader()

examples = enumerate(test\_loader, start=100)

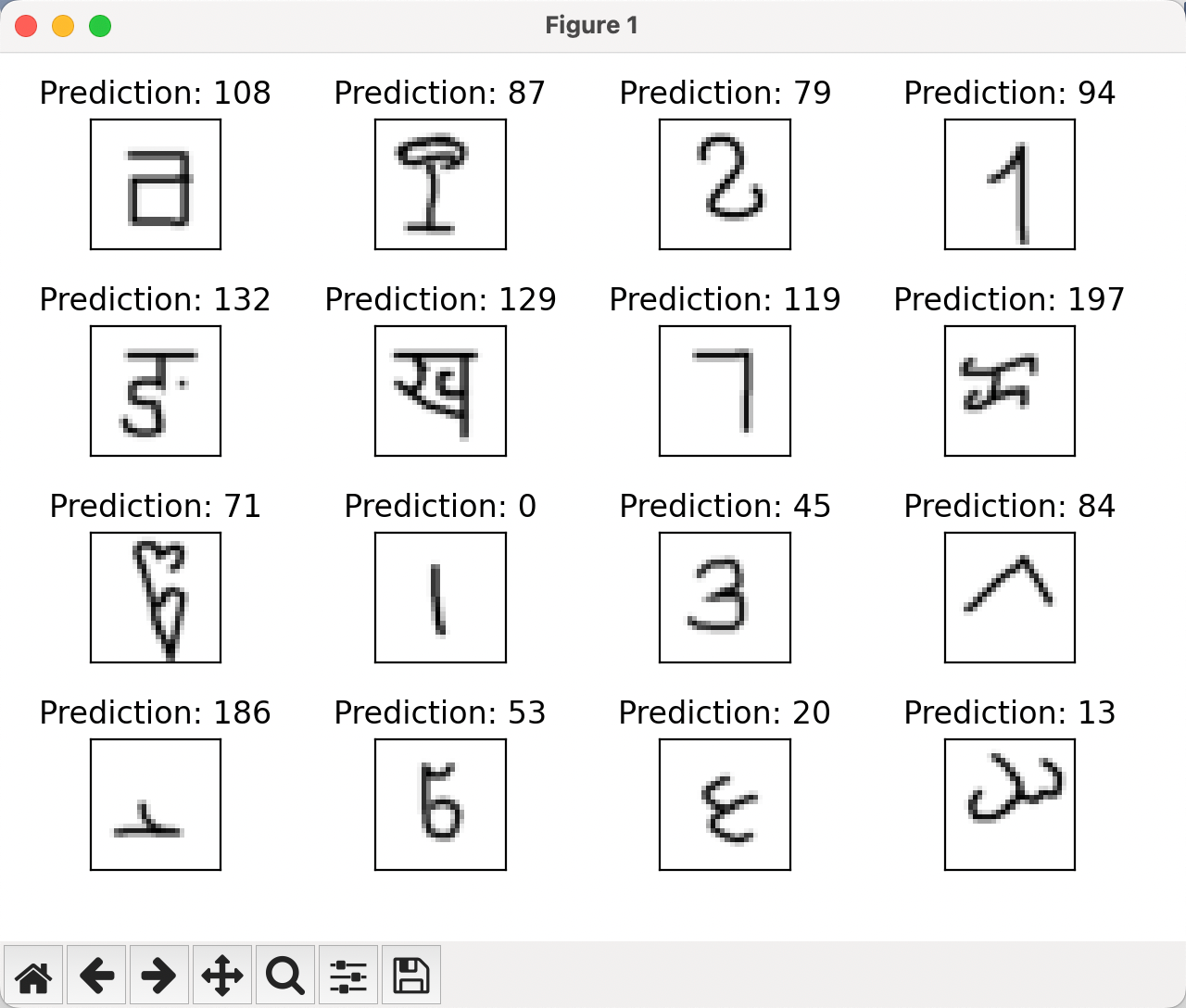
batch\_idx, (example\_data, example\_targets) = next(examples)

with torch.no\_grad():

output = model(example\_data)

show\_examples(output, example\_data)

一次随机结果如图



**3.4 CNN模型设计**

**Baseline通过两个简单Conv-BN-ReLU-Pooling组合后拉平放入两个通过ReLu连接的FC得到最终预测结果，效果一般。**

**Baseline2从Baseline的组合获取灵感，直接定义一组BasicBlock，按照具体的使用需要更改参数，可以实现降采样后Residual连接，Pooling等操作**

class BasicBlock(nn.Module):

def \_\_init\_\_(self, in\_channel, out\_channel, stride=1, downsample=None, if\_maxpool=True, \*\*kwargs):

super(BasicBlock, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(in\_channels=in\_channel, out\_channels=out\_channel,

kernel\_size=3, stride=stride, padding=1, bias=False)

self.bn1 = nn.BatchNorm2d(out\_channel)

self.relu = nn.ReLU()

self.maxpool = nn.MaxPool2d(kernel\_size=2)

self.conv2 = nn.Conv2d(in\_channels=out\_channel, out\_channels=out\_channel,

kernel\_size=3, stride=1, padding=1, bias=False)

self.bn2 = nn.BatchNorm2d(out\_channel)

self.downsample = downsample

self.if\_maxpool = if\_maxpool

def forward(self, x):

identity = x

if self.downsample:

identity = F.interpolate(identity, scale\_factor=0.5)

# print(f"x = {x.shape}")

out = self.conv1(x)

out = self.bn1(out)

out = self.relu(out)

# out = self.maxpool(out)

out = self.conv2(out)

out = self.bn2(out)

if not self.downsample:

out += identity

out = self.relu(out)

if self.if\_maxpool:

out = self.maxpool(out)

return out

**Baseline2中使用3个无残差的BasicBlock，最后拉平后使用一个FC作为Classifier直接输出预测结果，取得了明显好于Baseline的效果**

class Baseline2(nn.Module):

def \_\_init\_\_(self):

super(Baseline2, self).\_\_init\_\_()

self.conv1 = BasicBlock(1, 16, downsample=True)

self.conv2 = BasicBlock(16, 32, downsample=True)

self.conv3 = BasicBlock(32, 64, downsample=True)

self.classifier = nn.Linear(576, 200)

self.cls = BasicClassifier(576, 200)

def forward(self, x):

x = self.conv1(x)

# print(f"conv1:{x.shape}")

x = self.conv2(x)

# print(f"conv2:{x.shape}")

x = self.conv3(x)

# print(f"conv3:{x.shape}")

x = x.view(x.size(0), -1)

# print(f"view x: {x.shape}")

# x = self.classifier(x)

x = self.cls(x)

# print(f"classify: {x.shape}")

# x = F.softmax(x, dim=1)

# x = F.log\_softmax(x, dim=1)

# print(f"softmax: {x.shape}")

return x

**Baseline3希望在Baseline2的基础上，更大程度的发挥残差的作用，在Baseline2的基础上做了两处修改：BasicBlock中使用残差，BasicClassfier中使用残差，但是效果往往比不上同参数条件的Baseline2，调参后效果平齐**

class Baseline3(nn.Module):

def \_\_init\_\_(self):

super(Baseline3, self).\_\_init\_\_()

self.conv1 = BasicBlock(1, 16, downsample=True)

self.conv2 = BasicBlock(16, 32, downsample=True)

self.conv3 = BasicBlock(32, 32, downsample=False, if\_maxpool=False)

self.conv4 = BasicBlock(32, 64, downsample=True, if\_maxpool=True)

# self.conv5 = BasicBlock(128, 128, downsample=False, if\_maxpool=False)

self.dropout = nn.Dropout(0.5)

self.classifier = BasicClassifier(576,200)

def forward(self, x):

x = self.conv1(x)

# print(f"conv1:{x.shape}")

x = self.conv2(x)

# print(f"conv2:{x.shape}")

x = self.conv3(x)

# print(f"conv3:{x.shape}")

x = self.conv4(x)

# print(f"conv4:{x.shape}")

# x = self.dropout(x)

# print(f"dropout:{x.shape}")

# x = self.conv5(x)

# print(f"conv5:{x.shape}")

x = x.view(x.size(0), -1)

# print(f"view x:{x.shape}")

x = self.classifier(x)

# print(f"cls x: {x.shape}")

return x

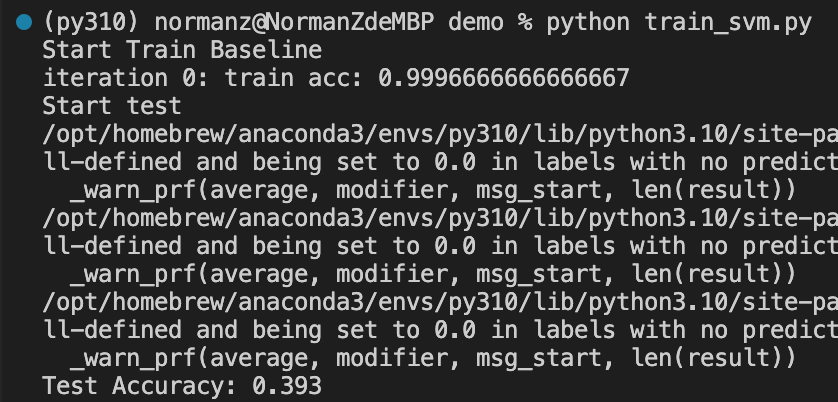
**最后使用了Kaiming He .el [5]的ResNet系列，复现了ResNet18、ResNet50、ResNet101 [6]三种进行了测试实验，发现调参后从0开始训练同等epoch（e.g.100 epoch）能得到优于Baseline的效果，对ResNet50魔改调参后收获了本轮实验的最高acc，详见四实验结果**

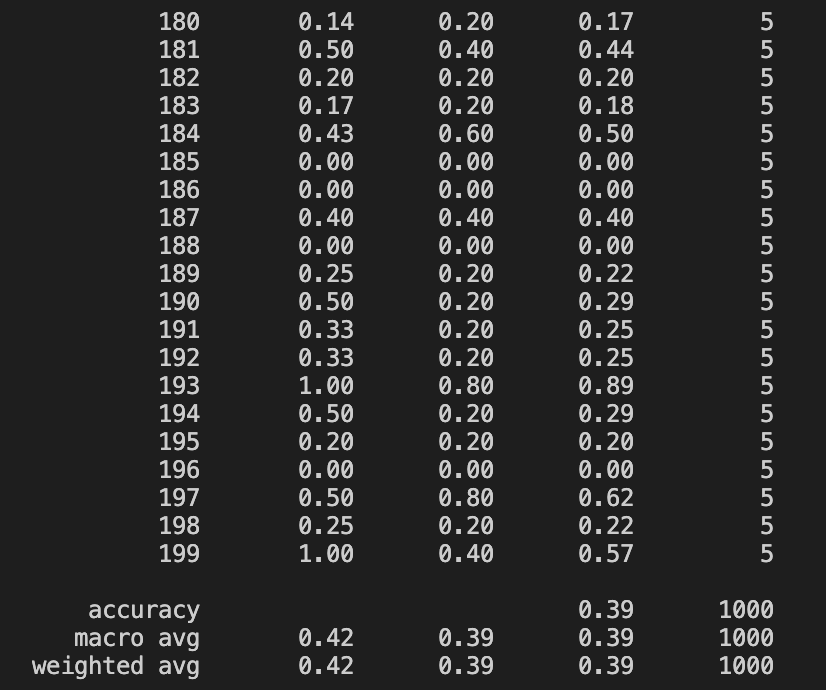
1. **实验结果**

**SVM**

原始baseline

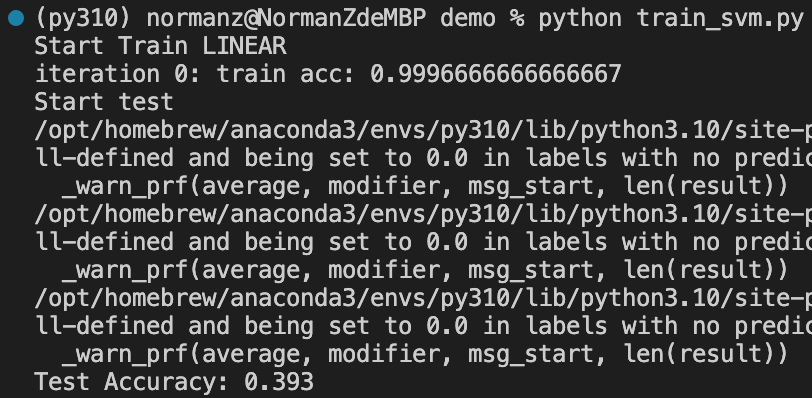
train acc = 0.9997 test acc = 0.393

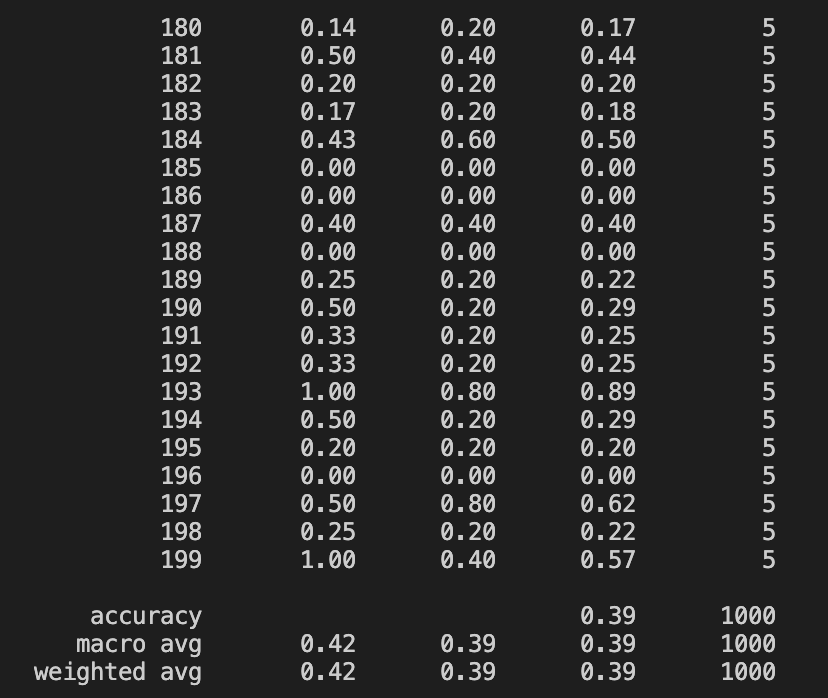




CV调参后linear kernel

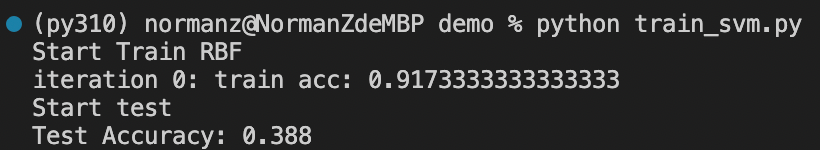
train acc = 0.9997 test acc = 0.393

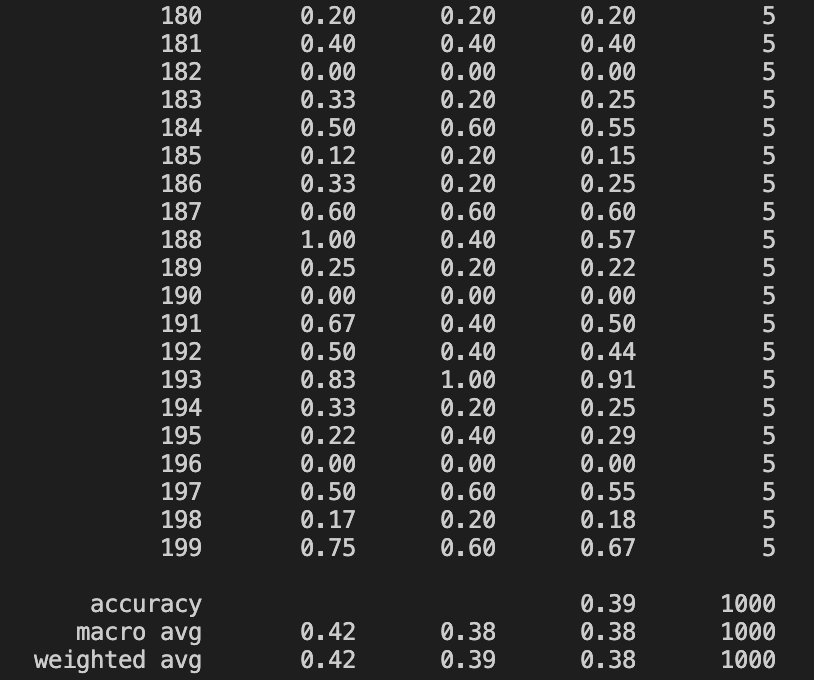




CV调参后rbf kernel

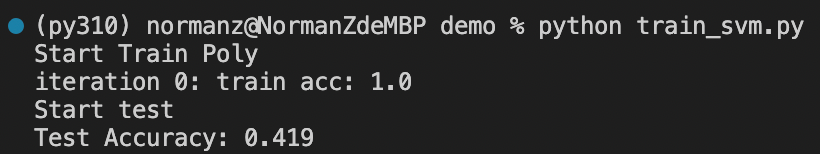
train acc = 0.9173 test acc = 0.388

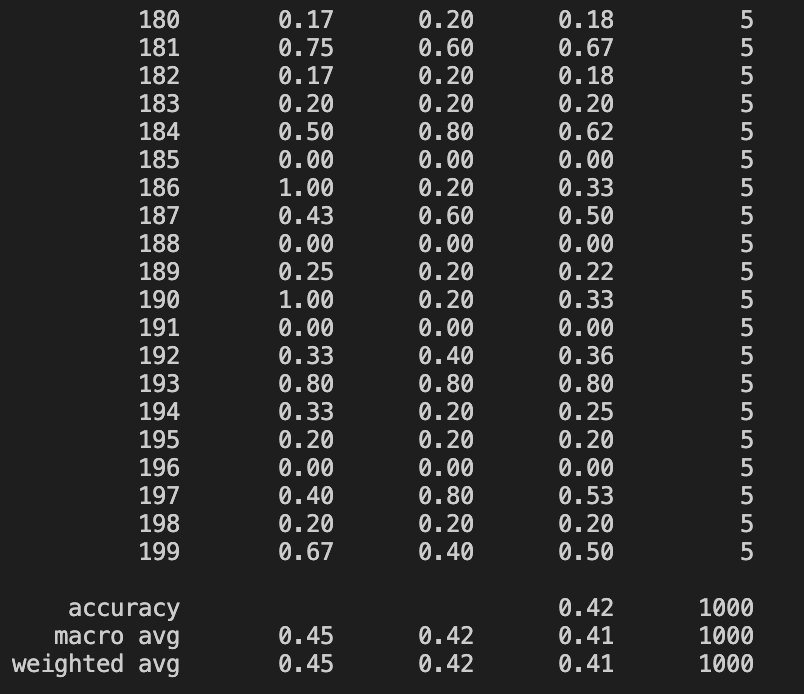




CV调参后poly kernel

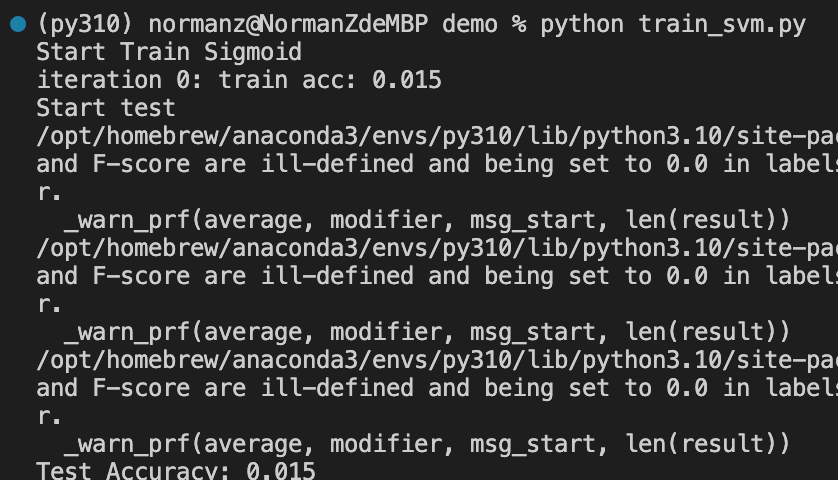
train acc = 1.0 test acc = 0.419

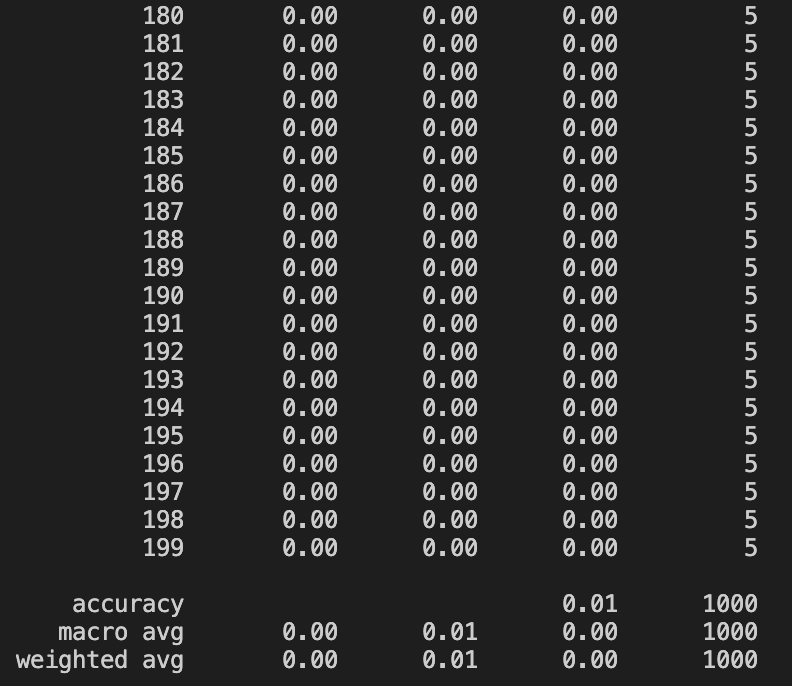




CV调参后sigmoid kernel

train acc = 0.015 test acc = 0.015

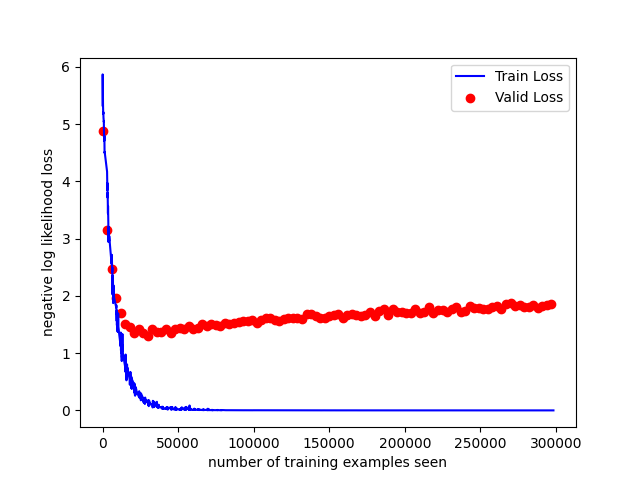
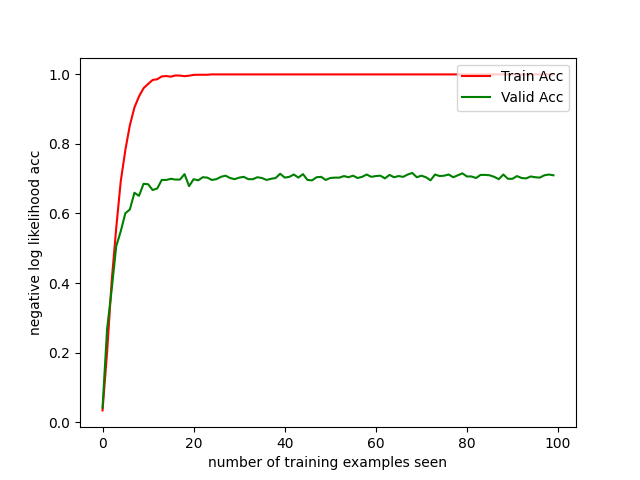




**CNN**

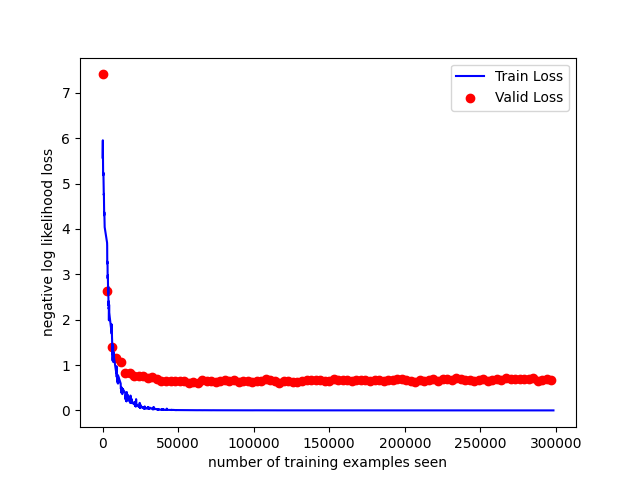
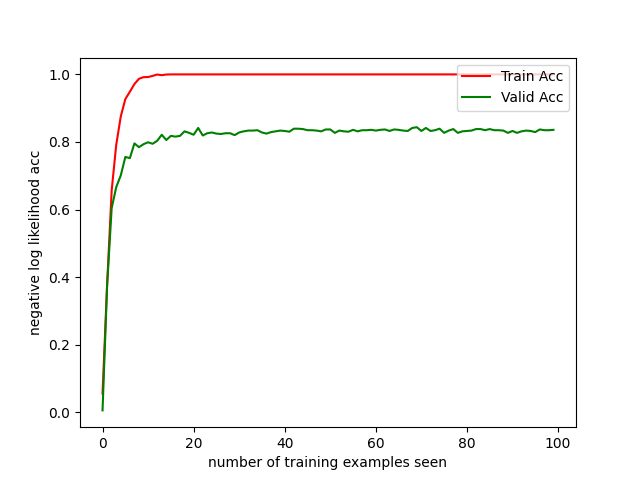
**原始baseline**

train acc = 1 test acc = 0.68



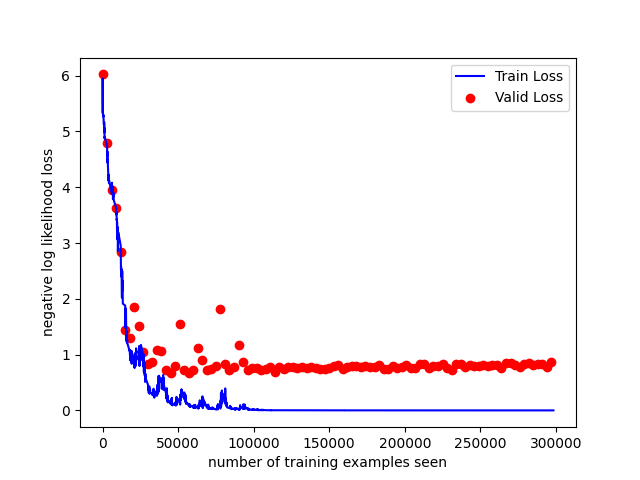
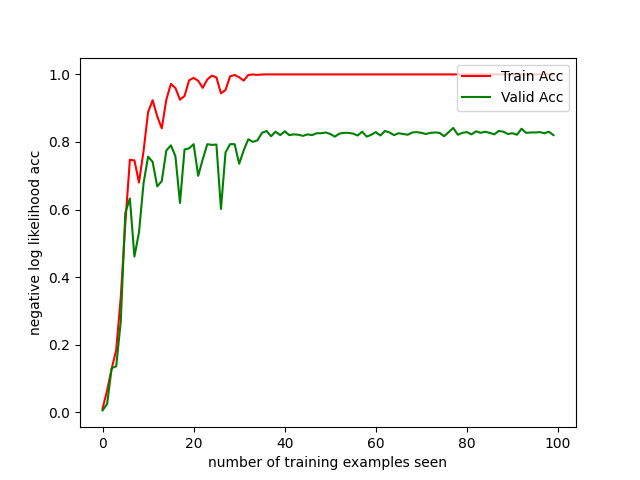
**Baseline2**

train acc = 1, test acc = 0.83



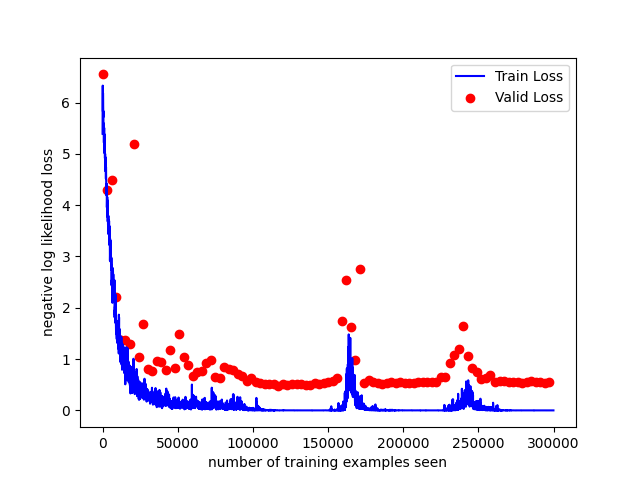
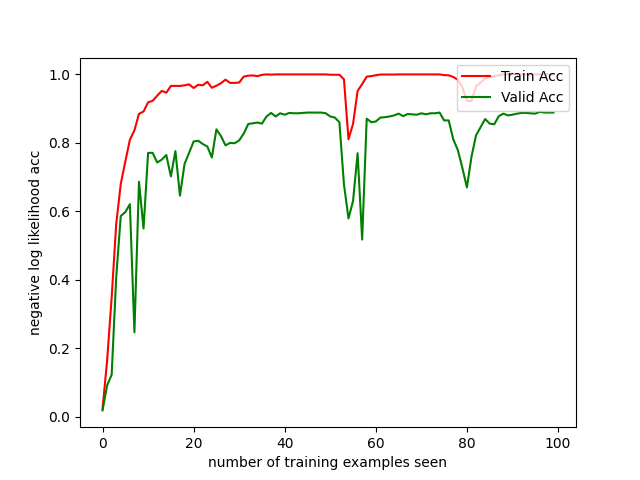
**Baseline3**

train acc = 1, test acc = 0.83



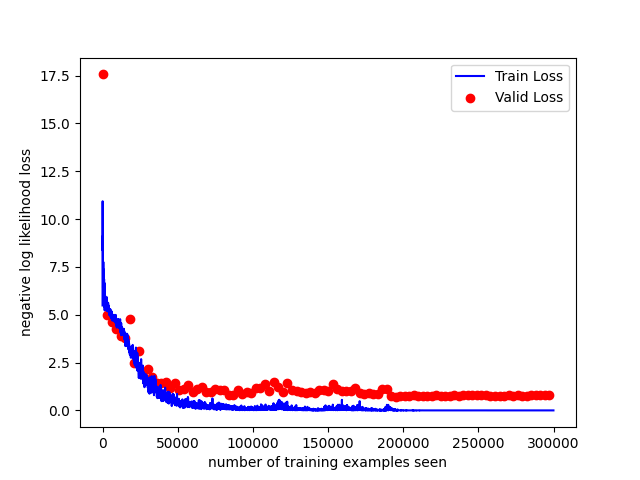
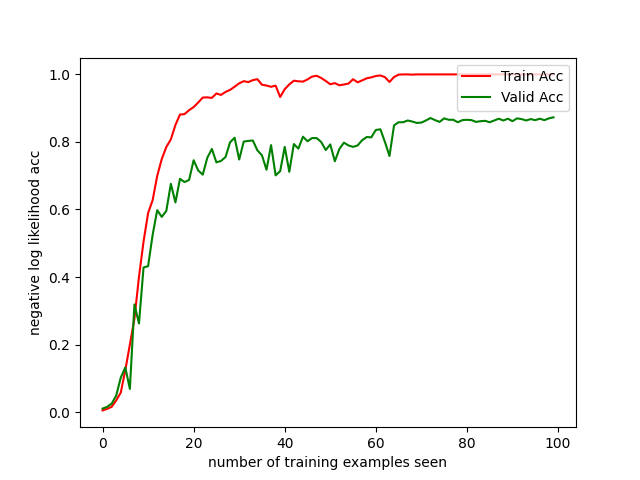
**ResNet18**

train acc = 1, test acc = 0.85



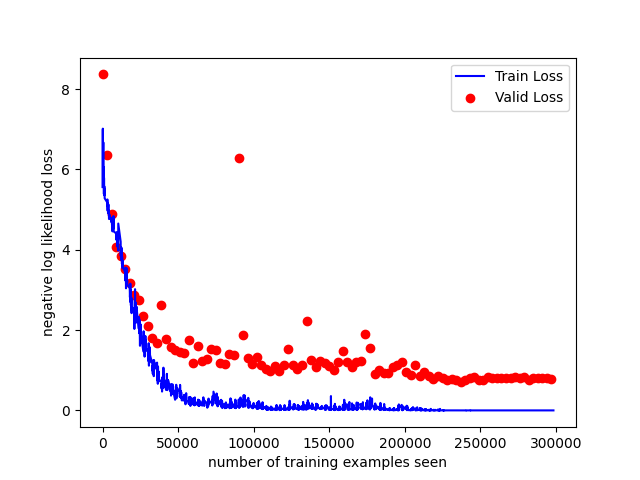
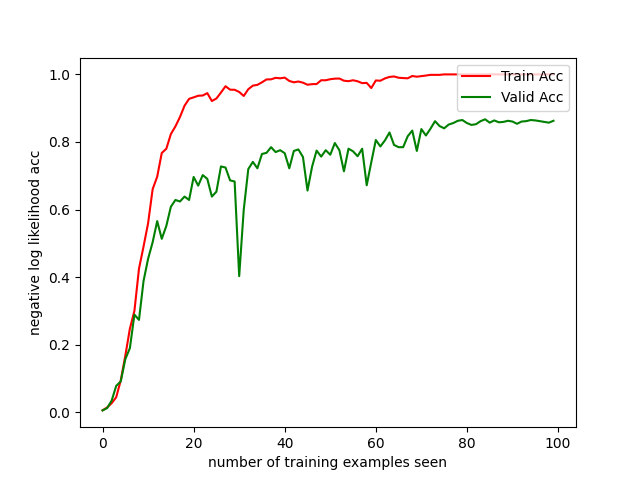
**ResNet50**

train acc = 1, test acc = 0.86



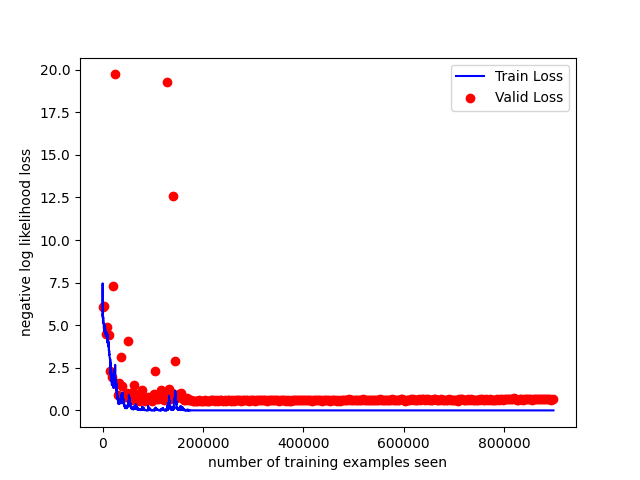
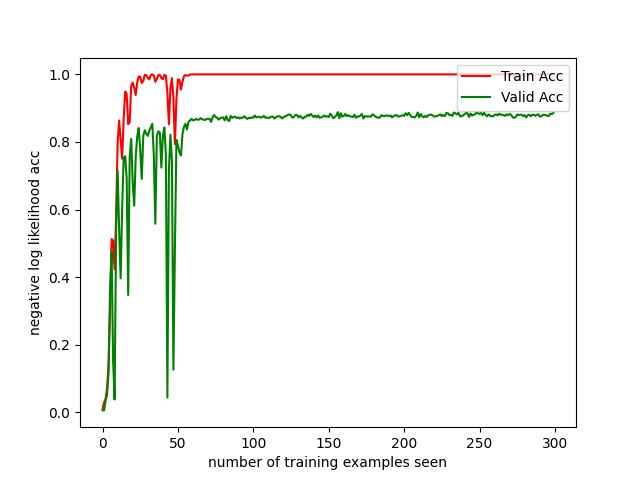
**ResNet101**

train acc = 1, test acc = 0.87



**ResNet50(opt=Adamw & scheduler=CosineLR)**

train acc = 1, test acc = 0.89



1. **评价分析**

SVM总结：

本次实验中，SVM直接使用200个label作为feature，而不是使用经过特征工程处理得到的特征来做分类，因此无法使用RFE等方法做特征选择，本来想画的特征相关热力图也无法绘制，label当feature进行预测可以预见的分类效果偏差

表现最好的是Poly kernel的SVC模型，train acc=1，test acc=0.419，在CV调参中最高达到0.43的accuracy

表现最差的是Sigmoid kernel的SVC模型，train和test acc都一塌糊涂，Sigmoid不适合做多分类任务

CNN总结：

本次实验中，相比于将每一次训练最后跑出的test acc（或将一次训练中最高的test acc&最低的train loss作为best checkpoint，使用earlystop的思路观测模型性能），我选择绘制全过程的loss与acc曲线

这样就显而易见的观察模型性能了，Baseline毫无疑问是全部实验中表现最差的；Baseline2使用3个BasicBlock衔接，巧妙控制feature map尺度，让最后拉平后的dim=1 shape不至于太大，可以直接一层FC拉到label shape，性能明显优于Baseline；Baseline3尝试结合更多的残差，做了大量的实验最后最好也是性能接近Baseline2（后续尝试Baseline2继续调参能拉到0.84 test acc大概，Baseline3更加追不上了）。

对于ResNet，最初尝试导入预训练权重，去除label的权重保留backbone权重，然后在此基础上进行训练；后来发现直接从0开始训练一个ResNet也能得到不错的结果，就直接用**无敌的随机种子3407** [7] 统一无预训练权重地跑100个epoch，（或300个epoch获得更好的性能），但为了公平期间一般只对比所有Baseline以同一random seed训练同样100个epoch的表现。

**附1:参考文献**

1. **[scikit-learn 1.4. Support Vector Machines](https://scikit-learn.org/stable/modules/svm.html" \l ":~:text=Support%20vector%20machines%20(SVMs)%20are,Effective%20in%20high%20dimensional%20spaces.)**
2. **[support-vector-machine](https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47)**
3. **[how-to-save-a-trained-model-by-scikit-learn](https://stackoverflow.com/questions/56107259/how-to-save-a-trained-model-by-scikit-learn)**
4. **[Wikipedia-Convolutional\_neural\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)**
5. **[Deep Residual Learning for Image Recognition](https://arxiv.org/abs/1512.03385v1)**
6. **[PaperwithCode-ResNet](https://paperswithcode.com/paper/deep-residual-learning-for-image-recognition/review/?hl=6136)**
7. **[Torch.manual\_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision](https://arxiv.org/abs/2109.08203)**

**附2：代码（只展示重点修改过的代码，完整代码请查阅src）**

**引用部分**

X1 模型配置参数（带kernel）实例:

model = SVC(C=1, kernel='rbf', degree=2, gamma='scale', coef0=0.0,

shrinking=True, probability=False, tol=1e-4, cache\_size=200,

class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape='ovo', random\_state=500)

**源码部分（CNN）**

model.py CNN Baseline1,2,3定义：

import cv2

# import kornia

import numpy

from matplotlib import pyplot as plt

import numpy as np

import torch

import torch.nn as nn

from torch import Tensor

import torch.nn.functional as F

class BasicBlock(nn.Module):

def \_\_init\_\_(self, in\_channel, out\_channel, stride=1, downsample=None, if\_maxpool=True, \*\*kwargs):

super(BasicBlock, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(in\_channels=in\_channel, out\_channels=out\_channel,

kernel\_size=3, stride=stride, padding=1, bias=False)

self.bn1 = nn.BatchNorm2d(out\_channel)

self.relu = nn.ReLU()

self.maxpool = nn.MaxPool2d(kernel\_size=2)

self.conv2 = nn.Conv2d(in\_channels=out\_channel, out\_channels=out\_channel,

kernel\_size=3, stride=1, padding=1, bias=False)

self.bn2 = nn.BatchNorm2d(out\_channel)

self.downsample = downsample

self.if\_maxpool = if\_maxpool

def forward(self, x):

identity = x

if self.downsample:

identity = F.interpolate(identity, scale\_factor=0.5)

# print(f"x = {x.shape}")

out = self.conv1(x)

out = self.bn1(out)

out = self.relu(out)

# out = self.maxpool(out)

out = self.conv2(out)

out = self.bn2(out)

if not self.downsample:

out += identity

out = self.relu(out)

if self.if\_maxpool:

out = self.maxpool(out)

# print(f"x view: {x.view(out.size(0), -1).shape}")

# print(f"out = {out.shape}")

return out

class Baseline(nn.Module):

def \_\_init\_\_(self):

super(Baseline, self).\_\_init\_\_()

self.conv\_features = nn.Sequential(

nn.Conv2d(1, 16, kernel\_size=3, padding=1),

nn.BatchNorm2d(16),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2),

nn.Conv2d(16, 32, kernel\_size=3, padding=1),

nn.BatchNorm2d(32),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2)

)

self.classifier = nn.Sequential(

nn.Linear(1568, 512),

nn.ReLU(inplace=True),

nn.Linear(512, 200)

)

def forward(self, x): # size(x) == (B,1,28,28)

x = self.conv\_features(x)

x = x.view(x.size(0), -1)

# print(x.shape)

x = self.classifier(x)

# x = F.softmax(x, dim=1)

return x

class Baseline2(nn.Module):

def \_\_init\_\_(self):

super(Baseline2, self).\_\_init\_\_()

self.conv1 = BasicBlock(1, 16, downsample=True)

self.conv2 = BasicBlock(16, 32, downsample=True)

self.conv3 = BasicBlock(32, 64, downsample=True)

self.classifier = nn.Linear(576, 200)

self.cls = BasicClassifier(576, 200)

def forward(self, x):

x = self.conv1(x)

# print(f"conv1:{x.shape}")

x = self.conv2(x)

# print(f"conv2:{x.shape}")

x = self.conv3(x)

# print(f"conv3:{x.shape}")

x = x.view(x.size(0), -1)

# print(f"view x: {x.shape}")

# x = self.classifier(x)

x = self.cls(x)

# print(f"classify: {x.shape}")

# x = F.softmax(x, dim=1)

# x = F.log\_softmax(x, dim=1)

# print(f"softmax: {x.shape}")

return x

class BasicClassifier(nn.Module):

def \_\_init\_\_(self, in\_channel, out\_channel, residual=True):

super(BasicClassifier, self).\_\_init\_\_()

self.cl1 = nn.Linear(in\_channel, in\_channel//2)

self.cl2 = nn.Linear(in\_channel//2, out\_channel)

self.cl3 = nn.Linear(in\_channel, 16)

self.cl4 = nn.Linear(16, in\_channel//2)

self.cl5 = nn.Linear(in\_channel, out\_channel)

self.relu = nn.ReLU()

self.residual = residual

def forward\_normal(self, x):

x = self.cl1(x)

# print(f"cl1: {x.shape}")

x = self.relu(x)

# print(f"relu1: {x.shape}")

x = self.cl2(x)

# print(f"cl2: {x.shape}")

return x

def forward\_residual(self, x):

x1 = self.cl1(x)

x1 = self.relu(x1)

print(f"x1: {x1.shape}")

x21 = self.cl3(x)

x21 = self.relu(x21)

print(f"x21: {x21.shape}")

x22 = self.cl4(x21)

x22 = self.relu(x22)

print(f"x22: {x22.shape}")

out = self.cl2(x1+x22)

print(f"out: {out.shape}")

return out

def forward(self, x):

if self.residual:

x = self.forward\_residual(x)

else:

x = self.forward\_normal(x)

return x

class Baseline3(nn.Module):

def \_\_init\_\_(self):

super(Baseline3, self).\_\_init\_\_()

self.conv1 = BasicBlock(1, 16, downsample=True)

self.conv2 = BasicBlock(16, 32, downsample=True)

self.conv3 = BasicBlock(32, 32, downsample=False, if\_maxpool=False)

self.conv4 = BasicBlock(32, 64, downsample=True, if\_maxpool=True)

# self.conv5 = BasicBlock(128, 128, downsample=False, if\_maxpool=False)

self.dropout = nn.Dropout(0.5)

self.classifier = BasicClassifier(576,200)

def forward(self, x):

x = self.conv1(x)

# print(f"conv1:{x.shape}")

x = self.conv2(x)

# print(f"conv2:{x.shape}")

x = self.conv3(x)

# print(f"conv3:{x.shape}")

x = self.conv4(x)

# print(f"conv4:{x.shape}")

# x = self.dropout(x)

# print(f"dropout:{x.shape}")

# x = self.conv5(x)

# print(f"conv5:{x.shape}")

x = x.view(x.size(0), -1)

# print(f"view x:{x.shape}")

x = self.classifier(x)

# print(f"cls x: {x.shape}")

x = F.log\_softmax(x, dim=1)

return x

class ResNet(nn.Module):

def \_\_init\_\_(self,

block,

blocks\_num,

num\_classes=200,

include\_top=True,

groups=1,

width\_per\_group=64):

super(ResNet, self).\_\_init\_\_()

self.include\_top = include\_top

self.in\_channel = 64

self.groups = groups

self.width\_per\_group = width\_per\_group

self.conv1 = nn.Conv2d(3, self.in\_channel, kernel\_size=7,

stride=2, padding=3, bias=False)

self.bn1 = nn.BatchNorm2d(self.in\_channel)

self.relu = nn.ReLU(inplace=True)

self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=2, padding=1)

self.layer1 = self.\_make\_layer(block, 64, blocks\_num[0])

self.layer2 = self.\_make\_layer(block, 128, blocks\_num[1], stride=2)

self.layer3 = self.\_make\_layer(block, 256, blocks\_num[2], stride=2)

self.layer4 = self.\_make\_layer(block, 512, blocks\_num[3], stride=2)

if self.include\_top:

self.avgpool = nn.AdaptiveAvgPool2d((1, 1)) # output size = (1, 1)

self.fc = nn.Linear(512 \* block.expansion, num\_classes)

for m in self.modules():

if isinstance(m, nn.Conv2d):

nn.init.kaiming\_normal\_(m.weight, mode='fan\_out', nonlinearity='relu')

def \_make\_layer(self, block, channel, block\_num, stride=1):

downsample = None

if stride != 1 or self.in\_channel != channel \* block.expansion:

downsample = nn.Sequential(

nn.Conv2d(self.in\_channel, channel \* block.expansion, kernel\_size=1, stride=stride, bias=False),

nn.BatchNorm2d(channel \* block.expansion))

layers = []

layers.append(block(self.in\_channel,

channel,

downsample=downsample,

stride=stride,

groups=self.groups,

width\_per\_group=self.width\_per\_group))

self.in\_channel = channel \* block.expansion

for \_ in range(1, block\_num):

layers.append(block(self.in\_channel,

channel,

groups=self.groups,

width\_per\_group=self.width\_per\_group))

return nn.Sequential(\*layers)

def forward(self, x):

x = self.conv1(x)

x = self.bn1(x)

x = self.relu(x)

x = self.maxpool(x)

x = self.layer1(x)

cond1 = x

x = self.layer2(x)

cond2 = x

x = self.layer3(x)

cond3 = x

x = self.layer4(x)

cond4 = x

if self.include\_top:

x = self.avgpool(x)

x = torch.flatten(x, 1)

x = self.fc(x)

return cond1, x

def resnet34(num\_classes=200, include\_top=True):

# https://download.pytorch.org/models/resnet34-333f7ec4.pth

return ResNet(BasicBlock, [3, 4, 6, 3], num\_classes=num\_classes, include\_top=include\_top)

if \_\_name\_\_ == "\_\_main\_\_":

model = Baseline3()

input = torch.rand(16, 1, 28, 28)

output = model(input)

print(output.shape)

resnet.py Pytorch implementation for ResNet

"""resnet in pytorch

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.

Deep Residual Learning for Image Recognition

https://arxiv.org/abs/1512.03385v1

"""

import torch

import torch.nn as nn

class BasicBlock(nn.Module):

"""Basic Block for resnet 18 and resnet 34

"""

#BasicBlock and BottleNeck block

#have different output size

#we use class attribute expansion

#to distinct

expansion = 1

def \_\_init\_\_(self, in\_channels, out\_channels, stride=1):

super().\_\_init\_\_()

#residual function

self.residual\_function = nn.Sequential(

nn.Conv2d(in\_channels, out\_channels, kernel\_size=3, stride=stride, padding=1, bias=False),

nn.BatchNorm2d(out\_channels),

nn.ReLU(inplace=True),

nn.Conv2d(out\_channels, out\_channels \* BasicBlock.expansion, kernel\_size=3, padding=1, bias=False),

nn.BatchNorm2d(out\_channels \* BasicBlock.expansion)

)

#shortcut

self.shortcut = nn.Sequential()

#the shortcut output dimension is not the same with residual function

#use 1\*1 convolution to match the dimension

if stride != 1 or in\_channels != BasicBlock.expansion \* out\_channels:

self.shortcut = nn.Sequential(

nn.Conv2d(in\_channels, out\_channels \* BasicBlock.expansion, kernel\_size=1, stride=stride, bias=False),

nn.BatchNorm2d(out\_channels \* BasicBlock.expansion)

)

def forward(self, x):

return nn.ReLU(inplace=True)(self.residual\_function(x) + self.shortcut(x))

class BottleNeck(nn.Module):

"""Residual block for resnet over 50 layers

"""

expansion = 4

def \_\_init\_\_(self, in\_channels, out\_channels, stride=1):

super().\_\_init\_\_()

self.residual\_function = nn.Sequential(

nn.Conv2d(in\_channels, out\_channels, kernel\_size=1, bias=False),

nn.BatchNorm2d(out\_channels),

nn.ReLU(inplace=True),

nn.Conv2d(out\_channels, out\_channels, stride=stride, kernel\_size=3, padding=1, bias=False),

nn.BatchNorm2d(out\_channels),

nn.ReLU(inplace=True),

nn.Conv2d(out\_channels, out\_channels \* BottleNeck.expansion, kernel\_size=1, bias=False),

nn.BatchNorm2d(out\_channels \* BottleNeck.expansion),

)

self.shortcut = nn.Sequential()

if stride != 1 or in\_channels != out\_channels \* BottleNeck.expansion:

self.shortcut = nn.Sequential(

nn.Conv2d(in\_channels, out\_channels \* BottleNeck.expansion, stride=stride, kernel\_size=1, bias=False),

nn.BatchNorm2d(out\_channels \* BottleNeck.expansion)

)

def forward(self, x):

return nn.ReLU(inplace=True)(self.residual\_function(x) + self.shortcut(x))

class ResNet(nn.Module):

def \_\_init\_\_(self, block, num\_block, num\_classes=200):

super().\_\_init\_\_()

self.in\_channels = 64

self.conv1 = nn.Sequential(

nn.Conv2d(1, 64, kernel\_size=3, padding=1, bias=False),

nn.BatchNorm2d(64),

nn.ReLU(inplace=True))

#we use a different inputsize than the original paper

#so conv2\_x's stride is 1

self.conv2\_x = self.\_make\_layer(block, 64, num\_block[0], 1)

self.conv3\_x = self.\_make\_layer(block, 128, num\_block[1], 2)

self.conv4\_x = self.\_make\_layer(block, 256, num\_block[2], 2)

self.conv5\_x = self.\_make\_layer(block, 512, num\_block[3], 2)

self.avg\_pool = nn.AdaptiveAvgPool2d((1, 1))

self.fc = nn.Linear(512 \* block.expansion, num\_classes)

def \_make\_layer(self, block, out\_channels, num\_blocks, stride):

"""make resnet layers(by layer i didnt mean this 'layer' was the

same as a neuron netowork layer, ex. conv layer), one layer may

contain more than one residual block

Args:

block: block type, basic block or bottle neck block

out\_channels: output depth channel number of this layer

num\_blocks: how many blocks per layer

stride: the stride of the first block of this layer

Return:

return a resnet layer

"""

# we have num\_block blocks per layer, the first block

# could be 1 or 2, other blocks would always be 1

strides = [stride] + [1] \* (num\_blocks - 1)

layers = []

for stride in strides:

layers.append(block(self.in\_channels, out\_channels, stride))

self.in\_channels = out\_channels \* block.expansion

return nn.Sequential(\*layers)

def forward(self, x):

output = self.conv1(x)

output = self.conv2\_x(output)

output = self.conv3\_x(output)

output = self.conv4\_x(output)

output = self.conv5\_x(output)

output = self.avg\_pool(output)

output = output.view(output.size(0), -1)

output = self.fc(output)

return output

def resnet18():

""" return a ResNet 18 object

"""

return ResNet(BasicBlock, [2, 2, 2, 2])

def resnet34():

""" return a ResNet 34 object

"""

return ResNet(BasicBlock, [3, 4, 6, 3])

def resnet50():

""" return a ResNet 50 object

"""

return ResNet(BottleNeck, [3, 4, 6, 3])

def resnet101():

""" return a ResNet 101 object

"""

return ResNet(BottleNeck, [3, 4, 23, 3])

def resnet152():

""" return a ResNet 152 object

"""

return ResNet(BottleNeck, [3, 8, 36, 3])

if \_\_name\_\_ == "\_\_main\_\_":

model = resnet50()

input = torch.rand(16, 1, 28, 28)

output = model(input)

print(output.shape)

train.py 训练CNN Baseline pipeline

import os.path

import torch

import torch.backends.cudnn as cudnn

from torch.utils.data import DataLoader

from torch.optim.lr\_scheduler import CosineAnnealingLR

from utils.dataloader\_cl import Dataset, dataset\_collate

from utils.trainer import fit\_one\_epoch, draw\_loss, draw\_acc

from nets.model import Baseline2

# from nets.resnet import resnet18

train\_losses = []

train\_acc = []

train\_counter = []

test\_losses = []

test\_acc = []

if \_\_name\_\_ == "\_\_main\_\_":

Cuda = False

# ------------------------------------------------------#

# pretrained\_model\_path 网络预训练权重文件路径

# ------------------------------------------------------#

# pretrained\_model\_path = './logs/ep300-loss0.000.pth'

# pretrained\_model\_path = './checkpoints/baseline2\_ep100.pth'

pretrained\_model\_path = ''

# ------------------------------------------------------#

# input\_shape 输入的shape大小

# ------------------------------------------------------#

input\_shape = [28, 28]

batch\_size = 128

Init\_Epoch = 0

Epoch = 50

# ------------------------------------------------------#

# random seed 随机种子

# ------------------------------------------------------#

random\_seed = 3407

torch.manual\_seed(random\_seed)

# ------------------------------------------------------#

# Init\_lr 初始学习率

# ------------------------------------------------------#

Init\_lr = 0.002

# ------------------------------------------------------------------#

# save\_period 多少个epoch保存一次权值

# ------------------------------------------------------------------#

save\_period = 10

# ------------------------------------------------------------------#

# save\_dir 权值与日志文件保存的文件夹

# ------------------------------------------------------------------#

save\_dir = './checkpoints/'

if not os.path.exists(save\_dir):

os.makedirs(save\_dir)

num\_workers = 0

# ------------------------------------------------------#

# train\_val\_dataset\_path 训练和测试文件路径

# ------------------------------------------------------#

train\_val\_dataset\_path = 'dataset/NewDataset.mat'

# ------------------------------------------------------#

# 设置用到的显卡

# ------------------------------------------------------#

ngpus\_per\_node = torch.cuda.device\_count()

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

# ------------------------------------------------------#

# 创建模型

# ------------------------------------------------------#

model = Baseline2()

# model = resnet18()

if pretrained\_model\_path != '':

print('Load weights {}.'.format(pretrained\_model\_path))

# pretrained\_dict = torch.load(pretrained\_model\_path, map\_location= device)

# model.load\_state\_dict(pretrained\_dict)

layers\_False = ["fc.weight", "fc.bias"]

pretrained\_dict = torch.load(pretrained\_model\_path, map\_location= device)

# 删除有关分类类别的权重

for k in list(pretrained\_dict.keys()):

if k in layers\_False:

del pretrained\_dict[k]

print(model.load\_state\_dict(pretrained\_dict, strict=False))

model\_train = model.train()

if Cuda:

Generator\_train = torch.nn.DataParallel(model)

cudnn.benchmark = True

Generator\_train = Generator\_train.cuda()

# opt\_model = torch.optim.Adam(model.parameters(), lr=Init\_lr)

opt\_model = torch.optim.AdamW(model.parameters(), lr=Init\_lr, betas=(0.9, 0.999), eps=1e-08, weight\_decay=0.01, amsgrad=False)

scheduler = CosineAnnealingLR(opt\_model, T\_max=5, eta\_min=0)

# ---------------------------------------#

# 构建数据集加载器。

# ---------------------------------------#

train\_dataset = Dataset(train\_val\_dataset\_path, input\_shape, epoch\_length=Epoch, is\_train=True, random\_seed=random\_seed)

val\_dataset = Dataset(train\_val\_dataset\_path, input\_shape, epoch\_length=Epoch, is\_train=False, random\_seed=random\_seed)

shuffle = True

train\_gen = DataLoader(train\_dataset, shuffle=shuffle, batch\_size=batch\_size, num\_workers=num\_workers,

pin\_memory=True, drop\_last=True, collate\_fn=dataset\_collate, sampler=None)

val\_gen = DataLoader(val\_dataset, shuffle=shuffle, batch\_size=batch\_size, num\_workers=num\_workers,

pin\_memory=True, drop\_last=True, collate\_fn=dataset\_collate, sampler=None)

# ---------------------------------------#

# 开始模型训练

# ---------------------------------------#

total\_epochs = []

test\_counter = [i \* len(train\_gen.dataset) for i in range(Epoch - Init\_Epoch)]

for epoch in range(Init\_Epoch, Epoch):

epoch\_step = train\_dataset.length // batch\_size

epoch\_step\_val = val\_dataset.length // batch\_size

train\_gen.dataset.epoch\_now = epoch

val\_gen.dataset.epoch\_now = epoch

total\_epochs.append(epoch)

fit\_one\_epoch(model\_train, model, opt\_model, scheduler, epoch, epoch\_step, epoch\_step\_val, train\_gen, val\_gen, Epoch, Cuda, save\_period, save\_dir, train\_losses, train\_acc, train\_counter, test\_losses, test\_acc)

draw\_loss(train\_counter, train\_losses, test\_counter, test\_losses)

draw\_acc(total\_epochs, train\_acc, test\_acc)

trainer.py 单个训练epoch的pipeline以及画图部分

import os

import cv2

import kornia

import numpy

from torch import Tensor

import torch.nn as nn

import torch

from tqdm import tqdm

import matplotlib.pyplot as plt

def draw\_loss(train\_counter, train\_losses, test\_counter, test\_losses):

# draw loss curve

fig = plt.figure()

plt.plot(train\_counter, train\_losses, color='blue')

print(f"test\_counter = {len(test\_counter)}, test\_losses = {len(test\_losses)}")

plt.scatter(test\_counter, test\_losses, color='red')

plt.legend(['Train Loss', 'Valid Loss'], loc='upper right')

plt.xlabel('number of training examples seen')

plt.ylabel('negative log likelihood loss')

plt.savefig(f"./figure/resnet50\_loss.png")

plt.show()

def draw\_acc(total\_epochs, train\_acc, test\_acc):

# draw acc curve

fig = plt.figure()

plt.plot(total\_epochs,train\_acc, color='red')

plt.plot(total\_epochs, test\_acc, color='green')

plt.legend(['Train Acc', 'Valid Acc'], loc='upper right')

plt.xlabel('number of training examples seen')

plt.ylabel('negative log likelihood acc')

plt.savefig(f"./figure/resnet50\_acc.png")

plt.show()

def fit\_one\_epoch(model\_train, model, opt\_model, scheduler, epoch, epoch\_step, epoch\_step\_val, train\_gen, val\_gen, Epoch,

cuda, save\_period, save\_dir, train\_losses, train\_acc, train\_counter, test\_losses, test\_acc):

loss = 0

train\_set = set()

print('Start Train')

criterion = nn.CrossEntropyLoss()

if cuda:

criterion = criterion.cuda()

pbar = tqdm(total=epoch\_step, desc=f'Epoch {epoch + 1}/{Epoch}', postfix=dict, mininterval=0.3)

acc = 0

for iteration, batch in enumerate(train\_gen):

if iteration >= epoch\_step:

break

images, label = batch[0], batch[1] # image (B,C,H,W) label (B)

with torch.no\_grad():

if cuda:

images = images.cuda()

label = label.cuda()

model\_train.train()

prob\_tensor = model\_train(images)

# import pdb; pdb.set\_trace()

class\_index = torch.argmax(prob\_tensor, dim=1)

acc = acc + (label == class\_index).sum().item()

loss\_value = criterion(prob\_tensor, label)

opt\_model.zero\_grad()

loss\_value.backward()

opt\_model.step()

scheduler.step()

loss += loss\_value.item()

train\_losses.append(loss\_value.item())

train\_counter.append(

(iteration \* 64) + ((epoch) \* len(train\_gen.dataset)))

pbar.set\_postfix(\*\*{'loss': loss / (iteration + 1),

'acc': acc / ((iteration + 1) \* label.shape[0])

})

pbar.update(1)

train\_acc.append(acc / ((iteration + 1) \* label.shape[0]))

print('Start test')

pbar.close()

pbar = tqdm(total=epoch\_step\_val, desc=f'Epoch {epoch + 1}/{Epoch}', postfix=dict, mininterval=0.3)

acc = 0

loss = 0

for iteration, batch in enumerate(val\_gen):

if iteration >= epoch\_step\_val:

break

model\_train.eval()

images, label = batch[0], batch[1]

for i in range(label.shape[0]):

train\_set.add(int(label[i]))

with torch.no\_grad():

if cuda:

images = images.cuda()

label = label.cuda()

prob\_tensor = model\_train(images)

class\_index = torch.argmax(prob\_tensor, dim=1)

acc = acc + (label == class\_index).sum().item()

loss\_value = criterion(prob\_tensor, label)

loss += loss\_value.item()

pbar.set\_postfix(\*\*{'acc': acc / ((iteration + 1) \* label.shape[0]),

})

pbar.update(1)

pbar.close()

test\_losses.append(loss / (iteration + 1))

test\_acc.append(acc / ((iteration + 1) \* label.shape[0]))

save\_state\_dict = model.state\_dict()

# save\_state\_dict\_gen = Generator.state\_dict()

if (epoch + 1) % save\_period == 0 or epoch + 1 == Epoch:

torch.save(save\_state\_dict, os.path.join(save\_dir, "baseline2\_ep%03d.pth" % (

epoch + 1)))

torch.save(save\_state\_dict, os.path.join(save\_dir, "last\_epoch\_weights.pth"))

inference.py 可视化展示模型预测结果

import torch

from torch.utils.data import DataLoader

from matplotlib import pyplot as plt

from utils.dataloader\_cl import Dataset, dataset\_collate

from nets.model import Baseline2

def show\_examples(output, example\_data):

fig = plt.figure()

for i in range(16):

plt.subplot(4, 4, i + 1)

plt.tight\_layout()

# 设置字体为楷体

plt.rcParams['font.sans-serif'] = ['KaiTi']

plt.imshow(example\_data[i][0], cmap='gray', interpolation='none')

plt.title("Prediction: {}".format(

output.data.max(1, keepdim=True)[1][i].item()))

plt.xticks([])

plt.yticks([])

plt.show()

def get\_testloader():

train\_val\_dataset\_path = 'dataset/NewDataset.mat'

input\_shape = [28, 28]

random\_seed = 3407

batch\_size = 128

num\_workers = 0

test\_dataset = Dataset(train\_val\_dataset\_path, input\_shape, epoch\_length=1, is\_train=False, random\_seed=random\_seed)

test\_loader = DataLoader(test\_dataset, shuffle=True, batch\_size=batch\_size, num\_workers=num\_workers,

pin\_memory=True, drop\_last=True, collate\_fn=dataset\_collate, sampler=None)

return test\_loader

def process():

model = Baseline2()

network\_state\_dict = torch.load('./checkpoints/baseline2\_ep100.pth')

model.load\_state\_dict(network\_state\_dict)

test\_loader = get\_testloader()

examples = enumerate(test\_loader, start=100)

batch\_idx, (example\_data, example\_targets) = next(examples)

with torch.no\_grad():

output = model(example\_data)

show\_examples(output, example\_data)

if \_\_name\_\_ == "\_\_main\_\_":

process()

源码部分（SVM）

train\_svm.py与CNN的train.py差异不大，详见src

trainer\_svm.py 单轮训练&测试SVM，CV调参（热力图原来想画，但是参考demo没有提供手动特征工程处理方法，于是本次实验我也没有做特征工程，也就没有基于特征的热力图以及RFE特征选择等常规ML操作）

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn.svm import SVC

from sklearn.feature\_selection import RFE

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import accuracy\_score, classification\_report

import pickle

import seaborn as sns

def show\_sns(data, target):

cat = pd.concat((data,target),axis=1)

cat.corr()

plt.figure(figsize=(28,28))

sns.heatmap(cat.corr(),cmap="RdBu\_r", annot=True)

def test\_svm(val\_gen, model\_path):

with open(model\_path, 'rb') as f:

clf = pickle.load(f)

for iteration, batch in enumerate(val\_gen):

if iteration >= 1:

break

val\_images, val\_label = batch[0], batch[1]

val\_pred = clf.predict(val\_images)

accuracy = accuracy\_score(val\_label, val\_pred)

report = classification\_report(val\_label, val\_pred)

print("Test Accuracy:", accuracy)

print("Classification report:", report)

def fit\_one\_epoch(train\_gen, val\_gen):

loss = 0

train\_set = set()

need\_cv = False

need\_test = True

model\_path = './checkpoints/svm\_poly.pkl'

print('Start Train Poly')

# model = SVC(kernel='linear', C=1.0, gamma='scale')

model = SVC(C=1, kernel='poly', degree=8, gamma='scale', coef0=0.0,

shrinking=True, probability=False, tol=1e-4, cache\_size=200,

class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape='ovo', random\_state=500)

parameters = {

"C": [1],

"kernel": ["poly"],

"gamma": ["scale"],

"decision\_function\_shape": ['ovo'],

"tol": [1e-4],

"degree": [7,8,9],

"random\_state": [500],

}

# rfe = RFE(estimator=SVC, n\_features\_to\_select=14, step=1)

for iteration, batch in enumerate(train\_gen):

# if iteration >= 1:

# break

train\_images, train\_label = batch[0], batch[1] # image (B,C,H,W) label (B)

# print(np.shape(train\_images), np.shape(train\_label))

# model.fit(train\_images, train\_label)

# print(iteration)

if need\_cv:

gsearch = GridSearchCV(model, param\_grid=parameters, scoring='accuracy', cv=10)

gsearch.fit(train\_images, train\_label)

print("Best score: %0.4f" % gsearch.best\_score\_)

print("Best parameters set:")

best\_parameters = gsearch.best\_estimator\_.get\_params()

for param\_name in sorted(parameters.keys()):

print("\t%s: %r" % (param\_name, best\_parameters[param\_name]))

model.fit(train\_images, train\_label)

else:

model.fit(train\_images, train\_label)

train\_pred = model.predict(train\_images)

accuracy = accuracy\_score(train\_label, train\_pred)

print(f"iteration {iteration}: train acc: {accuracy}")

# save

with open(model\_path,'wb') as f:

pickle.dump(model,f)

# rfe.fit(train\_images, train\_label)

# print(f"ranking = {rfe.ranking\_}")

if need\_test:

print('Start test')

test\_svm(val\_gen, model\_path)

# acc = 0