Introduction to Convolutional Neural Networks

Moritz Wolter

September 26, 2023

High Performance Computing and Analytics Lab, University of Bonn

Overview

The convolution operation in machine learning

Understanding convolution

Convolutional neural networks

Network coding

Motivation [GBC16]

- sparse interactions
- parameter sharing
- equivariant representations (i.e. with respect to translation)
- efficiency
- Train deeper networks.

The invention of convolutional neural networks

Proposed in Yann le Cun's [LeC+89].

The convolution operation in machine learning

Defining convolution

For two one-dimensional signals $x \in \mathbb{R}^T$ and $k \in \mathbb{R}^T$, convolution is defined as

$$s(t) = (x * k)(t) = \sum_{a=0}^{T} x(a)k(t-a), \tag{1}$$

for numbers t, a. Possible t will depend on signal length and padding.
In 2D, we require a kernel matrix $K \in \mathbb{R}^{O,P}$ and a image matrix $I \in \mathbb{K}^{N,M}$

$$S(i,j) = (K * I)(i,j) = \sum_{m}^{M} \sum_{n}^{N} I(i-m, j-n)K(n,m) \tag{2}$$

Again not just any, i, j will do. We will see what this means in a minute.

Defining cross-correlation

$$S(i,j) = (K * I) = \sum_{m}^{M} \sum_{n}^{N} I(i+m, j+n) K(m,n) \tag{3}$$

Cross-correlation is convolution without flipping the kernel [GBC16]. Many machine learning libraries implement cross-correlation and call it convolution. In this course we will follow their example.

Illustrating the convolution operation

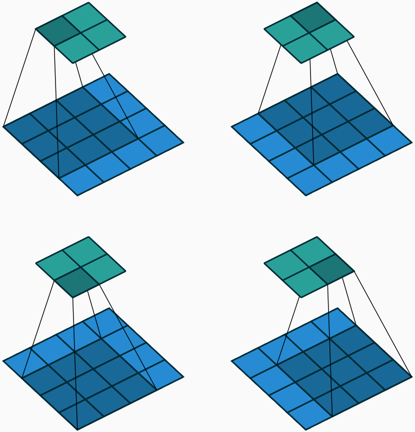


Figure: Illustration of the convolution operation without padding and unit strides [DV16].

Strided convolution

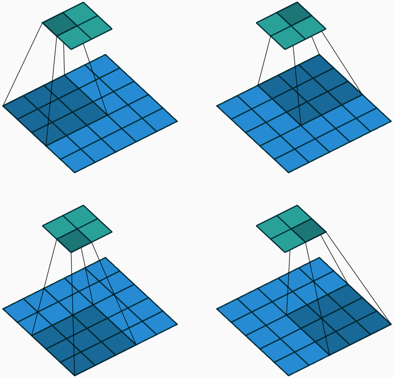


Figure: Visualization of stride two convolutions without padding [DV16].

Padded convolution

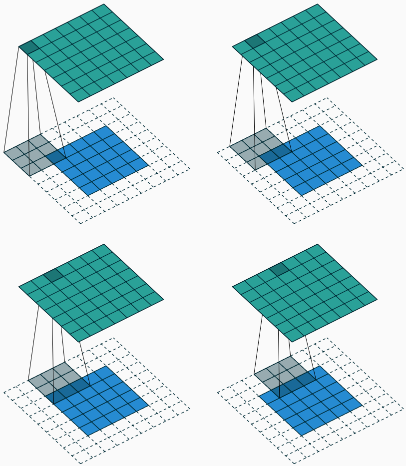


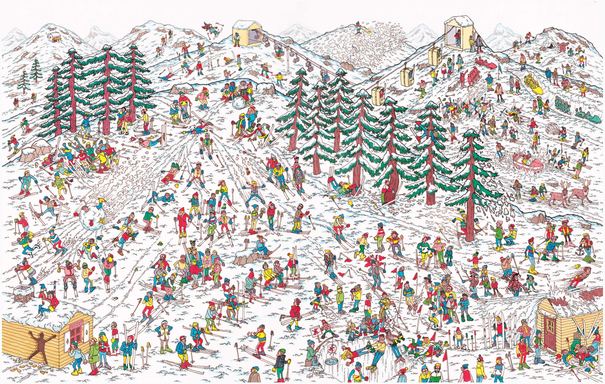
Figure: Visualization of fully padded convolutions with unit strides [DV16].

Summary

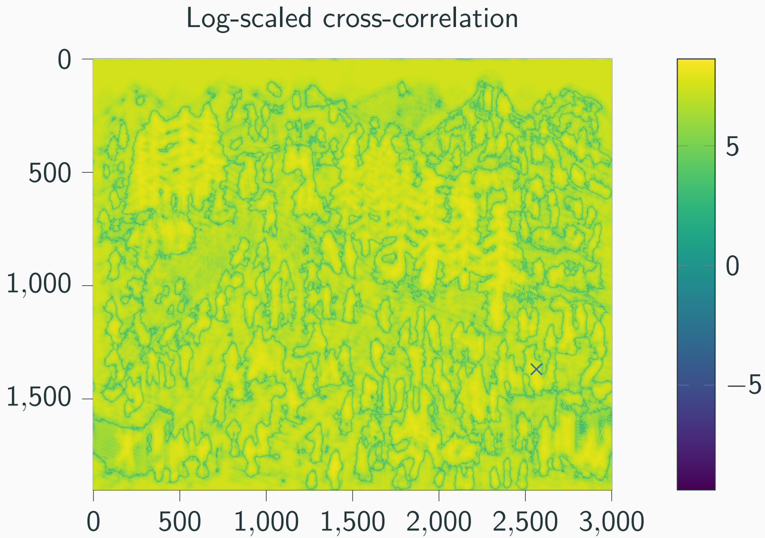
- The convolution operation slides convolution kernels over an image.
- Padding avoids losing pixels on the side.
- Strided convolutions downsample the input.
- Moving in steps of two pixels, for example, cuts the resolution in half.

Understanding convolution

Getting computers to find Waldo



Finding Waldo via cross-correlation.



Summary

- Cross-correlation is called convolution in the machine learning literature.
- Patterns can be located in signals via cross-correlation.

Convolutional neural networks

Motivating convolutional neural networks (CNN)

- Fixed filters work if we are looking for a very specific waldo.
- In other cases, we need a better solution.
- Convolutional neural networks rely on filter optimization via back-propagation.
- Filter optimization turns CNNs into very versatile tools!

Multichannel convolution

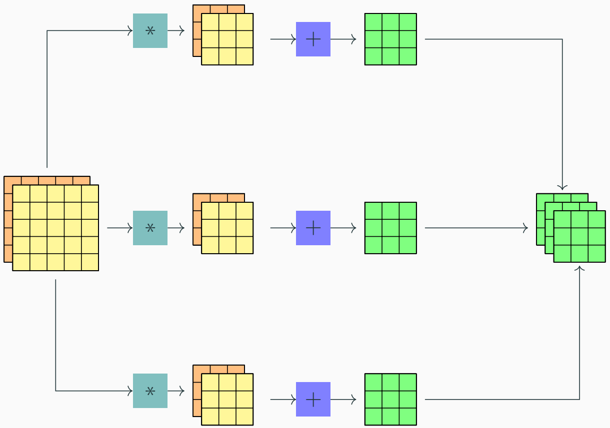


Figure: The plot shows a convolution computation using a $3x2x3x3$ kernel on a $2x5x5$ input. The kernel pairs convolve with the input, producing $3x3$ results. $+$ adds the two channels for each of the three tensors. Finally, everything is stacked. Inspired by [DV16, page 9].

Computing the output shape of a CNN layer

One can determine the output shape for each dimension individually. Without zero padding and a stride size of one

$$o = (i - k) + 1, \tag{4}$$

can be used to compute the output size. i denotes the input size, and k is the kernel size. [DV16] covers all cases which appear in practice.

Image to column and the forward pass

We already know how to train dense network layers using matrix multiplication. Training a CNN the same way requires restructuring the image to express convolution as matrix multiplication,

$$\bar{\mathbf{h}} = \mathbf{K}_f \mathbf{v}_I + \mathbf{b}, \tag{5}$$

$$\mathbf{h}_f = f(\bar{\mathbf{h}}). \tag{6}$$

$\mathbf{v}_I \in \mathbb{R}$ denotes the restructured image input. $\mathbf{K}_f \in \mathbb{R}^{k_o, k_i \cdot k_h \cdot k_w}$ the flattened restructured kernel. o, i, h, w denote the output, input, height, and width dimensions, respectively.

The backward pass

We apply the rules for dense layers to the restructured convolutional layer data,

$$\delta \mathbf{K}_f = [f'(\bar{\mathbf{h}}) \odot \triangle]_f \mathbf{v}_I^T, \qquad \delta \mathbf{b} = f'(\bar{\mathbf{h}}) \odot \triangle, \tag{7}$$

$$\delta \mathbf{x} = \left(\mathbf{K}_f^T [f'(\bar{\mathbf{h}}) \odot \triangle]_f\right)_{I^{-1}}. \tag{8}$$

With I and I^{-1} denoting the `im2col` and `col2im` operations. All major deep learning frameworks have both operations built in.

The classifier at the end

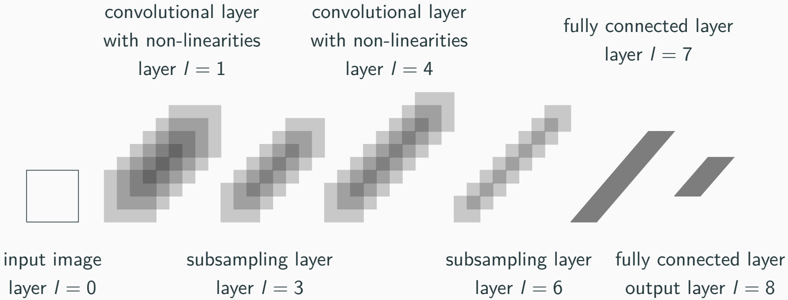


Figure: The LeNet-architecture[LeC+89] as illustrated by [Stu20].

The shifting input problem

- With the tools we have seen, shifting an input also shifts the CNN output before the dense classifier.
- Shifting the input would shift the input in front of the final dense-classifier neurons.
- We want invariance to translation.

Pooling

Max pooling layers choose maximum values in predefined regions. Two by two max pooling, for example, picks the maximum in neighboring areas of four pixels. If an input is shifted by two pixels, the result will remain the same! Pooling layers are used repeatedly for a cumulative effect.

MNIST

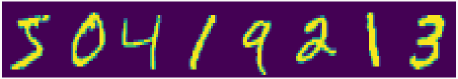


Figure: Sample digits from the MNIST-database.

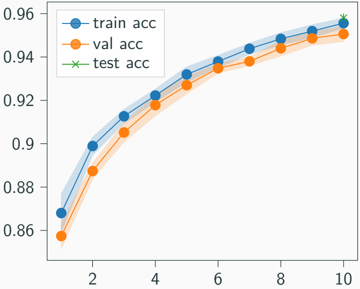


Figure: Mean convergence of two-layer CNN with a dense classifier.

Deep convolutional neural networks

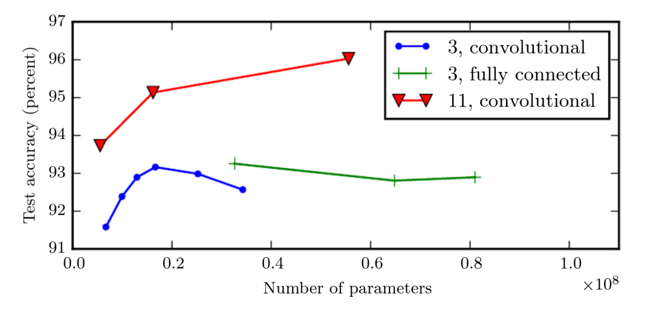


Figure: Comparing deep networks with and without convolutional structures on the Google-Street view dataset [GBC16, page 199].

References

[DV16] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning." In: *arXiv preprint arXiv:1603.07285* (2016).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Literature ii

[LeC+89] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. "Handwritten digit recognition with a back-propagation network." In: *Advances in neural information processing systems* 2 (1989).

[Stu20] David Stutz. *illustrating-convolutional-neural-networks*. `https://davidstutz.de/illustrating-convolutional-neural-networks-in-latex-with-tikz/`. Accessed: 2022-09-23. 2020.

Network coding

How to batch data

```
sample_no = data_train.shape[0]
img_batches = np.split(
    data_train,
    sample_no // batch_size,
    axis=0
)
label_batches = np.split(
    lbl_data_train,
    sample_no // batch_size,
    axis=0
)
```

In the listing above // denotes integer- or floor-division in python.

Forward pass and Gradients

```
def forward_pass(
    weights: FrozenDict,
    img_batch: jnp.ndarray,
    label_batch: jnp.ndarray
) -> jnp.ndarray:
    out = net.apply(weights,
        jnp.expand_dims(img_batch,
            -1))
    ce_loss = cross_entropy(
        label_batch,
        net_out)
    return ce_loss
```

Encode labels via `nn.one_hot`. Use `jax.value_and_grad` to obtain a function which allows gradient computations.

Tree-Map and gradient descent

To understand what the optimizers implemented in optax are doing. It's useful to use a lambda function here.

```
update_fun = \
    lambda w, g: w - lr * g
weights = jax.tree_util.tree_map(
    update_fun,
    weights, grads
)
```

More information on lambda functions:
https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions .
More information on the tree maps:
https://jax.readthedocs.io/en/latest/_autosummary/jax.tree_util.tree_map.html