Neural Networks for Visual Recognition

Gül Varol

IMAGINE team, École des Ponts ParisTech

gul.varol@enpc.fr

http://imagine.enpc.fr/~varolg/

@RecVis, 31.10.2023

Announcements

Assignment 2 due Tuesday Nov 14
http://imagine.enpc.fr/~varolg/teaching/recvis23/
Neural Networks

Last week: Introduction to neural networks
(A. Joulin)

This week: Neural networks for visual recognition
(G. Varol)

Next week: Beyond classification: Object detection, Segmentation, Human pose estimation
(G. Varol)
First words that come to your mind when hearing “neural networks for visual recognition”?

slido.com #2557 457
Disclaimer: Terminology

- Deep learning
  - Neural networks?
  - Artificial neural networks?
  - Multilayer neural networks?
  - ...
  - ...
This lecture

Computer Vision

"Deep" Learning

Machine Learning

"AI"

Slide credit: Justin Johnson
This lecture

“AI”

Computer Vision

Machine Learning

“Deep” Learning

NLP

Slide credit: Justin Johnson
Definitions

“AI”  
Any technique that enables **computers** to **mimic human** behavior

Machine Learning  
Ability to learn **without** explicitly being **programmed**

“Deep” Learning  
Extract patterns from data using **neural networks**

Computer Vision  
Extracting meaning from **visual** signals

NLP  
Extracting meaning from **textual** signals
Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification
Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification
Recap: Image recognition so far

*Instance-level recognition*
Category Recognition

- Image classification: assigning a class label to the image

Car: present
Cow: present
Bike: not present
Horse: not present
...

Slide: C. Schmid
Category Recognition

- Image classification: assigning a class label to the image

- Object localization: define the location and the category
Difficulties: within-class variations
Why machine learning?

• Early approaches: simple features + handcrafted models
• Can handle only few images, simple tasks


**ABSTRACT**

In order to make it possible for a computer to construct and display a three-dimensional array of solid objects from a single two-dimensional photograph, the rules and assumptions of depth perception have been carefully analyzed and mechanized. It is assumed that a photograph is a perspective projection of a set of objects which can be constructed from transformations of known three-dimensional models, and that the objects are supported by other visible objects or by a ground plane. These assumptions enable a computer to obtain a reasonable, three-dimensional description from the edge information in a photograph by means of a topological, mathematical process.

A computer program has been written which can process a photograph into a line drawing, transform the line drawing into a three-dimensional representation, and finally, display the three-dimensional structure with all the hidden lines removed, from any point of view. The 2-D to 3-D construction and 3-D to 2-D display processes are sufficiently general to handle most collections of planar-surfaced objects and provide a valuable starting point for future investigation of computer-aided three-dimensional systems.
Why machine learning?

- Early approaches: manual programming of rules
- Tedious, limited, and does not take data into account

Figure 3. A system developed in 1978 by Ohta, Kanade and Sakai \[33, 32\] for knowledge-based interpretation of outdoor natural scenes. The system is able to label an image (c) into semantic classes: S-sky, T-tree, R-road, B-building, U-unknown.

Why machine learning?

- Today lots of data, complex tasks

  Internet images, personal photo albums

  Movies, news, sports

- Instead of trying to encode rules directly, learn them from examples of inputs and desired outputs
Texture Classification

- Profound observation: Grass and sea pictures don’t look the same!

- Basic idea: Model the distribution of “texture” over the image (or over a region) and classify in different classes based on the texture models learned from training examples.
Image categorization

- Profound observation: Cows and buildings don’t look the same!

- Basic idea: Model the distribution of “texture” over the image (or over a region) and classify in different classes based on the texture models learned from training examples.
Bag-of-features for image classification

Origin: texture recognition

• Texture is characterized by the repetition of basic elements or textons

Bag-of-features for image classification

Universal texton dictionary

histogram
Political observers say that the government of Zorgia does not control the political situation. The government will not hold elections …

Analogy: Text fragment ↔ Image region
Word ↔ Texton

« Bag of words »
The ZH-20 unit is a 200Gigahertz processor with 2Gigabyte memory. Its strength is its bus and high-speed memory……
Bag-of-features for image classification

1. Extract regions
2. Compute descriptors
3. Find clusters and frequencies
4. Compute distance matrix
5. Classification

[Csurka et al. WS’2004], [Nowak et al. ECCV’06], [Zhang et al. IJCV’07]
Bag-of-features for image classification

**Step 1**
- Extract regions
- Compute descriptors

**Step 2**
- Find clusters and frequencies

**Step 3**
- Compute distance matrix
- Classification

SVM
Step 1: feature extraction

Sparse sampling
• SIFT as interest point detector

Dense sampling
• Interest points do not necessarily capture “all” features
Step 1: feature extraction

Sparse sampling
• SIFT as interest point detector

Dense sampling
• Interest points do not necessarily capture “all” features
• Spatial pyramid (Lazebnik, Schmid & Ponce, CVPR 2006)
Bag-of-features for image classification

Step 1
- Extract regions

Step 2
- Compute descriptors
- Find clusters and frequencies
- Compute distance matrix

Step 3
- Classification

SVM
Step 2: Quantization

Cluster descriptors
• K-means
• Gaussian mixture model

Assign each visual word to a cluster
• Hard or soft assignment

Build frequency histogram
Examples for visual words

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airplanes</td>
<td><img src="image1.png" alt="Airplanes Examples" /></td>
</tr>
<tr>
<td>Motorbikes</td>
<td><img src="image2.png" alt="Motorbikes Examples" /></td>
</tr>
<tr>
<td>Faces</td>
<td><img src="image3.png" alt="Faces Examples" /></td>
</tr>
<tr>
<td>Wild Cats</td>
<td><img src="image4.png" alt="Wild Cats Examples" /></td>
</tr>
<tr>
<td>Leaves</td>
<td><img src="image5.png" alt="Leaves Examples" /></td>
</tr>
<tr>
<td>People</td>
<td><img src="image6.png" alt="People Examples" /></td>
</tr>
<tr>
<td>Bikes</td>
<td><img src="image7.png" alt="Bikes Examples" /></td>
</tr>
</tbody>
</table>
Image representation

- Each image is represented by an aggregated histogram vector, typically 1000-4000 dimensional
- Normalized with L2 norm
- Fisher Vectors [Perronnin et al. ECCV’10]: improvements over Bag of Features
Bag-of-features for image classification

Step 1: Extract regions
Step 2: Compute descriptors and find clusters and frequencies
Step 3: Compute distance matrix, classification matrix and SVM
Step 3: Classification

Training data: Vectors are histograms, one from each image

Train classifier, e.g. SVM
Step 3: Classification

Learn a decision rule (classifier) assigning bag-of-features representations of images to different classes.
Traditional Recognition Approach

Image/Video Pixels $\rightarrow$ Hand-designed feature extraction $\rightarrow$ Trainable classifier $\rightarrow$ Object Class
Traditional Recognition Example

- SIFT features
- BOF: Bag of Features / Visual Words (inspired by Bag of Words in NLP)
- SVM: Support Vector Machines for classification
Analogy to the traditional visual recognition pipeline

- Features are not learned (e.g., HOG, SIFT, Bag of Features)
- Trainable classifier is often generic (e.g., SVM, Random Forest)
Analogy to the traditional visual recognition pipeline

What about learning the features?

- Features are learned “end-to-end” (i.e., pixels are input)
- “Feature hierarchy” all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly
Analogy to the traditional visual recognition pipeline

- Features are learned “end-to-end” (i.e., pixels are input)
- “Feature hierarchy” all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly
“Shallow” vs. “deep” models

Traditional recognition: “Shallow” architecture

Image/Video Pixels → Hand-designed feature extraction → Trainable classifier → Object Class

Deep learning: “Deep” architecture

Image/Video Pixels → Layer 1 → … → Layer N → Simple classifier → Object Class
Agenda

1. Recap: Bag of Visual Words, Analogy with NNs

2. Neural networks (NNs) for computer vision:
   - Applications
   - A brief history: from perceptron to MLPs to CNNs

3. Convolutional neural networks (CNNs)
   - Standard layers
   - Recap: Training NNs
   - Visualizing CNNs
   - Pretraining & finetuning NNs
   - Typical CNN architectures

4. Beyond CNNs
   - Attention & Transformer
   - Vision Transformers

5. Beyond classification
Neural Networks in Production
Face detection

Slide credit: Kosta Derpanis
Self-driving cars / Autonomous vehicles

“We’ve built an AV that is seamlessly integrating into traffic in Munich, Paris, Detroit, Jerusalem, New York, Tokyo, and other cities across the globe.”
Shopping

Slide credit: Kosta Derpanis
this course is so interesting

ce cours est tellement intéressant
What is “Deep” Learning?
Recap: Basics of supervised learning

• $n$ training data pairs

• Learn a predictor/decision function

• By minimizing

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$$

$$\hat{f} : \mathcal{X} \rightarrow \mathcal{A}$$

$$\sum_{i=1}^{n} l(f(x_i), y_i)$$
Recap: Basics of supervised learning

- $n$ training data pairs
- Learn a predictor/decision function
- By minimizing

\[
\hat{f} : \mathcal{X} \rightarrow \mathcal{A}
\]

\[
\sum_{i=1}^{n} l(f(x_i), y_i)
\]

Loss | Model | Input | Label
Deep learning

\[ \sum_{i=1}^{n} l(f(x_i), y_i) \]

Loss \rightarrow Model \rightarrow Input \rightarrow Label

Deep learning: Model = neural network
What is a “deep” neural network?

Stacking more than one layer
What is a layer?

Typically matrix multiplication! (But the function can take many forms*)

- **Fully-connected** layer
- **Convolution** layer
- **Pooling** layer (e.g., Max-pooling)
- **Non-linearity** layer (e.g., ReLU)
- **Attention** layer
- ...

*requirement to be differentiable if optimized with gradient descent algorithm variants
Recap: Perceptrons

Most basic form of a neural network

Sigmoid function:

\[
\sigma(t) = \frac{1}{1 + e^{-t}}
\]

Input

Weights

\( x_1 \)
\( w_1 \)

\( x_2 \)
\( w_2 \)

\( x_3 \)
\( w_3 \)

\( \ldots \)
\( \ldots \)

\( x_d \)
\( w_d \)

Output: \( \sigma(w \cdot x + b) \)

Bias

Non-linearity

Linear combination of inputs
Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) — The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo — the Weather Bureau’s $2,000,000 “704” computer — learned to differentiate between right and left after fifty attempts in the Navy’s demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism “capable of receiving, recognizing and identifying its surroundings without any human training or control.”

The “brain” is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today’s demonstration, the “704” was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Lears by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a “Q” for the left squares and “O” for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a “self-induced change in the wiring diagram.”

The first Perceptron will have about 1,000 electronic “association cells” receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.
Recap: Multi-Layer Perceptron (MLP)

Linear regression:

Perceptron:

MLP:
Images are numbers

An image is just a matrix of numbers $[0, 255]$!
i.e., $1080 \times 1080 \times 3$ for an RGB image

Slide credit: Alexander Amini
Review: Convolutional Neural Networks (CNN)

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end

Progress on ImageNet

![ILSVRC Top 5 Error on ImageNet](https://www.dsiac.org/resources/journals/dsiac/winter-2017/CNN_AlexNet.png)

Hand-crafted features

Deep Learning

Human

CNN (AlexNet)

https://www.dsiac.org
CNNs were not invented overnight

1998
LeCun et al.

10^6

2012
Krizhevsky et al.

10^9

10^14

# of transistors

# of pixels used in training

10^7

Large Scale Visual Recognition Challenge

Year 2010
NEC-UIUC

Year 2012
SuperVision

Year 2014
GoogLeNet

Year 2015
MSRA

Dense descriptor grid:
HOG, LBP

Coding: local coordinate,
super-vector

Pooling, SVM

Linear SVM

Conv-64

Conv-64

Maxpool

Conv-128

Conv-128

Maxpool

Conv-256

Conv-256

Maxpool

Conv-512

Conv-512

Maxpool

Fc-4096

Fc-4096

Fc-1000

Softmax

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Figure credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Neural Networks date back decades.

1952  Stochastic Gradient Descent
1958  Perceptron
     • Learnable Weights
1986  Backpropagation
     • Multi-Layer Perceptron
1995  Deep Convolutional NN
     • Digit Recognition
2012  AlexNet

1. Big Data
   • Larger datasets
   • Easier collection & storage

2. Hardware
   • Graphics Processing Units (GPUs)
   • Massively Parallelizable

3. Software
   • Improved Techniques
   • New Models
   • Toolboxes

Slide credit: Alexander Amini
CVPR: (Computer Vision Pattern Recognition Conference)
Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification
Agenda

1. Recap: Bag of Visual Words, Analogy with NNs

2. Neural networks (NNs) for computer vision:
   - Applications
   - A brief history: from perceptron to MLPs to CNNs

3. Convolutional neural networks (CNNs)
   - Standard layers
   - Recap: Training NNs
   - Visualizing CNNs
   - Pretraining & finetuning NNs
   - Typical CNN architectures

4. Beyond CNNs
   - Attention & Transformer
   - Vision Transformers

5. Beyond classification
Standard layers

1. **Fully-connected** layer
2. **Convolution** layer
3. **Pooling** layer (e.g., Max-pooling)
4. **Non-linearity** layer (e.g., ReLU)
5. **Normalization** layer (e.g., BatchNorm)
   ...

- **Convolutional block**
- **Fully-connected block**

Learnable parameters

Input image

[224 x 224 x 3]

Conv1 Conv2 Conv3 Conv4 Conv5 FC6 FC7 FC8

[1000] probability

Class probability
1. Fully-connected layer

\[ h = Wx + b; \ h_i = \sum_j w_{ij} x_j + b_i \]

\[ h_5 = \sum_{j=1}^{5} w_{5j} x_j + b_5 \]
2. Convolution layer

- Layer with a special connectivity structure
- Dependencies are local
- Translation invariance

\[ h = Wx + b; \quad h_i = \sum_j w_{ij} x_j + b_i \]

\[ h_1 = \sum_{j=0}^{2} w_{j} x_{j+1} + b \]

\[ h_i = \sum_j w_j x_{j+i} + b \]
2. Convolution layer

Fully-connected

$$h = Wx + b; \ h_i = \sum_j w_{ij}x_j + b_i$$

1D Convolutional

$$h_i = \sum_j w_{ij}x_{j+i} + b$$
2. Convolution layer

**Fully-connected**

\[ h = Wx + b; \quad h_i = \sum_j w_{ij}x_j + b_i \]

**1D Convolutional**

\[ h_i = \sum_j w_{ij}x_{j+i} + b \]
2. Convolution layer

2D Convolutions

\[(I \ast K)(i, j) = \sum_m \sum_n I(m, n)K(i + m, j + n)\]
2. Convolution layer

2D Convolutions

\[(I \ast K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i + m, j + n)\]
2. Convolution layer

2D Convolutions

\[(I \ast K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i + m, j + n)\]
2. Convolution layer

2D Convolutions

\[(I \ast K)(i, j) = \sum_m \sum_n I(m, n)K(i + m, j + n)\]
2. Convolution layer

2D Convolutions

\[(I \ast K)(i, j) = \sum_m \sum_n I(m, n)K(i + m, j + n)\]

Slide credit: Naila Murray
2. Convolution layer

The data manipulated by a CNN has the form of 3D tensors. These are interpreted as discrete vector fields \( \mathbf{x} \), assigning a feature vector \( (x_{uv1}, \ldots, x_{uvC}) \) at each spatial location \( (v,u) \).

A colour image is a simple example of a vector field with 3D features (RGB):
2. Convolution layer

With a bank of 3D filters

\[
\begin{align*}
    y_{v'u'q'} &= b_{q'} + \sum_{\tilde{v}=1}^{\tilde{H}} \sum_{\tilde{u}=1}^{\tilde{W}} \sum_{c=1}^{C} x_{v'+\tilde{v}-1,u'+\tilde{u}-1,c} f_{\tilde{v}\tilde{u}cq'} \\
    y &= F \ast x + b
\end{align*}
\]

Linear convolution applies a bank of linear filters \( F \) to the input tensor \( x \).

- **Input tensor** \( x = H \times W \times C \) array
- **Filter bank** \( F = \tilde{H} \times \tilde{W} \times C \times Q \) array
- **Output tensor** \( y = (H - \tilde{H} + 1) \times (W - \tilde{W} + 1) \times Q \) array
2. Convolution layer

\[ W_{[4, 4, 3]} \]
2. Convolution layer

As a neural network

input features

a bank of 2 filters

2-dimensional output features

Slide: A. Vedaldi
Filter bank example

• A bank of 256 filters (learned from data)
• Each filter has 1 channel (it applies to a grayscale image)
• Each filter is 16x16 pixels
Filtering

Each filter generates a “feature map”

Maximum response when filter matches signal
Convolution details

What is the output size?

Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):
- stride 1 => \((7 - 3)/1 + 1 = 5\)
- stride 2 => \((7 - 3)/2 + 1 = 3\)
- stride 3 => \((7 - 3)/3 + 1 = \ldots :\)
Example: What is the output volume?

Input volume: 32x32x3
Receptive fields: 5x5, stride 3
Number of neurons: 5

Output volume: \((32 - 5) / 3 + 1 = 10\), so: 10x10x5
Zero padding (in each channel)

- e.g. input 7x7
- neuron with receptive field 3x3, stride 1
- pad with 1 pixel border => what is the output?

7x7 => preserved size!

- in general, common to see stride 1, size F, and
- zero-padding with $(F-1)/2$.
- (Will preserve input size spatially)
What is the number of parameters?

- Consider an input gray-scale image of 1000x1000 pixels.
- What is the number of parameters of a filter bank of 100 7x7 filters?
- How does it compare to a fully connected layer that considers the entire input image?

Figure: A. Zisserman

Convolution:
100x 7x7
= 4900 parameters

vs.

Fully connected layer:
1000x1000
x
1000x1000
= 1B parameters.
3. Spatial Max Pooling

Single depth slice

\[
\begin{array}{cccc}
1 & 1 & 2 & 4 \\
5 & 6 & 7 & 8 \\
3 & 2 & 1 & 0 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

max pool with 2x2 filters and stride 2

\[
\begin{array}{cc}
6 & 8 \\
3 & 4 \\
\end{array}
\]

Slide credit: Andrej Karpathy & Fei-Fei Li
3. Spatial Max Pooling

**Dimensions of pooling outputs**

Input volume of size $[W_1 \times H_1 \times D_1]$
Pooling unit receptive fields $F \times F$ and applying them at strides of $S$ gives

Output volume: $[W_2, H_2, D_1]$
$W_2 = (W_1 - F)/S + 1$, $H_2 = (H_1 - F)/S + 1$

Note: pooling happens independently in each channel/slice
4. Non-linearity

- The non-linear activation functions are essential. Why?
4. Non-linearity

Why?

- Non-linearities allow us to approximate arbitrarily complex functions.

- **Universal approximation theorem**: A two-layer multilayer perceptron (MLP) with increasing continuous and bounded non-linearity can approximate any continuous function on a compact given enough hidden neurons. [Cybenko 1989]

- Linear activation functions produce linear decisions no matter what the model size, i.e., stacking multiple linear functions can be expressed with a single linear function.
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]
$y(x) = \tanh(x)$
$y(x) = \max(0, x)$
$y(x) = \max(0, x)$

What is the derivative of the ReLU?

$y = 0$

$y = x$
\[ y(x) = \max(\alpha x, x) \]
4. Non-Linearity

- Per-element (independent)

- Options:
  - **Sigmoid**: $\frac{1}{1+\exp(-x)}$
  - **Tanh**
  - **Rectified linear unit (ReLU)**
    - Simplifies backpropagation
    - Makes learning faster
    - Avoids saturation issues
  - **Variants of ReLU**, e.g. Leaky ReLU
5. Normalization

Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with $N$ as the batch axis, $C$ as the channel axis, and $(H, W)$ as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

[Wu & He, “Group normalization”, ECCV 2018]
CNN Successes

- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese [Ciresan et al. 2012]

- Simpler recognition benchmarks
  - CIFAR-10 (9.3% error [Wan et al. 2013])
  - Traffic sign recognition (0.56% error vs 1.16% for humans [Ciresan et al. 2011])

- But until recently, less good at more complex datasets
  - Caltech-101/256 (few training examples)
ImageNet Dataset

- ~14 million labeled images, 20k classes
- Challenge: 1.2 million training images, 1000 classes
- Images gathered from Internet
- Human labels via Amazon Mechanical Turk

[Deng et al. CVPR 2009]
ImageNet Challenge 2012 (ILSVRC)

- Similar framework to LeCun’98 but:
  - **Bigger model** (7 hidden layers, 60,000,000 params)
  - **More data** ($10^6$ vs. $10^3$ images)
  - **GPU** implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
    - Better regularization for training (DropOut)

AlexNet – 16.4% error (top-5)
Next best (non-convnet) – 26.2% error
Agenda

1. Recap: Bag of Visual Words, Analogy with NNs

2. Neural networks (NNs) for computer vision:
   - Applications
   - A brief history: from perceptron to MLPs to CNNs

3. Convolutional neural networks (CNNs)
   - Standard layers
   - Recap: Training NNs
   - Visualizing CNNs
   - Pretraining & finetuning NNs
   - Typical CNN architectures

4. Beyond CNNs
   - Attention & Transformer
   - Vision Transformers

5. Beyond classification
Gradient descent

- The objective function is an average over all $N$ training data points:

$$L(\theta) = \frac{1}{N} \sum_{i} l(\theta, X_i, Y_i)$$

- Performing a gradient descent is iterating.

$$\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_{i} \frac{\partial l(\theta, X_i, Y_i)}{\partial \theta}$$

- Need to choose the learning rate policy $\alpha_t$

- If the function is not convex, get stuck in a local minimum

- Each step can be expensive to compute if the dataset is large
Stochastic gradient descent

- Instead of computing the gradient, compute an approximation:

\[ \theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta} \]

\[ \theta_{t+1} \rightarrow \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta} \]

- Can take advantage of large datasets, in particular infinite* datasets!

- Introduce stochasticity, which might be good to get out of local minima in the non-convex case
Stochastic gradient descent with minibatch

- Some variance is good, too much can be bad

\[ \theta_{t+1} \to \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta} \]

- It’s faster to compute several gradients in parallel

\[ \theta_{t+1} \to \theta_t - \frac{\alpha_t}{K} \sum_{i=1}^{K} \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta} \quad \text{(with } K << N) \]

- In practice, using batches as large as possible so that the network fits in the GPU memory (e.g., between 1 and 256, depending on the task and network)
Summary: Stochastic Gradient Descent (SGD)

The objective function is an average over all $N$ training data points:

$$
\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}
$$

(gradient descent)

Key idea: approximate the gradient with 1 random datapoint:

$$
\theta_{t+1} \rightarrow \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}
$$

(stochastic gradient descent)

Pick $K$ random points instead of picking 1 (with $K \ll N$):

$$
\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{K} \sum_{i=1}^{K} \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}
$$

(stochastic gradient descent with mini-batches)

$\Rightarrow$ commonly used

Slide credit: Andrea Vedaldi
Quiz: 5 minutes

Let’s consider a training dataset of N samples. How many iterations (i.e., parameter updates) are there in one training epoch?

a. Gradient descent: ___
b. Stochastic gradient descent: ___
c. Stochastic gradient descent with minibatch of size K: ___
**Backpropagation**

**Computing the gradients:** While in theory, we just have the gradients of composite functions and for that apply **chain rule**, there is an **efficient** way to do it, called **backpropagation**.

Y: bike  
X: image  

Slide credit: Andrea Vedaldi  
Training a neural network

Given a \((X, Y)\) pair:

- **Forward pass**: apply network to \(X\) to produce an output \(\hat{Y}\)
- **Evaluation**: Compute loss function, i.e., \(\ell(\hat{Y}, Y)\)
- **Backward pass**: compute the gradient with backpropagation
- **Update**: Take a step in the direction of the gradient
Loss Function

• Regression:
  • L1 (absolute error) / L2 (squared error)

• Classification:
  • Cross-entropy loss
Loss Function: Regression

Estimating a continuous value

- L1 (absolute error)
  \[ L = \left| f(X_i, \theta) - Y_i \right| \]

- L2 (squared error)
  \[ L = \left( f(X_i, \theta) - Y_i \right)^2 \]

Prediction: output of the network \( f \) with parameters \( \theta \) given input \( X_i \)

Ground truth: (label, annotation)
**Loss Function: Classification**

- Cross-entropy loss = **softmax** + **negative log-likelihood**

\[
\text{loss}(x, \text{class}) = -\log\left(\frac{\exp(x|\text{class}|)}{\sum_j \exp(x|j|)}\right)
\]

\[
\begin{align*}
e^5 &+ e^4 + e^2 \\
e^5 &+ e^4 + e^2
\end{align*}
\]

**Fig: Micheleen Harris**

\[
\text{Loss} = 0.34 + 0.02 + 0.71 = 1.07
\]
“Problems” with training

• Making poor predictions on the training data (underfitting)

• Not generalizing to unseen data (overfitting)
Example: polynomial regression of degree $M$
“Typical” machine learning setup

Data split into three sets

- Training set
- Validation set
- Test set

- Allowed to make statistics, learn models, tune hyperparameters
- Not allowed to “see”

- Learn models on the training set
- Evaluate on the validation set many times (run experiments to find good hyperparameters, e.g., number of epochs, learning rate, batch size...)
- (Optional: Learn the final model on the combination of training and validation sets)
- Evaluate on the test set “once”
A few possible scenarios for learning curves

Good fit: both decreasing, converging, minimal gap

Unrepresentative validation set: easier than training set

Unrepresentative validation set: too few examples

Overfit: validation increasing

Underfit: training loss not decreasing

Underfit: training halted prematurely

Credit: Jason Brownlee
How to avoid overfitting?

Deep networks have many parameters. Some regularization techniques:

• Smaller network, i.e., less parameters
• Data augmentation
• Suboptimize, i.e., “early stopping”
• Force redundancy in hidden units, i.e., “dropout”
• Penalize parameter norms, i.e., “weight decay”

L2 penalty:
encourages the norm of the parameters to be low

\[
L(\theta) + \frac{\lambda}{2} \left| \theta \right|^2 \\
\theta_{t+1} \rightarrow \theta_t - \alpha \frac{\partial L}{\partial \theta_t} - \alpha \lambda \theta_t
\]
Look at your results

• When you train a network, you should try to really understand what is happening:
  • Train/val/test sets are important
  • **Look at loss** and performance on train/val sets during training
  • Choose LR, compare networks, try different initialization (random seeds)

• **Very important:** **Look** at your data and results (e.g., visualize predictions) on training and testing data.
Practical problems

• Data loading:
  • Loading “on the fly”: needed for big datasets, use efficient database structure, fast disk access, e.g., SSD
  • Loading to RAM: possible for smaller datasets, or pre-computed features
  • Speed: use GPUs, parallel data loading
  • Network size: get lots of memory on your GPU or/and use several GPUs

Good news: you don’t have to do all of it!
Many ready-to-use and efficient frameworks are available (e.g., Pytorch)
NN packages

- PyTorch (Python)
  - http://pytorch.org/

- TensorFlow (Python) - Google
  - https://www.tensorflow.org/

- Lua Torch
  - http://torch.ch/

- Caffe (C++, pycaffe, matcaffe)
  - http://caffe.berkeleyvision.org/

- MatConvNet (Matlab)
  - http://www.vlfeat.org/matconvnet/
Let’s look at some code (more in Assignment 2)

- The key objects are
  - model,
  - optimizer,
  - dataloader,
  - loss.

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

model = Net()
if args.cuda:
    model.cuda()

optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum)

def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if args.cuda:
            data, target = data.cuda(), target.cuda()

        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        for epoch in range(1, args.epochs + 1):
            train(epoch)
```

• Key part of pytorch code for CNN learning
Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
    • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification
Visualizing CNNs
What does CNN learn once it is trained?
Recap: AlexNet

- **Fully-connected layer**
- **Convolution layer**
- **Pooling layer** (e.g., Max-pooling)
- **Non-linearity layer** (e.g., ReLU)
- ...

AlexNet: [Krizhevsky 2012]
Layer 1: Top-9 patches

Patches from validation images that give maximal activation of a given feature map

 Learned filters

[Zeiler and Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014]
Layer 2: Top-9 patches
Layer 3: Top-9 patches
Layer 4: Top-9 patches
Layer 5: Top-9 patches
References: Visualizing and understanding NNs

Analysis tools

**Visualizing higher-layer features of a deep network**
Ethan et al. 2009
[intermediate features]

**Deep inside convolutional networks**
Simonyan et al. 2014
[deepest features, aka “deep dreams”]

**DeConvNets**
Zeiler et al. In ECCV, 2014
[intermediate features]

**Understanding neural networks through deep visualisation**
Yosinksi et al. 2015
[intermediate features]

Artistic tools

**Google’s “inceptionism”**
Mordvintsev et al. 2015

**Style synthesis and transfer**
Gatys et al. 2015
Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification
Transferring learnt representations

“pretraining”
“Pre-training” and transfer learning

CNN as universal representations
- First several layers in most CNNs are generic
- They can be reused when training data is comparatively scarce

Application
- Pre-train on ImageNet classification 1M images
- Cut at some deep conv or FC layer to get features

[Evaluations in A. S. Razavian, 2014, Chatfield et al., 2014]
"Pre-training" and transfer learning

Deep representations are generic

A general purpose deep encoder is obtained by chopping off the last layers of a CNN trained on a large dataset.
Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks

M. Oquab, L. Bottou, I. Laptev, J. Sivic
In CVPR 2014
http://www.di.ens.fr/willow/research/cnn/
ImageNet classification challenge

Object centric
1000 classes
1.2M images
What about other recognition tasks and datasets?

ImageNet classification challenge

Object centric
- 1000 classes
- 1.2M images

Complex scenes
- 20 classes
- 10k images
Background – Convolutional neural network of [Krizhevsky et al. 2012]

Input: ~1M labelled images (1000 images / 1000 classes)
Number of parameters: ~60 million
Training time: ~1 week on one GPU

Learn parameters using stochastic gradient descent on cross-entropy error function.

Can we transfer learnt parameters to other tasks with limited training data?
The dataset statistics between the source task (ImageNet) and the target task (Pascal VOC) can be very different.

- Type of objects and labels
- Object size, object location, scene clutter
- Object viewpoints, imaging conditions

![ImageNet](image1.jpg)  ![Pascal VOC](image2.jpg)

- Maltese terrier
- Dog

\[ \text{ImageNet} \quad \text{Pascal VOC} \]
Approach
 Approach [Oquab, Bottou, Laptev, Sivic, CVPR’14]

1. Design training/test procedure using sliding windows
2. Train adaptation layers to map labels

See also [Girshick et al.’13], [Donahue et al.’13], [Sermanet et al. ’14], [Zeiler and Fergus ’13]
Transfer learning workshop at ICCV’13, ImageNet workshop at ICCV’13
Pre-training helps

- Pascal VOC 2012 object classification

<table>
<thead>
<tr>
<th></th>
<th>plane</th>
<th>bike</th>
<th>bird</th>
<th>boat</th>
<th>btl</th>
<th>bus</th>
<th>car</th>
<th>cat</th>
<th>chair</th>
<th>cow</th>
<th>table</th>
<th>dog</th>
<th>horse</th>
<th>moto</th>
<th>pers</th>
<th>plant</th>
<th>sheep</th>
<th>sofa</th>
<th>train</th>
<th>tv</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO PRETRAIN</td>
<td>85.2</td>
<td>75.0</td>
<td>69.4</td>
<td>66.2</td>
<td>48.8</td>
<td>82.1</td>
<td>79.5</td>
<td>79.8</td>
<td>62.4</td>
<td>61.9</td>
<td>49.8</td>
<td>75.9</td>
<td>71.4</td>
<td>82.7</td>
<td>93.1</td>
<td>59.1</td>
<td>69.7</td>
<td>49.3</td>
<td>80.0</td>
<td>76.7</td>
<td>70.9</td>
</tr>
<tr>
<td>PRE-1000C</td>
<td>93.5</td>
<td>78.4</td>
<td>87.7</td>
<td>80.9</td>
<td>57.3</td>
<td>85.0</td>
<td>81.6</td>
<td>89.4</td>
<td>66.9</td>
<td>73.8</td>
<td>62.0</td>
<td>89.5</td>
<td>83.2</td>
<td>87.6</td>
<td>95.8</td>
<td>61.4</td>
<td>79.0</td>
<td>54.3</td>
<td>88.0</td>
<td>78.3</td>
<td>78.7</td>
</tr>
<tr>
<td>PRE-1000R</td>
<td>93.2</td>
<td>77.9</td>
<td>83.8</td>
<td>80.0</td>
<td>55.8</td>
<td>82.7</td>
<td>79.0</td>
<td>84.3</td>
<td>66.2</td>
<td>71.7</td>
<td>59.5</td>
<td>83.4</td>
<td>81.4</td>
<td>84.8</td>
<td>95.2</td>
<td>59.8</td>
<td>74.9</td>
<td>52.9</td>
<td>83.8</td>
<td>75.7</td>
<td>76.3</td>
</tr>
<tr>
<td>PRE-1512</td>
<td>94.6</td>
<td>82.9</td>
<td>88.2</td>
<td>84.1</td>
<td>60.3</td>
<td>89.0</td>
<td>84.4</td>
<td>90.7</td>
<td>72.1</td>
<td>86.8</td>
<td>69.0</td>
<td>92.1</td>
<td>93.4</td>
<td>88.6</td>
<td>96.1</td>
<td>64.3</td>
<td>86.6</td>
<td>62.3</td>
<td>91.1</td>
<td>79.8</td>
<td>82.8</td>
</tr>
</tbody>
</table>

- Pascal VOC 2012 action classification

<table>
<thead>
<tr>
<th>Action</th>
<th>jump</th>
<th>phon</th>
<th>instr</th>
<th>read</th>
<th>bike</th>
<th>horse</th>
<th>run</th>
<th>phot</th>
<th>comp</th>
<th>walk</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO PRETRAIN</td>
<td>43.2</td>
<td>30.6</td>
<td>50.2</td>
<td>25.0</td>
<td>76.8</td>
<td>80.7</td>
<td>75.2</td>
<td>22.2</td>
<td>37.9</td>
<td>55.6</td>
<td>49.7</td>
</tr>
<tr>
<td>PRE-1512</td>
<td>73.4</td>
<td>44.8</td>
<td>74.8</td>
<td>43.2</td>
<td>92.1</td>
<td>94.3</td>
<td>83.4</td>
<td>45.7</td>
<td>65.5</td>
<td>66.8</td>
<td>68.4</td>
</tr>
<tr>
<td>PRE-1512U</td>
<td>74.8</td>
<td>46.0</td>
<td>75.6</td>
<td>45.3</td>
<td>93.5</td>
<td>95.0</td>
<td>86.5</td>
<td>49.3</td>
<td>66.7</td>
<td>69.5</td>
<td>70.2</td>
</tr>
</tbody>
</table>
Other "pre-training" examples
Figure 8: We compare training on FHB only (Real) and pre-training on synthetic, followed by fine-tuning on FHB (Synth2Real). As the amount of real data decreases, the benefit of pre-training increases. For both the object and the hand reconstruction, synthetic pre-training is critical in low-data regimes.
Text-to-Video Retrieval

Pretraining on millions of images & videos
Finetuning on MSRVTT with 9K training videos

<table>
<thead>
<tr>
<th>Pre-training (for 1 epoch)</th>
<th>#pairs</th>
<th>( \uparrow R@1 )</th>
<th>( \uparrow R@10 )</th>
<th>( \downarrow \text{MedR} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>5.6</td>
<td>22.3</td>
<td>55</td>
</tr>
<tr>
<td>ImageNet</td>
<td></td>
<td>15.2</td>
<td>54.4</td>
<td>9.0</td>
</tr>
<tr>
<td>HowTo-17M subset</td>
<td>17.1M</td>
<td>24.1</td>
<td>63.9</td>
<td>5.0</td>
</tr>
<tr>
<td>CC3M</td>
<td>3.0M</td>
<td>24.5</td>
<td>62.7</td>
<td>5.0</td>
</tr>
<tr>
<td>WebVid2M</td>
<td>2.5M</td>
<td>26.0</td>
<td>64.9</td>
<td>5.0</td>
</tr>
<tr>
<td>CC3M + WebVid2M</td>
<td>5.5M</td>
<td>27.3</td>
<td>68.1</td>
<td>4.0</td>
</tr>
</tbody>
</table>

1. A man and a woman performing a musical.
2. A teenage couple perform in an amateur musical
3. Dancers are playing a routine.
4. People are dancing in a musical.
5. Some people are acting and singing for performance.

Pretraining on various tasks on different datasets
Finetuning on 50K videos from BSL-1K sign language dataset
Pretraining Summary

• Common practice: Pretrain on large data, finetune on small data.
  • Remove the last class-specific layer (e.g. 1000 categories)
  • Add new layer(s) for the new task randomly initialized
  • Either “freeze” the pretrained parameters and train a simple classifier on top,
  • Or train “end-to-end” all parameters.

• Avoids overfitting

• Shortens training time

• Lots of pretrained models available online

• Task and domain-relevant pretraining is usually better
Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification
Recall: the goal of this model is to map an input image to a class prediction.
Recall: The AlexNet model

A breakthrough in image understanding

Each large block represents a data tensor
Each smaller block represents a filter
The filter size and stride are shown

The number of filters can be deduced from the number of feature channels
There are two parallel streams in this network (for efficiency reasons)

[AlexNet by Krizhevsky et al. 2012]
How deep is deep enough?

AlexNet (2012)

5 convolutional layers

3 fully-connected layers
How deep is deep enough?

- AlexNet (2012)
- VGG-M (2013)
- VGG-VD-16 (2014)

Slide: A. Vedaldi
How deep is deep enough?

How deep is deep enough?

GoogLeNet (2014)  
VGG-VD-16 (2014)  
VGG-M (2013)  
AlexNet (2012)  
16 convolutional layers  
50 convolutional layers  
152 convolutional layers  
ResNet 50 (2015)  
ResNet 152 (2015)


Accuracy

3 × more accurate in 3 years

Slide: A. Vedaldi
**Remark:** 101 ResNet layers same size/speed as 16 VGG-VD layers

**Reason:** far fewer feature channels (quadratic speed/space gain)

**Moral:** optimize your architecture
CNN architectures – notes and details

- Increased depth of recent architectures
- Number of parameters matter (How to count parameters?)
- Power of small filters, e.g. 3x3 convolutions
- ResNet architecture
The power of small filters

Suppose we stack two CONV layers with receptive field size 3x3

Q: What region of input does each neuron in 2\textsuperscript{nd} CONV see?

Answer: [5x5]
The power of small filters

Suppose we stack three CONV layers with receptive filed size 3x3

Q: What region of input does each neuron in 3rd CONV see?

Answer: [7x7]
The power of small filters

Suppose input has depth C & we want output depth C as well.

1x CONV with 7x7 filters

3x CONV with 3x3 filters

Number of weights:

\[ C \cdot (7 \times 7 \cdot C) \]
\[ = 49 \ C^2 \]

Number of weights:

\[ C \cdot (3 \times 3 \cdot C) + C \cdot (3 \times 3 \cdot C) + C \cdot (3 \times 3 \cdot C) \]
\[ = 3 \cdot 9 \cdot C^2 \]
\[ = 27 \ C^2 \]
Residual networks [ResNets]

Plain net

\[ x \]

weight layer

\[ \text{relu} \]

weight layer

\[ \text{relu} \]

\[ H(x) \]

\[ H(x) \text{ is any desired mapping, hope the 2 weight layers fit } H(x) \]

Residual networks [ResNets]

Residual net

\[ H(x) = F(x) + x \]

\[ H(x) \text{ is any desired mapping,} \]
\[ \text{hope the 2 weight layers fit } H(x) \]
\[ \text{hope the 2 weight layers fit } F(x) \]
\[ \text{let } H(x) = F(x) + x \]
**Residual networks [ResNets]**

*F(x)* is a **residual mapping w.r.t. identity**

\[ H(x) = F(x) + x \]

- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations
Network design

Basic design (vgg-style)

- almost all 3x3 conv
- Spatial size /2 => # filters x2
- Simple design, just deep
- No fully connected layers
- No dropout
CIFAR-10 experiments

- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error** and lower test error
Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification
Do we need convolutions?

Attention Is All You Need

Published as a conference paper at ICLR 2021

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

NeurIPS 2017

NeurIPS 2021

https://github.com/KentoNishi/awesome-all-you-need-papers
Recent Hype#1: Transformers

- Transformers = neural network architectures stacking "attention" layers\(^1\)
- Initially successful for natural language processing
- Then applied to computer vision\(^2\). Better performance than CNNs given enough data.
- The hype still continues today.
- What is attention?

---

\(^1\) Vaswani et al. "Attention is all you need", NeurIPS 2017.
\(^2\) Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale", ICLR 2021.
Attention & Transformer

• Basic transformer model

• Image transformers
Attention

• Motivation: sequence-to-sequence models

\[
\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V
\]

\[
A_{i,j} = \text{softmax} \left( \frac{Q_i \cdot K_j}{\sqrt{d}} \right) \\
Y_i = \sum_j A_{i,j}V_j
\]
Attention mechanisms

Given a query sequence $Q$, a key sequence $K$, and a value sequence $V$, compute an attention matrix $A$ by matching $Q$s to $K$s, and weight $V$ with it to get the sequence $Y$.

$$A_{i,j} = \text{softmax} \left( \frac{Q_i \cdot K_j}{\sqrt{d}} \right)$$
Attention mechanisms

Given a query sequence $Q$, a key sequence $K$, and a value sequence $V$, compute an attention matrix $A$ by matching $Q$s to $K$s, and weight $V$ with it to get the sequence $Y$.

$$Y_i = \sum_j A_{i,j}V_j$$
Attention mechanisms

Given a query sequence $Q$, a key sequence $K$, and a value sequence $V$, compute an attention matrix $A$ by matching $Q$s to $K$s, and weight $V$ with it to get the sequence $Y$.

$$A_{i,j} = \text{softmax} \left( \frac{Q_i \cdot K_j}{\sqrt{d}} \right)$$
Attention mechanisms

Given a query sequence $Q$, a key sequence $K$, and a value sequence $V$, compute an attention matrix $A$ by matching $Q$s to $K$s, and weight $V$ with it to get the sequence $Y$.

$$Y_i = \sum_j A_{i,j}V_j$$
Attention mechanisms

- Query and Key dimensionalities are the same.
- Value dimensionality may be different.
- Output dimensionality will be the same as Value.

In "self-attention", \((Q, K, V)\) obtained from the same input, linearly projected three times.
Self-Attention Example

\[ A_{i,j} = \text{softmax} \left( \frac{Q_i \cdot K_j}{\sqrt{d}} \right) \]

\[ Y_i = \sum_j A_{i,j} V_j \]
Basic transformer model

- Sequence-to-sequence architecture using only point-wise processing and attention (no recurrent units or convolutions)

**Encoder**: receives entire input sequence and outputs encoded sequence of the same length

**Decoder**: predicts next token conditioned on encoder output and previously predicted tokens


NLP application: Machine Translation

Image source
Key-Value-Query attention model

• Key vectors: \( K = XW_K \)
• Value Vectors: \( V = XW_V \)
• Query vectors
• Similarities: scaled dot-product attention

\[
E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}} \quad \text{or} \quad E = QK^T / \sqrt{D}
\]

\( (D \) is the dimensionality of the keys)\)
• Attn. weights: \( A = \text{softmax}(E, \text{dim} = 1) \)
• Output vectors:

\[
Y_i = \sum_j A_{i,j}V_j \quad \text{or} \quad Y = AV
\]

Adapted from J. Johnson
Attention mechanisms

- **Encoder self-attention**: queries, keys, and values come from previous layer of encoder
- **Decoder self-attention**: values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention**: queries come from previous decoder layer, keys and values come from output of encoder
Self-attention

- Used to capture context *within the sequence*

As we are encoding “it”, we should focus on “the animal”

As we are encoding “it”, we should focus on “the street”
Self-attention layer

- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*
  $$E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}} \quad \text{or} \quad E = QK^T / \sqrt{D}$$
  ($D$ is the dimensionality of the keys)
- Attn. weights: $A = \text{softmax}(E, \dim = 1)$
- Output vectors:
  $$Y_i = \sum_j A_{i,j}V_j \quad \text{or} \quad Y = AV$$

Adapted from J. Johnson
Positional encoding

- Self attention doesn’t “know” the order of the vectors it is processing!
- In order to make processing position-aware, concatenate input with **positional encoding**
- $E$ can be learned lookup table, or fixed function

Adapted from J. Johnson
Positional encoding

- To give transformer information about ordering of tokens, add function of position (based on sines and cosines) to every input.
Attention mechanisms: Overview

- **Encoder self-attention:** queries, keys, and values come from previous layer of encoder
- **Decoder self-attention:** values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention:** queries come from previous decoder layer, keys and values come from output of encoder
Decoder: **Masked self-attention**

- The decoder should not “look ahead” in the output sequence

![Diagram of masked self-attention](image)

Adapted from J. Johnson
Decoder: **Masked self-attention**

- The decoder should not "look ahead" in the output sequence

Adapted from J. Johnson
Decoder: **Masked self-attention**

- The decoder should not “look ahead” in the output sequence

---

Adapted from J. Johnson
Attention mechanisms: Overview

- **Encoder self-attention**: queries, keys, and values come from previous layer of encoder
- **Decoder self-attention**: values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention**: queries come from previous decoder layer, keys and values come from output of encoder
Transformer architecture: Details

A. Vaswani et al., *Attention is all you need*, NeurIPS 2017
Multi-head attention

- Run $h$ attention models in parallel on top of different linearly projected versions of $Q$, $K$, $V$; concatenate and linearly project the results.

- Intuition: enables model to attend to different kinds of information at different positions (see visualization tool).
Transformer blocks

- A Transformer is a sequence of transformer blocks

- Vaswani et al.: \( N = 12 \) blocks, embedding dimension = 512, 6 attention heads

- **Add & Norm**: residual connection followed by layer normalization

- **Feedforward**: two linear layers with ReLUs in between, applied independently to each vector

- Attention is the only interaction between inputs!
Transformer architecture: Zooming back out

A. Vaswani et al., *Attention is all you need*, NeurIPS 2017
Transformer implementation

class MultiHeadedAttention(nn.Module):
    """
    Multi-Head Attention module from "Attention is All You Need"
    Implementation modified from OpenNMT-py.
    https://github.com/OpenNMT/OpenNMT-py
    """

    def __init__(self, num_heads: int, size: int, dropout: float = 0.1):
        """
        Create a multi-headed attention layer.
        :param num_heads: the number of heads
        :param size: model size (must be divisible by num_heads)
        :param dropout: probability of dropping a unit
        """
        super(MultiHeadedAttention, self).__init__()

        assert size % num_heads == 0

        self.head_size = head_size = size // num_heads
        self.model_size = size
        self.num_heads = num_heads

        self.k_layer = nn.Linear(size, num_heads * head_size)
        self.v_layer = nn.Linear(size, num_heads * head_size)
        self.q_layer = nn.Linear(size, num_heads * head_size)

        self.output_layer = nn.Linear(size, size)
        self.softmax = nn.Softmax(dim=-1)
        self.dropout = nn.Dropout(dropout)

nn.Linear:
Learnable params
Transformer implementation

```python
def forward(
    self, k: Tensor, v: Tensor, q: Tensor, rel_mouth_times=None, mask: Tensor = None
):
    ...
    Computes multi-headed attention.
    :param k: keys [B, K, D] with K being the sentence length.
    :param v: values [B, K, D]
    :param q: query [B, Q, D] with Q being the sentence length.
    :param mask: optional mask [B, 1, K]
    :return:
    
    batch_size = k.size(0)  # B
    num_heads = self.num_heads  # H
    # project the queries (q), keys (k), and values (v)
    k = self.k_layer(k)
    v = self.v_layer(v)
    q = self.q_layer(q)

    # reshape q, k, v for our computation to [B, H, ..., D/H]
    k = k.view(batch_size, -1, num_heads, self.head_size).transpose(1, 2)
    v = v.view(batch_size, -1, num_heads, self.head_size).transpose(1, 2)
    q = q.view(batch_size, -1, num_heads, self.head_size).transpose(1, 2)

    # compute scores
    q = q / math.sqrt(self.head_size)
```

K = XW_K
V = XW_V
Q = XW_Q
Transformer implementation

```python
# [B, H, Q, K]
scores = torch.matmul(q, k.transpose(2, 3))

# apply the mask (if we have one)
# we add a dimension for the heads to it below: [B, 1, 1, K]
if mask is not None:
    scores = scores.masked_fill(~mask.unsqueeze(1), float("-inf"))

# apply attention dropout and compute context vectors.
# [B, H, Q, K]
attention_map = self.softmax(scores)
attention = self.dropout(attention_map)

# get context vector (select values with attention)
# [B, H, Q, D/H]
context = torch.matmul(attention, v)
# reshape back to [B, Q, D]
context = (context.transpose(1, 2)
    .contiguous()
    .view(batch_size, -1, num_heads * self.head_size)
)

# [B, Q, D]
output = self.output_layer(context)

return output, attention_map
```

\[ E = \frac{QK^T}{\sqrt{D}} \]

\[ A = \text{softmax}(E, \text{dim} = 1) \]

\[ Y = AV \]
class TransformerEncoderLayer(nn.Module):
    # One Transformer encoder layer has a Multi-head attention layer plus
    # a position-wise feed-forward layer.

    def __init__(
        self, size: int = 0, ff_size: int = 0, num_heads: int = 0, dropout: float = 0.1
    ):
        super(TransformerEncoderLayer, self).__init__()

        self.layer_norm = nn.LayerNorm(size, eps=1e-6)
        self.src_src_att = MultiHeadedAttention(num_heads, size, dropout=dropout)
        self.feed_forward = PositionwiseFeedForward(
            size, ff_size=ff_size, dropout=dropout
        )
        self.dropout = nn.Dropout(dropout)
        self.size = size

    # pylint: disable=arguments-differ
    def forward(self, x: Tensor, mask: Tensor) -> Tensor:
        # First applies layer norm, then self attention, then dropout with residual connection (adding the input to the result), and then a position-wise feed-forward layer.
        x_norm = self.layer_norm(x)
        h, att_map_src_src = self.src_src_att(k=x_norm, v=x_norm, q=x_norm, mask=mask)
        h = self.dropout(h) + x
        o = self.feed_forward(h)
        return o
Original transformer results on machine translation

Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
    • Vision Transformers

• 5. Beyond classification
Attention & Transformers

• Basic transformer model

• Image transformers
**Image transformer – Google**  
Self-attention only locally

- Image generation and super-resolution with 32x32 output, attention restricted to local neighborhoods

*Table 2. On the left are image completions from our best conditional generation model, where we sample the second half. On the right are samples from our four-fold super-resolution model trained on CIFAR-10. Our images look realistic and plausible, show good diversity among the completion samples and observe the outputs carry surprising details for coarse inputs in super-resolution.*

N. Parmar et al., *Image transformer*, ICML 2018
Sparse transformers – OpenAI

R. Child et al., Generating Long Sequences with Sparse Transformers, arXiv 2019

*Figure 5.* Unconditional samples from ImageNet 64x64, generated with an unmodified softmax temperature of 1.0. We are able to learn long-range dependencies directly from pixels without using a multi-scale architecture.
Image GPT* – OpenAI

*GPT: Generative pre-trained Transformer

M. Chen et al., Generative pretraining from pixels, ICML 2020

https://openai.com/blog/image-gpt/

works on reduced resolutions
Vision transformer (ViT) - Google

- Split an image into patches, feed linearly projected patches into standard transformer encoder
- With patches of 14x14 pixels, you need 16x16=256 patches to represent 224x224 images

A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. ICLR 2021
Vision transformer (ViT)

Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

BiT: **Big Transfer** (ResNet)
ViT: Vision Transformer (Base/Large/Huge, patch size of 14x14, 16x16, or 32x32)

Internal Google dataset (not public)

A. Dosovitskiy et al. *An image is worth 16x16 words: Transformers for image recognition at scale*. ICLR 2021
**Masked autoencoders are scalable vision learners**

Figure 1. **Our MAE architecture.** During pre-training, a large random subset of image patches (e.g., 75%) is masked out. The encoder is applied to the small subset of visible patches. Mask tokens are introduced after the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images to produce representations for recognition tasks.

K. He et al. [Masked autoencoders are scalable vision learners](https://arxiv.org/abs/2111.06377). arXiv 2021
Figure 2. Example results on ImageNet validation images. For each triplet, we show the masked image (left), our MAE reconstruction† (middle), and the ground-truth (right). The masking ratio is 80%, leaving only 39 out of 196 patches. More examples are in the appendix.

†As no loss is computed on visible patches, the model output on visible patches is qualitatively worse. One can simply overlay the output with the visible patches to improve visual quality. We intentionally opt not to do this, so we can more comprehensively demonstrate the method’s behavior.
Detection Transformer (DETR)

- Hybrid of CNN and transformer, aimed at standard recognition task

N. Carion et al., *End-to-end object detection with transformers*, ECCV 2020
Do we need attention?

[R] Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprisingly Well on ImageNet

Research

https://www.reddit.com/r/MachineLearning/comments/n62qhn/r_do_you_even_need_attention_a_stack_of/
Do we need attention?

MLP-Mixer: An all-MLP Architecture for Vision

Byya Toltikhin*, Neil Houlsby*, Alexander Kolesnikov*, Lucas Beyer*, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy
*equal contribution
Google Research, Brain Team
(tolstikhin, neilhoulsby, aekolesnikov, lbeyer,
xzhai, unterthiner, jyung, asteiner, dkeysers, juszkoreit, mlucic, adosovitskiy)@google.com

Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprisingly Well on ImageNet

Luke Melas-Kyriazi
Oxford University
lukemk@robots.ox.ac.uk

Pay Attention to MLPs

Hanjiao Liu, Zihang Dai, David R. So, Quoc V. Le
Google Research, Brain Team
(hanjiaol, zihangd, davidso, qv)@google.com
Recent Hype#2: MLPs (!)

- Back to basics
- MLPs perform similar to Transformers while being more efficient
- CNNs and MLPs complexity linear with the number of input pixels, Transformers quadratic
Summary: Beyond CNNs

- CNNs (convolution), Transformers (attention), MLPs (fully connected)
- There is no answer to which architecture is better.
- Often depends on the data.
- If you have infinite data, more complex can be better (e.g., MLP ~ Transformers > CNN).
- Similar performance can be obtained with more efficient models (e.g., MLP ~ Transformers)
- It is possible there will be newer/better architectures/hypes before you graduate. Stay tuned.
Agenda

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification - preview
The field makes progress

Beyond Classification
Computer vision tasks

Extracting meaning from visual signals*

Object recognition, Object detection, Pixel-level segmentation, 3D localization, etc.

*Visual signal: Image, video, depth, 3D point cloud, MRI, scans, …
Example tasks

Slide credit: Naila Murray
Object recognition and localization (detection)

Slide credit: Naila Murray
Q: Is this an outdoor scene?
A: Yes

Q: What is the weather like?
A: Cloudy but dry
Activity recognition

walking

Slide credit: Naila Murray
Pose estimation
Captioning

A jackal walking across a rural asphalt road
Semantic segmentation
Depth estimation
3D shape estimation
Visual localization
Object detection

Finally we do NMS and threshold detections.
Segmentation

Human pose estimation

Guler et al. DensePose, CVPR 2018
Text-to-image retrieval

Contrastive Language-Image Pretraining (CLIP)

Contrastive objective: in a batch of N image-text pairs, classify each text string to the correct image and vice versa

A. Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021
   https://openai.com/blog/clip/
Billy reveals the truth to Louis about the Duke’s bet which changed both their lives.
Frozen in Time

Video Search Demo

Visual search of ~2.6M videos are based on research described in
Frozen in time: A joint video and image encoder for end-to-end retrieval.

[Bain, Nagrani, Varol, Zisserman, ICCV 2021]
Text-based image generation: DALL-E

(a) a tapir made of accordion.  
(b) an illustration of a baby hedgehog in a christmas sweater walking a dog  
(c) a neon sign that reads “backprop”. a neon sign that reads “backprop”.  backprop neon sign

A. Ramesh et al., Zero-Shot Text-to-Image Generation, ICML 2021
https://openai.com/blog/dall-e/
Summary of today

• 1. Recap: Bag of Visual Words, Analogy with NNs

• 2. Neural networks (NNs) for computer vision:
  • Applications
  • A brief history: from perceptron to MLPs to CNNs

• 3. Convolutional neural networks (CNNs)
  • Standard layers
  • Recap: Training NNs
  • Visualizing CNNs
  • Pretraining & finetuning NNs
  • Typical CNN architectures

• 4. Beyond CNNs
  • Attention & Transformer
  • Vision Transformers

• 5. Beyond classification
Key elements of DL for CV

1. **Model** (i.e., architectural definition of connectivity and learnable parameters)
2. **Loss**
3. **Data**

- Initially in CV
- Next in CV
- These days in CV !
- ML community

- Optimization algorithm (i.e., variations of SGD)
Still many open questions

• 3D

• Videos

• Visual perception in robotics