A

机器学习实验课程设计

****

**题 目 五位验证码识别**

组长姓名 于晨

组员姓名 沈秋凡 胡睿 赵新怡 程浩

学 院 人工智能学院

专 业 信息工程

任课教师 周媛

**一.课题概论**

**1.背景**

验证码（CAPTCHA）是“Completely Automated Public Turing test to tell Computers and Humans Apart”（全自动区分计算机和人类的图灵测试）的缩写，是一种区分用户是计算机还是人的公共全自动程序。可以防止：恶意破解密码、刷票、论坛灌水，有效防止某个黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登陆尝试，实际上用验证码是现在很多网站通行的方式，我们利用比较简易的方式实现了这个功能。这个问题可以由计算机生成并评判，但是必须只有人类才能解答。由于计算机无法解答CAPTCHA的问题，所以回答出问题的用户就可以被认为是人类。

**二.课题内容**

**1.主要任务**

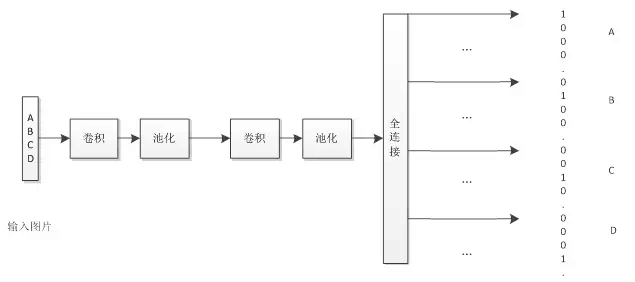
①读懂 5 位验证码的识别程序（验证码包含数字和字母）

②调整网络结构/程序并训练（建议使用 GPU），能够对测试集进行验证显示，并分析网络性能

**2.主要内容**

这其实是一个多标签分类问题，每个验证码图片有5个字符（标签），并且顺序固定；只要将卷积神经网络的最后一层稍加修改就能实现多标签分类。

如下图所示，我们的验证码一共有5个数字，将5个数字转换成40位one\_hot形式，输出层的[0-9]输出值对应第一个字符的onehot编码，[10-19]输出值对应第二个字符的onehot编码，[20-29]输出值对应第三个字符，[30-39]输出值对于第四个字符，并使用pytorch的多标签分类函数nn.MultiLabelSoftMarginLoss作为损失函数。

训练集9999张图片，测试集200张，每张图片大小20\*60

5位验证码为数字和字母的组合，有10个数字和24个英文字母

验证码没有O和I，原因是0和O相近，1和I相近

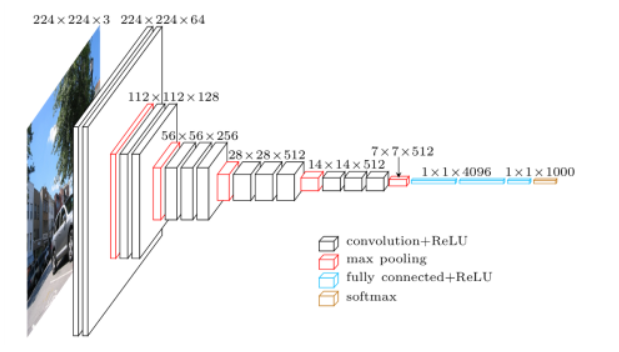
**3.理论技术**

①One-hot热编码

One-Hot编码，又称为一位有效编码，主要是采用N位状态寄存器来对N个状态进行编码，每个状态都由他独立的寄存器位，并且在任意时候只有一位有效。

②VGG神经网络

VGGNet由牛津大学的视觉几何组（Visual Geometry Group）和 Google DeepMind公司的研究员一起研发的的深度卷积神经网络，在 ILSVRC 2014 上取得了第二名的成绩，将 Top-5错误率降到7.3%。目前使用比较多的网络结构主要有ResNet（152-1000层），GooleNet（22层），VGGNet（19层），大多数模型都是基于这几个模型上改进，采用新的优化算法，多模型融合等。到目前为止，VGG Net 依然经常被用来提取图像特征。



VGGNet的特点：

（1）针对网络架构VGGNet的特点

1.针对网络架构:

1.全部使用3\*3的卷积核和2\* 2的池化核,通过不断加深网络结构来提升性能

2.使用多个小卷积核串联组成卷积层,和先前的大卷积核相比,拥有同样的感受野,却有着更少的参数,强的非线性变换,因此有着更强的特征提取能力。

3.使用1\*1卷积层, 1\*1卷积的意义主要在于线性变换,输入通道和输出通道数不变,没有发生降维

4. LRN层作用不大(在本文模型上效果表现不佳,且计算量会增大很多)

（2）.针对过拟合现象:

1.数据增强,使用Multi-Scale的方法做数据增强,将原始图像缩放到不同的尺寸S ,在随机裁取固定大小的图片,这样能增加很多数据量,可以防止模型过拟合

2.在预测时:使眵裁剪(Multi-Scale)的多尺度(Multi-Scale)配合(论文的结构正是这两者配合使用,效果比单使用更好,可以说是两者互补(起码不是互斥K)) ,可以很好的提升模型的性能。

**（3）.针对训练速度:**

1.可以先训练底深度的A网络,再复用A网络的权重初始化后面的几个复杂模型,这样训练收敛的速度更快

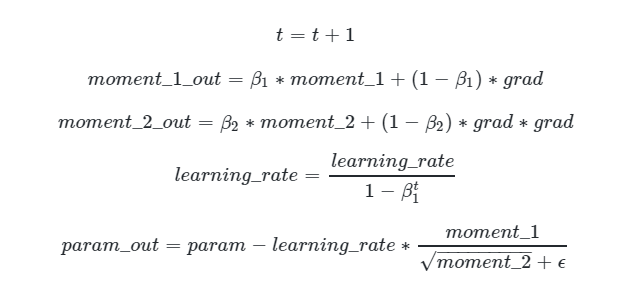
③torch.optim

torch.optim是一个实现了各种优化算法的库。大部分常用的方法得到支持，并且接口设备具有足够的通用性，使得能够集成更加复杂的算法。

为了使用torch.optim我们需要构建一个optimizer对象，这个对象能够保持当前参数状态并基于计算得到的梯度进行参数更新。在这一个实验中，我们给予它的是一个adam优化器。

Adam优化器出自 Adam论文 的第二节，能够利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。

其参数更新的计算公式如下：



参数：

·params (iterable) – 待优化参数的iterable或者是定义了参数组的dict

·lr (float, 可选) – 学习率（默认：1e-3）

·betas (Tuple[float, float], 可选) – 用于计算梯度以及梯度平方的运行平均值的系数（默认：0.9，0.999）

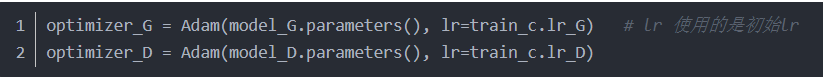
·eps (float, 可选) – 为了增加数值计算的稳定性而加到分母里的项（默认：1e-8）

·weight-decay (float, 可选) – 权重衰减（L2惩罚）（默认: 0）

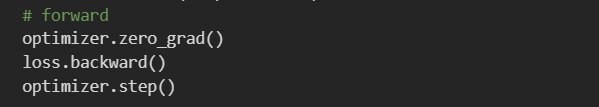
# ④Pytorch optimizer.step() 和loss.backward()的关系

首先需要明确optimzier优化器的作用, 形象地来说，优化器就是需要根据网络反向传播的梯度信息来更新网络的参数，以起到降低loss函数计算值的作用，这也是机器学习里面最一般的方法论。

从优化器的作用出发，要使得优化器能够起作用，需要主要两个东西：【1】. 优化器需要知道当前的网络或者别的什么模型的参数空间，这也就是为什么在训练文件中，正式开始训练之前需要将网络的参数放到优化器里面，比如使用pytorch的话总会出现类似如下的代码：



【2】. 需要知道反向传播的梯度信息



从上面的代码可以看到step这个函数使用的是参数空间中的grad,也就是当前参数空间对应的梯度。因为如果不清零，那么使用的这个grad就得同上一个mini-batch有关，这不是我们需要的结果。再回过头来看，我们知道optimizer更新参数空间需要基于反向梯度，因此，当调用optimizer.step()的时候应当是loss.backward()的时候（loss.backward()的具体运算过程可以参看Pytorch 入门），这也就是经常会碰到,如下情况

loss.backward()在前，然后跟一个step。

那么为什么optimizer.step()需要放在每一个batch训练中，而不是epoch训练中，这是因为现在的mini-batch训练模式是假定每一个训练集就只有mini-batch这样大，因此实际上可以将每一次mini-batch看做是一次训练，一次训练更新一次参数空间，因而optimizer.step()放在这里。

scheduler.step（）按照Pytorch的定义是用来更新优化器的学习率的，一般是按照epoch为单位进行更换，即多少个epoch后更换一次学习率，因而scheduler.step()放在epoch这个大循环下。

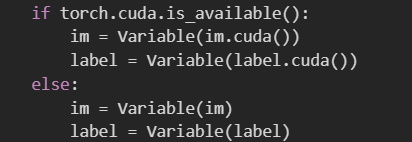
⑤cuda

CUDA（Compute Unified Device Architecture），是显卡厂商NVIDIA推出的运算平台。 CUDA™是一种由NVIDIA推出的通用并行计算架构，该架构使GPU能够解决复杂的计算问题。 它包含了CUDA指令集架构（ISA）以及GPU内部的并行计算引擎。

由于实验中数据集较大，单独使用CPU运算将耗费大量的时间，所以我们调用GPU并行计算。



为CUDA提供数据



**4.理解和想法**

实验过程前期，我们小组发现实验结果的准确率一直为0，但是损失却是较为稳定，我们考虑了很多种情况。我们怀疑实产生了过拟合，为此我们增加了惩罚项，可是结果还是和一开始一样没有产生什么变化。

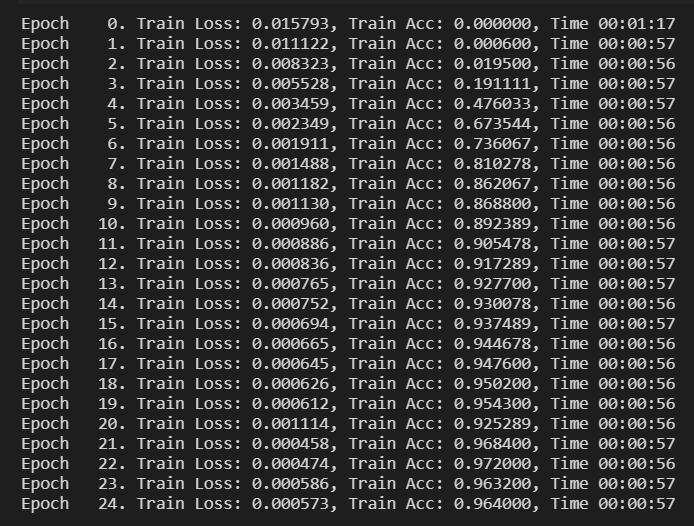
为了研究问题所在，我们将三张训练集中的图片复制到测试集中，很快我们发现准确率也产生了变化而不是一开始的0，由此我们认为主要问题并不在算法身上。

同时经过仔细的检查和分析，我们认为算法本身并不存在问题。根本问题是训练集太小导致机器学习的训练广度不够，以至于产生过拟合，无法得出应有的结果。我们通过验证码生成算法获得了9999张图片，将它们作为训练集代替原来的1000个样本的训练集，验证结果的准确率也很快到达了我们所期望的数字。

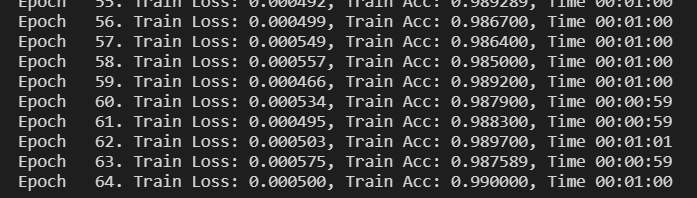
神经元学习系统一定要依赖一个足够大训练集来满足学习系统的需求，不然的话很容易因为样本数量太少，从而产生过拟合。

**5.项目结果**

通过GPU运算，batsize为10，Epoch为65，耗时约1h



……



6/29/1/28/6 6/29/1/28/6

6/20/28/31/31 6/20/28/31/4

27/31/2/11/7 27/31/2/11/7

20/7/26/14/18 20/7/26/14/18

17/11/2/11/8 17/11/2/11/8

3/29/6/6/0 3/27/29/6/0

24/21/21/19/11 24/21/21/19/11

33/19/19/10/21 33/19/19/10/21

20/30/16/7/10 20/30/16/7/10

14/3/0/12/25 14/3/0/12/25

22/2/1/2/27 22/2/1/2/27

7/7/32/3/20 7/6/32/3/20

0/23/22/6/13 0/23/22/6/13

23/10/16/30/17 23/10/16/30/17

7/20/15/6/14 7/20/15/6/14

5/25/17/18/24 5/25/17/18/24

29/15/8/20/20 29/15/8/20/20

31/21/15/10/17 31/21/15/10/17

21/16/24/27/5 21/16/24/27/5

16/1/11/11/27 16/1/11/11/27

28/21/9/17/31 28/21/9/17/31

18/27/10/14/28 18/27/10/14/28

10/16/7/30/8 10/16/7/30/8

30/13/4/31/10 30/13/4/31/10

3/8/21/29/22 3/8/21/29/22

8/22/29/11/22 8/22/29/11/22

32/5/12/13/23 32/5/12/13/23

4/10/11/9/23 4/10/11/9/23

31/33/23/0/25 31/33/23/0/25

19/4/7/1/29 19/4/7/1/31

19/12/10/31/15 19/12/10/31/15

9/6/24/5/29 9/6/24/5/29

17/26/30/23/28 17/26/30/23/28

33/24/16/23/9 33/24/16/23/9

25/32/21/2/22 25/32/21/2/22

14/3/12/7/7 14/3/12/7/29

19/7/29/12/6 19/7/29/12/6

13/5/26/22/33 13/5/26/22/33

19/14/3/7/20 19/14/3/7/20

23/16/11/6/31 23/16/11/6/31

11/0/12/32/10 11/0/12/32/10

22/28/33/5/21 22/28/33/5/21

2/6/16/23/26 2/6/16/23/26

5/26/24/6/28 5/26/24/6/28

14/27/14/26/28 14/27/14/26/28

6/0/3/6/2 6/0/3/6/2

24/2/14/27/11 24/2/14/27/11

8/21/23/0/6 8/21/23/0/6

23/10/22/14/3 23/10/22/14/3

30/6/29/30/33 30/6/29/30/33

27/15/13/10/15 27/15/13/10/15

2/13/1/31/12 2/13/1/31/12

11/13/30/12/10 11/13/30/12/10

28/0/27/27/26 28/0/27/27/26

18/24/28/29/19 18/24/28/29/19

2/5/27/8/24 2/5/27/8/24

24/2/26/14/24 24/2/26/14/24

21/3/10/24/9 21/3/10/24/9

14/17/11/32/32 14/17/11/32/32

25/29/29/9/16 25/14/29/9/16

15/3/14/22/2 15/3/14/22/2

18/1/7/18/27 18/1/7/18/27

19/20/32/2/17 19/20/32/2/17

29/33/12/11/30 29/33/12/11/30

17/16/15/29/26 17/16/15/29/26

19/17/17/12/6 19/17/17/12/6

32/18/4/3/0 32/18/4/3/0

1/32/16/29/16 1/32/16/29/16

24/12/14/28/30 24/12/14/28/30

8/15/15/9/2 8/15/15/9/2

21/29/13/11/13 21/29/13/11/13

25/22/20/11/20 25/22/20/11/20

0/8/9/15/3 0/8/9/15/3

28/16/9/0/18 28/16/9/0/18

30/33/14/12/29 30/33/14/12/29

22/24/2/18/25 22/24/2/18/25

23/5/23/14/6 23/5/23/14/6

1/2/8/8/5 1/2/8/8/5

5/1/1/17/19 5/1/1/17/19

3/13/13/2/8 3/13/13/2/8

9/22/2/28/12 9/22/2/28/12

26/33/11/7/30 26/33/11/7/30

21/30/23/25/30 21/30/23/25/30

12/3/20/19/33 12/3/20/19/33

10/16/14/20/7 10/16/14/20/7

31/10/29/21/0 31/10/29/21/0

22/31/12/18/25 22/31/12/18/25

15/28/28/25/12 15/25/28/25/12

28/21/11/13/7 28/21/11/13/7

30/4/10/26/2 30/4/10/26/2

18/3/7/12/10 18/3/7/12/10

22/26/3/13/0 22/26/3/13/0

10/31/26/16/9 10/31/26/16/9

28/27/20/20/15 28/27/20/20/15

11/2/23/9/20 11/2/23/9/20

4/32/21/32/19 4/32/21/32/19

7/11/10/12/12 7/11/10/12/12

26/17/29/21/24 26/17/29/21/24

0/3/29/24/20 0/3/29/24/20

21/25/16/9/7 21/25/16/9/7

23/4/9/4/13 23/4/9/4/13

25/3/12/9/21 25/3/12/9/21 1

2/29/32/2/23 12/29/32/2/23

0/30/6/4/1 0/30/6/4/1

11/16/20/13/10 11/16/20/13/10

22/22/3/8/32 22/22/3/8/32

0/7/17/13/16 0/7/17/13/16

26/6/32/31/32 26/6/32/31/32

5/3/4/23/5 5/3/4/23/5

14/11/1/24/4 14/11/1/24/4

26/23/28/12/0 26/23/28/12/0

16/22/6/29/33 16/22/6/29/33

3/31/22/24/23 3/31/22/24/23

1/29/22/10/33 1/29/22/10/33

13/5/32/25/7 13/5/32/25/7

9/22/17/12/0 9/22/17/12/0

17/15/4/30/25 17/15/4/30/25

1/1/22/11/27 1/1/22/11/27

19/11/7/31/10 19/11/7/31/10

10/20/7/0/21 10/20/7/0/21

17/10/26/15/8 17/10/26/15/8

8/16/15/12/14 8/16/15/12/14

32/26/0/13/26 32/26/0/13/26

6/23/17/25/32 6/23/17/25/32

26/20/23/16/7 26/20/23/16/7

25/9/6/14/2 25/9/6/14/2

13/19/27/15/14 13/19/27/15/14

12/19/13/5/22 12/19/13/5/22

11/12/19/4/24 11/12/19/32/24

8/6/25/21/20 8/6/25/21/20

11/14/1/14/15 11/14/1/14/15

32/17/11/17/27 32/17/11/17/27

17/21/19/18/11 17/21/19/18/11

23/33/18/12/32 23/33/18/12/32

30/1/18/2/33 30/1/18/2/33

28/24/4/7/7 28/24/4/7/7

5/21/5/12/26 5/21/5/12/26

2/20/11/10/1 2/20/11/10/1

33/17/19/3/28 33/17/19/3/28

30/14/4/31/10 30/14/4/31/10

26/1/26/23/32 26/1/26/23/32

28/31/29/19/10 28/31/29/19/10

13/12/14/0/29 13/12/14/0/29

26/7/27/0/1 26/7/27/25/1

20/2/1/2/9 20/2/1/2/9

14/30/6/2/30 14/30/6/2/30

24/12/29/23/8 24/12/29/23/8

3/8/19/24/20 3/8/19/24/20

25/20/14/14/31 25/20/14/14/31

33/29/28/8/33 33/29/28/8/33

2/1/25/11/16 2/1/25/11/16

16/32/27/28/10 16/32/27/28/10

25/18/12/12/12 25/18/12/12/12

11/6/0/4/32 11/6/0/4/32

8/18/5/12/18 8/18/5/12/18

27/23/33/27/31 27/23/33/27/31

11/6/24/27/18 11/6/24/27/18

30/13/11/26/4 30/13/26/26/4

11/33/11/25/9 11/33/11/25/9

26/9/2/12/23 26/9/2/12/23

16/27/1/28/25 16/27/1/28/25

29/13/14/21/8 29/13/14/21/8

14/22/10/29/14 14/22/10/29/14

31/26/18/23/24 31/26/28/23/24

26/20/27/27/8 26/20/27/27/8

17/0/31/3/28 17/0/31/3/28

21/10/17/22/27 21/10/17/22/27

16/8/33/32/14 16/8/33/32/14

29/13/26/2/17 29/13/26/2/17

17/6/2/6/17 17/6/2/6/17

10/6/31/2/9 10/6/31/2/9

11/11/29/25/33 11/11/29/25/33

14/9/1/23/27 14/9/1/23/27

26/26/20/9/27 26/26/20/9/27

2/25/2/32/31 2/25/2/32/31

14/8/8/23/31 14/8/8/23/31

29/31/6/6/31 29/31/6/6/31

3/12/26/12/2 3/12/26/12/2

6/33/14/15/21 6/33/14/15/21

32/21/1/30/20 32/21/1/30/20

2/33/21/1/14 2/33/21/1/14

10/8/6/16/10 10/8/6/16/10

7/1/15/16/21 7/1/15/16/21

24/8/29/8/8 24/8/29/8/8

12/19/3/10/12 12/19/3/10/12

25/25/13/1/9 25/25/13/1/9

12/32/14/12/12 12/32/14/12/12

1/25/5/27/15 1/25/5/27/15

18/7/30/21/25 18/7/30/21/25

0/22/4/23/4 0/22/4/23/4

9/22/7/17/11 9/22/7/17/11

14/16/13/9/6 14/16/13/9/6

9/0/16/12/23 9/0/16/12/23

20/10/8/10/27 20/10/8/10/6

19/22/10/11/11 19/22/10/21/11

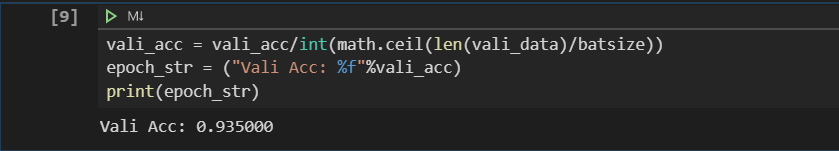
15/8/6/10/25 15/8/6/10/25

20/22/3/13/27 20/22/3/13/27

31/2/21/4/23 31/2/21/4/23

20/19/29/22/30 20/19/29/22/30

25/0/0/28/3 25/0/0/28/3



准确率为93.5%

**三.课题工作安排**

·于晨（20171309003）负责编写验证集套用训练模型进行验证的代码，并修改了验证码生成代码以扩充训练集

·程浩（20171309081）负责代码调试

·沈秋凡（20161309015）负责代码运行、实验报告的撰写

·胡睿（20171309072）、赵新怡（20171309076）负责搜集资料

**四.参考文献资料**

[1].飞桨Paddle API文档 图像分类 <https://www.paddlepaddle.org.cn/documentation/docs/zh/beginners_guide/basics/image_classification/index.html#optimizer-function>

[2].Pytorch多标签CNN端到端验证码识别

https://www.jianshu.com/p/08e9d2669b42

[3]. <https://arxiv.org/pdf/1409.1556.pdf>

[4]. <https://pytorch-cn.readthedocs.io/zh/latest/package_references/torch-optim/#torchoptim>

[5]. <https://blog.csdn.net/xiaoxifei/article/details/87797935>

**附录1：**

**主程序**：

import sys

from PIL import Image

import numpy as np

import csv

import torch

from torch import nn

from torch.autograd import Variable

import cv2

import datetime

import math

LETTERSTR = "0123456789ABCDEFGHJKLMNPQRSTUVWXYZ"

batsize=10

num\_epochs=65

def toonehot(text):

    labellist = []

    for letter in text:

        onehot = [0 for \_ in range(34)]

        num = LETTERSTR.find(letter)

        onehot[num] = 1

        labellist.extend(onehot)  #every label size is 1\*170, according with LETTERSTR

#         labellist.append(onehot)

    return labellist

def vgg\_block(num\_convs, in\_channels, out\_channels):

    net = [nn.Conv2d(in\_channels, out\_channels, kernel\_size=3, padding=1), nn.ReLU(True)]

    for i in range(num\_convs - 1):  # 定义后面的许多层

        net.append(nn.Conv2d(out\_channels, out\_channels, kernel\_size=3, padding=1))

        net.append(nn.ReLU(True))

    net.append(nn.BatchNorm2d(out\_channels))

    net.append(nn.MaxPool2d(2, 2))  # 定义池化层

    net.append(nn.Dropout(0.3))

    return nn.Sequential(\*net)

# 下面我们定义一个函数对这个 vgg block 进行堆叠

def vgg\_stack(num\_convs, channels):

    net = []

#     print(zip(num\_convs, channels))

    for n, c in zip(num\_convs, channels):

        in\_c = c[0]

        out\_c = c[1]

        net.append(vgg\_block(n, in\_c, out\_c))

    return nn.Sequential(\*net)

class vgg(nn.Module):

    def \_\_init\_\_(self):

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

        self.feature = vgg\_net

        self.fc = nn.Sequential(

            nn.Linear(3\*12\*256, 2560),

            nn.Dropout(0.3),

            nn.Linear(2560, 170)# 34\*5=170

        )

    def forward(self, x):

        x = self.feature(x)

        x = x.view(x.shape[0], -1)

        x = self.fc(x)

        return x

    # 然后我们可以训练我们的模型看看在 cifar10 上的效果

    def data\_tf(x):

        x = np.array(x, dtype='float32') / 255

        x = (x - 0.5) / 0.5

        x = x.transpose((2, 0, 1))  ## 将 channel 放到第一维，只是 pytorch 要求的输入方式

        x = torch.from\_numpy(x)

        return x

net = vgg()

optimizer = torch.optim.Adam(net.parameters(), lr=0.001)

criterion = nn.MultiLabelSoftMarginLoss()

# print(net)

print("Reading training data...")

traincsv = open('data/5\_imitate\_train\_set/captcha\_train.csv', 'r', encoding = 'utf8')

train\_data = np.stack([np.array(cv2.imread("data/5\_imitate\_train\_set/" + row[0] + ".jpg"))/255.0 for row in csv.reader(traincsv)])

train\_data = train\_data.transpose(0,3,1,2)

traincsv = open('data/5\_imitate\_train\_set/captcha\_train.csv', 'r', encoding = 'utf8')

train\_label = [toonehot(row[1]) for row in csv.reader(traincsv)]

# print(train\_label[0:2])

print("Shape of train data:", train\_data.shape)

# print("Shape of train label:", len(train\_label))

print("Reading validation data...")

valicsv = open('data/5\_imitate\_vali\_set/captcha\_vali.csv', 'r', encoding = 'utf8')

vali\_data = np.stack([np.array(Image.open("./data/5\_imitate\_vali\_set/" + row[0] + ".jpg"))/255.0 for row in csv.reader(valicsv)])

vali\_data = vali\_data.transpose(0,3,1,2)

valicsv = open('data/5\_imitate\_vali\_set/captcha\_vali.csv', 'r', encoding = 'utf8')

vali\_label = [toonehot(row[1]) for row in csv.reader(valicsv)]

# vali\_label = [[] for \_ in range(5)]

# for arr in read\_label:

#     for index in range(5):

#         vali\_label[index].append(arr[index])

# vali\_label = [arr for arr in np.asarray(vali\_label)]

print("Shape of validation data:", vali\_data.shape)

# print("Shape of validation label:", vali\_label[0].shape)

def get\_acc(output, label):

    correct\_num =0

    for i in range(output.size()[0]):

        c0 = np.argmax(output[i, 0:34].data.cpu().numpy())

        c1 = np.argmax(output[i, 34:68].data.cpu().numpy())

        c2 = np.argmax(output[i, 68:102].data.cpu().numpy())

        c3 = np.argmax(output[i, 102:136].data.cpu().numpy())

        c4 = np.argmax(output[i, 136:170].data.cpu().numpy())

        c = '%s/%s/%s/%s/%s' % (c0, c1, c2, c3,c4)

        l0 = np.argmax(label[i, 0:34].data.cpu().numpy())

        l1 = np.argmax(label[i, 34:68].data.cpu().numpy())

        l2 = np.argmax(label[i, 68:102].data.cpu().numpy())

        l3 = np.argmax(label[i, 102:136].data.cpu().numpy())

        l4 = np.argmax(label[i, 136:170].data.cpu().numpy())

        l = '%s/%s/%s/%s/%s' % (l0, l1, l2, l3,l4)

        print(c,l)

        if l==c:

            correct\_num += 1

    return float(correct\_num)/ len(output)

if torch.cuda.is\_available():

    net = net.cuda()

prev\_time = datetime.datetime.now()

iters = int(math.ceil(train\_data.shape[0]/batsize))

for epoch in range(num\_epochs):

    train\_loss = 0

    train\_acc = 0

    net = net.train()

    for i in range(iters):

        temp\_acc = 0

        im = train\_data[i\*batsize: (i+1)\*batsize]

        im = torch.tensor(im).float()

        label = train\_label[i\*batsize: (i+1)\*batsize]

        label = torch.tensor(label).float()

        if torch.cuda.is\_available():

            im = Variable(im.cuda())

            label = Variable(label.cuda())

        else:

            im = Variable(im)

            label = Variable(label)

        # forward

        output = net(im)

#         print(output.shape)

#         print(label.shape)

        loss = criterion(output, label)

        # forward

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

        train\_loss += loss.item()

#         print(output.shape)

#         print(label.shape)

        train\_acc += get\_acc(output, label)

    cur\_time = datetime.datetime.now()

    h, remainder = divmod((cur\_time - prev\_time).seconds, 3600)

    m, s = divmod(remainder, 60)

    time\_str = "Time %02d:%02d:%02d" % (h, m, s)

    prev\_time = cur\_time

    epoch\_str = ("Epoch %4d. Train Loss: %f, Train Acc: %f, " %

                     (epoch, train\_loss / len(train\_data),

                      train\_acc / iters))

    print(epoch\_str + time\_str)

torch.save(net,'./models/model')

net=torch.load('./models/model')

net.eval()

vali\_acc = 0

for i in range(int(math.ceil(len(vali\_data)/batsize))):

    im = vali\_data[i\*batsize: (i+1)\*batsize]

    im = torch.tensor(im).float()

    label = vali\_label[i\*batsize: (i+1)\*batsize]

    label = torch.tensor(label).float()

    if torch.cuda.is\_available():

        im = Variable(im.cuda())

        label = Variable(label.cuda())

    else:

        im = Variable(im)

        label = Variable(label)

    output = net(im)

    vali\_acc += get\_acc(output, label)

vali\_acc = vali\_acc/int(math.ceil(len(vali\_data)/batsize))

epoch\_str = ("Vali Acc: %f"%vali\_acc)

print(epoch\_str)

**附录二：**

**验证码生成代码：**

from PIL import Image, ImageDraw, ImageFont

from random import randint

import sys

import os

import csv

import numpy as np

FONTPATH = ["./data/font/times-bold.ttf", "./data/font/courier-bold.ttf"]

ENGSTR = "ABCDEFGHJKLMNPQRSTUVWXYZ" # æ²’æœ‰Oå’ŒI

class rect:

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

self.size = (randint(5, 21), randint(5, 21))

self.location = (randint(1, 199), randint(1, 59))

self.luoverlay = True if randint(1, 10) > 6 else False

self.rdoverlay = False if self.luoverlay else True if randint(

1, 10) > 8 else False

self.lucolor = 0 if randint(0, 1) else 255

self.rdcolor = 0 if self.lucolor == 255 else 255

self.ludrawn = False

self.rddrawn = False

self.pattern = randint(0, 1)

def draw(self, image, overlay):

if((overlay or not self.luoverlay) and not self.ludrawn):

self.ludrawn = True

stp = self.location

transparent = int(255 \* 0.45 if self.lucolor == 0 else 255 \* 0.8)

color = (self.lucolor, self.lucolor, self.lucolor, transparent)

uline = Image.new("RGBA", (self.size[0], 1), color)

lline = Image.new("RGBA", (1, self.size[1]), color)

image.paste(uline, stp, uline)

image.paste(lline, stp, lline)

if((overlay or not self.rdoverlay) and not self.rddrawn):

self.rddrawn = True

dstp = (self.location[0], self.location[1] + self.size[1])

rstp = (self.location[0] + self.size[0], self.location[1])

transparent = int(255 \* 0.45 if self.rdcolor == 0 else 255 \* 0.8)

color = (self.rdcolor, self.rdcolor, self.rdcolor, transparent)

dline = Image.new("RGBA", (self.size[0], 1), color)

rline = Image.new("RGBA", (1, self.size[1]), color)

image.paste(dline, dstp, dline)

image.paste(rline, rstp, rline)

class captchatext:

def \_\_init\_\_(self, priority, offset, captchalen, engletter, ENGNOLIMIT):

self.engletter = engletter

if ENGNOLIMIT:

engletter = True if randint(1, 34) <= 24 else False

if engletter:

self.letter = ENGSTR[randint(0, len(ENGSTR) - 1)]

else:

self.letter = str(randint(0, 9))

self.color = [randint(10, 140) for \_ in range(3)]

self.angle = randint(-55, 55)

self.priority = priority

self.offset = offset

self.next\_offset = 0

self.captchalen = captchalen

def draw(self, image):

color = (self.color[0], self.color[1], self.color[2], 255)

font = ImageFont.truetype(

FONTPATH[randint(0, 1)], randint(25, 27) \* 10)

text = Image.new("RGBA", (font.getsize(

self.letter)[0], 300), (0, 0, 0, 0))

textdraw = ImageDraw.Draw(text)

textdraw.text((0, 0), self.letter, font=font, fill=color)

text = text.rotate(self.angle, expand=True)

text = text.resize((int(text.size[0] / 10), int(text.size[1] / 10)))

base = int(self.priority \* (200 / self.captchalen))

rand\_min = (self.offset - base - 4) if (self.offset -

base - 4) >= -15 else -15

rand\_min = 0 if self.priority == 0 else rand\_min

avg\_dp = int(200 / self.captchalen)

rand\_max = (

avg\_dp - text.size[0]) if self.priority == self.captchalen - 1 else (avg\_dp - text.size[0] + 10)

try:

displace = randint(rand\_min, rand\_max)

except:

displace = rand\_max

location = (base + displace, randint(3, 23))

self.next\_offset = location[0] + text.size[0]

image.paste(text, location, text)

def generate(GENNUM, SAVEPATH, ENGP=25, FIVEP=0, ENGNOLIMIT=False):

captchacsv = open(SAVEPATH + "captcha\_train.csv",

'w', encoding='utf8', newline='')

numcsv = open(SAVEPATH+"len\_train.csv", 'w', encoding='utf8', newline='')

letterlist = []

numlist = []

for index in range(1, GENNUM + 1, 1):

print(index, end='\r')

captchastr = ""

captchalen = 5 if randint(1, 100) <= FIVEP else 5

engat = randint(0, captchalen - 1) if randint(1, 100) <= ENGP else -1

bgcolor = [randint(180, 250) for \_ in range(3)]

captcha = Image.new('RGBA', (200, 60),

(bgcolor[0], bgcolor[1], bgcolor[2], 255))

rectlist = [rect() for \_ in range(32)]

for obj in rectlist:

obj.draw(image=captcha, overlay=False)

offset = 0

for i in range(captchalen):

newtext = captchatext(i, offset, captchalen,

(True if engat == i else False), ENGNOLIMIT)

newtext.draw(image=captcha)

offset = newtext.next\_offset

captchastr += str(newtext.letter)

if index > 999:

picname = str(index)

elif index > 99:

picname = '0'+str(index)

elif index > 9:

picname = '00'+str(index)

else:

picname = '000'+str(index)

letterlist.append([picname, captchastr])

numlist.append([picname, '5'])

for obj in rectlist:

obj.draw(image=captcha, overlay=True)

captcha.convert("RGB").save(SAVEPATH + picname + ".jpg", "JPEG")

writer = csv.writer(captchacsv)

writer.writerows(letterlist)

captchacsv.close()

writer = csv.writer(numcsv)

writer.writerows(numlist)

numcsv.close()

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

generate(9999, "./data/5\_imitate\_train\_set/", ENGP=100, FIVEP=50, ENGNOLIMIT=True)