**1.选择题**

1. A
2. AC
3. D
4. BC
5. B

**2.概念学习**

1. 样本一和样本二：

S0：{<<Ф Ф Ф Ф>,<Ф Ф Ф Ф>>}

S1: {<<male brown tall US> ，<female black short US>>}

S2: {<<male brown ？？>，<female black short US>>}

G0，G1，G2:<< ？ ？ ？ ？> , < ？ ？ ？ ？>>

②样本三：

S3: {<<male brown ？？>，<female black short US>>}

G3：{<<male ？？？>，<？？？？>>; <<？？？？>，<？？？US>>}

③样本四：

S4: {<<male brown ？？>，<female ？ short ？>>}

G4：{<<male ？？？>，<？？？？>>}

1. 28
2. 最短查询序列：

<<**female** black short Portuguese> ，<female blonde tall Indian>>

<<male **brown** short Portuguese> ，<female blonde tall Indian>>

<<male black **tall** Portuguese> ，<female blonde tall Indian>>

<<male black short **US**> ，<female blonde tall Indian>>

<<male black short Portuguese> ，<**male** blonde tall Indian>>

<<male black short Portuguese> ，<female **brown** tall Indian>>

<<male black short Portuguese> ，<female blonde **short** Indian>>

<<male black short Portuguese> ，<female blonde tall **US**>>

由上述可知，序列的长度为8，为（b）中答案的指数即 = 8。

1. 如果需要扩展这一语言，使其能够表达该实例语言上的所有概念，则假设空间变为实例集X的幂集。X的元素数目为：

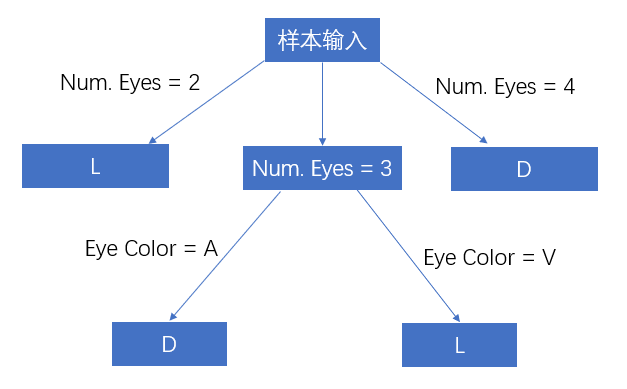
|X| = 2\*3\*3\*7\*2\*3\*3\*7 = 15876

序列的长度为：

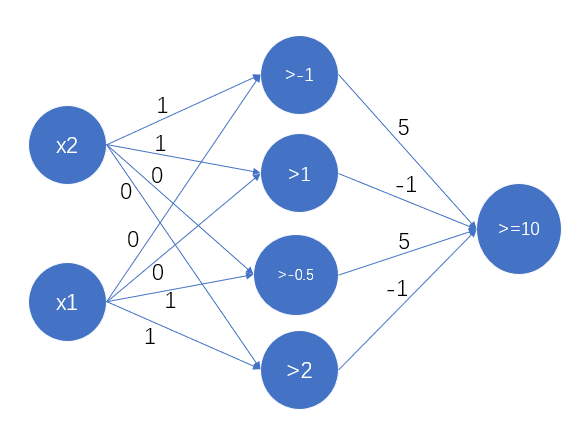
-1+ = 2\*3\*3\*7\*2\*3\*3\*7 – 1 =15875

更改后的序列应该每次只确定1个属性的值，使其与最后的单个正确的假设相一致，以此类推得到整个查询序列。

**3.决策树**

1. H(EyeColor|Weight = N) = -3/5 log 3/5- 2/5 log 2/5 = 0.966Num. Eyes
2. Num. Eyes
3. 
4. 无错误

**4.神经网络**



**5.上机题**

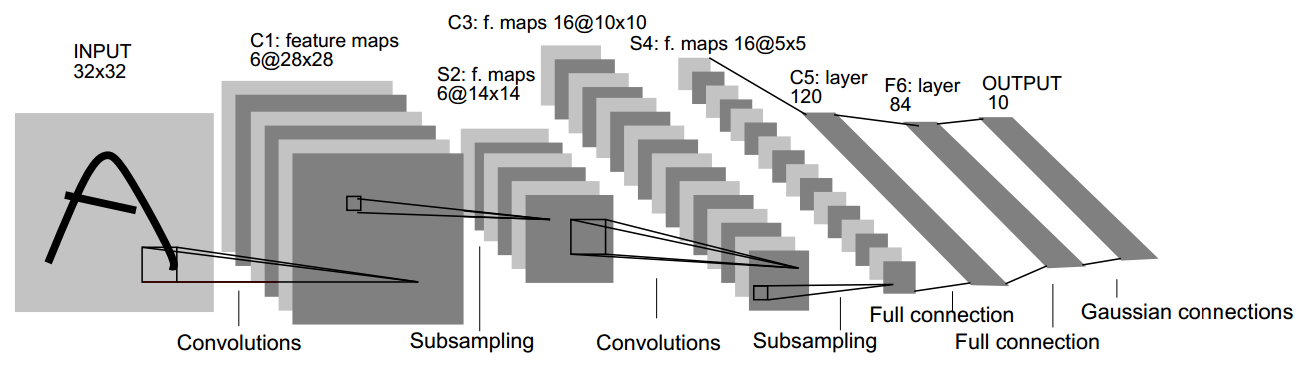
**交通灯的识别与分类**

上机题将使用深度学习的方法来识别交通灯，对交通灯进行分类。共有9类，分别是红色圆圈，绿色圆圈，红色左转向，绿色右转向，红色前进，绿色前进，红色右转，绿色右转，黄灯，负样本。

**一、背景及研究现状**

比起深度学习，机器学习这一词更加耳熟能祥。机器学习致力于通过算法从经验中自动获取所需要的知识。而经验往往对应以特征形式存储的数据，算法则是通过数据得到最符合数据分布的机器学习模型。这里的第一步就是要获取特征数据，而这也是最难的一步。人们一开始通过试错的方式来设计特征，但这样太耗时耗力，同时迁移到其他类型的任务，效果不好。而在有些问题上，人们是不知道如何去设计特征的，特别是图像问题，如何去识别一只猫在过去是很困难的一项任务。根据这一问题，人们就想能不能跟自动学到最优模型一样，将最优特征也通过学习的方式习得。这样，就诞生了表示学习。而深度学习则是表示学习的一个经典代表。深度学习以数据原始形态（没有经过任何处理）作为原始输入，通过算法将数据层层抽象成任务所需要的特征，然后将特征映射到目标，这中间没有任何的人工操作，全是由机器自动完成整个操作。层层叠进和多模块叠加，这也许是被称之“深度”学习的原因吧。深度学习中的一类代表性算法是神经网络算法，包括卷积神经网络和递归神经网络等等。其中卷积神经网络在图像领域应用十分广泛，也是主要探讨和应用的算法。

**二、基本原理**



以上图的LeNet-5来介绍原理，根据LeNet-5模型，我们可以知道cnn的各个组件，分别是输入层、卷积层、激活函数层、池化层、全连接层以及输出层。输入层很简单，就是大小为32x32的字符图片。卷积层是cnn的核心，它会对输入卷积层的图片进行卷积操作，上图的第一层卷积则是输入图像与6个卷积核进行卷积，得到6张特征图，卷积核可看做一种滤波器，它会学习输入图像的局部信息，不同的卷积核所学到的局部信息不尽相同。然后就是激活函数层，激活函数是层与层之前的桥梁，也是模型复杂度的关键。LeNet-5用的是sigmiod，现在用得更多的是Relu以及它的各种变体。池化层又称之下采样，它的目的是为减小数据量计算以及保留最有用的信息，同时防止过拟合，属于一种特征降维方式，LeNet-5用的是平均池化，即所选取的像素块用像素块所有像素和的均值替换，常见还有最大池化，即保留像素块中像素值最大的像素，也是最具有信息量的部分。全连接层是为了将前面层所学习到的特征映射到最终目标，最后得到输出层。得到输出层还没完，还需要将得到的输出与目标进行误差计算，然后将误差反向传播，更新网络的参数。

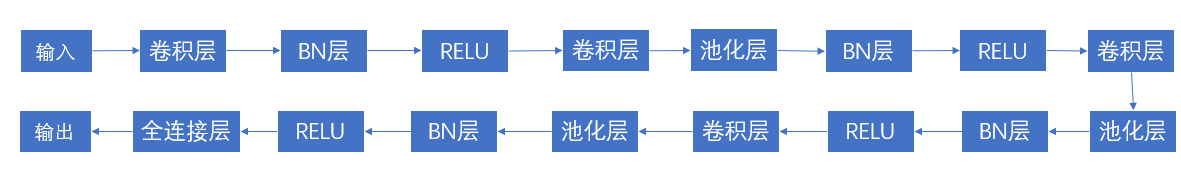
总的来说，cnn可分别两部分，分别是前馈运算和反馈运算。前馈运算就是输入层向输出层进行值传递，其中卷积层，激活函数层以及池化层是将输入数据层层映射到高维特征，全连接层则是将高维特征组合映射到目标，得到输出。反馈运算则是从输出层向输入层进行误差传播，更新每一层的参数值，以达到最优模型。

**三、算法流程以及网络结构**

**算法流程：**

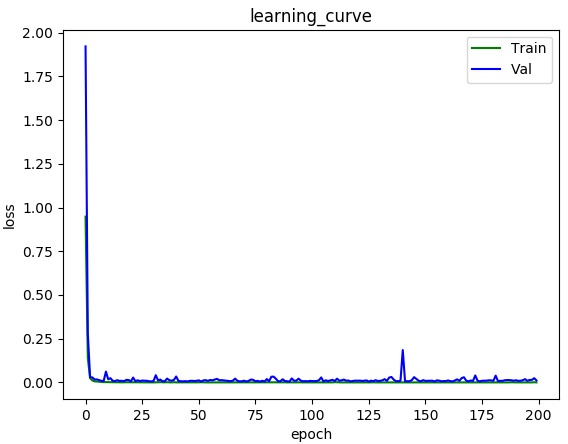
具体结构与上述的LeNet类似，不过加入了batchnorm层，加速收敛，以及更好的激活函数RELU。算法流程是先清洗数据，构造dataloader导入到算法当中去。然后构造分类模型，输出标签与类别标签数一致，然后写训练的过程以及相应的测试过程，最后在主函数里进行汇总，开始训练，同时测试。

下图是网络结构图。



使用的是搜集好的数据集，链接<https://pan.baidu.com/s/1voBHwdX2hH1p_jn4_6EhTg>

**四、实验结果**



最终得到97.425%的精确度。

**五、结论**

通过此次试验，我基本掌握了CNN网络的基本原理，体会到了CNN的强大。在之后的学习中，我将会进一步学习其他的网络模型，加强自己的实践能力，提升自我。

代码附录：

**Main.py文件**

import torch

import argparse

from model import A2NN

from dataset import Traffic\_Light

from utils import get\_train\_val\_names, check\_folder

from trainer import Trainer

from validator import Validator

from logger import Logger

from torch.utils.data import DataLoader

def main():

parse = argparse.ArgumentParser()

parse.add\_argument('--dataset\_path', type=str, default='TL\_Dataset/')

parse.add\_argument('--remove\_names', type=list, default=['README.txt',

'README.png',

'Testset'])

parse.add\_argument('--img\_resize\_shape', type=tuple, default=(32, 32))

parse.add\_argument('--batch\_size', type=int, default=1024)

parse.add\_argument('--lr', type=float, default=0.001)

parse.add\_argument('--num\_workers', type=int, default=4)

parse.add\_argument('--epochs', type=int, default=200)

parse.add\_argument('--val\_size', type=float, default=0.3)

parse.add\_argument('--save\_model', type=bool, default=True)

parse.add\_argument('--save\_path', type=str, default='logs/')

args = vars(parse.parse\_args())

check\_folder(args['save\_path'])

# pylint: disable=E1101

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

# pylint: disable=E1101

model = A2NN().to(device)

names = get\_train\_val\_names(args['dataset\_path'], args['remove\_names'])

train\_dataset = Traffic\_Light(names['train'], args['img\_resize\_shape'])

val\_dataset = Traffic\_Light(names['val'], args['img\_resize\_shape'])

train\_dataload = DataLoader(train\_dataset,

batch\_size=args['batch\_size'],

shuffle=True,

num\_workers=args['num\_workers'])

val\_dataload = DataLoader(val\_dataset,

batch\_size=args['batch\_size'],

shuffle=True,

num\_workers=args['num\_workers'])

loss\_logger = Logger(args['save\_path'])

logger\_dict = {'train\_losses': [],

'val\_losses': []}

for epoch in range(args['epochs']):

print('<Main> epoch{}'.format(epoch))

trainer = Trainer(model, train\_dataload, epoch, args['lr'], device)

train\_loss = trainer.train()

if args['save\_model']:

state = model.state\_dict()

torch.save(state, 'logs/nn\_state.t7')

validator = Validator(model, val\_dataload, epoch,

device, args['batch\_size'])

val\_loss = validator.eval()

logger\_dict['train\_losses'].append(train\_loss)

logger\_dict['val\_losses'].append(val\_loss['val\_loss'])

loss\_logger.update(logger\_dict)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**model.py文件**

import torch.nn as nn

from torchsummary import summary

class A2NN(nn.Module):

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

super(A2NN, self).\_\_init\_\_()

self.main = nn.Sequential(

nn.Conv2d(3, 16, 3, 1, 1),

nn.BatchNorm2d(16),

nn.ReLU(inplace=True),

nn.Conv2d(16, 32, 3, 1, 1),

nn.MaxPool2d(2, 2),

nn.BatchNorm2d(32),

nn.ReLU(inplace=True),

nn.Conv2d(32, 32, 3, 1, 1),

nn.MaxPool2d(2, 2),

nn.BatchNorm2d(32),

nn.ReLU(inplace=True),

nn.Conv2d(32, 64, 3, 1, 1),

nn.MaxPool2d(2, 2),

nn.BatchNorm2d(64),

nn.ReLU(inplace=True),

)

self.linear = nn.Linear(4\*4\*64, 9)

def forward(self, inp):

x = self.main(inp)

x = x.view(x.shape[0], -1)

x = self.linear(x)

return x

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

nn = A2NN()

summary(nn, (3, 32, 32))

**dataset.py**

import torch

import cv2

import torch.utils.data as data

class\_light = {

'Red Circle': 0,

'Green Circle': 1,

'Red Left': 2,

'Green Left': 3,

'Red Up': 4,

'Green Up': 5,

'Red Right': 6,

'Green Right': 7,

'Red Negative': 8,

'Green Negative': 8

}

class Traffic\_Light(data.Dataset):

def \_\_init\_\_(self, dataset\_names, img\_resize\_shape):

super(Traffic\_Light, self).\_\_init\_\_()

self.dataset\_names = dataset\_names

self.img\_resize\_shape = img\_resize\_shape

def \_\_getitem\_\_(self, ind):

img = cv2.imread(self.dataset\_names[ind])

img = cv2.resize(img, self.img\_resize\_shape)

img = img.transpose(2, 0, 1)-127.5/127.5

for key in class\_light.keys():

if key in self.dataset\_names[ind]:

label = class\_light[key]

# pylint: disable=E1101,E1102

return torch.from\_numpy(img), torch.tensor(label)

# pylint: disable=E1101,E1102

def \_\_len\_\_(self):

return len(self.dataset\_names)

if \_\_name\_\_ == '\_\_main\_\_':

from torch.utils.data import DataLoader

from glob import glob

import os

path = 'TL\_Dataset/Green Up/'

names = glob(os.path.join(path, '\*.png'))

dataset = Traffic\_Light(names, (32, 32))

dataload = DataLoader(dataset, batch\_size=1)

for ind, (inp, label) in enumerate(dataload):

print("{}-inp\_size:{}-label\_size:{}".format(ind, inp.numpy().shape,

label.numpy().shape))

**util.py**

import os

from glob import glob

def get\_train\_val\_names(dataset\_path, remove\_names, radio=0.3):

train\_names = []

val\_names = []

dataset\_paths = os.listdir(dataset\_path)

for n in remove\_names:

dataset\_paths.remove(n)

for path in dataset\_paths:

sub\_dataset\_path = os.path.join(dataset\_path, path)

sub\_dataset\_names = glob(os.path.join(sub\_dataset\_path, '\*.png'))

sub\_dataset\_len = len(sub\_dataset\_names)

val\_names.extend(sub\_dataset\_names[:int(radio\*sub\_dataset\_len)])

train\_names.extend(sub\_dataset\_names[int(radio\*sub\_dataset\_len):])

return {'train': train\_names, 'val': val\_names}

def check\_folder(path):

if not os.path.exists(path):

os.mkdir(path)

**trainer.py**

import torch.nn as nn

from torch.optim import Adam

class Trainer:

def \_\_init\_\_(self, model, dataload, epoch, lr, device):

self.model = model

self.dataload = dataload

self.epoch = epoch

self.lr = lr

self.device = device

self.optimizer = Adam(self.model.parameters(), lr=self.lr)

self.criterion = nn.CrossEntropyLoss().to(self.device)

def \_\_epoch(self, epoch):

self.model.train()

loss\_sum = 0

for ind, (inp, label) in enumerate(self.dataload):

inp = inp.float().to(self.device)

label = label.long().to(self.device)

self.optimizer.zero\_grad()

out = self.model.forward(inp)

loss = self.criterion(out, label)

loss.backward()

loss\_sum += loss.item()

self.optimizer.step()

print('epoch{}\_step{}\_train\_loss\_: {}'.format(epoch,

ind,

loss.item()))

return loss\_sum/(ind+1)

def train(self):

train\_loss = self.\_\_epoch(self.epoch)

return train\_loss

**validator.py**

import torch.nn as nn

class Validator:

def \_\_init\_\_(self, model, dataload, epoch, device, batch\_size):

self.model = model

self.dataload = dataload

self.epoch = epoch

self.device = device

self.batch\_size = batch\_size

self.criterion = nn.CrossEntropyLoss().to(self.device)

def \_\_epoch(self, epoch):

self.model.eval()

loss\_sum = 0

for ind, (inp, label) in enumerate(self.dataload):

inp = inp.float().to(self.device)

label = label.long().to(self.device)

out = self.model.forward(inp)

loss = self.criterion(out, label)

loss\_sum += loss.item()

return {'val\_loss': loss\_sum/(ind+1)}

def eval(self):

val\_loss = self.\_\_epoch(self.epoch)

return val\_loss

**logger.py**

import matplotlib.pyplot as plt

import os

class Logger:

def \_\_init\_\_(self, save\_path):

self.save\_path = save\_path

def update(self, Kwarg):

self.\_\_plot(Kwarg)

def \_\_plot(self, Kwarg):

save\_img\_path = os.path.join(self.save\_path, 'learning\_curve.png')

plt.clf()

plt.plot(Kwarg['train\_losses'], label='Train', color='g')

plt.plot(Kwarg['val\_losses'], label='Val', color='b')

plt.xlabel('epoch')

plt.ylabel('loss')

plt.legend()

plt.title('learning\_curve')

plt.savefig(save\_img\_path)