Deep Learning (IST, 2021-22)

Practical 8: Convolutional Neural Networks

João Santinha, José Maria Moreira, Luís Sá-Couto, and Andreas Wichert

Introduction

We have now learned about the Perceptron, Linear and Logistic Regression, Multi-layer Perceptron, Auto-encoders, and the gradient backpropagation algorithm.

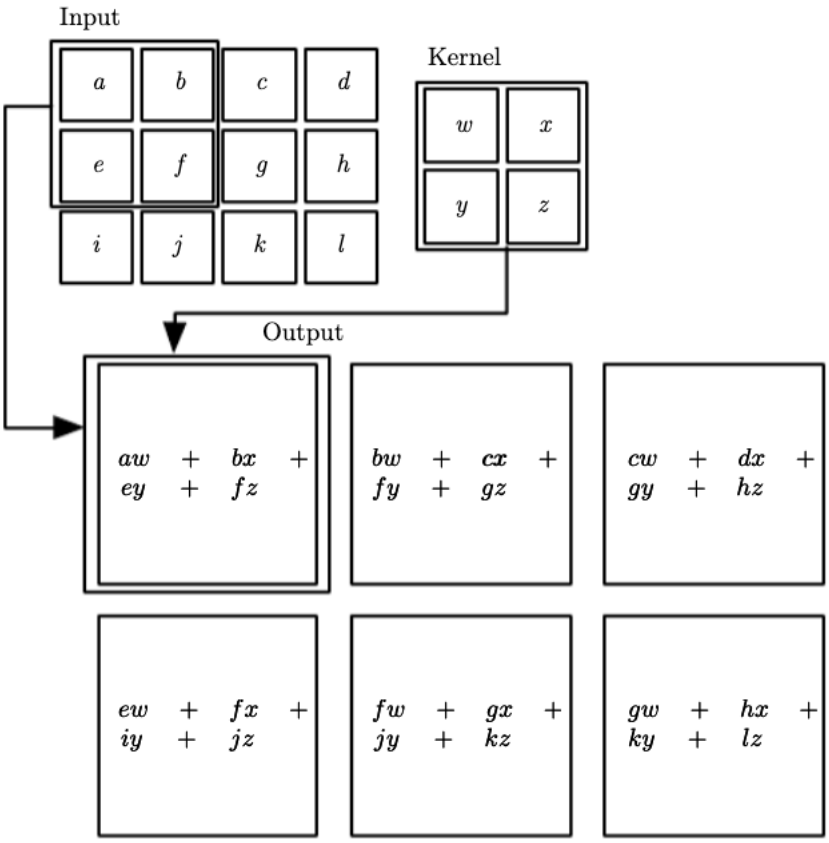
Today we will talk about Convolutional Neural Networks (CNNs). We will start with a pen-and-pencil exercise, and will move to Google Colab, where we will do some programming hands-on exercises.

CNNs provide a clever way of exploiting input structure. Natural images are an example of data where this structure exists and where CNNs have been particularly successful.

Important properties of CNNs:

- Invariance → learn useful representations with fewer parameters
  - Locality (learning small convolutional kernels/filters will dramatically decrease the number of trainable parameters with features being translational invariant)
  - Spatial Invariance / Equivariant Representations
- Sparse Interactions (induced by kernels smaller than the input)
- Parameter Sharing

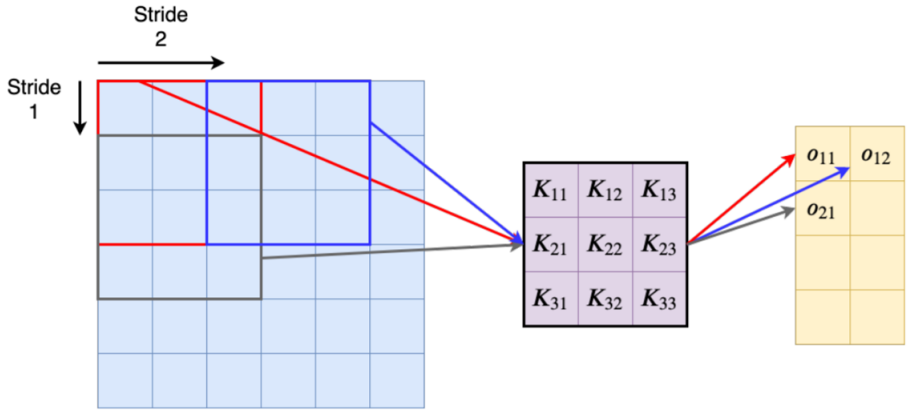
Their name comes from the fact that the network employs the mathematical operation called convolution which can be represented by (with kernel size 2 and stride 1):



[From Goodfellow I., Bengio Y., Courville A. Deep learning. MIT press; 2016.]

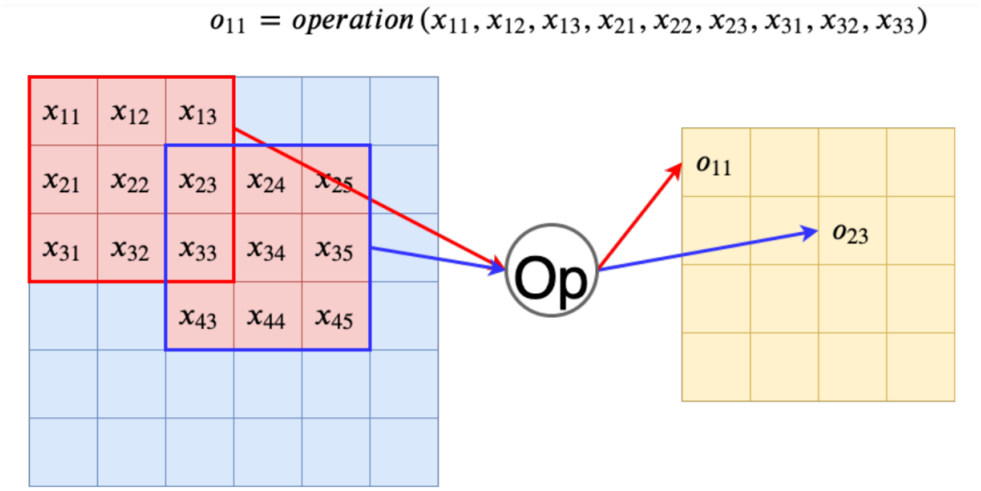
In its simplest form, and to be correct, this performs a cross-correlation operation on the two-dimensional input data and the kernel (and then adds a bias).

It is possible to change the the stride and the kernel shape ($s_w = 2$, $s_h = 1$, and $k : (3, 3)$), leading to:



Another operation in the context of CNNs is the pooling operation, which is very similar to the convolution operation, however without the kernel. Pooling applies an operation to each window it is evaluating from the input. As examples, there is the max pooling and the average

pooling, which finds the maximum value or calculates the average of the values in the window, respectively:



Question 1

Consider the following input image:

$$\mathbf{I} = \begin{pmatrix} 20 & 35 & 35 & 35 & 35 & 20 \\ 29 & 46 & 44 & 42 & 42 & 27 \\ 16 & 25 & 21 & 19 & 19 & 12 \\ 66 & 120 & 116 & 154 & 114 & 62 \\ 74 & 216 & 174 & 252 & 172 & 112 \\ 70 & 210 & 170 & 250 & 170 & 110 \end{pmatrix}$$

1. What is the output provided by a convolution layer with the following properties:
   - Stride: (1,1)
   - Kernel:
   $$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$
2. Take the output from 1. and apply a max pooling layer with the following properties:
   - Stride: (2,2)
   - Window Shape: (2,2)
3. For the following kernels, describe what kind of feature they extract from the image:

$$\mathbf{F1} = \begin{pmatrix} -10 & -10 & -10 \\ 5 & 5 & 5 \\ -10 & -10 & -10 \end{pmatrix}$$

$$\mathbf{F2} = \begin{pmatrix} 2 & 2 & 2 \\ 2 & -12 & 2 \\ 2 & 2 & 2 \end{pmatrix}$$

$$\mathbf{F3} = \begin{pmatrix} -20 & -10 & 0 & 5 & 10 \\ -10 & 0 & 5 & 10 & 5 \\ 0 & 5 & 10 & 5 & 0 \\ 5 & 10 & 5 & 0 & -10 \\ 10 & 5 & 0 & -10 & -20 \end{pmatrix}$$

Question 2

Consider that your input images have size 227×227×3 and filter of size 11×11×3 in the first convolution layer. In the case of AlexNet we have in the first convolution layer a total of 96 filters, a stride of 4 and a padding of 0.

1. Determine the width/height of the output of this first convolution layer.
2. Determine the number of units within the first convolutional layer.
3. Determine the number of trainable parameters/weights within the convolutional layer with weight sharing.
4. How many parameters would we have if we had used a FF layer with 256 units instead of a convolutional layer?

Question 3

Now it's time to try the Convolutional Neural Networks on real data and compare it with other approaches previously seen in our practical sessions.

1. Download and create the MNIST train and validation datasets using `torchvision`:

   ```
   from torchvision import datasets, transforms
   mnist_train_dataset = datasets.MNIST('../data', download=True, train=True,
                                        transform=mnist_transform)
   mnist_val_dataset = datasets.MNIST('../data', download=True, train=False,
                                      transform=mnist_transform)
   ```

   where

   ```
   mnist_transform = transforms.Compose([transforms.ToTensor(),
                                 transforms.Normalize((0.1307,), (0.3081,))])
   ```

   Implement a Logistic Regression using `scikit-learn` and a Feedforward Neural Network with 3 hidden layers of dimensions 256, 128, 64, using `pytorch`. Train the Feedforward Neural Network using as optimizer `SGD` with a learning rate of 0.01 and a momentum 0.5, `nn.NLLLoss()` as your loss function. These models will be our baselines to be compared with CNNs.

   Run your implementation of the Logistic Regression and the Feedforward Neural Network algorithms on this dataset. Measure the training and validation accuracy.

   Plot the coefficients of the Logistic Regression for each class.

2. Implement a CNN with the following structure:

- convolutional layer with 10 output channels, kernel size of 5 x 5, and default padding and stride.
- max pooling layer with kernel size of 2 x2.
- a ReLU activation function.
- convolutional layer with 20 output channels, kernel size of 5 x 5, and default padding and stride.
- dropout layer with default probability.
- max pooling layer with kernel size of 2 x2.
- a ReLU activation function.
- affine transformation with 50 output features.
- a ReLU activation function.
- affine transformation with 10 output features followed by a `log_softmax()`.

Use the Adam optimizer with a learning rate of 0.001 and the negative log likelihood loss (`nn.NLLLoss()`).

3. Determine the number of learnable parameters of the Feedforward Neural Network and the CNN.

4. Change the number of hidden layers and respective number of units so that it approximates the number of learnable parameters of the CNN. Run your implementation and measure the training and validation accuracy of your new Neural Network.

   Are you able to maintain your performance?

5. What happens in each of the classifiers if for some reason your validation set does not always have the digits centered in your input image?

   Hint:

```
mnist_transform_val = transforms.Compose([transforms.ToTensor(),
                                transforms.RandomAffine(0, translate=[0.1, 0]),
                                transforms.Normalize((0.1307,), (0.3081,))])
mnist_val_dataset = datasets.MNIST('../data', download=True, train=False,
                                        transform=mnist_transform_val)
```

   What property of CNNs allows it to maintain its performance?