Convolutional Neural Networks

Image Processing — scc0251/scc5830

www.icmc.usp.br/∼moacir — moacir@icmc.usp.br

ICMC/USP — São Carlos, SP, Brazil

2017

Agenda

Image classification example

Problem — given two classes of images:

- class 1: **desert**,
- class 2: **beach**,

and also a set of 9 images taken from each class, develop a program able to classify a new, and unseen image, into one of those two classes.

- **Object**: image

Image classification example

- **Feature**: set of values extracted from images that can be used to measure the (dis)similarity between images **Any suggestion?**
  - Requantize the image to obtain only 64 colours per image, use the two most frequent colours as features!
  - Each image is represented by 2 values: 2D feature space.

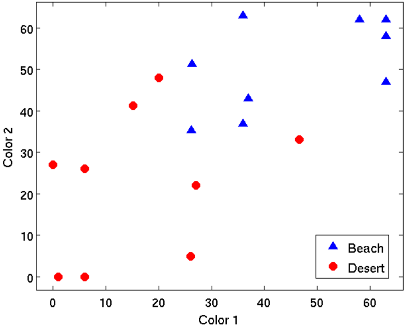
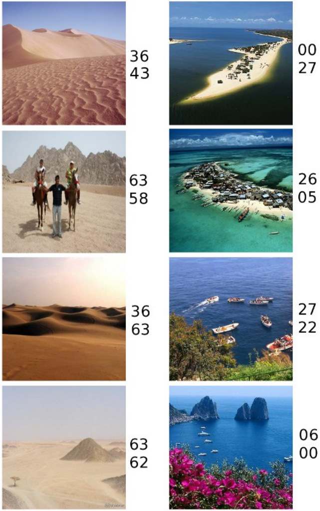


Image classification example



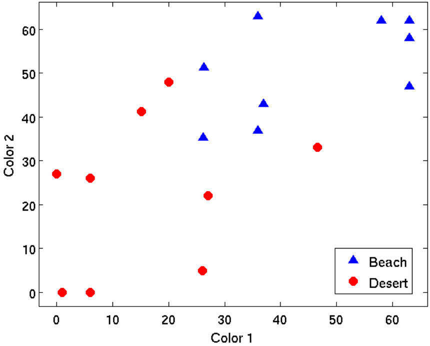
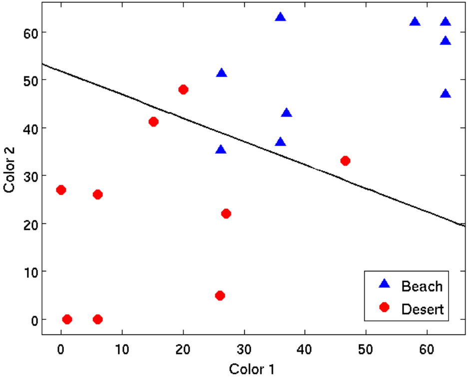


Image classification example

- **Classifier**: a model build using labeled examples (images for which the classes are known). This model must be able to predict the class of a new image. **Any suggestion?**
    - To find a partition of the space, using the data distribution.



,

Image classification example

- Examples used to build the classifier : **training set**.
- Training data is seldom linearly separable
- Therefore there is a **training error**

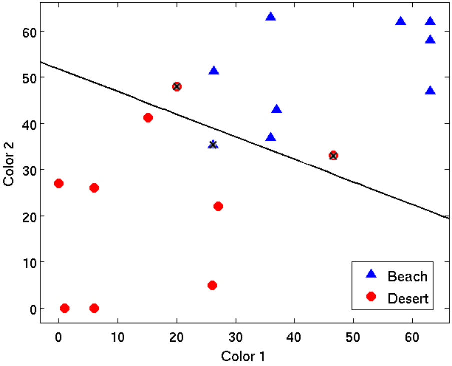


Image classification example

- The model, or **classifier**, can then be used to predict/infer the class of a new **example**.

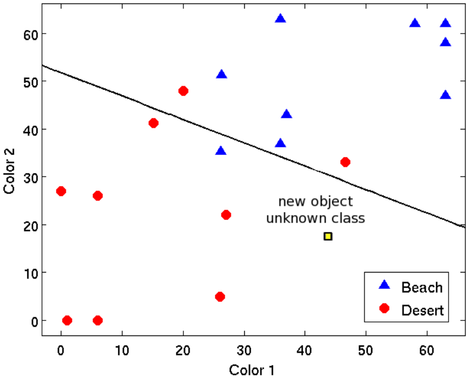
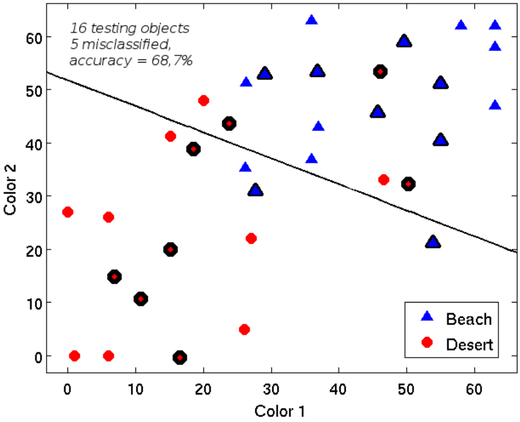


Image classification example

- Now we want to test, for future data (not used in training), the classifier error rate (or alternatively, its accuracy)
- The examples used in this stage is known as **test set**.



Terminology

Class: label/category, $\Omega = \{\omega_1, \omega_2, ..., \omega_c\}$

Dataset: $X = \{x_1, x_2, ..., x_N\}$, for $x_i \in \mathbb{R}^M$

$\mathrm{x}_i \in \mathbb{R}^M$ **example** (object) in the feature space: the *feature vector*

$l(\mathrm{x}_i) = y_i \in \Omega$ **labels** assigned to the each example

matrix N examples $\times$ M features:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,M} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,M} \\ \ldots & \ldots & & \ldots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,M} \end{bmatrix}, \text{ labels} = Y = \begin{bmatrix} l(\mathrm{x}_1) = y_1 \\ l(\mathrm{x}_2) = y_2 \\ \ldots \\ l(\mathrm{x}_N) = y_N \end{bmatrix}$$

Agenda

Introduction

Recent history that tries to solve the problem of image classification:

- Color, shape and texture descriptors (1970-2000)
- SIFT (1999)
- Histogram of Gradients (2005)
- Spatial Pyramid Matching (2006),

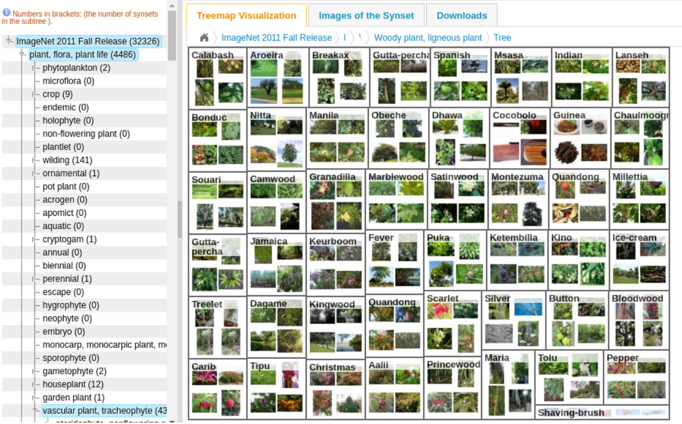
Pipeline

1. Descriptor grid: HoG, LBP, SIFT, SURF
2. Fisher Vectors
3. Spatial Pyramid Matching
4. Classifier

Image Net/ Large Scale Visual Recognition Challenge

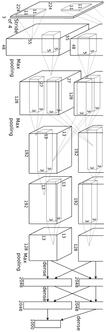
ImageNet: 22000 categories, 14 million images

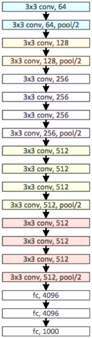
ImageNet Challenge: $\sim$ 1.4 million images, 1000 classes.

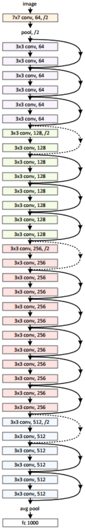


Architectures and number of layers

AlexNet (9) GoogLeNet (22) VGG (16/19) ResNet (34+)

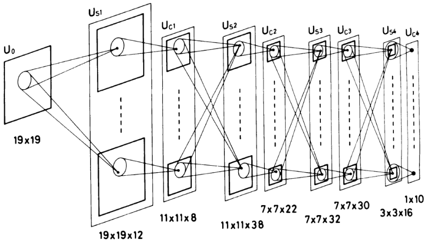




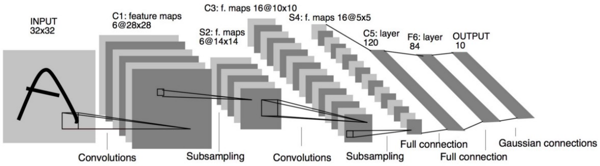


CNNs were not invented in 2012...

Fukushima's Neocognitron (1989)



LeCun's LeNet (1998)



Agenda

A linear classifier

Input  $\rightarrow$ x

weight matrix — image — bias term

$$f(W, x) = W \; x + b$$

= scores for possible classes of x

Linear classifier for image classification

- Input: image (with $N \times M \times 3$ numbers) vectorized into column x
- Classes: cat, turtle, owl
- Output: class scores

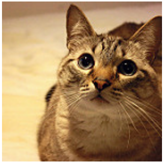
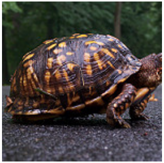
$$= \mathrm{x} = [1, 73, 227, 82]$$

$$f(\mathrm{x}, W) = s \quad \rightarrow \text{ 3 numbers with class scores}$$

$$W\mathrm{x} + \mathrm{b}$$

$$\begin{bmatrix} 0.1 & -0.25 & 0.1 & 2.5 \\ 0 & 0.5 & 0.2 & -0.6 \\ 2 & 0.8 & 1.8 & -0.1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 73 \\ 227 \\ 82 \end{bmatrix} + \begin{bmatrix} -2.0 \\ 1.7 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -337.3 \\ -38.6 \\ 460.30 \end{bmatrix}$$

Linear classifier for image classification

cat	-337.3	**380.3**	8.6
owl	**460.3**	160.3	**26.3**
turtle	38.6	17.6	21.8

We need:

- a **loss function** that quantifies undesired scenarios in the training set
- an **optimization algorithm** to find W so that the loss function is minimized!

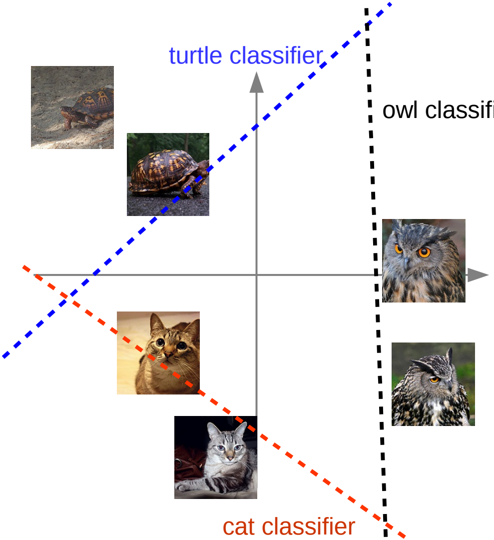
Linear classifier for image classification

- We want to optimize some function to produce the best classifier
- This function is often called **loss function**,

Let (X, Y) be the training set: X are the features, Y are the class labels, and $f(.)$ a classifier that maps any value in X into a class:

$$\ell(f(W, \mathrm{x}_i, y_i) = (\underbrace{f(W, \mathrm{x}_i)}_{\text{predicted label}} - \underbrace{y_i}_{\text{true label}})^2 \quad (1)$$

A linear classifier we would like



Minimizing the loss function

Use the slope of the loss function over the space of parameters!
For each dimension j:

$$\frac{df(x)}{dx} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

$$\frac{d\ell(f(w_j, \mathrm{x}_i))}{dw_j} = \lim_{\delta \to 0} \frac{f(w_j+\delta, \mathrm{x}_i) - f(w_j, \mathrm{x}_i)}{\delta}$$

We have multiple dimensions, therefore a gradient (vector of derivatives).

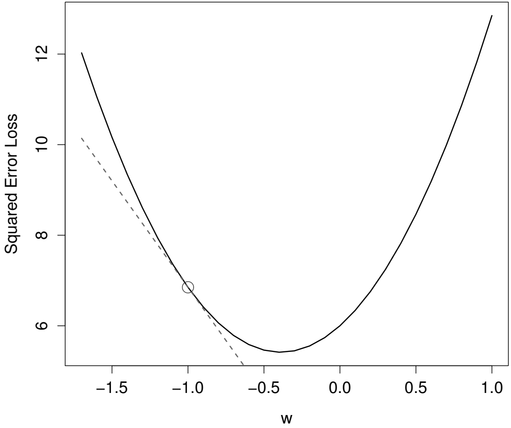
We may use:

1. Numerical gradient: approximate
2. Analytic gradient: exact

Gradient descent — search for the valley of the function, moving in the direction of the negative gradient.

Gradient descent

Changes in a parameter affects the loss (ideal example)



Gradient descent

W	$w_i + \delta$	dw_i
$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} 0.1 + 0.001, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} ?, \\ , \\ , \\ , \\ , \\ ..., \\ \end{bmatrix}$
$\ell(f(W)) = 2.31298$	$\ell(f(W')) = 2.31201$	$(f(w_i + \delta) - f(w_i))/\delta$

Gradient descent

W	$w_i + \delta$	dw_i
$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} 0.1 + 0.001, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} -0.97, \\ , \\ , \\ , \\ , \\ ..., \\ \end{bmatrix}$
$\ell(f(W)) = 2.31298$	$\ell(f(W')) = 2.31201$	$(f(w_i + \delta) - f(w_i))/\delta$

Gradient descent

$$W \qquad\qquad w_i + \delta \qquad\qquad dw_i$$

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix} \qquad \begin{bmatrix} 0.1, \\ -0.25 + 0.001, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix} \qquad \begin{bmatrix} -0.97, \\ 0.0, \\ , \\ , \\ , \\ ..., \\ \end{bmatrix}$$

$$\ell(f(W)) = 2.31298 \qquad \ell(f(W')) = 2.31298 \qquad (f(w_i + \delta) - f(w_i))/\delta$$

Gradient descent

W	$w_i + \delta$	dw_i
$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1 + 0.001, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} -0.97, \\ 0.0, \\ +1.61, \\ -, \\ -, \\ ..., \\ - \end{bmatrix}$
$\ell(f(W)) = 2.31298$	$\ell(f(W1)) = 2.31459$	$(f(w_i + \delta) - f(w_i))/\delta$

Gradient descent

$$W \qquad\qquad w_i + \delta \qquad\qquad dw_i$$

$$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix} \qquad \begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix} \qquad \begin{bmatrix} -0.93, \\ 0.0, \\ -1.61, \\ +0.02, \\ +0.5, \\ ..., \\ -3.7 \end{bmatrix}$$

$$\ell(f(W)) = 2.31298 \qquad \ell(f(W')) = 2.08720 \qquad (f(w_i + \delta) - f(w_i))/\delta$$

Regularization

regularization

$$\ell(W) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(x_i, y+i, W) + \lambda R(W)$$

$$\nabla_W \ell(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W \ell_i(x_i, y+i, W) + \lambda \nabla_W R(W)$$

Regularization will help the model to keep it simple. Possible methods

- $L2$: $R(W) = \sum_i \sum_j W_{i,j}^2$
- $L1$: $R(W) = \sum_i \sum_j |W_{i,j}|$
- others (dropout, batch normalization)

Stochastic Gradient Descent (SGD)

It is hard to compute the gradient, when N is large.

SGD:
Approximate the sum using a **minibatch** (random sample) of instances: something between 32 and 512.
Because it uses only a fraction of the data:

- fast
- often gives bad estimates on each iteration, needing more iterations

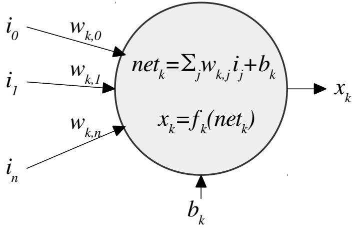
Stochastic Gradient Descent (SGD)

Naïve approach (α is the learning rate):

```
repeat until convergence (or a fixed number of iterations) {
    sample a minibatch of examples
    for each w(i) {
        tmp(i) = w(i) - alpha (d / d theta(i)) l(theta)
    }
    for each w(i) {
        w(i) = tmp(i)
    }
}
```

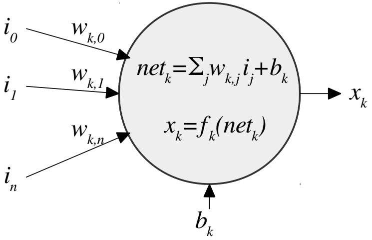
Neuron

- input: 1+ values
- output: 1 value
- each connection associated with a weight w (connection strength)
- often there is a bias value b (intercept)
- to learn is to adapt the parameters: weights w and b
- function $f(.)$ is called activation function (transforms output)



Neuron

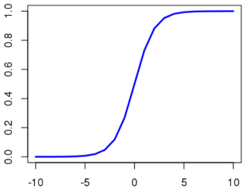
- input: 1+ values
- output 1 value
- each connection associated with a weight w (connection strength)
- often there is a bias value b
- to learn is to adapt the parameters: weights w and b



Some activation functions

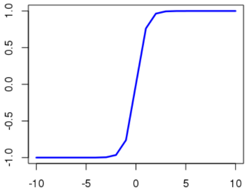
Sigmoid

$$f(x) = \frac{1}{1+e^{-x}}$$



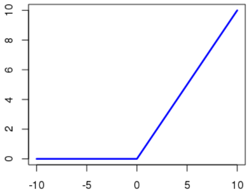
Hiperbolic Tangent

$$f(x) = \tanh(x)$$



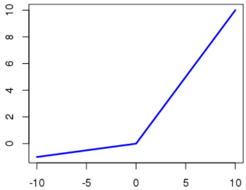
ReLU

$$f(x) = \max(0, x)$$



Leaky ReLU

$$f(x) = \max(0.1x, x)$$



Backpropagation

- Algorithm that recursively apply chain rule to compute weight adaptation for all parameters.
- **Forward**: compute result of the operation in some input over all neurons, up to the loss function
- **Backward**: apply chain rule to compute the gradient of the loss function, propagating through all layers of the network, in a graph structure

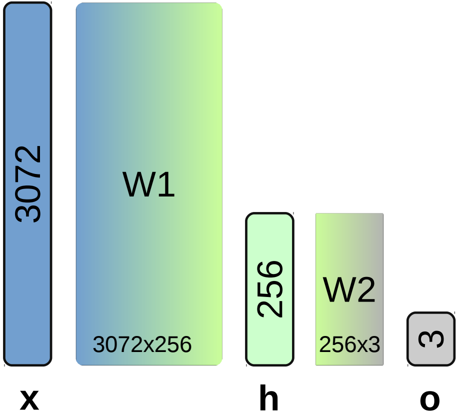
Simple NN with two layers

The linear classifier was defined as $f(W, \mathrm{x}) = W\mathrm{x}$

A two-layer neural network could be seen as: $f(W_2 \max(0, W_1\mathrm{x}))$

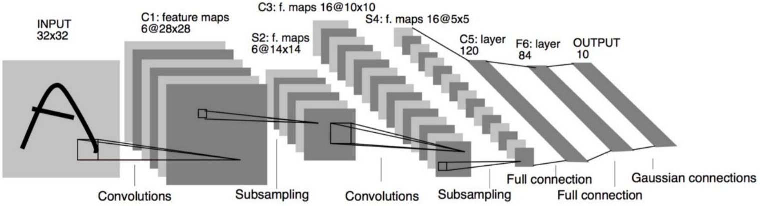
- input: image $32 \times 32 \times 3$
- hidden layer: 256 neurons
- output: vector with 3 scores

Simple NN with two layers



Agenda

Architecture LeNet



New terminology:

- Convolutions / convolutional layer
- Subsampling / pooling
- Feature maps
- Full connection

Convolutional layer

Input $(N \times M \times L)$  e.g. $32 \times 32 \times 3$

Filter (neuron) w with $P \times Q \times D$, e.g. $5 \times 5 \times 3$ (keeps depth)

- Each neuron/filter performs a convolution with the input image

Centred at a specific pixel, we have, mathematically

$$w^T x + b$$

Convolutional layer: input x filter x stride

The convolutional layer must take into account

- input size
- filter size
- convolution stride

An input with size $N_I \times N_I$, filter size $P \times P$ and stride s will produce an output with size:

$$N_O = \frac{(N_I - P)}{s} + 1$$

Examples:

- $(7 - 3)/1 + 1 = 5$
- $(7 - 3)/2 + 1 = 3$
- $(7 - 3)/3 + 1 = 2.3333$

Convolutional layer

- Feature maps are stacked images generated after convolution with filters followed by an activation function (e.g. ReLU)

(32 × 32 × 3)

convolution layer
10 filters of 5 × 5 × 3 →

01
02
...
10

size

(28 × 28 × 10)

Convolutional layer: zero padding

In practice, zero padding is used to avoid losing borders. Example:

- input size: 10×10
- filter size: 5×5
- convolution stride: 1
- zero padding: 1
- output: 10×10

General rule: zero padding size to preserve image size: $(P-1)/2$

Example: $32 \times 32 \times 3$ input with $P = 5$, $s = 1$ and zero padding $z = 2$

Output size: $(N_I + (2 \cdot z) - P)/s + 1 = (32 + (2 \cdot 2) - 5)/1 + 1 = 32$

Convolutional layer: number of parameters

Parameters in a convolutional layer is $[(P \times P \times d) + 1] \times K$:

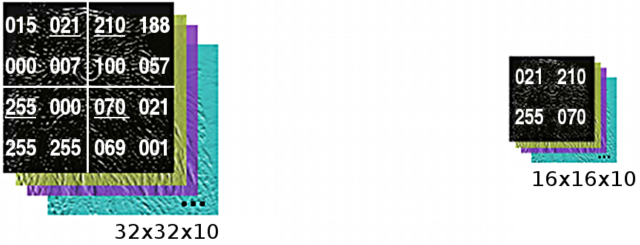
- filter weights: $P \times P \times d$, *d is given by input depth*
- number of filters/neurons: K *(each processes input in a different way)*
- $+1$ is the bias term

Example, with an image input $32 \times 32 \times 3$:

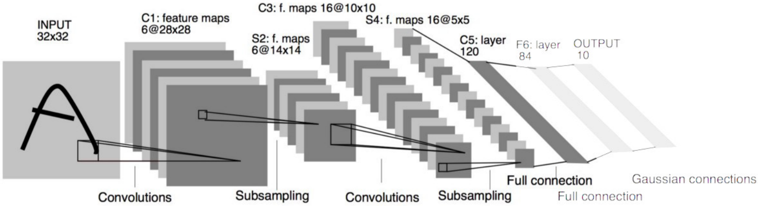
- Conv Layer 1: $P = 5$, $K = 8$
- Conv Layer 2: $P = 5$, $k = 16$
- Conv Layer 3: $P = 1$, $k = 32$
- # parameters Conv layer 1: $[(5 \times 5 \times 3) + 1] \times 8 = 608$
- # parameters Conv layer 2: $[(5 \times 5 \times 8) + 1] \times 16 = 3216$
- # parameters Conv layer 3: $[(1 \times 1 \times 16) + 1] \times 32 = 544$

Convolutional layer: pooling

Operates over each feature map, to make the data smaller
Example: max pooling with downsampling factor 2 and stride 2.

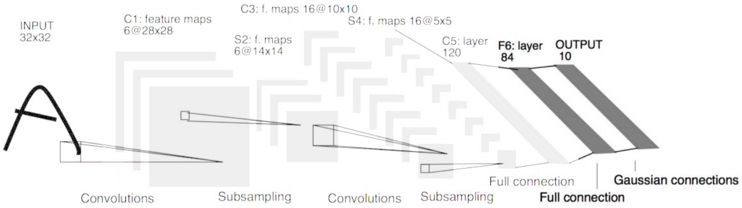


Convolutional layer: convolution + activation + pooling



- Convolution: as seen before
- Activation: ReLU
- Pooling: maxpooling

Fully connected layer + Output layer



Fully connected (FC) layer:

- FC layers work as in a regular Multilayer Perceptron
- A given neuron operates over all values of previous layer

Output layer:

- each neuron represents a class of the problem

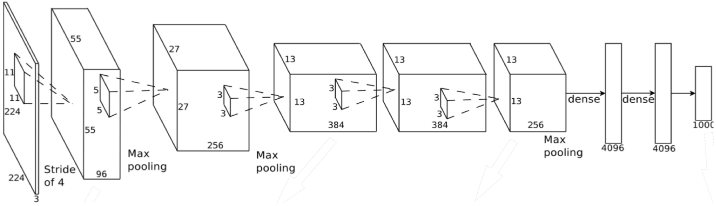
Visualization

figs/single_layer.png

Agenda

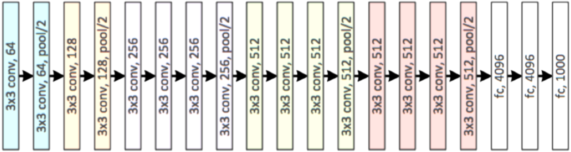
AlexNet (Krizhevsky, 2012)

- 60 million parameters.
- input 224×224
- conv1: $K = 96$ filters with $11 \times 11 \times 3$, stride 4,
- conv2: $K = 256$ filters with $5 \times 5 \times 48$,
- conv3: $K = 384$ filters with $3 \times 3 \times 256$,
- conv4: $K = 384$ filters with $3 \times 3 \times 192$,
- conv5: $K = 256$ filters with $3 \times 3 \times 192$,
- fc1, fc2: $K = 4096$.



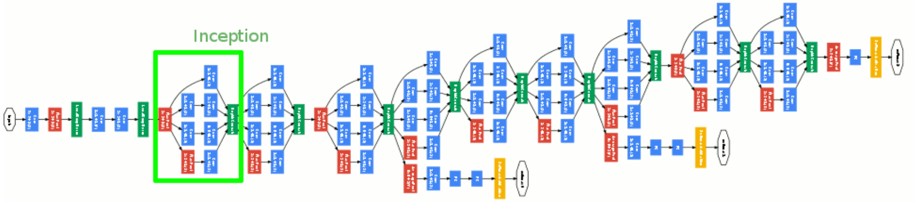
VGG 19 (Simonyan, 2014)

- $+$layers, $-$filter size $=$ less parameters
- input 224×224,
- filters: all 3×3,
- conv 1-2: $K = 64$ + maxpool
- conv 3-4: $K = 128$ + maxpool
- conv 5-6-7-8: $K = 256$ + maxpool
- conv 9-10-11-12: $K = 512$ + maxpool
- conv 13-14-15-16: $K = 512$ + maxpool
- fc1, fc2: $K = 4096$

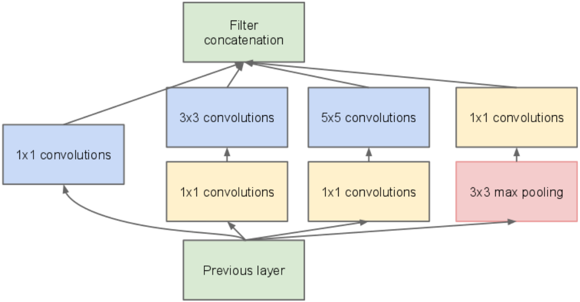


GoogLeNet (Szegedy, 2014)

- 22 layers
- Starts with two convolutional layers
- *Inception layer* ("filter bank"):
  - filters 1×1, 3×3, 5×5 + max pooling 3×3;
  - reduce dimensionality using 1×1 filters.
  - 3 classifiers in different parts
- Blue = convolution,
- Red = pooling,
- Yellow = Softmax loss fully connected layers
- Green = normalization or concatenation



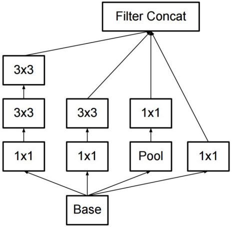
GoogLeNet: inception module



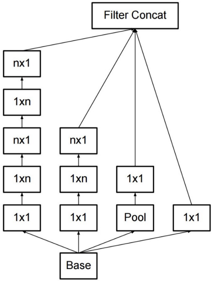
- 1×1 convolution reduces the depth of previous layers by half
- this is needed to reduce complexity (e.g. from 256 to 128 d)
- concatenates 3 filters plus an extra max pooling filter (because).

Inception modules (V2 and V3)

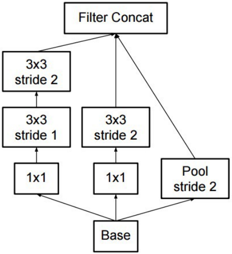
multiple 3×3 convs.



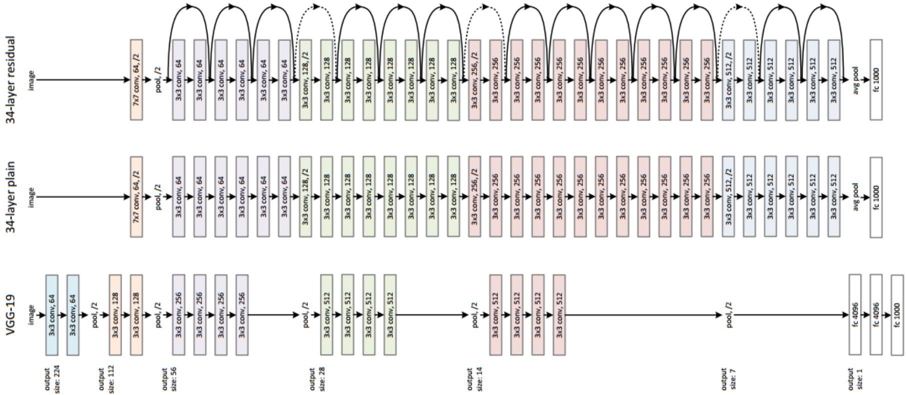
flattened conv.



decrease size

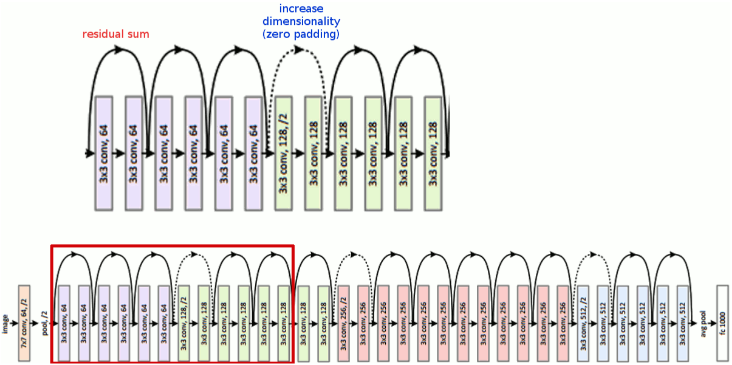


VGG19 vs "VGG34" vs ResNet34

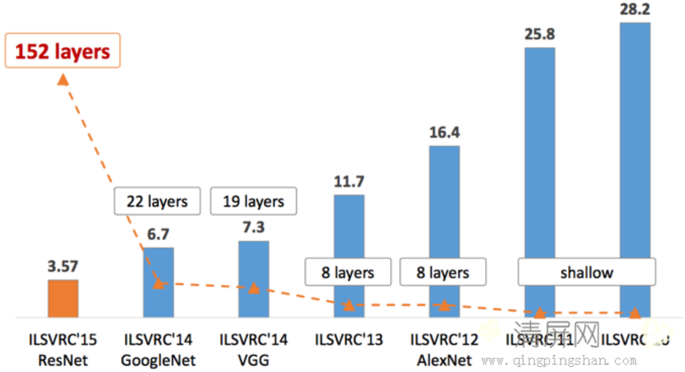


Residual Network — ResNet (He et al, 2015)

Reduces number of filters, increases number of layers (34-1000).
Residual architecture: add identity before activation of next layer.

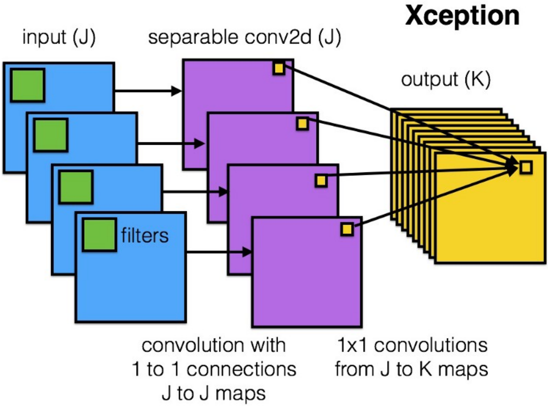


Comparison

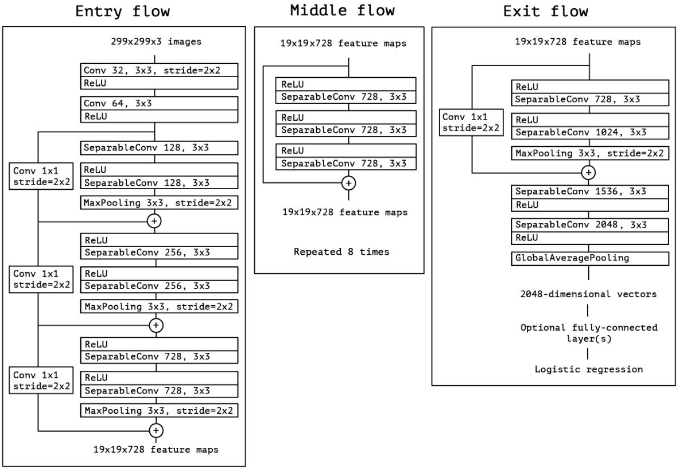


Thanks to Qingping Shan `www.qingpingshan.com`

Xception


Xception
input (J)
separable conv2d (J)
output (K)
filters
convolution with
1 to 1 connections
J to J maps
1x1 convolutions
from J to K maps

Xception


Entry flow
299x299x3 images
Conv 32, 3x3, stride=2x2
ReLU
Conv 64, 3x3
ReLU
SeparableConv 128, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 128, 3x3
MaxPooling 3x3, stride=2x2
ReLU
SeparableConv 256, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 256, 3x3
MaxPooling 3x3, stride=2x2
ReLU
SeparableConv 728, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 728, 3x3
MaxPooling 3x3, stride=2x2
19x19x728 feature maps
Middle flow
19x19x728 feature maps
ReLU
SeparableConv 728, 3x3
ReLU
SeparableConv 728, 3x3
ReLU
SeparableConv 728, 3x3
19x19x728 feature maps
Repeated 8 times
Exit flow
19x19x728 feature maps
ReLU
SeparableConv 728, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 1024, 3x3
MaxPooling 3x3, stride=2x2
SeparableConv 1536, 3x3
ReLU
SeparableConv 2048, 3x3
ReLU
GlobalAveragePooling
2048-dimensional vectors
Optional fully-connected
layer(s)
Logistic regression

Agenda

Tricks

Batch

- **Mini-batch**: in order to make it easier to process, on SGD use several images at the same time,
- **Mini-batch size**: 128 or 256, if not enough memory, 64 or 32,
- **Batch normalization**: when using ReLU, normalize the batch.

Convergence and training set

- **Learning rate**: in SGD apply a decaying learning rate, a fixed momentum,
- **Clean data**: cleaniness of the data is very important,
- **Data augmentation**: generate new images by perturbation of existing ones,
- **Loss, validation and training error**: plot values for each epoch.

Guidelines for new data

Classification (finetuning)

- Data similar to ImageNet: fix all Conv Layers, train FC layers

- Data not similar to ImageNet: fix lower Conv Layers, train others

Guidelines for new data

Feature extraction for image classification and retrieval

- Perform forward, get activation values of higher Conv and/or FC layers
- Apply some dimensionality reduction: e.g. PCA, Product Quantization, etc.
- Use external classifier: e.g. SVM, k-NN, etc.

References

- LeCun, Y. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998. Demos : http://yann.lecun.com/exdb/lenet
- Szegedy, C. et al. Going Deeper with Convolutions. CVPR 2015 http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf.
- He, K et al Deep Residual Learning for Image Recognition https://arxiv.org/abs/1512.03385
- Donglai et al. Understanding Intra-Class Knowledge Inside CNN, 2015, Tech Report. http://vision03.csail.mit.edu/cnn_art/index.html
- Mahendran and Vedaldi. Understanding Deep Image Representations by Inverting Them, 2014.
- Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions https://arxiv.org/abs/1610.02357.