Neural Networks for Classification

Convolutional Networks for Image Classification

Joseph Le Roux

18/10/25

Outline

- Convolutions
- For Image Processing
- For Feature Extraction

Topic

- Convolutions
- For Image Processing
- For Feature Extraction

Convolutions

Discrete Convolution between two vectors

Convolution of filter a over input v

$$f = a * v \text{ such that } f_n = \sum_{m=-\infty}^{+\infty} a_m v_{n-m}$$

Can be difficult to read. Let's rewrite it:

$$f_n = \sum_{p,q \text{ s.t. } p+q=n} a_p v_q$$

Example $a = [1, 2, 3]$ and $v = [4, 5, 6, 7]$ give $f = [4, 13, 28, 34, 32, 21]$

- $f_0 = a_0 v_0 = 4$
- $f_1 = a_0 v_1 + a_1 v_0 = 13$
- $f_2 = a_0 v_2 + a_1 v_1 + a_2 v_0 = 28$
- $f_3 = a_0 v_3 + a_1 v_2 + a_2 v_1 = 34$
- $f_4 = a_1 v_3 + a_2 v_2 = 32$
- $f_5 = a_2 v_3 = 21$

```
import numpy as np

print(np.convolve([1,2,3], [4,5,6,7]))
```

```
[ 4 13 28 34 32 21]
```

In Pytorch this was called a 1D convolution.

Let's code this with PyTorch API

```
import torch
conv = torch.nn.Conv1d(1, #each input position in (4,5,6) is 1 number
                       1, #each output is 1 number (1 filter)
                       kernel_size=3, #convolution has size 3 (1,2,3)
                       padding=2, #at most vector a has 2 positions out
                       bias=False) #combination is linear, not affine
conv.weight.requires_grad=False # so we can change values by hand
#print(conv.weight.size()) # (1,1,3) (out,in,kernel)
conv.weight[0][0] = torch.tensor([3,2,1]) # in reverse order

print(conv(torch.tensor([4.,5.,6.,7.]).unsqueeze(0)).squeeze(0))
```

```
tensor([ 4., 13., 28., 34., 32., 21.])
```

A Little Picture

Here be drawing. . .

Topic

- Convolutions
- For Image Processing
- For Feature Extraction

Convolutions for Images (1)

(we forget about reversing filters from now on. . .)

Why convolution?

- we want to represent a linear transformation of a patch
- need to fix a small discrepancy between the *mathematical* convolution and the *computer vision* convolution.

Instead of vectors, matrices

- our image is a matrix M, i.e. a 32×32, where each point is value between 0 (black) and 1 (white) (we'll change that later)
- filter A is a matrix (usually square) $k \times k$, ($k = 3, 5 \ldots$ odd)

if we adapt the previous formula, the output is a matrix f where:

$$f_{ij} = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} A_{nm} M_{i-n,j-m}$$

p little drawing shows this is not exactly what we want

let's have the filter "centered" around position ij

let's define $p = \frac{k}{2}$ (rounded down)

$$f_{ij} = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} A_{p+n,p+m} M_{i-n,j-m}$$

- now, little drawing shows this is *exactly* what we want
- really?

More realistically

images are not matrices

- each point can have several pieces of information, called channels, encoded as vectors
- for instance:
  - 3 channels for RGB coding (red/green/blue)
  - either integer between 0 and 255, or
  - float between 0 and 1
- Images are encoded as 3d tensors: $C \times W \times H$ with $C = 3$
- Bias term b : compute affine functions instead of linear

$$f_{ij} = b + \sum_{c=0}^{C-1} \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} A_{c,p+n,p+m} M_{c,i-n,j-m}$$

Convolutions for Images (4)

What can we do with convolutions? Alter each pixel!

Example Average (blurring effect)

First, load a picture

```
import torch
from torchvision import io
import matplotlib.pyplot as plt

img = io.read_image("../static/IMG_20220726_161509.jpg")

fig = plt.figure(figsize=(img.size(1)/ 150,img.size(2) / 150))
plt.imshow(img.permute(1,2,0))
```

Average (blurring effect)

Second, apply an average convolution

```
ks, n_ch = 20,3 # kernel size (patch size) and number of channels (RGB)

conv = torch.nn.Conv2d(n_ch, n_ch, ks, padding=ks-n_ch + 1, bias=False) #no bias
conv.weight.requires_grad=False #so we can replace values manually
print(conv.weight.size()) #(3,3,20,20)
conv.weight[:,:,:,:] = torch.zeros(n_ch,n_ch,ks,ks) #reset all values to zero

patch = torch.ones(ks,ks) / (ks*ks) # uniform dist.

conv.weight[0][0] = patch # for R channel
conv.weight[1][1] = patch # for G channel
conv.weight[2][2] = patch # for B channel

img = img.to(torch.float)/256 # all values between 0 and 1

blurred = conv(img)
io.write_jpeg((256*blurred).to(torch.uint8), "blurred.jpg", 100)
```

Convolutions for Images (6)

Average (blurring effect)

Convolutions for Images (7)

Sharpening (edge detection)

```
ks, n_ch = 20,3 # kernel size (patch size) and number of channels (RGB)

conv = torch.nn.Conv2d(n_ch, n_ch, ks, padding=ks-n_ch + 1, bias=False) #no bias
conv.weight.requires_grad=False #so we can replace values manually
print(conv.weight.size()) #(3,3,20,20)
conv.weight[:,:,:,:] = torch.zeros(n_ch,n_ch,ks,ks) #reset all values to zero

ks, n_ch = 3, 3  # kernel size (patch size) and number of channels (RGB)

conv = torch.nn.Conv2d(n_ch, n_ch, ks, padding=ks - n_ch + 1, bias=False)  # no bias
conv.weight.requires_grad = False  # so we can replace values manually
print(conv.weight.size())  # (3,3,3,3)
conv.weight[:, :, :, :] = torch.zeros(n_ch, n_ch, ks, ks)  # reset all values to zero

patch = (torch.tensor([[0, -1, 0],
                       [-1, 4, -1],
                       [0, -1, 0]]) / 4)

conv.weight[0][0] = patch # for R channel
conv.weight[1][1] = patch # for G channel
conv.weight[2][2] = patch # for B channel

img = img.to(torch.float)/256 # all values between 0 and 1
sharpened = conv(img)
io.write_jpeg((256*sharpened).to(torch.uint8), "sharpened.jpg", 100)
```

Convolutions for Images (8)

Average (blurring effect)

Fun... but that is not what we will use convolutions for :(

Topic

- Convolutions
- For Image Processing
- For Feature Extraction

- channels represent information about points (samples) in the image
- at the beginning, we only have information about pixel color (RGB)
- we want:
  1. incorporate information about neighbours
  2. move from information about points to information about zones/areas
  3. bigger and bigger areas until we have information about the entire image and make a prediction
- steps 1 and 2 will be repeated
- step 3 will be a MLP from the final image information

To sum up:

- step 1 realized by convolution
- step 2 realized by pooling (seen next)
- step 3 realized by MLP

Convolutions to extract information about neighbours (1)

Compute several filters for each position

```
ks, n_in_ch, n_out_ch = 5,3,12 # kernel size, input channels, output channels
conv = torch.nn.Conv2d(n_in_ch, n_out_ch, ks) # compute n_out_ch filters
print(conv.weight.size(), conv.bias.size()) # (12,3,5,5), (12)

print(img.size()) # (3, 640, 480) each pixel has 3 informations
img_w_neighbour_info= conv(img)
print(img_w_neighbour_info)# # (12, 636, 476) whaaaat!!!
```

- We have the right number of output channels, but the image is shrunk
- Convolutions are *centered*, so extreme image points are ignored

Convolutions to extract information about neighbours (2)

Size of the resulting matrix

How many consecutive k positions in a vector of size n

- $n - k + 1$ (draw picture if need be)
- this means that each convolution *loses* some positions

Padding to control the output size

- use padding to add *fake* positions around image (zero by default)

```
ks, n_in_ch, n_out_ch = 5,3,12
conv = torch.nn.Conv2d(n_in_ch, n_out_ch, ks, padding=2)

print(img.size()) # (3, 640, 480) each pixel has 3 informations
img_w_neighbour_info= conv(img)
print(img_w_neighbour_info)# # (12, 640, 480)
```

With convolutions

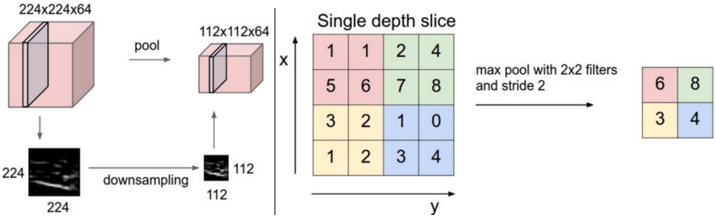
- represent each point by taking information from neighbourhood
- represent the same number of points (approx, with padding)
- change only de number of channels (aka *depth*)
- how do we represent *zones* of the images instead of points?

Pooling

- create a summary of zone
- do not change the depth
- change the size (aka the *spatial representation*)

Example: Max-Pooling

Implement pooling by returning for each channel the maximum value on a patch.



from cs231n.github.io

```
print(img_w_neighbour_info)# just a reminder (12, 640, 480)
mp = torch.nn.MaxPool2D(2)
img_small = mp(img_w_neighbour_info)
print(img_small.size()) # 12,320,240
```

Other Pooling Methods, Instead of max use:

- min
- average
- ...

Putting it all together

```python
class ConvNet1(torch.nn.Module):
  def __init__(self,):
    super().__init__()

    self.c1 = torch.nn.Conv2d(1, 10, 3, padding=1)
    self.m1 = torch.nn.MaxPool2d(2)
    self.c2 = torch.nn.Conv2d(10, 20, 5)
    self.m2 = torch.nn.MaxPool2d(2)
    self.c3 = torch.nn.Conv2d(20, 40, 3)
    self.m3 = torch.nn.MaxPool2d(3)
    self.fc = MLP(40, [30, 20],NB_CLASSES, torch.nn.ReLU)

  def forward(self,x):
    #print(x.size())
    x = self.c1(x)
    x = self.m1(x)
    x = self.c2(x)
    x = self.m2(x)
    x = self.c3(x)
    x = self.m3(x)
    x = self.fc(x)
    return x
```

Things we did not cover

Dropout

Randomly set neurons to zero during training, avoid overfitting, make the learned model more robust

Normalization

Like in image processing, it is often a good idea to normalize input (center values around zero, with variance 1. . .)

Transformers

Can we develop an architecture where the $\text{kernel}_{\text{size}}$ can be as large as the size of the picture?

Generation

Can we use convolutions to generate image instead of classifying them?