Advanced Neural Network Architectures

CS 530
Chapman

Spring 2021

1

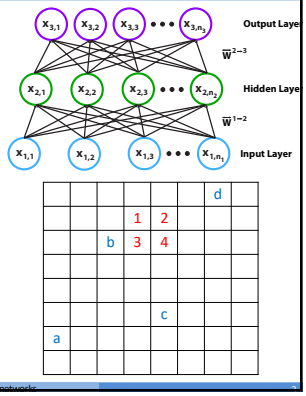
CONVOLUTION NEURAL NETWORKS (CONVNETS OR CNNS)

2

Neural networks for image analysis

Example: We want a neural network to classify 28x28 grayscale images of handwritten digits (MNIST dataset).

One option is to use our usual fully connected layers. We could have 28x28=784 input neurons, 100 neurons in the hidden layer, and 10 neurons in the output layer. So, all in all, 784+100+10 = 894 neurons and 784x100+100x10 = 79400 weights to update. The best attempts with this such a fully connected network approach result in ~5% error.



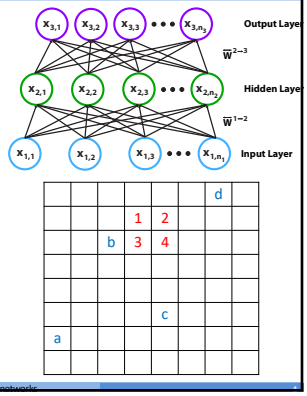
3

Neural networks for image analysis

Example: We want a neural network to classify 28x28 grayscale images of handwritten digits (MNIST dataset).

However, the 784x100=78400 all-to-all connections between the input and midden layer assumes that any collection of *n* pixels anywhere in the image contributes equally to image classification.

For example, on the right, it assumes that pixels (1,2,3,4) are as likely to contribute as (a,b,c,d). But nearby pixels are probably better joint features. So, we seem to be training a lot of weights for no reason.



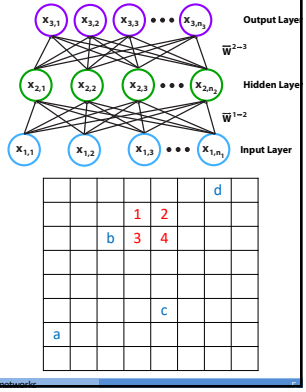
4

Neural networks for image analysis

Instead of looking for feature (pixel) combinations anywhere in the image, we could learn from the early human visual system. It possesses retinotopic representations that search for local features in the image that falls on the retina.

Such a local search might find a feature along (1,2,3,4) but would not search the combination (a,b,c,d).

How do we do this?



5

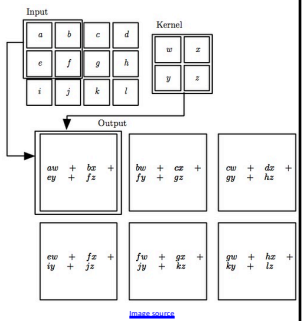
Convolution

Convolution is the mathematical operation

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\,w(t-a).$$

In machine learning, convolution typically refers to what is technically known as cross correlation (i.e., without kernel flipping),

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)\,K(m,n).$$



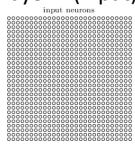
6

Receptive fields and filters in convnets

It is easier to co conceptualize the input layer, Layer 1, in convolutional neural networks as a grid rather than a vector. In this case, say, a 28x28 grid.

Layer 1 input neurons are connected to Layer 2 hidden neurons, but differently, and much more sparsely, than for a dense network. Each neuron in the Layer 2 is only connected to a 5x5 region in Layer 1 (instead of to all the neurons of Layer 1). That region of the input image is called that neuron's receptive field. And that hidden neuron learns to analyze that particular receptive field.

Layer 1 (input)



Layer 1 to 2

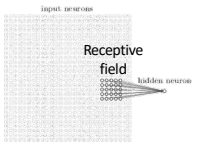


Image source

Neural networks

7

Receptive fields and filters in convnets

On the right, we can see the receptive fields of the first 2 neurons in Layer 2.

Note that if we have a 28x28 input image and 5x5 local receptive fields, there will be only 24x24=576 neurons in the hidden layer instead of 78,400.

Often receptive fields are moved with a stride length of 2 or more rather than 1. This results in less neurons in Layer 2 and less connections to it.

$$x_{2,j} = f\left((\vec{x}^1 * \vec{w}_{5\times5}) + b\right).$$

Receptive fields of first 2 neurons

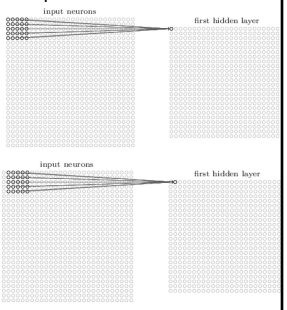


Image source

Neural networks

8

Receptive fields and filters in convnets

Importantly, convolutional networks use the same weights (and biases) across all the receptive fields in Layer 1. This is also termed *weight tying* or *weight sharing*.

These weights, also termed filters, are the kernel that the network learns to carry out in Layer 2. As all the weights are the same, all Layer 2 neurons detect the same feature over Layer 1, though at different locations.

Therefore, if the network learns that some edge detection is beneficial on Layer 1, it will carry it out across the entire image. This results in translational invariance for features in the image.

Receptive fields of first 2 neurons

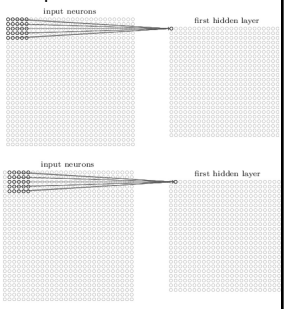


Image source

Neural networks

9

Receptive fields and filters in convnets

We need more than one feature to recognize images. So typical networks have several feature maps (a few, some tens of maps, or more). Here we illustrate 3.

Each of these feature maps typically learns to extract a different feature from the image.

Weight sharing reduced our weights from (28x28) x 100 = 78400 to 3 x (24x24) = 1728 weights, so 45-fold.

It seems that translational invariance dramatically reduced the number of parameters. What about the network's performance?

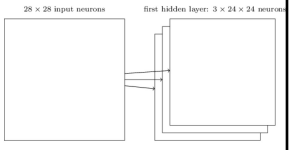


Image source

Neural networks

10

What do the filters detect?

If the filter (i.e., the learned value of the weights) is a good match for the part of the image (Layer 1) that it is scanning at that moment, it will result in a large number in Layer 2.

If the filter is a bad match for the scanned part of the image, it will result in a small number in Layer 2.

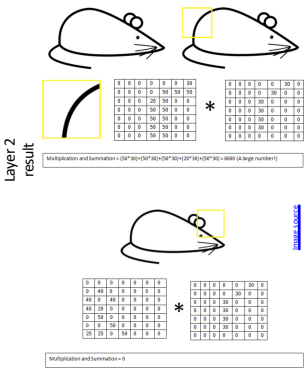


Image source

Neural networks

11

A bit more about filters

This filter is simple (for simple images). Actual filters are much more complex.

Also, many more-complex filters are typically learned to analyze an image dataset, not just one. And the filters are often 3D (RGB).

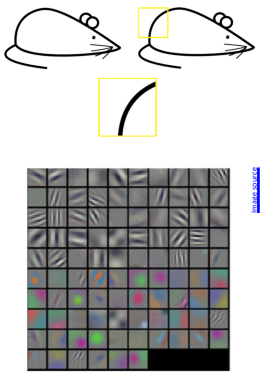


Image source

Neural networks

12

Pooling layers

After convolutional connections, these networks often have *pooling layers*. These condense the feature maps. For example, every 2x2 region of Layer 2 might be mapped to Layer 3, with a stride of 2 pixels. The function for the mapping is often the maximum among the pixels, termed *max-pooling*.

This reduces each 24x24 feature map down to at least 12x12 (for 2x2 pooling)—dimensionality reduction.

Some swap pooling layers for larger strides on the convolutional layers.

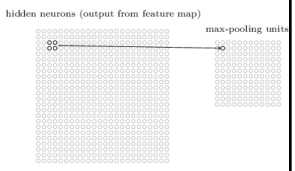


Image source

13

Pooling layers

Max pooling parameters are its filter size and stride size.

The larger the filter size, the larger the invariance it creates, but also the more "blurred" is the image.

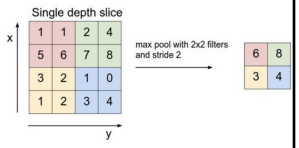


Image source

14

Putting it all together

We could then put all the layers together, with a 10-neuron, fully connected output layer as Layer 4.

Or we could add another convolution-pooling pair of layers as Layers 4 and 5. This would result in more condensed, higher-level spatial structures abstracted on top of the Layers 1/2 features.

Some small modifications are required in the backprop algorithm for convolutional networks because they are not fully connected.

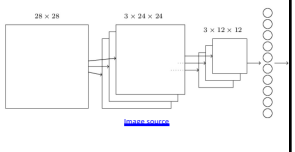


Image source

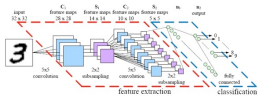


Image source

15

Why do convolution nets work?

Convolution leverages 3 important ideas that are known to improve machine learning:

1. Sparse interactions: due to small the kernels with respect to image size.
2. Parameter sharing: we learn the same kernel for every position of the input image.
3. Invariant representation: this convolution makes the image invariant to translation, which is a desirable property for object recognition.

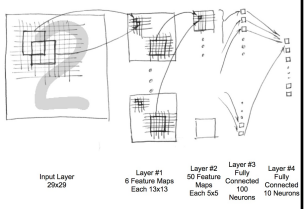


Image source

16

Why do convolution nets work?

Sparse connectivity

Small kernels with respect to the image size. Not all pixels in the input are connected to all pixels in the output—limited receptive field

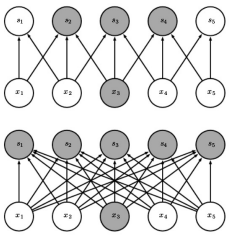


Image source

17

Why do convolution nets work?

Sparse connectivity

Small kernels with respect to the image size. Not all pixels in the input are connected to all pixels in the output.

Note that—through the deep architecture—the receptive fields of the output neurons eventually tend to extend to many, or all, of the input neurons

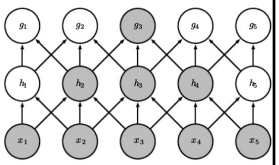


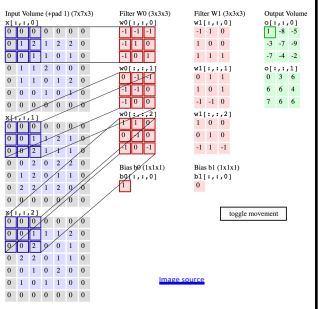
Image source

18

Why do convolution nets work?

Parameter sharing

We learn the same kernels (or filters) for every position of the input image

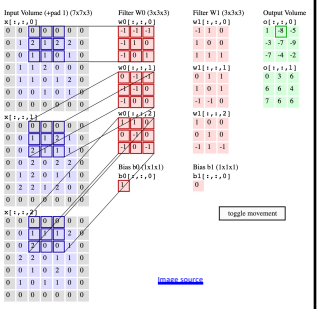


19

Why do convolution nets work?

Parameter sharing

We learn the same kernels (or filters) for every position of the input image

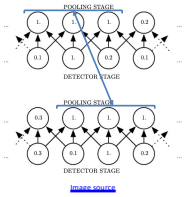


20

Why do convolution nets work?

Invariant representation

The convolution makes the network equivariant to translation—i.e., shifting an image and then applying convolution is the same as applying convolution and then shifting the same way.



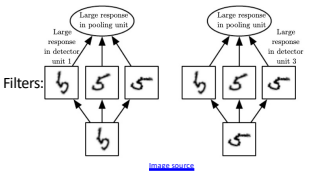
21

Why do convolution nets work?

Invariant representation

The convolution makes the network equivariant to translation—i.e., shifting an image and then applying convolution is the same as applying convolution and then shifting the same way.

Max-pooling gives us further robustness to some other small affine transformations.

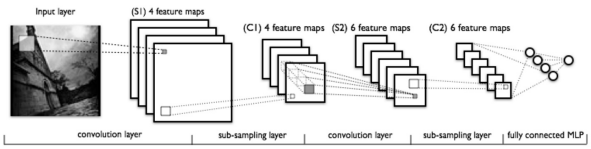


22

A full conv net

Left to right:

- Input image, scanned by light rectangle is the filter that passes over it
- Stacked activation maps for each filter employed. Larger rectangle is one patch to be down-sampled
- Max pooling (not always used)
- Re-filtering over first down-sampled stack, create 2nd activation maps
- Second down-sampling, further condensing 2nd activation maps
- Fully-connected layer that classifies output with one label per node.



23

How much do conv-nets save?

Example

- Image is 320x280
- Filter: edge detection (each pixel subtracts value of that on its left)
- Conv-net: 319 x 280 x 3 = 297,960 float operations
- Fully connected: 320 x 280 x 319 x 280 = 8,003,072,000
- 26,860 times less floating-point operations

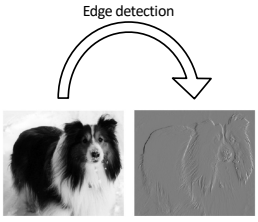


Image source

24

Advantages of convnets

Advantages of convnets over fully connected networks for image processing:

1. Sparse interactions and parameter sharing results in much less parameters to train. So, more accuracy can be achieved on a dataset with the same number of samples (images)
2. Invariant representation makes the image invariant to translation, desirable for object recognition.

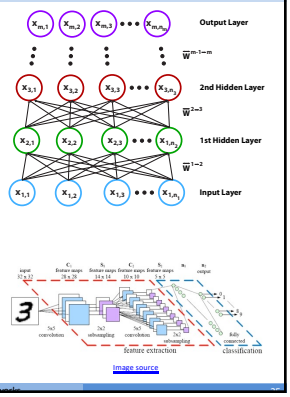
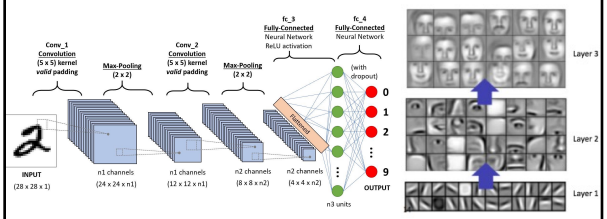


Image source

25

CNN on MNIST

- Can you see why CNNs result in better accuracy than fully connected neural networks?



26

A neural-network architecture for unsupervised learning

AUTOENCODERS

27

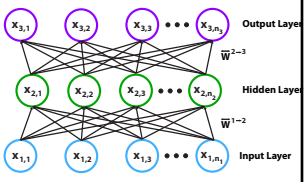
PCA using neural networks

We create an "hourglass network", where $n_1=n_3$ and $n_2<<n_1$. We set it up with purely linear neurons. And we set it to learn

$$\vec{W}^{1\to 2} = \text{argmin}\left(\left(\vec{x}^3 - \vec{x}^1\right)^2\right).$$

We will get a network that learns to perform PCA.

If the neurons are not linear, we could learn *non-linear PCA* or *kernel PCA*.



28

Autoencoders

Autoencoders are a generalization of the PCA concept in neural networks. They carry out unsupervised learning of a lower-dimensional-space representation of the input.

This is a deeper network (9 layers instead of 3). The larger number of layers help the network learn more complex kernels in the kernel PCA sense.

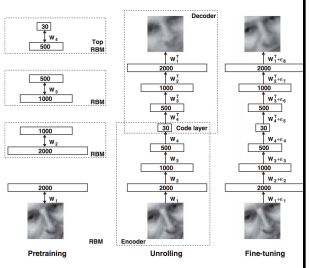


Image source

29

Autoencoders

Autoencoders are trained using backprop with a common cost function being

$$(\vec{x}^1 - \vec{x}^9)^2.$$

So, the weights satisfy

$$\vec{W}^{1\to 2}, \cdots, \vec{W}^{4\to 5} = \text{argmin}\left(\left(\vec{x}^1 - \vec{x}^9\right)^2\right).$$

Note that we do not require labels to train this network. So, this is an example of unsupervised learning in neural networks.

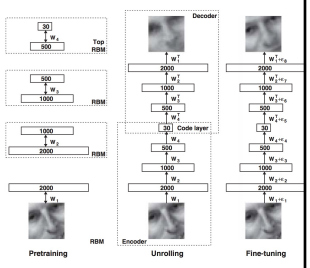


Image source

30

What do encoders learn?

- We want our encoder to learn some important features of the inputs to be able to decode it in the code layer, which is much narrower than the input and output layers
- If we make the network too powerful, and especially the code layer too wide, the network will tend to overfit and not be able to well represent from outside its training set

31