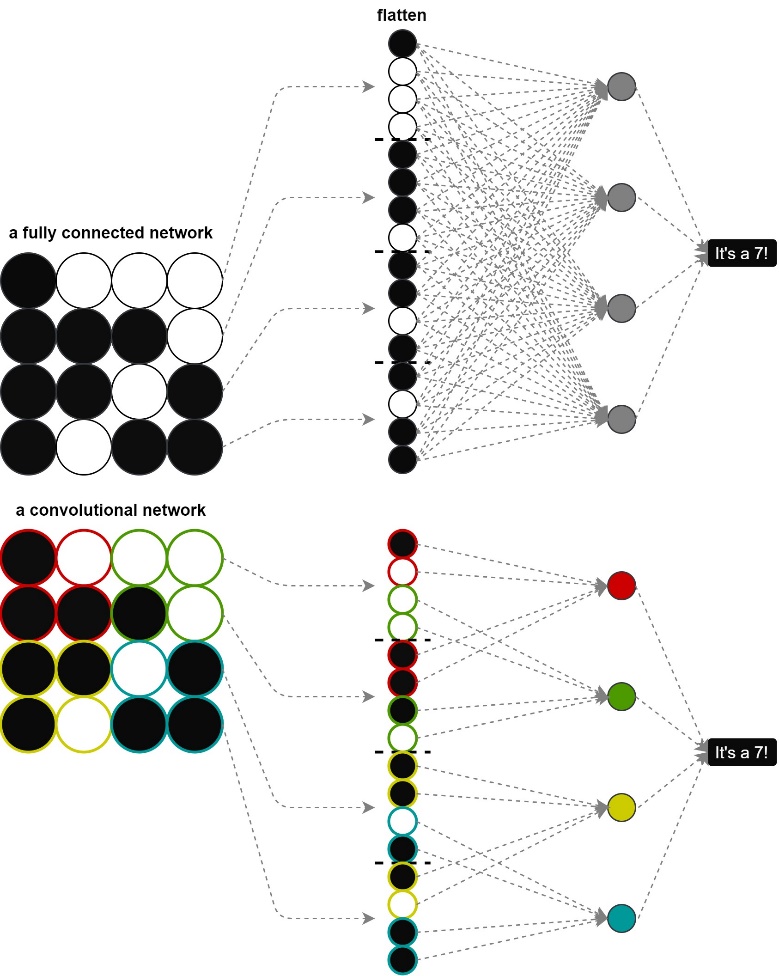
# 2.6.3 卷积神经网络

### 2.6.3.1 卷积神经网络（Convolutional neural network，CNN）—— 卷积原理与卷积神经网络

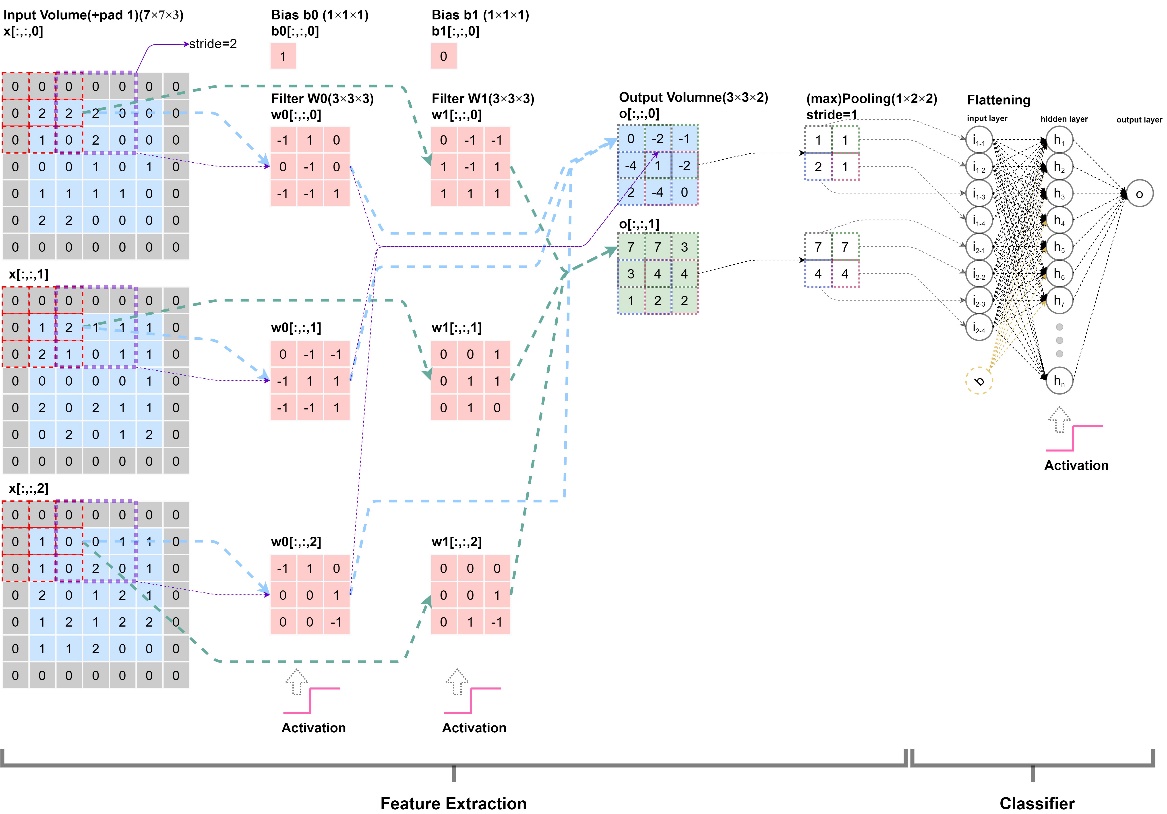
在阅读卷积神经网络（CNNs）之前，如果已经阅读“卷积”，“计算机视觉，特征提取和尺度空间”等部分章节，可以更好的理解CNNs。其中“尺度空间”的概念，通过降采样的不同空间分辨率影像（类似池化，pooling）和不同值的高斯核卷积（即卷积计算或称为互相关运算），来提取图像的概貌特征，这与CNNs的多层卷积网络如出一辙，只是CNNs除了卷积层，还可以自由加入其它的数据处理层，提升图像特征捕捉的几率。

一幅图像的意义来自于邻近的一组像素，而不是单个像素自身，因为单个像素并不包含关于整个图像的信息。例如下图[1]很好的说明了这两者的差异。一个全连接的神经网络（密集层，dense layer），一层的每个节点都连接到下一层的每个节点，并不能很好的反映节点之间的关系，也具有较大的计算量；但是卷积网络利用像素之间的空间结构减少了层之间的连接数量，显著提高了训练的速度，减少了模型的参数，更重要的是反映了图像特征像素之间的关系，即图像内容中各个对象是由自身与邻近像素关系决定的空间结构（即特征）所表述。

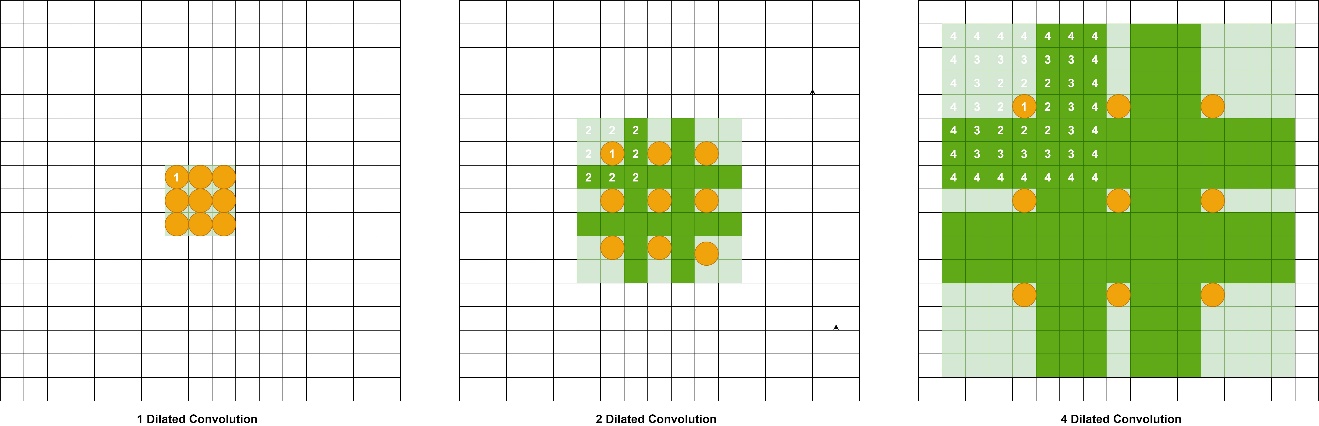


二维卷积层的卷积计算，就是表征图像的数组shape(c,h,w)（如果为灰度图像则只有一个颜色通道，如果是彩色图像（RGB）则有三个图像通道），每个通道(h,w)二维数组中每一像素，与卷积核（filter/kernal）的加权和计算。这个过程存在一些可变动的因素：一个是步幅（stride），卷积窗口（卷积核）从输入数组（二维图像数组）的最左上方开始，按从左到右，从上往下的顺序，依次在输入数组上滑动，每次滑动的行数和列数称为步幅。步幅越小代表捕捉不同内容对象的精度越高。下述图表[2]卷积核的滑动步幅为2，即行列均跨2步后计算；二是填充（padding），如果不设置填充，并且从左上角滑动，会有一部分卷积核对应空值（没有图像/数据）。同时要保持四周填充的行列数相同，通常使用奇数高宽的卷积核；三是，如果是对图像数据实现卷积，通常包括1个单通道，或3个多通道的情况。对于多通道的计算是可以配置不同的卷积核对应不同的通道，各自通道分别卷积后，计算和（累加）作为结果输出。同时可以增加并行的新的卷积计算，获取多个输出，例如下图的Filter W0，和Filter W1的()卷积核，W0和W1各自包含3个卷积核对应3个通道输入，并各自输出。

卷积运算，步幅、填充配置，及多通道卷积都没有改变图像的空间尺寸（空间分辨率），即尺度空间概念下降采样的表述（可以反映不同对象的尺度大小，或理解为只有在不同的尺度下才可以捕捉到对象的特征）。池化层（pooling）正是降采样在卷积神经网络中的表述，可以降低输入的空间维数，保留输入的深度。在图像卷积过程中，识别出比实际像素信息更多的概念意义，识别保留输入的关键信息，丢弃冗余部分。池化层不仅捕捉尺度空间下的对象特征，同时可以减少训练所需时间，减小模型的参数数量，降低模型复杂度，更好的泛化等。池化层可以用nn.MaxPool2d，取最大值；nn.AvgPool2d，取均值等方法。



卷积层和池化层都有一个dilation（扩张率/膨胀系数）参数，通过dilation配置，可以调整卷积核与图像的作用域，即感受野（receptive filed）。下图[3]中，当的卷积核dilation=1时，没有膨胀效应；当dilation=2时，感受野扩展至；当dilation=24时，感受野扩展至。可以确定当dilation线性增加时，感受野呈指数增加。



建立图表中的输入数据t\_input，张量形状为(batchsize, nChannels, Height, Width)=(1,3,7,7)，即只有一幅图像，通道数为3，高度为7，宽度为7。通常查看时，只要确定最后一维内的数据为图像每一行的像素值（由上至下）。PyTorch提供的nn.Conv2d卷积方法不需自定义卷积核，其参数为torch.nn.Conv2d(in\_channels: int, out\_channels: int, kernel\_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding\_mode: str = 'zeros')。如果需要自定义卷积核，则调用torch.nn.functional.conv2d方法，其中weight参数即为卷积核，其输入参数为torch.nn.functional.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1) → Tensor。

import torch  
import torch.nn as nn  
  
# batchsize, nChannels, Height, Width  
t\_input=torch.tensor([[[[0,0,0,0,0,0,0],[0,2,2,2,0,0,0],[0,1,0,2,0,0,0],[0,0,0,1,0,1,0],[0,1,1,1,1,0,0],[0,2,2,0,0,0,0],[0,0,0,0,0,0,0]],  
 [[0,0,0,0,0,0,0],[0,1,2,1,1,1,0],[0,2,1,0,1,1,0],[0,0,0,0,0,1,0],[0,2,0,2,1,1,0],[0,0,2,0,1,2,0],[0,0,0,0,0,0,0]],  
 [[0,0,0,0,0,0,0],[0,1,0,0,1,1,0],[0,1,0,2,0,1,0],[0,2,0,1,2,1,0],[0,1,2,1,2,2,0],[0,1,1,2,0,0,0],[0,0,0,0,0,0,0]],  
 ]],dtype=torch.float)  
print("t\_input.shape={}".format(t\_input.shape))  
conv\_2d\_3c=nn.Conv2d(in\_channels=3, out\_channels=2, kernel\_size=3,stride=2,padding=0,bias=1)  
conv\_2d\_3c(t\_input)

t\_input.shape=torch.Size([1, 3, 7, 7])  
  
  
  
  
  
tensor([[[[ 0.0716, 0.4418, 0.2253],  
 [ 0.5419, 1.3328, 0.6916],  
 [-0.5093, -0.2429, 0.3159]],  
  
 [[-0.5181, 0.4435, -0.2968],  
 [-0.5857, -0.9900, -0.4077],  
 [-0.4969, -0.1312, -0.1665]]]], grad\_fn=<ThnnConv2DBackward0>)

import torch.nn.functional as F  
w\_0=torch.tensor([[[[-1, 1, 0],  
 [ 0,-1, 0],  
 [-1,-1, 1]],  
  
 [[0,-1,-1],  
 [-1,1,1],  
 [-1,-1,1]],  
  
 [[-1,1,0],  
 [0,0,1],  
 [0,0,-1]]]],dtype=torch.float)  
  
w\_1=torch.tensor([[[[0, -1, -1],  
 [ 1,-1, 1],  
 [1,1, 1]],  
  
 [[0,0,1],  
 [0,1,1],  
 [0,1,0]],  
  
 [[0,0,0],  
 [0,0,1],  
 [0,1,-1]]]],dtype=torch.float)  
  
b\_0=torch.tensor([1])  
b\_1=torch.tensor([0])  
output\_0=F.conv2d(input=t\_input,weight=w\_0,stride=2,padding=0,bias=b\_0)  
output\_1=F.conv2d(input=t\_input,weight=w\_1,stride=2,padding=0,bias=b\_1)  
print(output\_0)  
print(output\_1)

tensor([[[[ 0., -2., -1.],  
 [-4., 1., -2.],  
 [ 2., -4., 0.]]]])  
tensor([[[[7., 7., 3.],  
 [3., 4., 4.],  
 [1., 2., 2.]]]])

最大池化，即设定池化（卷积核）大小，返回覆盖范围内最大值，其参数为torch.nn.MaxPool2d(kernel\_size: Union[T, Tuple[T, ...]], stride: Optional[Union[T, Tuple[T, ...]]] = None, padding: Union[T, Tuple[T, ...]] = 0, dilation: Union[T, Tuple[T, ...]] = 1, return\_indices: bool = False, ceil\_mode: bool = False)。

pooling\_input=torch.tensor([[[0,-2,-1],[-4,1,-2],[2,-4,0]]],dtype=torch.float)  
print("输入数据形状={}".format(pooling\_input.shape))  
  
maxPool\_2d=nn.MaxPool2d(kernel\_size=2,stride=1)  
maxPool\_2d(pooling\_input)

输入数据形状=torch.Size([1, 3, 3])  
  
  
  
  
  
tensor([[[1., 1.],  
 [2., 1.]]])

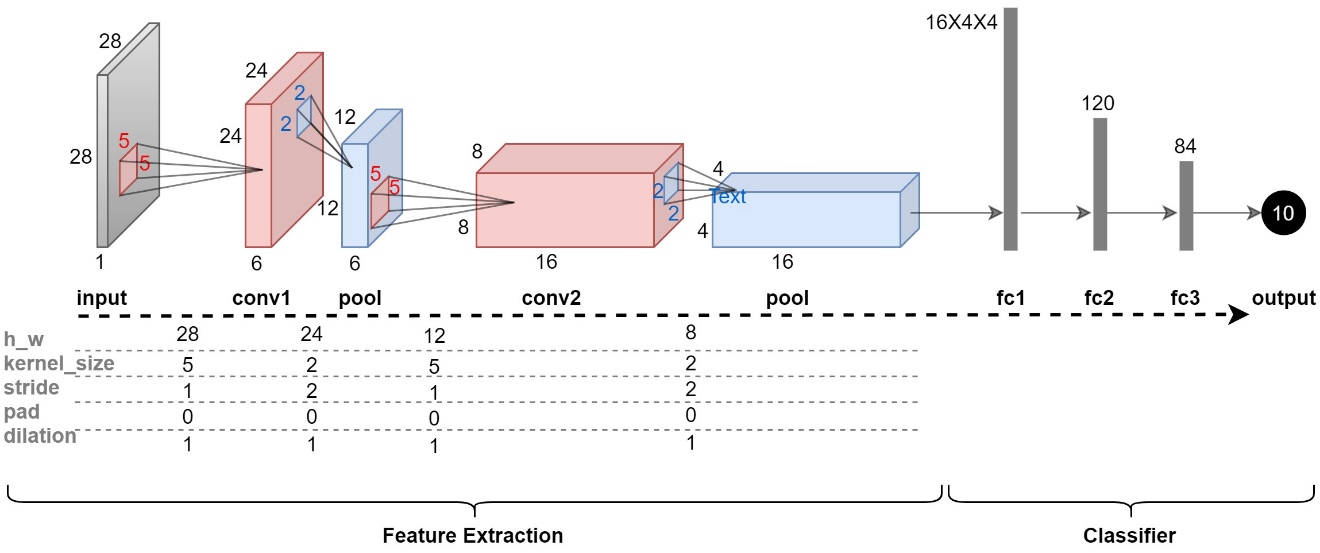
### 2.6.3.2 卷积\_特征提取器—>\_分类器，可视化卷积层/卷积核，及tensorboard

#### 1）卷积层，池化层输出尺寸（形状）计算，及根据输入输出尺寸反推填充pad

构建卷积神经网络，很重要的一步是确定图像经过一次或多次卷积后，输出的尺寸，用于分类器（线性模型，全连接层）的输入。在PyTorch的[torch.nn.Conv2d](https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html)①类方法说明中都会给出卷积输出的尺寸计算公式，但是手工的计算方式容易出错，并且耗时耗力，因此通常将其编写为代码程序。池化层的输出尺寸计算实际上与卷积的输出相同，但是为了区分，仍然在conv2d\_output\_size\_calculation类中，增加了pooling\_output\_shape方法（直接调用conv2d\_output\_shape）。卷积的方式除了给定h\_w图像高宽、kernel\_size卷积核（过滤器，filter）、stride步幅，pad填充，及dilation膨胀系数，卷积核的初始位置可以分为：以卷积核左上角对位图像左上角第一个像素值开始由左–>右，由上–>下卷积计算，即conv2d\_output\_shape函数； 以卷积核右下角对位图像左上角第一个像素值开始由左–>右，由上–>下卷积计算，即convtransp2d\_output\_shape函数。

同时，也可以根据卷积的输入和输出的尺寸，反推填充的大小，对应卷积核初始位置的不同分别为conv2d\_get\_padding和convtransp2d\_get\_padding函数。

对于卷积神经网络“net\_fashionMNIST”的网络结构如下：



输入图像的大小为，经过卷积层conv1–>池化层pool–>卷积层conv2–>池化层pool后，图像尺寸的大小变为，该值（卷积层的输出值）由自定义conv2d\_output\_size\_calculation计算。注意两次池化，其参数值相同，因此卷积神经网络结构定义时，可以仅定义一个池化方法，用于生成不同位置的池化层。计算应用卷积提取特征部分的图像输出大小后，因为conv2+pool之后，输出的out\_channels输出通道数配置为16，因此到分类器部分（线性函数/全连接层/展平层）的输入大小为。

class conv2d\_output\_size\_calculation:  
 '''  
 class - PyTorch 卷积层，池化层输出尺寸(shape)计算，及根据输入，输出尺寸(shape)反推pad填充大小  
   
 @author:sytelus Shital Shah  
 Updated on Tue Jan 12 19:17:22 2021 @author: Richie Bao  
 '''   
   
 def num2tuple(self,num):  
 '''  
 function - 如果num=2，则返回(2,2)；如果num=(2,2)，则返回(2,2).  
 '''  
 return num if isinstance(num, tuple) else (num, num)  
   
 def conv2d\_output\_shape(self,h\_w, kernel\_size=1, stride=1, pad=0, dilation=1):   
 '''  
 funciton - 计算PyTorch的nn.Conv2d卷积方法的输出尺寸。以卷积核左上角对位图像左上角第一个像素值开始由左-->右，由上-->下卷积计算  
 '''  
 import math  
  
 h\_w, kernel\_size, stride, pad, dilation = self.num2tuple(h\_w), \  
 self.num2tuple(kernel\_size), self.num2tuple(stride), self.num2tuple(pad), self.num2tuple(dilation)  
 pad = self.num2tuple(pad[0]), self.num2tuple(pad[1])  
   
 h = math.floor((h\_w[0] + sum(pad[0]) - dilation[0]\*(kernel\_size[0]-1) - 1) / stride[0] + 1)  
 w = math.floor((h\_w[1] + sum(pad[1]) - dilation[1]\*(kernel\_size[1]-1) - 1) / stride[1] + 1)  
   
 return h, w  
   
 def convtransp2d\_output\_shape(self,h\_w, kernel\_size=1, stride=1, pad=0, dilation=1, out\_pad=0):  
 '''  
 function - 以卷积核右下角对位图像左上角第一个像素值开始由左-->右，由上-->下卷积计算  
 '''  
 h\_w, kernel\_size, stride, pad, dilation, out\_pad = self.num2tuple(h\_w), \  
 self.num2tuple(kernel\_size), self.num2tuple(stride), self.num2tuple(pad), self.num2tuple(dilation), self.num2tuple(out\_pad)  
 pad = self.num2tuple(pad[0]), self.num2tuple(pad[1])  
   
 h = (h\_w[0] - 1)\*stride[0] - sum(pad[0]) + dilation[0]\*(kernel\_size[0]-1) + out\_pad[0] + 1  
 w = (h\_w[1] - 1)\*stride[1] - sum(pad[1]) + dilation[1]\*(kernel\_size[1]-1) + out\_pad[1] + 1  
   
 return h, w  
   
 def conv2d\_get\_padding(self,h\_w\_in, h\_w\_out, kernel\_size=1, stride=1, dilation=1):   
 '''  
 function - conv2d\_output\_shape 方法的逆，求填充pad  
 '''  
 import math  
  
 h\_w\_in, h\_w\_out, kernel\_size, stride, dilation = self.num2tuple(h\_w\_in), self.num2tuple(h\_w\_out), \  
 self.num2tuple(kernel\_size), self.num2tuple(stride), self.num2tuple(dilation)  
   
 p\_h = ((h\_w\_out[0] - 1)\*stride[0] - h\_w\_in[0] + dilation[0]\*(kernel\_size[0]-1) + 1)  
 p\_w = ((h\_w\_out[1] - 1)\*stride[1] - h\_w\_in[1] + dilation[1]\*(kernel\_size[1]-1) + 1)  
   
 return (math.floor(p\_h/2), math.ceil(p\_h/2)), (math.floor(p\_w/2), math.ceil(p\_w/2)) # ((pad\_up, pad\_bottom)， (pad\_left, pad\_right))  
   
 def convtransp2d\_get\_padding(self,h\_w\_in, h\_w\_out, kernel\_size=1, stride=1, dilation=1, out\_pad=0):   
 '''  
 function - convtransp2d\_output\_shape 方法的逆，求填充pad  
 '''  
 import math  
  
 h\_w\_in, h\_w\_out, kernel\_size, stride, dilation, out\_pad = self.num2tuple(h\_w\_in), self.num2tuple(h\_w\_out), \  
 self.num2tuple(kernel\_size), self.num2tuple(stride), self.num2tuple(dilation), self.num2tuple(out\_pad)  
   
 p\_h = -(h\_w\_out[0] - 1 - out\_pad[0] - dilation[0]\*(kernel\_size[0]-1) - (h\_w\_in[0] - 1)\*stride[0]) / 2  
 p\_w = -(h\_w\_out[1] - 1 - out\_pad[1] - dilation[1]\*(kernel\_size[1]-1) - (h\_w\_in[1] - 1)\*stride[1]) / 2  
   
 return (math.floor(p\_h/2), math.ceil(p\_h/2)), (math.floor(p\_w/2), math.ceil(p\_w/2))  
   
 def pooling\_output\_shape(self,h\_w, kernel\_size=1, stride=1, pad=0, dilation=1):  
 '''  
 function - pooling池化层输出尺寸，同conv2d\_output\_shape  
 '''  
 return self.conv2d\_output\_shape(h\_w, kernel\_size=kernel\_size, stride=stride, pad=pad, dilation=dilation)

conv2dSize\_cal=conv2d\_output\_size\_calculation()  
size\_conv1=conv2dSize\_cal.conv2d\_output\_shape(28, kernel\_size=5, stride=1, pad=0, dilation=1)  
size\_pool1=conv2dSize\_cal.pooling\_output\_shape(24, kernel\_size=2, stride=2, pad=0, dilation=1)  
size\_conv2=conv2dSize\_cal.conv2d\_output\_shape(12, kernel\_size=5, stride=1, pad=0, dilation=1)  
size\_pool2=conv2dSize\_cal.pooling\_output\_shape(8, kernel\_size=2, stride=2, pad=0, dilation=1)  
  
print("conv1\_size={}\nsize\_pool1={}\nsize\_conv2={}\nsize\_pool2={}".format(size\_conv1,size\_pool1,size\_conv2,size\_pool2))

conv1\_size=(24, 24)  
size\_pool1=(12, 12)  
size\_conv2=(8, 8)  
size\_pool2=(4, 4)

在设计卷积层，或者包含很多卷积层时，上述的方法可以进一步改进，定义一个函数可以一次性计算所有的卷积层。在输入参数设置时，使用了列表和元组的形式，固定输入参数input，conv和pool，值的参数依次为[kernel\_size, stride, pad, dilation]。

convs\_params=[  
 ('input',(28,28)),  
 ('conv',[5,1,0,1]), # kernel\_size, stride, pad, dilation  
 ('pool',[2,2,0,1]),  
 ('conv',[5,1,0,1]),  
 ('pool',[2,2,0,1]),   
 ]  
  
def conv2d\_outputSize\_A\_oneTime(convs\_params):  
 '''  
 fucntion - 一次性计算卷积输出尺寸  
 '''  
 conv2dSize\_cal=conv2d\_output\_size\_calculation()  
 for v in convs\_params:  
 if v[0]=='input':  
 h\_w=v[1]  
 elif v[0]=='conv' or v[0]=='pool':   
 kernel\_size, stride, pad, dilation=v[1]  
 h\_w, kernel\_size, stride, pad, dilation=conv2dSize\_cal.num2tuple(h\_w),conv2dSize\_cal.num2tuple(kernel\_size), conv2dSize\_cal.num2tuple(stride),conv2dSize\_cal.num2tuple(pad),conv2dSize\_cal.num2tuple(dilation)  
 h\_w=conv2dSize\_cal.conv2d\_output\_shape(h\_w, kernel\_size, stride, pad, dilation)   
 return h\_w  
   
output\_h\_w=conv2d\_outputSize\_A\_oneTime(convs\_params)   
print("卷积层输出尺寸={}".format(output\_h\_w))

卷积层输出尺寸=(4, 4)

#### 2）基于fashionMNIST数据集构建简单的卷积神经网络识别，及tensorboard可视化

此处构建上图给出的神经网络结构，input–>conv1(relu)–>pool–>conv2(relu)–>pool—>fc1(relu)–>fc2(relu)–>fc3–>output，同时应用tensorboard可以写入并自动的根据写入的内容图示卷积神经网络结构、损失曲线、预测结果、样本信息、及自定义的内容等。详细内容可以查看[torch.utils.tensorboard](https://pytorch.org/docs/stable/tensorboard.html)②。

参考：[Visualizing Models, Data, and Training with TensorBoard](https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html?highlight=fashion%20mnist)[4]

# 01-下载/读取fashinMNIST数据，及构建训练、测试可迭代对象。（如果已经下载，则直接读取）  
  
def load\_fashionMNIST(root,batchsize=4,num\_workers=2,resize=None,n\_mean=0.5,n\_std=0.5):   
 '''  
 function - 下载读取fashionMNIST数据集，并建立训练、测试可迭代数据集  
 '''  
 import torchvision  
 import torchvision.transforms as transforms  
  
 trans= [transforms.ToTensor(), # 转换PIL图像或numpy.ndarray为tensor张量  
 transforms.Normalize((0.5,), (0.5,))] # torchvision.transforms.Normalize(mean, std, inplace=False)，用均值和标准差，标准化张量图像   
 if resize:  
 trans.append(transforms.Resize(size=resize))   
 transform=transforms.Compose(trans)  
 mnist\_train=torchvision.datasets.FashionMNIST(root=root, train=True, download=True, transform=transform)   
 mnist\_test=torchvision.datasets.FashionMNIST(root=root, train=False, download=True, transform=transform)  
  
 # DataLoade-读取小批量  
 import torch.utils.data as data\_utils  
 batch\_size=batchsize  
 num\_workers=num\_workers  
 trainloader=data\_utils.DataLoader(mnist\_train, batch\_size=batch\_size, shuffle=True, num\_workers=num\_workers)  
 testloader=data\_utils.DataLoader(mnist\_test, batch\_size=batch\_size, shuffle=False, num\_workers=num\_workers)  
   
 return trainloader,testloader  
  
trainloader,testloader=load\_fashionMNIST(root='./dataset/FashionMNIST\_norm')  
classes=('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat','Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot')

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ./dataset/FashionMNIST\_norm\FashionMNIST\raw\train-images-idx3-ubyte.gz  
  
  
  
 0%| | 0/26421880 [00:00<?, ?it/s]  
  
  
Extracting ./dataset/FashionMNIST\_norm\FashionMNIST\raw\train-images-idx3-ubyte.gz to ./dataset/FashionMNIST\_norm\FashionMNIST\raw  
  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ./dataset/FashionMNIST\_norm\FashionMNIST\raw\train-labels-idx1-ubyte.gz  
  
  
  
 0%| | 0/29515 [00:00<?, ?it/s]  
  
  
Extracting ./dataset/FashionMNIST\_norm\FashionMNIST\raw\train-labels-idx1-ubyte.gz to ./dataset/FashionMNIST\_norm\FashionMNIST\raw  
  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ./dataset/FashionMNIST\_norm\FashionMNIST\raw\t10k-images-idx3-ubyte.gz  
  
  
  
 0%| | 0/4422102 [00:00<?, ?it/s]  
  
  
Extracting ./dataset/FashionMNIST\_norm\FashionMNIST\raw\t10k-images-idx3-ubyte.gz to ./dataset/FashionMNIST\_norm\FashionMNIST\raw  
  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz  
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ./dataset/FashionMNIST\_norm\FashionMNIST\raw\t10k-labels-idx1-ubyte.gz  
  
  
  
 0%| | 0/5148 [00:00<?, ?it/s]  
  
  
Extracting ./dataset/FashionMNIST\_norm\FashionMNIST\raw\t10k-labels-idx1-ubyte.gz to ./dataset/FashionMNIST\_norm\FashionMNIST\raw

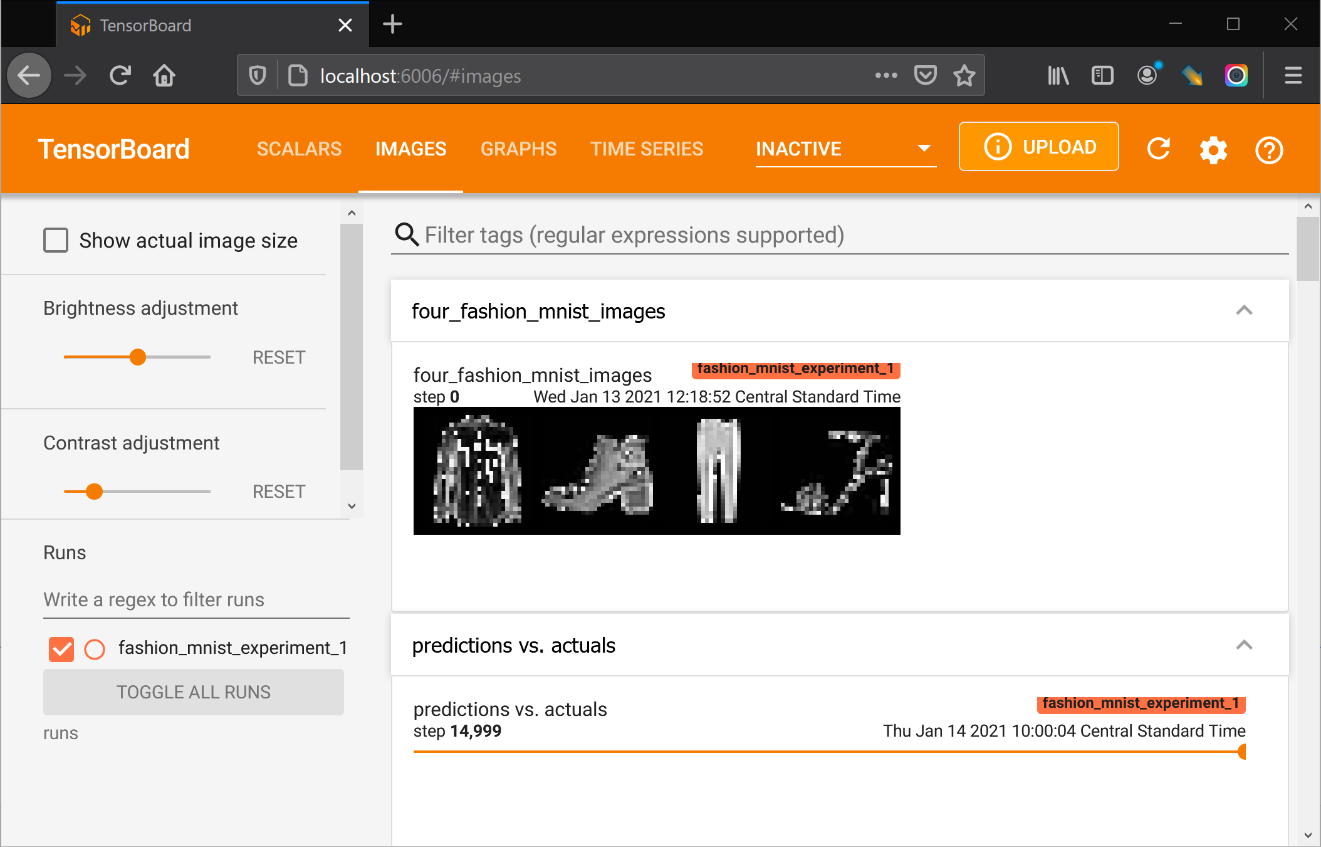
# 02-定义网络结构，特征提取层+分类器  
  
import torch.nn as nn  
import torch.nn.functional as F  
  
class net\_fashionMNIST(nn.Module):  
 def \_\_init\_\_(self):  
 super(net\_fashionMNIST,self).\_\_init\_\_()  
 self.conv1=nn.Conv2d(1,6,5)  
 self.pool=nn.MaxPool2d(2,2)  
 self.conv2=nn.Conv2d(6,16,5)  
 self.fc1=nn.Linear(16\*4\*4,120) # torch.nn.Linear(in\_features: int, out\_features: int, bias: bool = True)  
 self.fc2=nn.Linear(120,84)  
 self.fc3=nn.Linear(84,10)  
   
 def forward(self,x):  
 x=self.pool(F.relu(self.conv1(x)))  
 x=self.pool(F.relu(self.conv2(x)))  
 x=x.view(-1,16\*4\*4)  
 x=F.relu(self.fc1(x))  
 x=F.relu(self.fc2(x))  
 x=self.fc3(x)  
 return x  
net\_fashionMNIST\_=net\_fashionMNIST()  
print(net\_fashionMNIST\_)  
  
# 03-定义损失函数核优化算法  
  
import torch.optim as optim  
criterion=nn.CrossEntropyLoss() # 定义损失函数  
optimizer=optim.SGD(net\_fashionMNIST\_.parameters(), lr=0.001, momentum=0.9) # 定义优化算法

net\_fashionMNIST(  
 (conv1): Conv2d(1, 6, kernel\_size=(5, 5), stride=(1, 1))  
 (pool): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (conv2): Conv2d(6, 16, kernel\_size=(5, 5), stride=(1, 1))  
 (fc1): Linear(in\_features=256, out\_features=120, bias=True)  
 (fc2): Linear(in\_features=120, out\_features=84, bias=True)  
 (fc3): Linear(in\_features=84, out\_features=10, bias=True)  
)

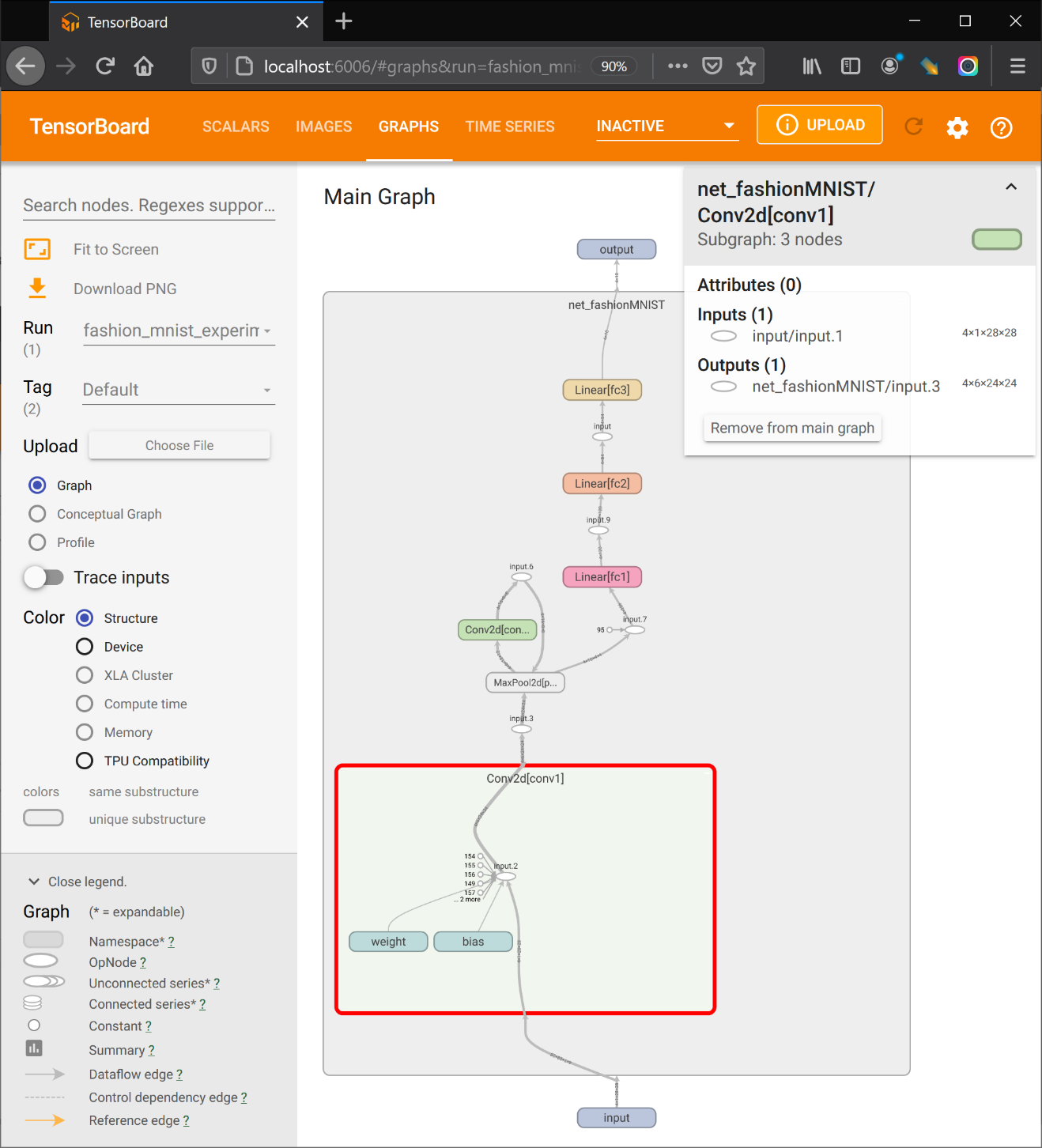
# 04-定义显示图像函数（一个batch批次）  
  
import torchvision  
# helper functions  
def matplotlib\_imshow(img, one\_channel=False):  
 import matplotlib.pyplot as plt  
 import numpy as np   
 if one\_channel:  
 img = img.mean(dim=0)  
 img = img / 2 + 0.5 # unnormalize  
 npimg = img.numpy()  
 if one\_channel:  
 plt.imshow(npimg, cmap="Greys")  
 else:  
 plt.imshow(np.transpose(npimg, (1, 2, 0)))  
  
# 05-调入初始化tensorboard.SummaryWriter（指定数据写入的保存路径），并写入训练图像与模型  
   
from torch.utils.tensorboard import SummaryWriter  
# default `log\_dir` is "runs" - we'll be more specific here  
writer=SummaryWriter(r'./runs/fashion\_mnist\_experiment\_1')  
  
# 提取图像（数量为batch\_size） get some random training images  
dataiter=iter(trainloader)  
images,labels=dataiter.next()  
# 建立图像格网 create grid of images  
img\_grid = torchvision.utils.make\_grid(images)  
# 显示图像 show images  
matplotlib\_imshow(img\_grid, one\_channel=True)  
  
# write to tensorboard  
writer.add\_image('four\_fashion\_mnist\_images', img\_grid)  
  
writer.add\_graph(net\_fashionMNIST\_, images)  
writer.close()



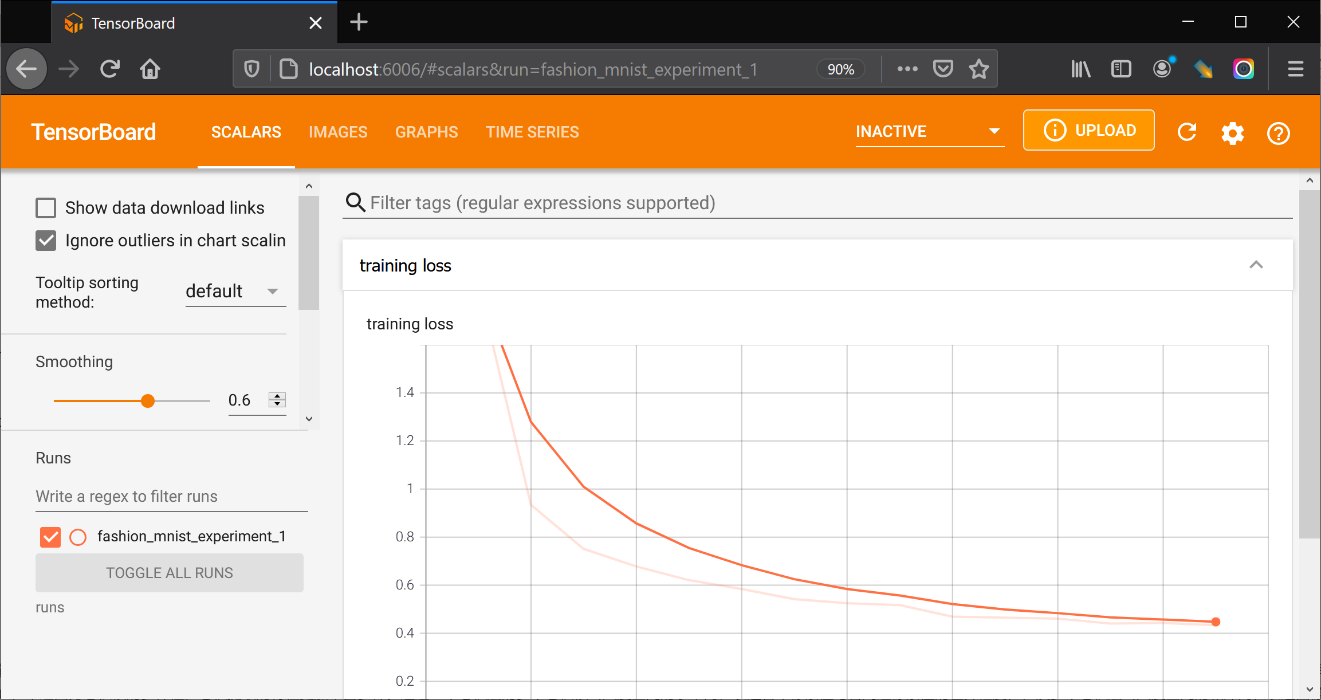
在终端，切换到tensorboard数据保存文件夹runs所在的目录，执行tensorboard --logdir=runs后，通常提示在浏览器中打开http://localhost:6006/地址，可以查看writer.add\_image('four\_fashion\_mnist\_images', img\_grid)写入的训练图像信息内容（其语法为：add\_image(tag, img\_tensor, global\_step=None, walltime=None, dataformats='CHW')），如下：



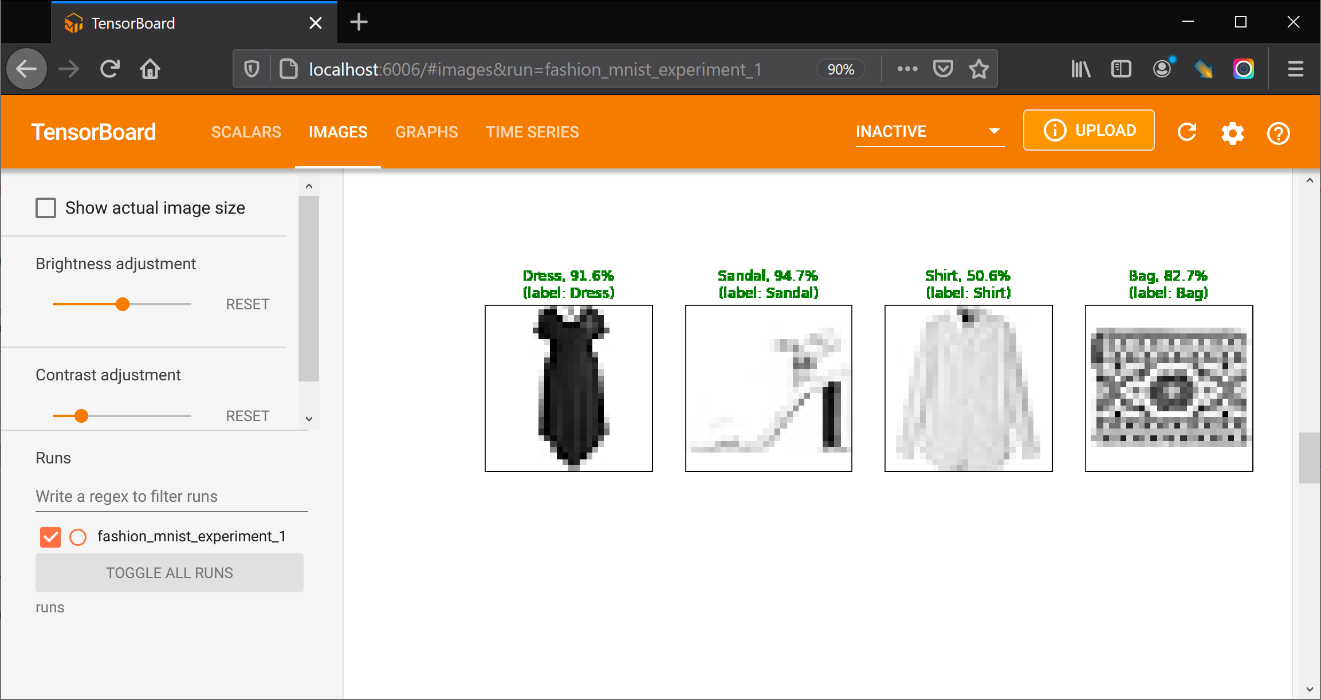
writer.add\_graph(net\_fashionMNIST\_, images)（其语法为：add\_graph(model, input\_to\_model=None, verbose=False)），写入的信息内容下，可以查看网络结构及运算的流程，这对模型的构建与调整有所帮助。



下述在训练过程中，riter.add\_scalar('training loss',running\_loss / 1000,epoch \* len(trainloader) + i)（其语法为：add\_scalar(tag, scalar\_value, global\_step=None, walltime=None)），写入损失数据图表，如下：



writer.add\_figure('predictions vs. actuals',plot\_classes\_preds(net\_fashionMNIST\_, inputs, labels),global\_step=epoch \* len(trainloader) + i)（其语法为：add\_figure(tag, figure, global\_step=None, close=True, walltime=None)），写入plot\_classes\_preds返回图表，包括图像，预测值及其概率，如下：



# 06-定义将图像应用训练的网络（模型）预测及其概率待写入tensorboard文件的函数  
  
def images\_to\_probs(net, images):  
 '''  
 function - 用训练的网络预测给定的一组图像，并计算相应的概率 Generates predictions and corresponding probabilities from a trainednetwork and a list of images  
 '''  
 output = net(images)  
 # convert output probabilities to predicted class  
 \_, preds\_tensor = torch.max(output, 1)  
 preds = np.squeeze(preds\_tensor.numpy())  
 return preds, [F.softmax(el, dim=0)[i].item() for i, el in zip(preds, output)]  
  
def plot\_classes\_preds(net, images, labels):   
 '''  
 function - 用训练的网络预测给定的一组图像，并计算概率后，显示图像、预测值以及概率，和实际的标签 Generates matplotlib Figure using a trained network, along with images and labels from a batch, that shows the network's top prediction along with its probability, alongside the actual label, coloring this information based on whether the prediction was correct or not.  
 Uses the "images\_to\_probs" function.  
 '''  
 import matplotlib.pyplot as plt  
  
 preds, probs = images\_to\_probs(net, images)  
 # plot the images in the batch, along with predicted and true labels  
 fig = plt.figure(figsize=(12, 48))  
 for idx in np.arange(4):  
 ax = fig.add\_subplot(1, 4, idx+1, xticks=[], yticks=[])  
 matplotlib\_imshow(images[idx], one\_channel=True)  
 ax.set\_title("{0}, {1:.1f}%\n(label: {2})".format(  
 classes[preds[idx]],  
 probs[idx] \* 100.0,  
 classes[labels[idx]]),  
 color=("green" if preds[idx]==labels[idx].item() else "red"))  
 return fig

# 07-训练模型，同时向tensorboard文件写入相关信息  
  
from tqdm.auto import tqdm  
import torch  
import numpy as np  
running\_loss = 0.0  
epochs=1  
for epoch in tqdm(range(epochs)): # loop over the dataset multiple times  
 for i, data in enumerate(trainloader, 0):  
 # get the inputs; data is a list of [inputs, labels]  
 inputs, labels = data  
  
 # zero the parameter gradients  
 optimizer.zero\_grad()  
  
 # forward + backward + optimize  
 outputs = net\_fashionMNIST\_(inputs)  
 loss = criterion(outputs, labels)  
 loss.backward()  
 optimizer.step()  
  
 running\_loss += loss.item()   
 if i % 1000==999: # every 1000 mini-batches...  
 # ...log the running loss  
 writer.add\_scalar('training loss',running\_loss / 1000,epoch \* len(trainloader) + i) # 写入损失数值  
  
 # ...log a Matplotlib Figure showing the model's predictions on a  
 # random mini-batch  
 writer.add\_figure('predictions vs. actuals',plot\_classes\_preds(net\_fashionMNIST\_, inputs, labels),global\_step=epoch \* len(trainloader) + i) # 写入自定义函数 plot\_classes\_preds 返回的图表，包括图像，预测值及其概率  
 loss\_temp=running\_loss  
 running\_loss=0.0  
   
 print("epoch={},running\_loss={}".format(epoch,loss\_temp))   
 loss\_temp=0  
   
print('Finished Training')

0%| | 0/1 [00:00<?, ?it/s]  
  
  
epoch=0,running\_loss=365.6714180938143  
Finished Training

#### 3）可视化卷积层/卷积核

通常一个卷积层的输出通道（output channel）有6，16，32，64，256，等不确定的输出个数及更多的输出个数，即一个卷积层包含输出通道数目的filter/kernal过滤器/卷积核；而卷积核通过，，等不确定尺寸（通常为奇数）或者更大尺寸来提取图像的特征，不同的卷积核提取的图像特征不同，或者表述为不同的卷积核关注不同的特征提取，这类似于图像的关键点描述子（位于不同的尺度空间下）。不过因为卷积核的多样性，卷积提取的特征更加丰富多样。通过一个卷积层的多个卷积核（尺度空间的水平向，表述各个像素值与各周边像素值的差异程度），及多个卷积层（尺度空间的纵深向，表述图像特征所在的（对应的）空间分辨率，例如遥感影像看清建筑轮廓的空间分辨率约为5~15m，看清行人的空间分辨率约为0.3~1m，而若要看清人的五官，空间分辨率率则约为0.01~0.05m，这与对象（特征）的尺寸有关）提取了大量的图像特征，将这些图像特征flatten展平，就构成了该图像的特征集合（feature maps）。

下述定义的conv\_retriever类用于取回卷积神经网络中所有卷积层及其权重值（卷积核），取回的卷积层可以直接输入图像数据计算该卷积。例如上述网络取回的卷积层有两个。函数visualize\_convFilter则可以打印显示卷积核。函数visualize\_convLayer则能打印显示指定数目的所有卷积图像结果。

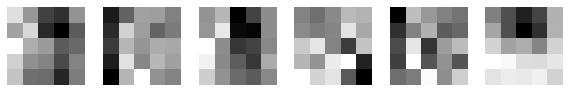
参考 [Visualizing Filters and Feature Maps in Convolutional Neural Networks using PyTorch](https://debuggercafe.com/visualizing-filters-and-feature-maps-in-convolutional-neural-networks-using-pytorch/)[5]

class conv\_retriever:  
 '''  
 class - 取回卷积神经网络中所有卷积层及其权重  
 '''  
 def \_\_init\_\_(self,net):  
 self.net=net  
 self.model\_weights=[] # we will save the conv layer weights in this list  
 self.conv\_layers=[] # we will save the 49 conv layers in this list  
 self.model\_children=list(net.children()) # get all the model children as list  
 self.counter=0  
   
 def retriever(self):  
 import torch.nn as nn  
 for i in range(len(self.model\_children)):  
 if type(self.model\_children[i])==nn.Conv2d:   
 self.counter+=1  
 self.model\_weights.append(self.model\_children[i].weight)  
 self.conv\_layers.append(self.model\_children[i])  
 elif type(self.model\_children[i])==nn.Sequential:  
 for j in range(len(self.model\_children[i])):  
 for child in self.model\_children[i][j].children():  
 if type(child)==nn.Conv2d:  
 self.counter+=1  
 self.model\_weights.append(child.weight)  
 self.conv\_layers.append(child)   
   
   
conv\_retriever\_=conv\_retriever(net\_fashionMNIST\_)  
conv\_retriever\_.retriever()  
# print(conv\_retriever\_.conv\_layers) #conv\_retriever\_.model\_weights  
for weight, conv in zip(conv\_retriever\_.model\_weights, conv\_retriever\_.conv\_layers):  
 print(f"CONV: {conv} ====> SHAPE: {weight.shape}")

CONV: Conv2d(1, 6, kernel\_size=(5, 5), stride=(1, 1)) ====> SHAPE: torch.Size([6, 1, 5, 5])  
CONV: Conv2d(6, 16, kernel\_size=(5, 5), stride=(1, 1)) ====> SHAPE: torch.Size([16, 6, 5, 5])

卷积核显示中，像素越黑，值越小，趋于0；而像素越亮，值越大，趋于255.0。因此越白的像素，在卷积过程中对应位置图像的像素的权重越大，对特征影响越重。

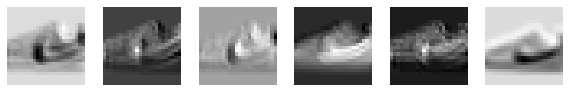
def visualize\_convFilter(conv\_layer,model\_weight,output\_name,figsize=(10,10)):   
 '''  
 function - 可视化卷积核 visualize the conv layer filters  
 '''  
 import matplotlib.pyplot as plt  
  
 plt.figure(figsize=figsize)  
 kernel\_size=conv\_layer.kernel\_size  
 for i,filter in enumerate(model\_weight):  
 plt.subplot(kernel\_size[0]+1,kernel\_size[1]+1,i+1)  
 plt.imshow(filter[0,:,:].detach(),cmap='gray')  
 plt.axis('off')  
 plt.savefig(r'./results/%s'%output\_name)  
 plt.show()   
   
visualize\_convFilter(conv\_retriever\_.conv\_layers[0],conv\_retriever\_.model\_weights[0],output\_name='fashion\_MNIST\_filter.png')



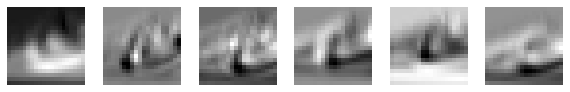
卷积（层）结果的显示，往往可以观察到不同的特征，例如明显的黑色区域轮廓为商标标识，对象的轮廓也能够通过代表不同颜色的值区分开来等。这为观察不同的卷积核提取了图像哪些特征，为相关研究或者网络调试提供参考。

def visualize\_convLayer(imgs\_batch,conv\_layers,model\_weights,num\_show=6,figsize=(10,10)):   
 '''  
 function - 可视化所有卷积层（卷积结果）  
 '''  
 import matplotlib.pyplot as plt  
  
 results=[conv\_layers[0](imgs\_batch)]  
 for i in range(1,len(conv\_layers)):  
 results.append(conv\_layers[i](results[-1]))  
 outputs=results  
   
 for num\_layer in range(len(outputs)):  
 plt.figure(figsize=figsize)  
 layer\_viz=outputs[num\_layer][0,:,:,:]  
 layer\_viz=layer\_viz.data  
 print("num\_layer:{},layer.size={}".format(num\_layer,layer\_viz.size()))  
   
 print()  
 kernel\_size=conv\_layers[num\_layer].kernel\_size  
 for i,filter in enumerate(layer\_viz):  
 if i==num\_show:  
 break  
   
 plt.subplot(kernel\_size[0]+1,kernel\_size[1]+1,i+1)  
 plt.imshow(filter,cmap='gray')  
 plt.axis('off')  
 print(f"Saving layer {num\_layer} feature maps...")  
 plt.savefig(f'./results/layer\_{num\_layer}.png')  
 plt.show()  
 # plt.close() # 如果只保存，不需要显示打印，则可以开启plt.close()，并注释掉plt.show()   
   
visualize\_convLayer(images,conv\_retriever\_.conv\_layers,conv\_retriever\_.model\_weights,num\_show=6)

num\_layer:0,layer.size=torch.Size([6, 24, 24])  
  
Saving layer 0 feature maps...



num\_layer:1,layer.size=torch.Size([16, 20, 20])  
  
Saving layer 1 feature maps...



### 2.6.3.3 [torchvision.models](https://pytorch.org/vision/0.8/models.html)③

torchvision.models库包含有解决不同任务的预定义模型（通过配置参数pretrained=True，可以下载已经训练模型的参数），包括：image classification（图像分类）、pixelwise sementic segmentation（像素语义分割）、object detection（对象检测）、instance segmentation（实例分割）、person keypoint detection （人关键点检测）和video classification（视频分类）等。目前包括的模型如下：

| 用途 | 模型/网络 |
| --- | --- |
| Classification | AlexNet, VGG,ResNet, SqueezeNet, DenseNet, Inception v3, GoogLeNet,ShuffleNet v2,MobileNet v2,ResNeXt, Wide ResNet, MNASNet |
| Semantic Segmentation | FCN ResNet50, ResNet101; DeepLabV3 ResNet50, ResNet101 |
| Object Detection, Instance Segmentation and Person Keypoint Detection | Faster R-CNN ResNet-50 FPN ,Mask R-CNN ResNet-50 FPN |
| Video classification | ResNet 3D, ResNet Mixed Convolution,ResNet (2+1)D |

#### 1） 自定义VGG网络

VGG作者[6]研究了在大规模图像识别设置中，卷积深度网络对精度的影响。其贡献是使用非常小的卷积核()滤波器的架构对深度增加的网络进行全面的评估，表面深度推进到16~19个权重层，可以实现对现有配置的显著改进。同时该模型可以很好的泛化到其它数据集。VGG的网络结构可以看作不断重复的模块，因此在定义模型时可以先定义规律性的模块，然后给定配置的卷积层数(num\_convs)，输入通道数(in\_channels)和输出通道数(out\_channels)，调用模块实现完整的架构，避免代码冗长。其中包括5个卷积模块（block），前2块为单层卷积，后3层为双层卷积。该网络总共8个卷积层和3个全连接层，所以也称为VGG-11。

def func\_asterisk(\*s\_position,\*\*d\_keywords):  
 print("\*s\_position={}\n\*\*d\_keywords={}".format(s\_position,d\_keywords))  
  
func\_asterisk('a','b','c','d','e',param\_a=1,param\_b=2,param\_c=3)

\*s\_position=('a', 'b', 'c', 'd', 'e')  
\*\*d\_keywords={'param\_a': 1, 'param\_b': 2, 'param\_c': 3}

'''  
@author:《动手学深度学习》/'Dive into PyTorch'  
Updated on Thu Jan 14 17:45:17 2021 @author: Richie Bao  
'''  
  
import torch  
from torch import nn,optim  
  
# 将每批次样本X的形状转换为(batch\_size,-1)  
from torch import nn  
class flattenLayer(nn.Module):  
 def \_\_init\_\_(self):  
 super(flattenLayer,self).\_\_init\_\_()  
 def forward(self,x):  
 return x.view(x.shape[0],-1)  
  
device=torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
  
def vgg\_block(num\_convs,in\_channels,out\_channels):  
 blk=[]  
 for i in range(num\_convs):  
 if i==0:  
 blk.append(nn.Conv2d(in\_channels,out\_channels,kernel\_size=3,padding=1))  
 else:  
 blk.append(nn.Conv2d(out\_channels,out\_channels,kernel\_size=3,padding=1))  
 blk.append(nn.ReLU())  
 blk.append(nn.MaxPool2d(kernel\_size=2,stride=2))  
 return nn.Sequential(\*blk)   
  
conv\_arch=((1, 1, 64), (1, 64, 128), (2, 128, 256), (2, 256, 512), (2, 512, 512))  
fc\_features=512\*7\*7  
fc\_hidden\_units=4096  
  
def vgg(conv\_arch,fc\_features,fc\_hidden\_units=4096):  
 net=nn.Sequential()  
 for i,(num\_convs,in\_channels,out\_channels) in enumerate(conv\_arch):  
 net.add\_module('vgg\_block\_'+str(i+1),vgg\_block(num\_convs,in\_channels,out\_channels))  
 net.add\_module('fc',nn.Sequential(flattenLayer(),  
 nn.Linear(fc\_features,fc\_hidden\_units),  
 nn.ReLU(),  
 nn.Dropout(0.5),  
 nn.Linear(fc\_hidden\_units,fc\_hidden\_units),  
 nn.ReLU(),  
 nn.Dropout(0.5),  
 nn.Linear(fc\_hidden\_units,10)   
 ))  
 return net  
  
VGG\_net=vgg(conv\_arch,fc\_features,fc\_hidden\_units)  
print("VGG网络结构：\n",VGG\_net)

VGG网络结构：  
 Sequential(  
 (vgg\_block\_1): Sequential(  
 (0): Conv2d(1, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (vgg\_block\_2): Sequential(  
 (0): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (vgg\_block\_3): Sequential(  
 (0): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (3): ReLU()  
 (4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (vgg\_block\_4): Sequential(  
 (0): Conv2d(256, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (3): ReLU()  
 (4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (vgg\_block\_5): Sequential(  
 (0): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (3): ReLU()  
 (4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (fc): Sequential(  
 (0): flattenLayer()  
 (1): Linear(in\_features=25088, out\_features=4096, bias=True)  
 (2): ReLU()  
 (3): Dropout(p=0.5, inplace=False)  
 (4): Linear(in\_features=4096, out\_features=4096, bias=True)  
 (5): ReLU()  
 (6): Dropout(p=0.5, inplace=False)  
 (7): Linear(in\_features=4096, out\_features=10, bias=True)  
 )  
)

配置完卷积层（特征提取层），可以通过上述定义的conv2d\_outputSize\_A\_oneTime函数，计算全连接层的输入尺寸。计算结果为(7,7)，将其与卷积层最后一层的输出通道数相乘就为全连接层的输入尺寸，这与VGG的输入值相同。

VGG\_convs\_params=[  
 ('input',(224,224)),  
 ('conv',[3,1,1,1]), #kernel\_size, stride, pad, dilation  
 ('pool',[2,2,0,1]),  
   
 ('conv',[3,1,1,1]),  
 ('pool',[2,2,0,1]),  
   
 ('conv',[3,1,1,1]),  
 ('conv',[3,1,1,1]),  
 ('pool',[2,2,0,1]),   
   
 ('conv',[3,1,1,1]),  
 ('conv',[3,1,1,1]),  
 ('pool',[2,2,0,1]),   
   
 ('conv',[3,1,1,1]),  
 ('conv',[3,1,1,1]),  
 ('pool',[2,2,0,1])   
 ]  
  
VGG\_output\_h\_w=conv2d\_outputSize\_A\_oneTime(VGG\_convs\_params)   
print("卷积层输出尺寸={}".format(VGG\_output\_h\_w))

卷积层输出尺寸=(7, 7)

可以先生成随机的样本数据(batchsize, nChannels, Height, Width)，逐个循环每一模块，即Sequential对象，获取每一模块计算后数据的形状，从而观察、验证并用于辅助调整模型参数。

X=torch.rand(1, 1, 224, 224)  
for name, blk in VGG\_net.named\_children():   
 X = blk(X)  
 print(name, 'output shape: ', X.shape)

vgg\_block\_1 output shape: torch.Size([1, 64, 112, 112])  
vgg\_block\_2 output shape: torch.Size([1, 128, 56, 56])  
vgg\_block\_3 output shape: torch.Size([1, 256, 28, 28])  
vgg\_block\_4 output shape: torch.Size([1, 512, 14, 14])  
vgg\_block\_5 output shape: torch.Size([1, 512, 7, 7])  
fc output shape: torch.Size([1, 10])

VGG的原始输入图像尺寸为224，目前实验的数据为fashionMNIST数据集，一幅图像的尺寸为，因此使用torchvision.transforms.Resize的方法调整图像大小，该方法已经包含于自定义load\_fashionMNIST中，因此只需要配置输入参数resize=224就可以修改图像尺寸，满足VGG输入数据尺寸的要求。

因为卷积后的图像输出尺寸只与图像的输入尺寸，卷积层、池化层的卷积核大小，填充、步幅等有关，因此可以自由的修改卷积层的输入、输出通道，及全连接层隐含层的数量。在修改时只需要配置一个缩放比例参数ratio，将所有的相关值除以该值完成新的配置。

ratio = 8  
small\_conv\_arch = [(1, 1, 64//ratio), (1, 64//ratio, 128//ratio), (2, 128//ratio, 256//ratio), (2, 256//ratio, 512//ratio), (2, 512//ratio, 512//ratio)]  
VGG\_net\_= vgg(small\_conv\_arch, fc\_features // ratio, fc\_hidden\_units // ratio)  
print("VGG网络结构\_减少通道数：\n",VGG\_net\_)

VGG网络结构\_减少通道数：  
 Sequential(  
 (vgg\_block\_1): Sequential(  
 (0): Conv2d(1, 8, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (vgg\_block\_2): Sequential(  
 (0): Conv2d(8, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (vgg\_block\_3): Sequential(  
 (0): Conv2d(16, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): Conv2d(32, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (3): ReLU()  
 (4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (vgg\_block\_4): Sequential(  
 (0): Conv2d(32, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (3): ReLU()  
 (4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (vgg\_block\_5): Sequential(  
 (0): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU()  
 (2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (3): ReLU()  
 (4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (fc): Sequential(  
 (0): flattenLayer()  
 (1): Linear(in\_features=3136, out\_features=512, bias=True)  
 (2): ReLU()  
 (3): Dropout(p=0.5, inplace=False)  
 (4): Linear(in\_features=512, out\_features=512, bias=True)  
 (5): ReLU()  
 (6): Dropout(p=0.5, inplace=False)  
 (7): Linear(in\_features=512, out\_features=10, bias=True)  
 )  
)

batch\_size=64  
trainloader,testloader=load\_fashionMNIST(root='./dataset/FashionMNIST\_norm',batchsize=batch\_size,num\_workers=2,resize=224,n\_mean=0.5,n\_std=0.5)  
  
def dataiter\_view(dataiter):  
 '''  
 function - 查看可迭代数据形状  
 '''  
 dataiter\_=iter(dataiter)  
 images\_,labels\_=dataiter\_.next()  
 print('数据形状：',images\_.shape)  
 return images\_,labels\_  
images\_,labels\_=dataiter\_view(testloader)

C:\Users\richi\anaconda3\envs\USDA\_database\lib\site-packages\torchvision\io\image.py:11: UserWarning: Failed to load image Python extension: Could not find module 'C:\Users\richi\anaconda3\envs\USDA\_database\Lib\site-packages\torchvision\image.pyd' (or one of its dependencies). Try using the full path with constructor syntax.  
 warn(f"Failed to load image Python extension: {e}")  
  
  
数据形状： torch.Size([64, 1, 224, 224])

lr,num\_epochs=0.001,5  
optimizer=torch.optim.Adam(VGG\_net\_.parameters(),lr=lr)

def evaluate\_accuracy\_V2(data\_iter, net, device=None):  
 '''  
 function - 模型精度计算  
 '''   
 if device is None and isinstance(net,torch.nn.Module):  
 device=list(net.parameters())[0].device #如果没指定device就使用net的device  
 acc\_sum,n=0.0,0  
 with torch.no\_grad():  
 for X,y in data\_iter:  
 if isinstance(net,torch.nn.Module):  
 net.eval() #评估模式，会关闭dropout  
 acc\_sum += (net(X.to(device)).argmax(dim=1) == y.to(device)).float().sum().cpu().item()  
 net.train() #改回训练模式  
 n+=y.shape[0]  
 return acc\_sum/n  
   
def train\_v2(net, train\_iter, test\_iter,optimizer, device, num\_epochs):  
 '''  
 function - 训练模型，v2版  
 '''  
 from tqdm.auto import tqdm  
 import time  
   
 net = net.to(device)  
 print("training on-",device)  
 loss=torch.nn.CrossEntropyLoss()  
 for epoch in tqdm(range(num\_epochs)):  
 train\_l\_sum, train\_acc\_sum, n, batch\_count, start = 0.0, 0.0, 0, 0, time.time()  
 for X,y in train\_iter:  
 X=X.to(device)  
 y=y.to(device)  
 y\_pred=net(X)  
 l=loss(y\_pred,y)  
 optimizer.zero\_grad()  
 l.backward()  
 optimizer.step()  
 train\_l\_sum += l.cpu().item()  
 train\_acc\_sum += (y\_pred.argmax(dim=1) == y).sum().cpu().item()  
 n += y.shape[0]  
 batch\_count += 1  
 test\_acc=evaluate\_accuracy\_V2(test\_iter, net)  
 print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f, time %.1f sec'% (epoch + 1, train\_l\_sum / batch\_count, train\_acc\_sum / n, test\_acc, time.time() - start))

train\_v2(net=VGG\_net\_, train\_iter=trainloader, test\_iter=testloader,optimizer=optimizer, device=device, num\_epochs=num\_epochs)

training on- cuda  
  
  
  
 0%| | 0/5 [00:00<?, ?it/s]  
  
  
epoch 1, loss 0.6070, train acc 0.775, test acc 0.874, time 63.3 sec  
epoch 2, loss 0.3222, train acc 0.886, test acc 0.904, time 67.9 sec  
epoch 3, loss 0.2749, train acc 0.902, test acc 0.905, time 82.1 sec  
epoch 4, loss 0.2418, train acc 0.914, test acc 0.912, time 98.6 sec  
epoch 5, loss 0.2180, train acc 0.922, test acc 0.915, time 107.7 sec

#### 2）torchvision.models实现VGG网络

VGG网络包含于torchvision.models库中， 因此无需自行配置网络，直接下载使用。通常模型也包含预先训练的模型参数，可以配置pretrained=True下载，下载到本地的位置为C:\Users\<your name>\.cache\torch\hub\checkpoints，后缀名为.pth。有时直接下载的网络并不能直接应用到其它不同的数据集，例如fashionMNIST数据集，该数据集的图像为灰色，即只有一个通道；同时，只有10个标签。因此需要对应层修改输入、输出大小。在[Finetuning Torchvision Models](https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html)④中，包含了模型参数微调的方法，对于VGG而言，修改卷积层，可以通过.features[idx]的方式读取；修改全连接层，可以通过.classifier[idx]的方式修改。需要修改的层为features[0]和classifier[6]这两个层。其它层不需修改。

注意在训练模型之前，需要调整优化函数optimizer=torch.optim.Adam(VGG\_model.parameters(),lr=lr)的输入模型参数为当前的网络模型。

import torchvision   
VGG\_model=torchvision.models.vgg11(pretrained=False) #如果为真，返回在ImageNet上预先训练的模型

print(VGG\_model)

VGG(  
 (features): Sequential(  
 (0): Conv2d(3, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU(inplace=True)  
 (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (3): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (4): ReLU(inplace=True)  
 (5): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (6): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (7): ReLU(inplace=True)  
 (8): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (9): ReLU(inplace=True)  
 (10): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (11): Conv2d(256, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (12): ReLU(inplace=True)  
 (13): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (14): ReLU(inplace=True)  
 (15): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (16): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (17): ReLU(inplace=True)  
 (18): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (19): ReLU(inplace=True)  
 (20): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (avgpool): AdaptiveAvgPool2d(output\_size=(7, 7))  
 (classifier): Sequential(  
 (0): Linear(in\_features=25088, out\_features=4096, bias=True)  
 (1): ReLU(inplace=True)  
 (2): Dropout(p=0.5, inplace=False)  
 (3): Linear(in\_features=4096, out\_features=4096, bias=True)  
 (4): ReLU(inplace=True)  
 (5): Dropout(p=0.5, inplace=False)  
 (6): Linear(in\_features=4096, out\_features=1000, bias=True)  
 )  
)

from torch import nn  
VGG\_model.features[0]=nn.Conv2d(1, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
VGG\_model.classifier[6]=nn.Linear(in\_features=4096, out\_features=10, bias=True)  
print(VGG\_model)

VGG(  
 (features): Sequential(  
 (0): Conv2d(1, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (1): ReLU(inplace=True)  
 (2): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (3): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (4): ReLU(inplace=True)  
 (5): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (6): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (7): ReLU(inplace=True)  
 (8): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (9): ReLU(inplace=True)  
 (10): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (11): Conv2d(256, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (12): ReLU(inplace=True)  
 (13): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (14): ReLU(inplace=True)  
 (15): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 (16): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (17): ReLU(inplace=True)  
 (18): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))  
 (19): ReLU(inplace=True)  
 (20): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)  
 )  
 (avgpool): AdaptiveAvgPool2d(output\_size=(7, 7))  
 (classifier): Sequential(  
 (0): Linear(in\_features=25088, out\_features=4096, bias=True)  
 (1): ReLU(inplace=True)  
 (2): Dropout(p=0.5, inplace=False)  
 (3): Linear(in\_features=4096, out\_features=4096, bias=True)  
 (4): ReLU(inplace=True)  
 (5): Dropout(p=0.5, inplace=False)  
 (6): Linear(in\_features=4096, out\_features=10, bias=True)  
 )  
)

import torch  
device=torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
lr,num\_epochs=0.001,5  
optimizer=torch.optim.Adam(VGG\_model.parameters(),lr=lr)  
train\_v2(net=VGG\_model, train\_iter=trainloader, test\_iter=testloader,optimizer=optimizer, device=device, num\_epochs=num\_epochs)

training on- cuda  
  
  
  
 0%| | 0/5 [00:00<?, ?it/s]  
  
  
epoch 1, loss 0.6091, train acc 0.814, test acc 0.876, time 944.2 sec  
epoch 2, loss 0.2923, train acc 0.893, test acc 0.895, time 1089.5 sec  
epoch 3, loss 0.2509, train acc 0.910, test acc 0.908, time 1063.9 sec  
epoch 4, loss 0.2229, train acc 0.919, test acc 0.913, time 1047.5 sec  
epoch 5, loss 0.2006, train acc 0.926, test acc 0.919, time 1089.6 sec

注释（Notes）：

① torch.nn.Conv2d，（<https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html>）。

② torch.utils.tensorboard，（<https://pytorch.org/docs/stable/tensorboard.html>）。

③ torchvision.models，预定义深度学习模型，（<https://pytorch.org/vision/0.8/models.html>）。

④ Finetuning Torchvision Models，对 torchvision 模型进行微调和特征提取，（<https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html>）。

参考文献（References）:

[1] Jibin Mathew.PyTorch Artificial Intelligence Fundamentals: A recipe-based approach to design, build and deploy your own AI models with PyTorch 1.x[m].UK:Packt Publishing (February 28, 2020)

[2] Convolutional Neural Networks, <https://cs231n.github.io/convolutional-networks>.

[3] Understanding 2D Dilated Convolution Operation with Examples in Numpy and Tensorflow with Interactive Code, <https://towardsdatascience.com/understanding-2d-dilated-convolution-operation-with-examples-in-numpy-and-tensorflow-with-d376b3972b25>.

[4] [Visualizing Models, Data, and Training with TensorBoard, <https://pytorch.org/tutorials/intermediate/tensorboard\_tutorial.html?highlight=fashion%20mnist).

[5] [Visualizing Filters and Feature Maps in Convolutional Neural Networks using PyTorch, <https://debuggercafe.com/visualizing-filters-and-feature-maps-in-convolutional-neural-networks-using-pytorch/>.

[6] Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. (2014).