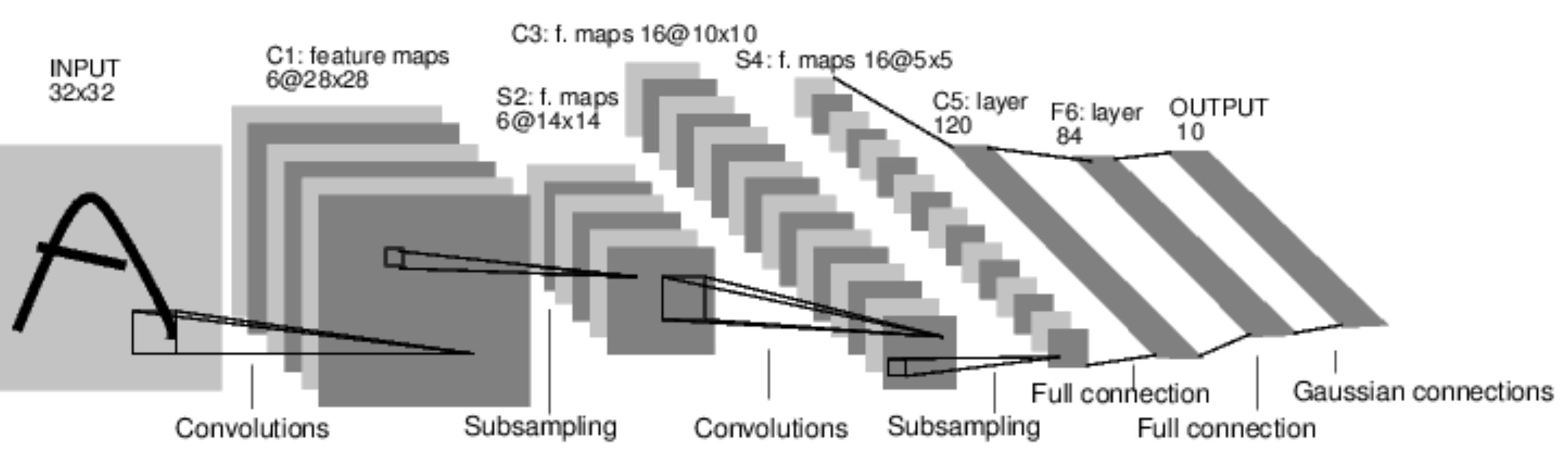
模型定义

网络结构如下：



模型结构代码如下：

```
import torch
import torch.nn as nn
import torch.nn.functional as F
class MinistModel(nn.Module):
    def __init__(self):
        super(MinistModel,self).__init__()
        self.conv1=nn.Conv2d(1,6,5)
        self.conv2=nn.Conv2d(6,16,5)
        self.fc1=nn.Linear(16*4*4,120)
        self.fc2=nn.Linear(120,84)
        self.fc3=nn.Linear(84,10)
    def num_flat_features(self,x):#在这里就是16*5*5
        size=x.shape[1:]#except batch size
        num_features=1
        for s in size:
            num_features*=s
        return num_features
    def forward(self, x):
        x=F.max_pool2d(F.relu(self.conv1(x)),(2,2))
        x=F.max_pool2d(F.relu(self.conv2(x)),(2,2))
        x=x.view(-1,self.num_flat_features(x))#摊平操作
        x=F.relu(self.fc1(x))
        x=F.relu(self.fc2(x))
        return self.fc3(x)
```

自定义数据进行框架测试

```
from model import *
import torch.optim as optim
net=MinistModel()
print(net)

# print(list(net.parameters()).__len__())

input=torch.randn(1,1,32,32)#(bs,c,w,h)
target=torch.randn(1,10)
print(target.size())

# input.requires_grad=True
out=net(input)
print(out)

# net.zero_grad()
# out.backward(torch.randn(1,10))
# print(net.conv1.bias.grad)

loss_func=nn.MSELoss()
loss=loss_func(out,target)
print(loss)

net.zero_grad()#这里是net，而不是loss。
loss.backward()

print(net.conv1.bias.grad)

#######################implement SGD##############

'''
weight=weight-lr*grad
'''
lr=0.001
for f in net.parameters():
    print(type(f.detach()))
    f.detach().sub_(lr*f.grad.detach())#0.3版本的pytorch，注意这里要取到data，以得到tensor数据；
    # 在0.4版本中，则要用.detach(),这个操作是不需要进行反向求导的，因此要detach
'''
 1).data返回一个新的requires_grad=False的Tensor! 然而新的这个Tensor与以前那个Tensor是共享内存的. 所以不安全
    y = x.data # x需要进行autograd
    # y和x是共享内存的,但是这里y已经不需要grad了，所以会导致本来需要计算梯度的x也没有梯度可以计算.从而x不会得到更新!
 2)推荐用x.detach()，这个仍旧是共享内存的，也是使得y的requires_grad为False，但是,如果x需要求导，仍旧是可以自动求导的!
'''

'''
or use pre-defined optimizers
# optimizer=optim.SGD(net.parameters(),lr)
# optimizer.zero_grad()#重要，否则会累加起来
# loss.backward()
# optimizer.step()
'''
```

在minist数据集上训练及测试

```
import torch
import os
import numpy as  np
import torchvision
import torchvision.transforms as transforms

import  torch.optim as optim
from model import *

#在GPU上训练
device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("trained on ",device)

#每个通道都要归一化
trainsform=transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.1307,), (0.3081,))])

trainset=torchvision.datasets.MNIST(root='./',train=True,download=True,transform=trainsform)
trainloader=torch.utils.data.DataLoader(trainset,batch_size=4,shuffle=True,num_workers=2)
testset=torchvision.datasets.MNIST(root='./',train=False,download=True,transform=trainsform)
testloader=torch.utils.data.DataLoader(testset,batch_size=4,shuffle=False,num_workers=0)

classes=('0', '1', '2', '3','4', '5', '6', '7', '8', '9')

net=MinistModel()
net=net.to(device)

optimizer=optim.SGD(net.parameters(),lr=0.003,momentum=0.9)
criterion=nn.CrossEntropyLoss()

def test():
    results=[]
    labels=[]

    # for evry category
    class_correct = np.array(list(0. for i in range(10)))
    class_total = np.array(list(0. for i in range(10)))

    for i,data in enumerate(testloader):
        img,label=data
        labels.extend(label)
        output=net(img.to(device))

        _, predicted = torch.max(output, 1)
        # print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
        #                               for j in range(4)))
        results.extend(predicted.cpu())

    # 计算准确率、精度、召回
    results=np.array(results)
    labels=np.array(labels)
    acc=np.equal(results,labels).sum()/results.shape[0]
    TP=((labels+results)==2).sum()
    FP=((labels-results)==-1).sum()
    FN=((labels-results)==1).sum()
    TN=((labels+results)==0).sum()
    rec=TP/(FN+TP)
    precision=TP/(FP+TP)
    print("epoch %d: accuracy: %3f---precision: %.3f---recall: %.3f" % (epoch + 1, acc, precision, rec))

    #计算每个类的准确率
    c = (results == labels).squeeze()
    for i in range(labels.shape[0]):
        class_correct[labels[i]] += c[i].item()
        class_total[labels[i]] += 1
    for i in range(10):
        print('Accuracy of %5s : %2d %%' % (
            classes[i], 100 * class_correct[i] / class_total[i]))



#train the network
EPOCH=100
for epoch in range(EPOCH):
    running_loss=0
    for i,data in enumerate(trainloader):
        img,label=data
        output=net(img.to(device))
        loss=criterion(output,label.to(device))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        running_loss+=loss.cpu()
        if(i%2000==1999):
            print('[%d,%5d] loss: %.3f' % (epoch+1,i+1,running_loss/2000))
            running_loss=0
        pass
    if((epoch+1)%2==0):
        print("testing.........")
        test()
        torch.save(net.state_dict(),os.path.join("./pretrained","epoch-"+str(epoch+1)+".pkl"))

print('Finished Training')
```

实验效果：

```
testing.........
epoch 10: accuracy: 0.987000---precision: 0.992---recall: 0.989
Accuracy of     0 : 98 %
Accuracy of     1 : 99 %
Accuracy of     2 : 97 %
Accuracy of     3 : 99 %
Accuracy of     4 : 99 %
Accuracy of     5 : 99 %
Accuracy of     6 : 98 %
Accuracy of     7 : 99 %
Accuracy of     8 : 98 %
Accuracy of     9 : 97 %
```