周学习总结

许典

CNN结构设计

卷积层1

- 输入 128 x 128 x 3(channels)
- 卷积核 3x3x16 池化 4
- 输出 32 x 32 x 16(channels)

卷积层2

- 输入 32 x 32 x 16(channels)
- 卷积核 3x3x32 池化 8
- 输出 4 x 4 x 32(channels)

全连接层

- 输入 4 x 4 x 32
- 输出 11

Model

```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 16, 3, 1, 1),  # (16 128 128)
            nn.ReLU(),
            nn.MaxPool2d(4, 4, 0),  # (16 32 32)

            nn.Conv2d(16, 32, 3, 1, 1),  # (32 32 32)
            nn.ReLU(),
            nn.MaxPool2d(8, 8, 0),  # (32 4 4)
        )
        self.fc = nn.Sequential(
            nn.ReLU(),
            nn.Linear(32*4*4, 11),
        )

    def forward(self, x):
        out = self.cnn(x)
        out = out.view(out.size()[0], -1)
        return self.fc(out)
```

卷积层1-卷积

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

x16

x16

输入: 128x128x3(channels)

卷积核: 3x3x3x16

输出: 128x128x16(channels)

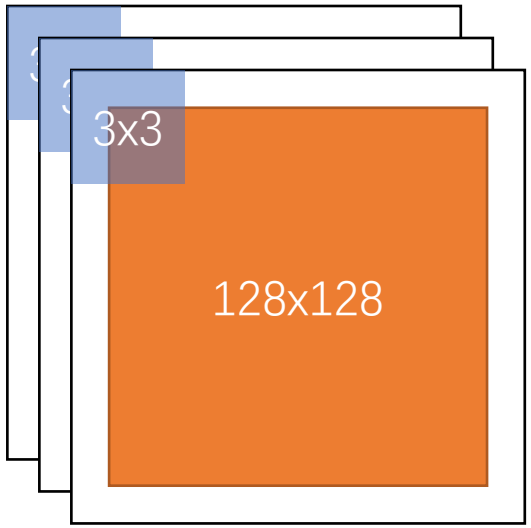
Docs > torch.nn > Conv2d

CONV2D

CLASS `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')` [SOURCE]

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

In channels: 3
Out channels: 16
Kernel size: 3
Stride: 1
Padding: 1




```cpp
/**
 * T: 数据类型名称
 * K: 卷积核大小
 * ICH: 输入通道数
 * OCH: 输出通道数
 * ISZ: 输入尺寸
 * OSZ: 输出尺寸
 * PADDING: 卷积时添加的边缘
 * STRIDE: 卷积的步幅
 * */
template<typename T, int K, int ICH, int OCH, int ISZ, int OSZ, int PADDING, int STRIDE>
void conv2d(T din[ISZ][ISZ][ICH], T dout[OSZ][OSZ][OCH], T weight[K][K][ICH][OCH], T bias[OCH]) {
    T kernel[K][K][ICH];
    int nexi = -1;
    int nexj = 0;
    for(int ii = -PADDING; ii < ISZ+PADDING; ii+=STRIDE) {
        // 判断kernel是否出界
        if(ii + K > ISZ + PADDING) break;
        nexi++;
        nexj = 0;
        for(int jj = -PADDING; jj < ISZ+PADDING; jj+=STRIDE) {
            //判断kernel是否出界
            if(jj + K > ISZ + PADDING) break;
            for(int i = 0; i < K; ++i) {
                for(int j = 0; j < K; ++j) {
                    // 判断kernel中的元素是否在输入图片范围之内
                    if(ii + i < 0 or ii + i >= ISZ or jj + j < 0 or jj + j >= ISZ) {
                        for(int c = 0; c < ICH; ++c) {
                            kernel[i][j][c] = 0;
                        }
                    } else {
                        for(int c = 0; c < ICH; ++c) {
                            kernel[i][j][c] = din[ii + i][jj + j][c];
                        }
                        //memcpy(kernel[i][j], din[ii+i][jj+j], sizeof(T)*ICH);
                        //如果把通道放到末尾的话，可以直接整个通道拷贝
                    }
                }
            }
            for(int cho = 0; cho < OCH; ++cho) {
                dout[nexi][nexj][cho] = bias[cho];
                for(int i = 0; i < K; ++i) {
                    for(int j = 0; j < K; ++j) {
                        for(int chi = 0; chi < ICH; ++chi) {
                            dout[nexi][nexj][cho] += kernel[i][j][chi] * weight[i][j][chi][cho];
                        }
                    }
                }
            }
            ++nexj;
        }
    }
}
```

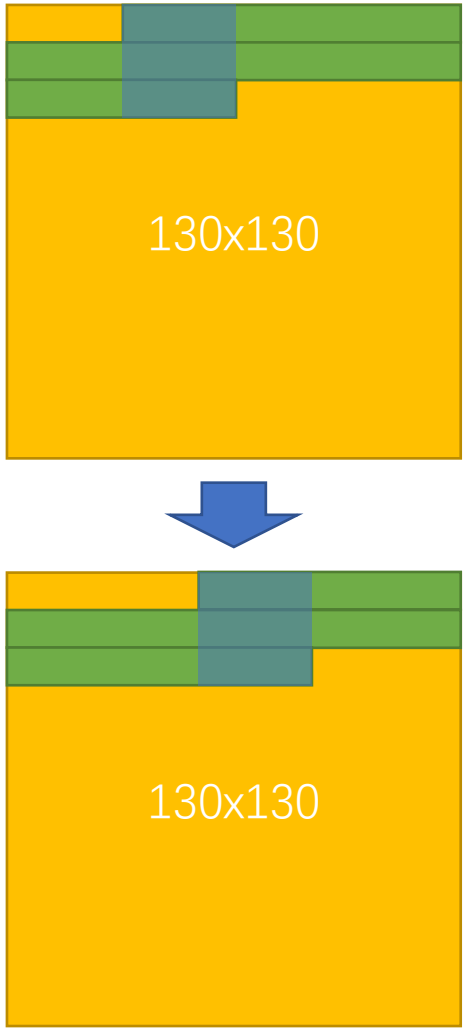
卷积操作的预处理

- 预处理要不要纳入统计，不纳入统计的预处理的操作的边界在哪？
  - 模型参数可以任意处理而不纳入统计
  - 输入数据也可以任意处理而不纳入统计
  - 和模型参数和输入数据都相关的预处理需要纳入统计

预处理方式1-line buffer

在FPGA中BRAM中申请一个小一点的数组，本样例中数组大小为263，然后每次请求一个新的数据，就能重新计算一次卷积核

缺点：换行的时候比较麻烦



预处理方式2-Stream

- 陈双的论文中提到的方法
- 将输入输出预处理成流的方式，可以解除数据之间的依赖关系。

Figure 4. Data rearrangement.