

Convolutional Neural Networks: Introduction

Lecture 24

by Marina Barsky

Problem with number of layers and number of neurons

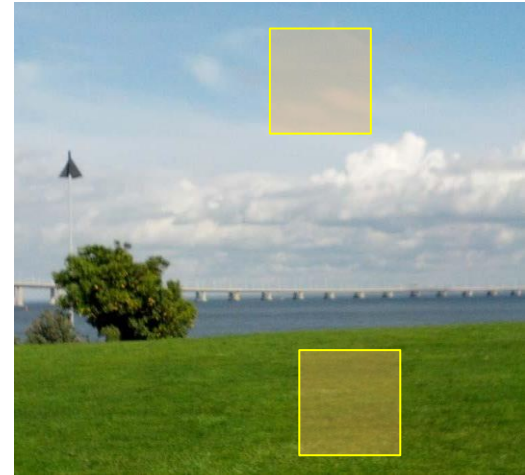
- The number of parameters in NN grows very fast as we add more layers
- If a single input vector contains n features, and we create another layer of size m - we add $n*m$ additional weights edges
For $n=1000$ and $m=1000$ we need to learn 1,000,000 additional weights
- Optimizing such big models is computationally expensive
- When our training examples are images, the input is already very high-dimensional
- If we use MLP with more than 3 layers - the optimization problem quickly becomes intractable

Convolutional Neural Network (CNN)

- A *Convolutional Neural Network* (CNN) is a special kind of Feed Forward NN that significantly reduces the number of parameters in a deep neural network
- CNNs are used in **image** and **text recognition** where they beat many previously established benchmarks

Recognizing local regions of the image

- Pixels that are close to each other usually represent the same type of information: sky, water, leaves, fur, bricks... This information can likely be combined into a smaller number of features
- The parts of an image where two different types of information “touch” one another - represent a shape
- The idea is to train the neural network to recognize regions of the same information as well as the edges, and use these learned features as new less-dimensional vectors for classifying images



Detecting image patterns

- We can split the image into *square patches* using a sliding frame approach
- We can then train multiple smaller models at once, each small model receiving a square patch as input
- The goal of each small model is to learn to detect a specific kind of pattern in the input patch
- For example, one small model will learn to detect the sky, another one will detect the grass, the third one will detect edges of a building...

The “filter” idea

- If we train a network using a set of labeled images (say, set of cats) the model will learn a set of local patterns which are most common to all cats
- We call these local patterns *filters*
- The CNN learns the filter shapes on its own during training - all you need to decide is the size of each filter, and the network will learn what each filter should look like
- Later, during classification, it will apply each filter to a new image and compute the output - image class

Comparing input patch to a filter

- Once you decide on the size of each filter, the regular training will produce the optimal values for each filter matrix
- Let's assume - for simplicity - that the input image is black and white, with 1 representing black and 0 representing white pixels
- Assume that our patches are 3 by 3 pixels ($p = 3$). Some patch could then look like matrix P (for "patch")
- Let's say we want to detect a pattern called "cross" in the image data. We initialize the matrix F (filter) with some random values

Patch in a black-and-white image

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Filter initialized

$$F = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$

Learning matrix F

- The small regression model that will detect “cross” patterns (and only them) would need to learn a 3 by 3 parameter matrix F where parameters at positions corresponding to the 1s in the input patch would be positive numbers, while the parameters in positions corresponding to 0s would be close to zero

Patch in a black-and-white image

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Filter: to be learned from data

$$F = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$

Learning matrix F

- The learning proceeds by calculating the dot-product between matrices P and F and then summing up all values from the resulting vector
- The higher the value, the more similar F is to P:

$$P \cdot F = [0 \cdot 0 + 2 \cdot 1 + 0 \cdot 0, 2 \cdot 1 + 4 \cdot 1 + 3 \cdot 1, 3 \cdot 0 + 1 \cdot 1 + 0 \cdot 1] = [2, 9, 1]$$

The sum is $2 + 9 + 1 = 12$

- This operation — **the dot product between a patch and a filter and then summing the values** — is called *convolution*

Patch in a black-and-white image

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Filter: randomly initialized

$$F = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$

Optimizing (matching) filter to a patch

- If our input patch P had a different pattern - then the convolution would give a lower result: $0 + 9 + 0 = 9$
- The more the patch “looks” like the filter, the higher the value of the convolution is
- There’s also a bias parameter b associated with each filter F which is added to the result of a convolution before applying the nonlinearity

Patch in a black-and-white image

P=

1	1	1
0	1	0
0	1	0

Filter: randomly initialized

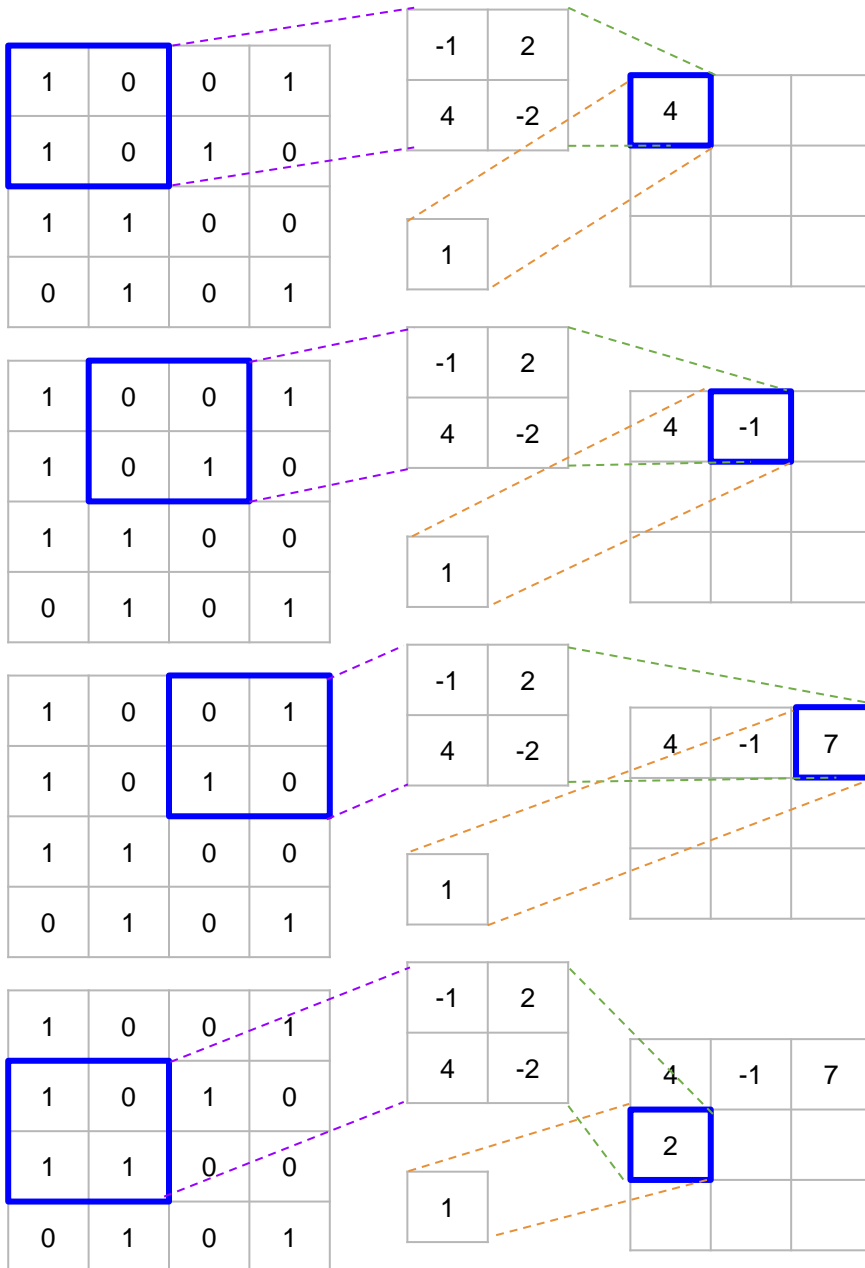
F=

0	2	3
2	4	1
0	3	0

Computing convolution for the entire image

- One hidden layer of a CNN consists of multiple convolution filters (each with its own bias parameter), just like one layer in a vanilla NN consists of multiple units
- Each filter of the first (leftmost) hidden layer slides — or *convolves* — across the input image, left to right, top to bottom, and convolution is computed for each sliding input frame
- So each neuron in a CNN layer is a filter which learns a single pattern. The number of such units generally is much smaller than the number of original pixels

Filter convolving across the image

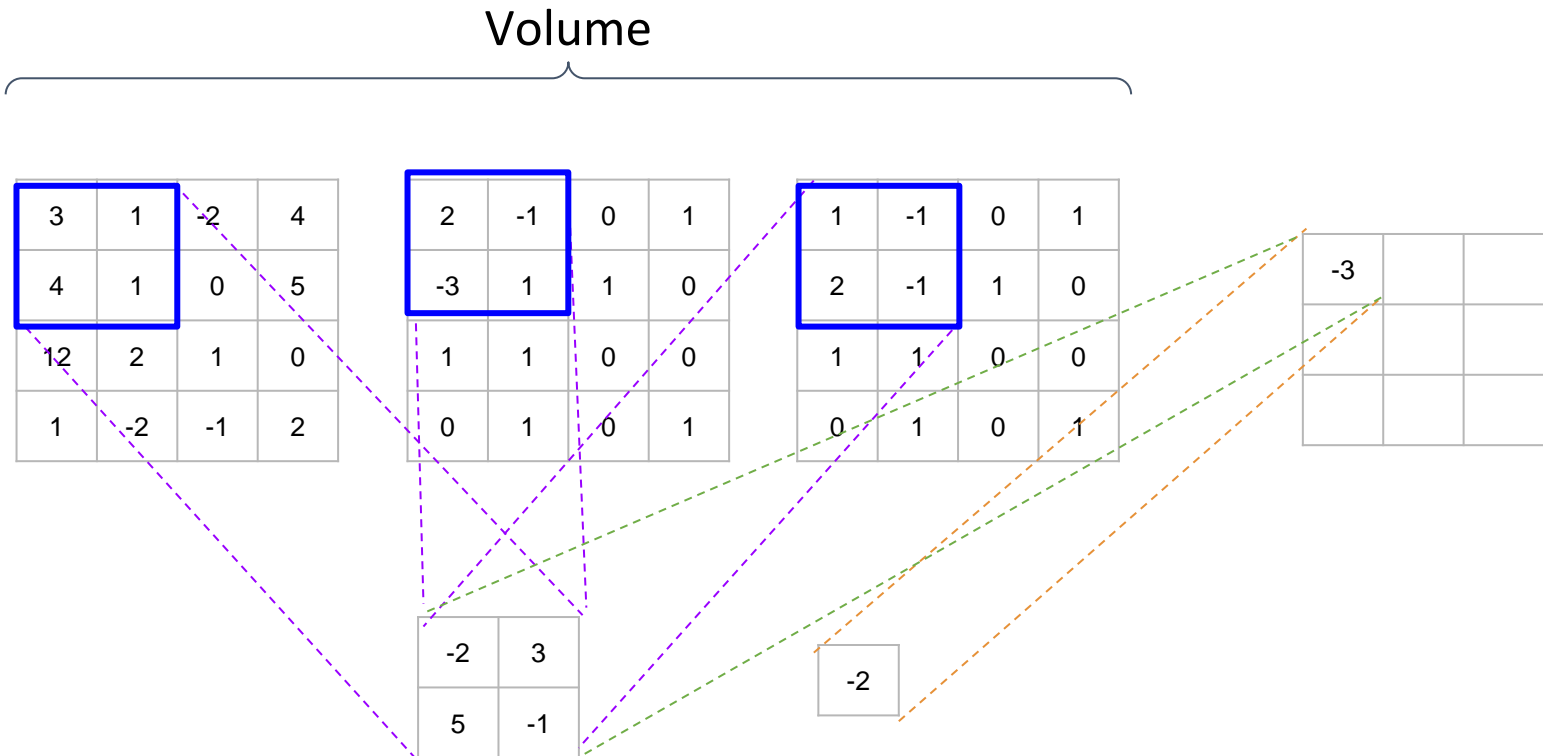


- As a result of this convolution, instead of a 4x4 image matrix we get a 3x3 matrix
- This matrix becomes an input to the next hidden layer
- A nonlinearity (ReLU) is applied to the [sum of the convolution plus the bias]
- The numbers for each filter matrix F and the value of the bias term b , are found by the gradient descent with backpropagation

Volume: collection of matrices

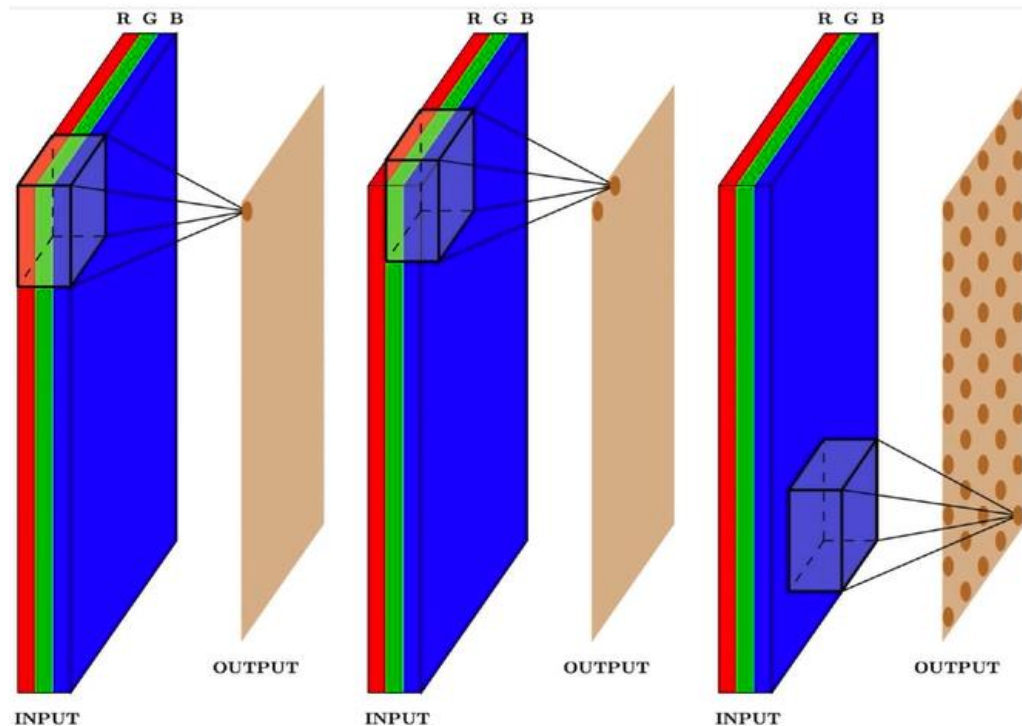
- Each layer with m filters produces m matrices of size $p \times p$ which serve as an input for the next hidden layer
- If the next layer is also a convolution layer, then layer $i + 1$ treats the output of the preceding layer i as a collection of m images
- Such a collection is called a *volume*. Each filter of layer $i + 1$ convolves the whole volume produced by filter i
- The convolution of a patch of a volume is simply the sum of convolutions of the corresponding patches of individual matrices in this volume

The volume convolves as a single input



$$(-2 \cdot 3 + 3 \cdot 1 + 5 \cdot 4 + -1 \cdot 1) + (-2 \cdot 2 + 3 \cdot (-1) + 5 \cdot (-3) + -1 \cdot 1) + (-2 \cdot 1 + 3 \cdot (-1) + 5 \cdot 2 + -1 \cdot (-1)) + (-2) = -3$$

Input image as a volume of 3 color channels



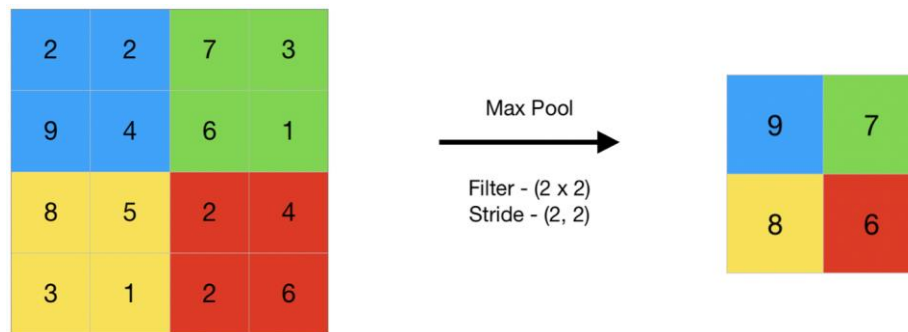
CNNs also often get volumes as input, since an image is usually represented by three channels: R, G, and B, each channel being a monochrome picture

CNN: parameters

This is just a very high-level picture of the CNN architecture

Other essential features include strides, padding, and pooling

- Strides and padding are hyperparameters of the convolution filter and the sliding window
- Pooling example: max pooling



All this is designed to reduce the number of parameters of a CNN even more

Learn more:

https://e2eml.school/how_convolutional_neural_networks_work.html

<https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/>

<https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>

Neural Nets: generalization and overfitting

The greatest challenge you will face with deep learning is convincing your neural network to **generalize instead of just memorize**

Exactly as before you will detect the overfitting if the difference in accuracy between train and test set is significant

- Same regularization as before: L1 and L2 norm
- **NN-specific regularization:**
 - Early stopping
 - Dropout

Early stopping

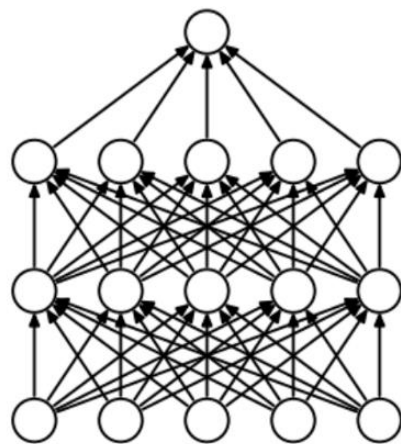
- Stop training the network when it starts getting worse
- Neural networks, even though they are randomly generated, start by learning the biggest, most broad sweeping features before learning too much about the noise
- Most of the signal is found in the more general characteristics (for images - big shapes and color)
- We don't let the network train long enough to learn the details of the training set

How do we know when to stop?

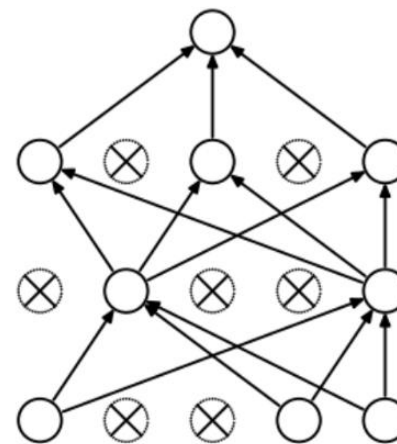
The only real way to know is to run the model on validation data and stop when the error rate on a validation set starts increasing

Industry Standard Regularization: Dropout

- Randomly turn neurons off (setting to 0) during training
- At each training epoch, individual nodes are either dropped out of the net with probability $1-p$ or kept with probability p . Incoming and outgoing edges to a dropped-out node are also removed
- This forces train random subnetworks



(a) Standard Neural Net



(b) After applying dropout.

Millions of random subnetworks are probably telling the truth

- What happens when you take a big neural network and only use a small part of it? It behaves like a small neural network!
- However, when we do this randomly over potentially millions of different "sub-networks", the sum total of the entire network still maintains the expressive power of the complex NN!
- Dropout is a form of training a bunch of small networks and averaging the result

Different subnetworks learn different noise

- NN always start out randomly and learn by trial and error
- This means that every neural network learns just a little bit differently: no two neural networks ever produce exactly the same weights
- It means that when you overfit two neural networks, no two neural networks overfit in exactly the same way
- Each neural network starts by predicting randomly, then adjusting its weights to make better predictions, and each network inevitably makes different mistakes, resulting in different updates

Averaging learning of subnetworks: to detect main signal

- While it is very likely for a single unregularized neural network to overfit to noise, each **overfits to a different noise**
- If you train 100 neural networks (all initialized randomly), they will each tend to latch onto **different noise** but **similar broad signal**
- This means that if we allowed them to vote equally, their noise would tend to cancel out, revealing only what they have all learned in common - the important signal

Regularization for deep learning implemented in *keras*

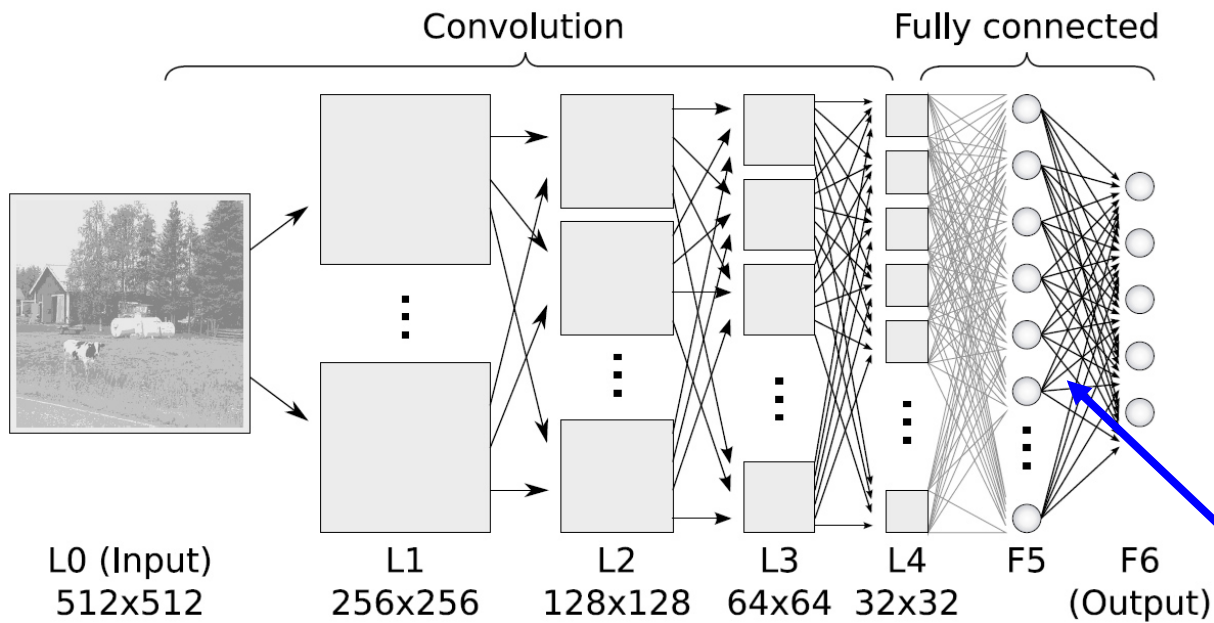
L1 L2 regularization: [link](#)

Early stopping: [link](#)

Dropout: [link](#)

Experiment with Convolutional Neural Networks in this demo:

https://github.com/mgbarsky/labs_ml_img_classification/blob/main/cats_dogs_CNN.ipynb



What is going on inside?

What does the network learn about images?

How and what does it learn?

The exact way neural networks see and interpret the world remains a black box

We want to better understand of how exactly they recognize specific patterns or objects in order to:

- improve the quality of NN learning
- solve legal problems since in many cases the outputs have to be interpretable by humans

How CNN sees images after learning

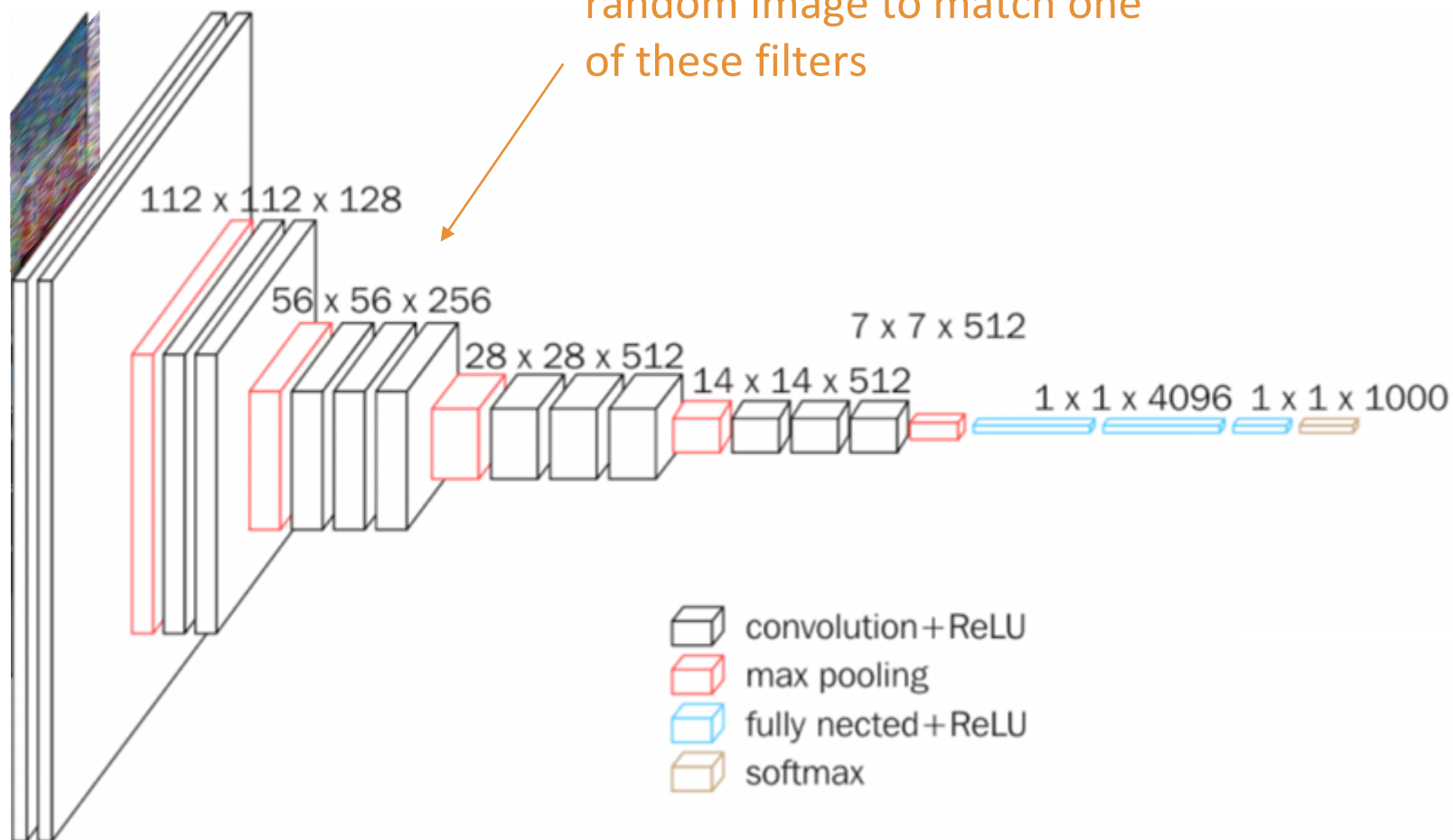
- Neural networks learn to transform images into successive layers of increasingly meaningful and complex representations (filters)
- We can think of a deep network as a “multistage information-distillation operation, where information goes through successive filters and comes out increasingly purified”. (François Chollet, “Deep Learning with Python”)
- We can generate patterns that maximize the mean activation of a chosen feature map in a certain layer

Experiment: understanding learned patterns (filters)

- Get a network which already learned to recognize thousands of image types
 - Pre-trained model is available from *torchvision.models*
 - Data is from <http://www.image-net.org/>
 - The network is VGG-16
- The goal is NOT to train the model, but use it in an evaluation mode
- Then show an image with random pixels to the model and optimize the pixel values to best match each filter at each hidden level (Erhan, D. et al. “Visualizing Higher-Layer Features of a Deep Network”, 2009).

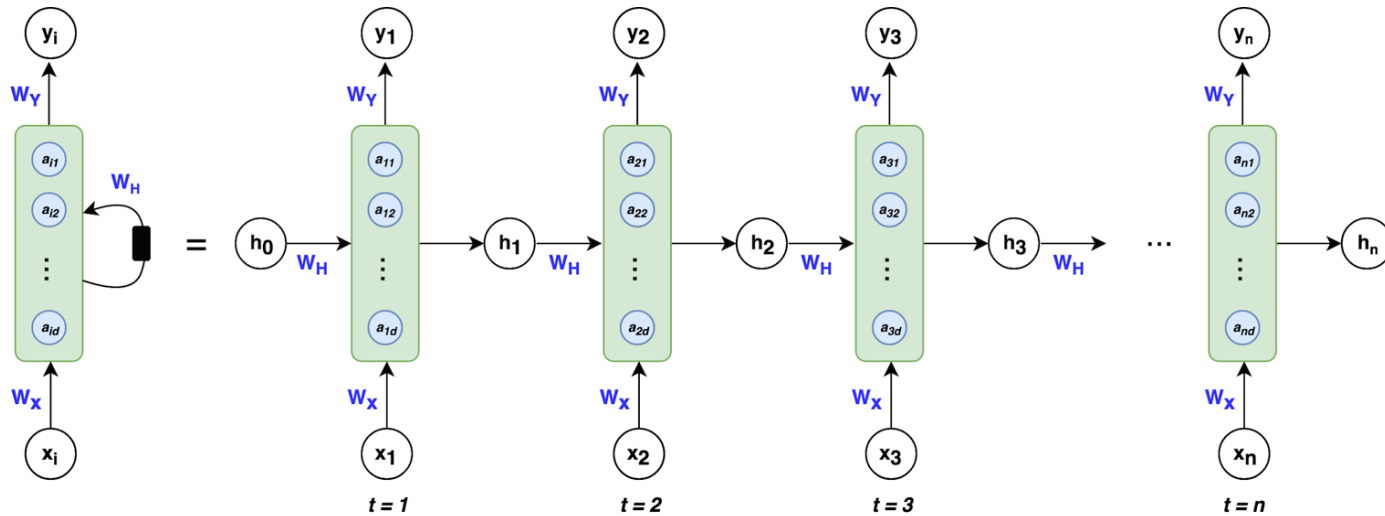
VGG-16

The idea: optimize pixels in a random image to match one of these filters



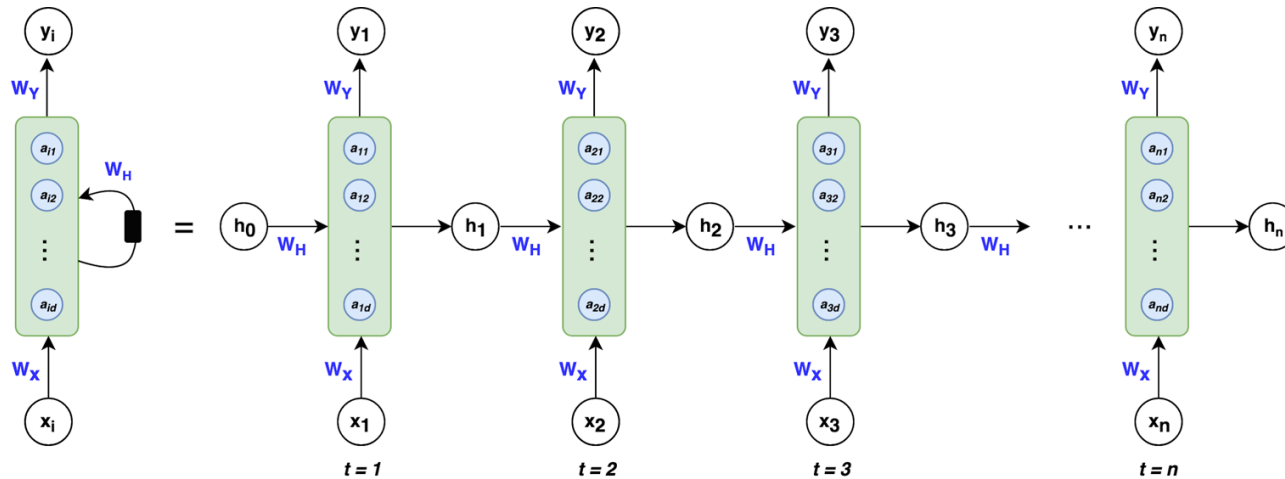
Here is a [blog](#) and the [code](#)

Recurrent neural networks



Used to label, classify, or generate **sequences**

- **Labeling** a sequence means predicting a class to each feature vector in a sequence
- **Classifying** a sequence means predicting a class for the entire sequence
- **Generating** a sequence means to output another sequence (of a possibly different length) somehow relevant to the input sequence



Recurrent neural networks

These are generally discussed in the Algorithms on Strings or possibly in the Natural Language Processing courses