Everything you wanted to know about Deep Learning for Computer Vision but were afraid to ask

Tutorial: SIBGRAPI 2017

Moacir Ponti, Leonardo Ribeiro, Tiago Nazare
ICMC, Universidade de São Paulo

Tu Bui, John Collomosse
CVSSP, University of Surrey

Contact: www.icmc.usp.br/~moacir — moacir@icmc.usp.br

Rio de Janeiro/Brazil – October, 2017

Agenda

1. Image classification basics
2. Searching for human-like image classification methods
3. Neural networks: from shallow to deep
   - Motivation and definitions
   - Linear function, loss function, optimization
   - Simple Neural Network
4. Convolutional Neural Networks
   - Current Architectures
   - Guidelines for training
5. How it Works, Limitations and final remarks

Image classification example

Task: learn how to distinguish two types of images:

- **desert**;
- **beach**.

Objective: given some images, develop a model able to classify unseen images into one of those two classes.

Image classification example

Task: learn how to distinguish two types of images:

- **desert**;
- **beach**.

Objective: given some images, develop a model able to classify unseen images into one of those two classes.

Image classification example

Task: learn how to distinguish two types of images:

- **desert**;
- **beach**.

Objective: given some images, develop a model able to classify unseen images into one of those two classes.

Image classification example

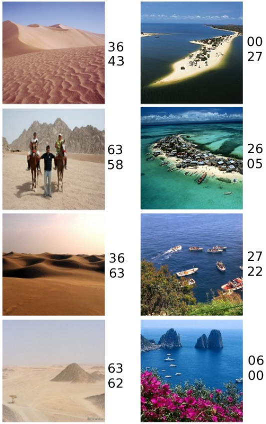
- **Features**: set of values extracted from images that can be used to measure the (dis)similarity between images **Any suggestion?**

Image classification example

- **Features**: set of values extracted from images that can be used to measure the (dis)similarity between images **Any suggestion?**
  - Use the two most frequent colors as a descriptor!

Image classification example

- **Features**: set of values extracted from images that can be used to measure the (dis)similarity between images **Any suggestion?**
  - Use the two most frequent colors as a descriptor!



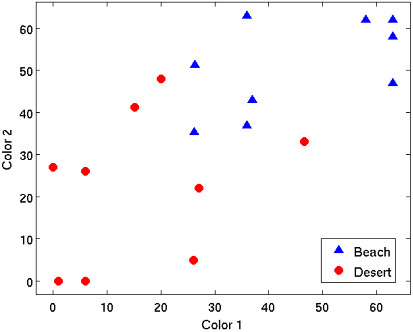


Image classification example

- **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?**

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?**
  - A linear classifier, for instance!

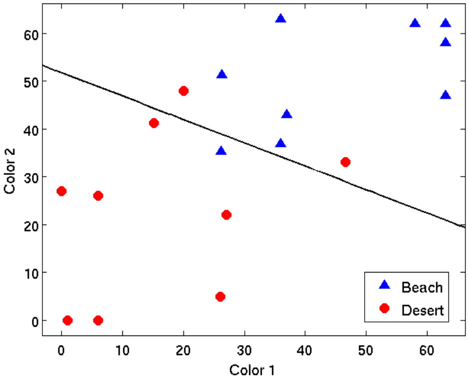


Image classification example

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

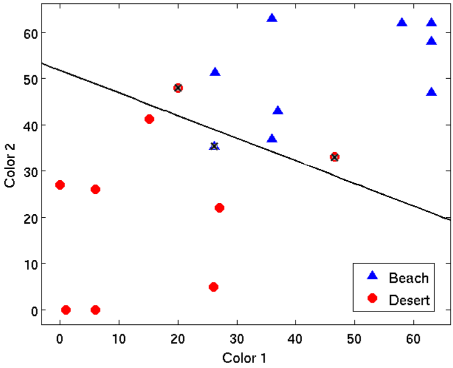


Image classification example

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

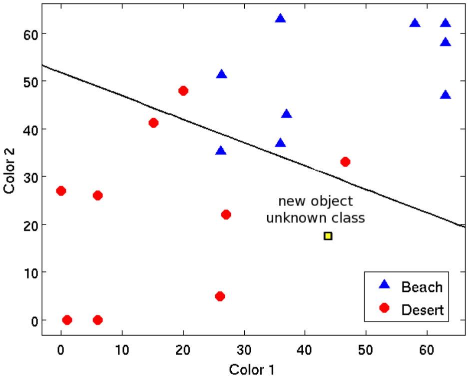
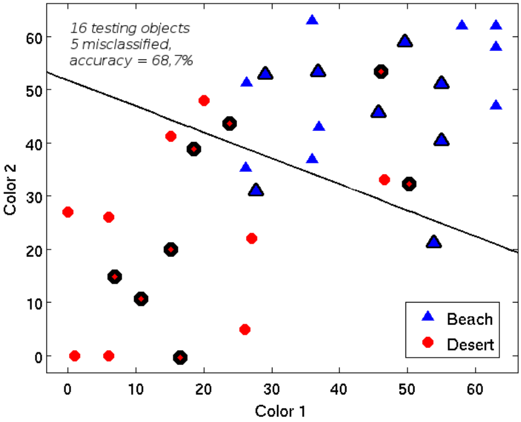


Image classification example

- Now we want to test, for future data (not used in training), the classifier error rate (or alternatively, its accuracy)

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**.



Agenda

1. Image classification basics
2. Searching for human-like image classification methods
3. Neural networks: from shallow to deep
   - Motivation and definitions
   - Linear function, loss function, optimization
   - Simple Neural Network
4. Convolutional Neural Networks
   - Current Architectures
   - Guidelines for training
5. How it Works, Limitations and final remarks

Examples

Some methods to try to tackle image classification:

- Color, shape and texture descriptors (1970-2000)
- SIFT (>1999)
- Histogram of Gradients (>2005)
- Bag of Features (>2004)
- Spatial Pyramid Matching (>2006),

Pipeline

General idea:

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

Not so versatile!

And then, in 2012...

The world didn't end

And then, in 2012...

The world didn't end and AlexNet won the ImageNet chalange!

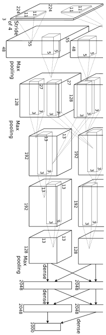
And then, in 2012...

The world didn't end and AlexNet won the ImageNet chalange!

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

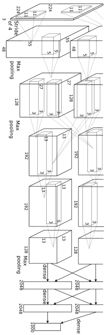
And CNNs continued to dominate image classification

AlexNet (9)



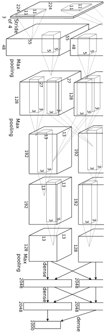
And CNNs continued to dominate image classification

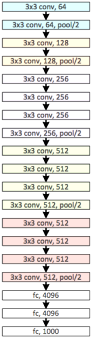
AlexNet (9) GoogLeNet (22)



And CNNs continued to dominate image classification

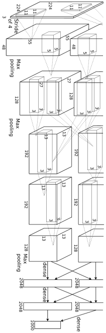
AlexNet (9) GoogLeNet (22) VGG (16/19)

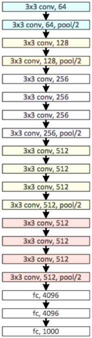


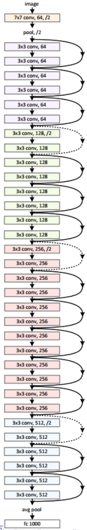


And CNNs continued to dominate image classification

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

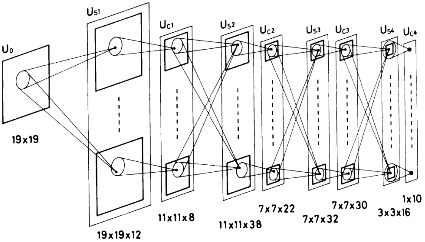




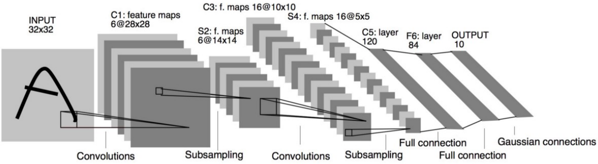


But CNNs were not invented in 2012...

Fukushima's Neocognitron (1989)



LeCun's LeNet (1998)



Agenda

Motivation with two problems

We want to find a function in the form $f(\mathbf{x}) = y$ (depends on the task)

Motivation with two problems

We want to find a function in the form $f(\mathbf{x}) = y$ (depends on the task)

Image classification: animals

- Data available: pairs of images and labels from dogs, cats and turtles,
- Input: RGB image in the form $\mathbf{x}$,
- Output: predicted label y (e.g. cat, dog) assigned to the input image.

Motivation with two problems

We want to find a function in the form $f(\mathbf{x}) = y$ (depends on the task)

Image classification: animals

- Data available: pairs of images and labels from dogs, cats and turtles,
- Input: RGB image in the form $\mathbf{x}$,
- Output: predicted label y (e.g. cat, dog) assigned to the input image.

Anomaly detection in audio for Parkinsons Disease diagnosis

- Data available: speech audio recorded from people without Parkinsons Disease,
- Input: time series with 16-bit audio content $\mathbf{x}$,
- Output: probability y of observing an anomalous audio input (indicating possible disease).

Machine Learning (ML) vs Deep Learning (DL)

Machine Learning

A more broad area that includes DL. Algorithms aims to infer $f()$ from a space of admissible functions given training data.

- "shallow" methods often infer a single function $f(.)$.
- e.g. a linear function $f(x) = w \cdot x + \theta$,
- Common algorithms: The Perceptron, Support Vector Machines, Logistic Classifier, etc.

Machine Learning (ML) vs Deep Learning (DL)

Deep Learning

Involves learning a sequence of representations via composite functions. Given an input $\mathbf{x}_1$ several intermediate representations are produced:

$$\mathbf{x}_2 = f_1(\mathbf{x}_1)$$
$$\mathbf{x}_3 = f_2(\mathbf{x}_2)$$
$$\mathbf{x}_4 = f_3(\mathbf{x}_3)$$
$$\cdots$$

The output is achieved by several L nested functions in the form:

$$f_L\left(\cdots f_3(f_2(f_1(\mathbf{x}_1, W_1), W_2), W_3)\cdots, W_L\right),$$

W_i are hyperparameters associated with each function i.

A shallow linear classifier

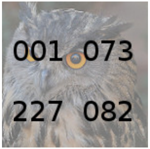
Input  $\rightarrow \mathbf{x}$

$$f(W, \mathbf{x}) = \underbrace{W}_{\text{weight matrix}} \underbrace{\mathbf{x}}_{\text{image}} + \underbrace{\mathbf{b}}_{\text{bias term}}$$

$= $ scores for possible classes of $\mathbf{x}$

Linear classifier for image classification

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



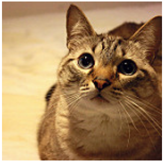
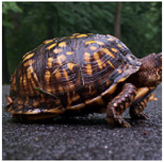
$$= \mathbf{x} = [1, 73, 227, 82]$$

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

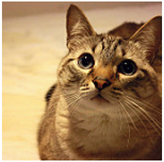
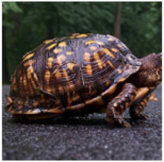
$$W\mathbf{x} + \mathbf{b}$$

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

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

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_i, y_i) be a training example: x_i are the features, y is the label, and $f(.)$ a classifier that maps any x_i into a class using parameters W.
A loss for a single example is some function in the form:

$$\ell(f(W, \mathbf{x}_i), y_i) \quad (1)$$

Linear classifier for image classification

In practice, we measure the loss $\mathcal{L}$, over a set X, Y of N examples. Common functions are:

Mean squared error (continuous values)

$$\mathcal{L}\left(f(W,X,Y)\right) = \mathcal{L}\left(\hat{Y}, Y\right) = \frac{1}{N}\sum_{i=1}^{N}\big(\underbrace{\hat{y}_i}_{\text{predicted label}} - \underbrace{y_i}_{\text{true label}}\big)^2$$

Linear classifier for image classification

In practice, we measure the loss $\mathcal{L}$, over a set X, Y of N examples.
Common functions are:

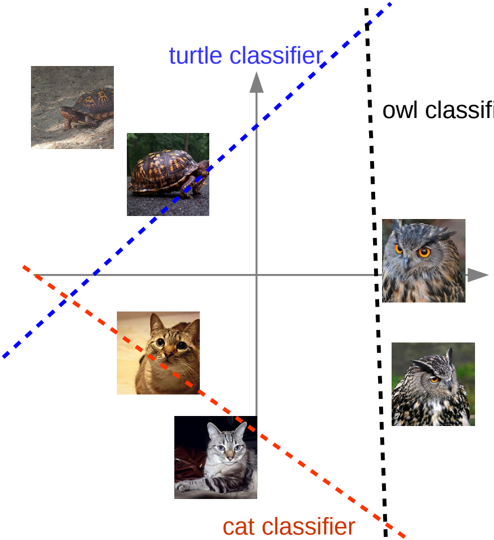
Mean squared error (continuous values)

$$\mathcal{L}\left(f(W,X,Y)\right) = \mathcal{L}\left(\hat{Y}, Y\right) = \frac{1}{N}\sum_{i=1}^{N}(\underbrace{\hat{y}_i}_{\text{predicted label}} - \underbrace{y_i}_{\text{true label}})^2$$

Cross entropy (bits or probability vectors)

$$\mathcal{L}\left(\hat{Y}, Y\right) = \frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1-\hat{y}_i)\log(1-\hat{y}_i)$$

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:

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

Minimizing the loss function

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

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

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

Minimizing the loss function

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

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

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.

Minimizing the loss function

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

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

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

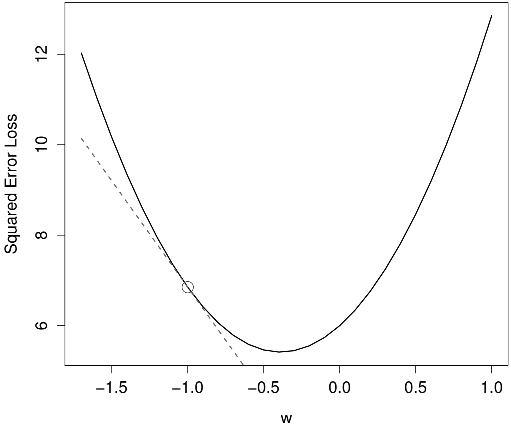
We may use:

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
$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} 0.1 + \mathbf{0.001}, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} ?, \\ , \\ , \\ , \\ , \\ ..., \\ \end{bmatrix}$
$\ell(f(W)) = 2.31298$	$\ell(f(W')) = 2.31\mathbf{201}$	$(f(w_i + \delta) - f(w_i))/\delta$

Gradient descent

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

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

$$\ell(f(W)) = 2.31298 \qquad \ell(f(W')) = 2.31\mathbf{201} \qquad (f(w_i + \delta) - f(w_i))/\delta$$

Gradient descent

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

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

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

Gradient descent

W	$w_i + \delta$	dw_i
$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} 0.1, \\ -0.25, \\ 0.1 + \mathbf{0.001}, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} -0.97, \\ 0.0, \\ +1.61, \\ -, \\ -, \\ ..., \\ - \end{bmatrix}$
$\ell(f(W)) = 2.31298$	$\ell(f(W1)) = 2.31\mathbf{459}$	$(f(w_i + \delta) - f(w_i))/\delta$

Gradient descent

$$W \quad \begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix} \quad \ell(f(W)) = 2.31298$$

$$w_i + \delta \quad \begin{bmatrix} 0.1, \\ -0.25, \\ 0.1, \\ 2.5, \\ 0, \\ ..., \\ -0.1 \end{bmatrix} \quad \ell(f(W')) = \mathbf{2.08720}$$

$$dw_i \quad \begin{bmatrix} -0.93, \\ 0.0, \\ -1.61, \\ +0.02, \\ +0.5, \\ ..., \\ -3.7 \end{bmatrix} \quad (f(w_i + \delta) - f(w_i))/\delta$$

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

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

Stochastic Gradient Descent (SGD)

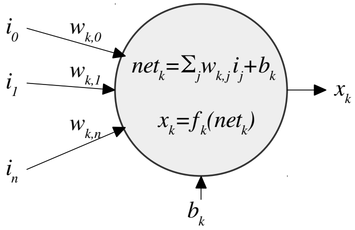
It is hard to compute the gradient, when N is large.

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

Neuron

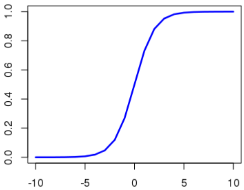
- input: several values $(i_0, i_1, \ldots, i_n)$
- output: a single value $\mathbf{x}_k$.
- each connection associated with a weight w (connection strength)
- often there is a bias value b (intercept)
- to learn is to adapt the parameters: weights w and b
- function $f(.)$ is called activation function (transforms output)



Some activation functions

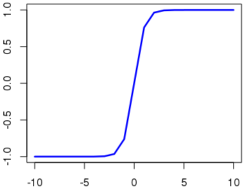
Sigmoid

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



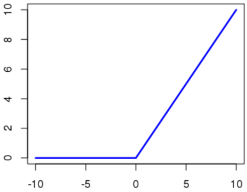
Hiperbolic Tangent

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



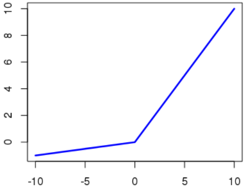
ReLU

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



Leaky ReLU

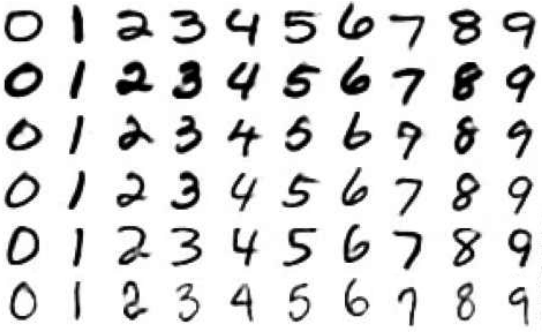
$$f(x) = \max(0.1x, x)$$



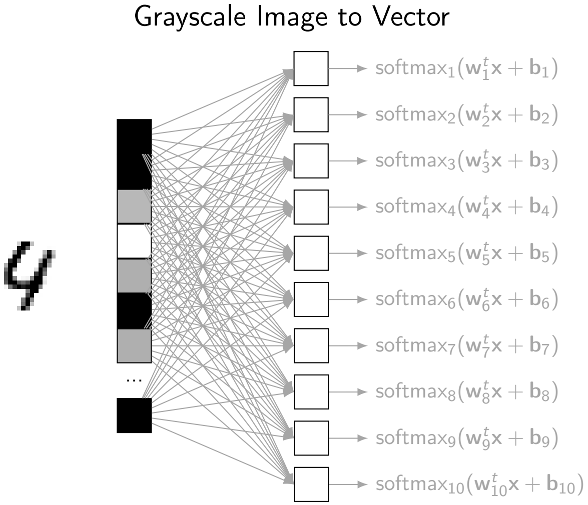
Backpropagation

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

A simple problem: digit classification



Neural Network with Single Layer



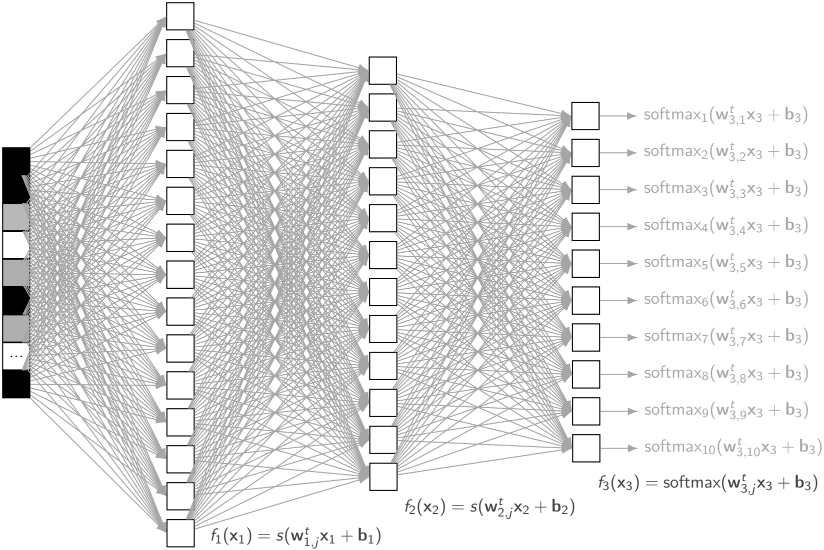
A simpler problem: digit classification

$$\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,783} \\ x_{1,0} & x_{0,1} & x_{1,2} & \cdots & x_{0,783} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{63,0} & x_{63,1} & x_{63,2} & \cdots & x_{63,783} \end{bmatrix} \cdot \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,9} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,9} \\ w_{2,0} & w_{2,1} & \cdots & w_{2,9} \\ \vdots & \vdots & \ddots & \vdots \\ w_{783,0} & w_{783,1} & \cdots & w_{783,9} \end{bmatrix} + \begin{bmatrix} b_0 & b_1 & b_2 & \ldots & b_9 \end{bmatrix}$$

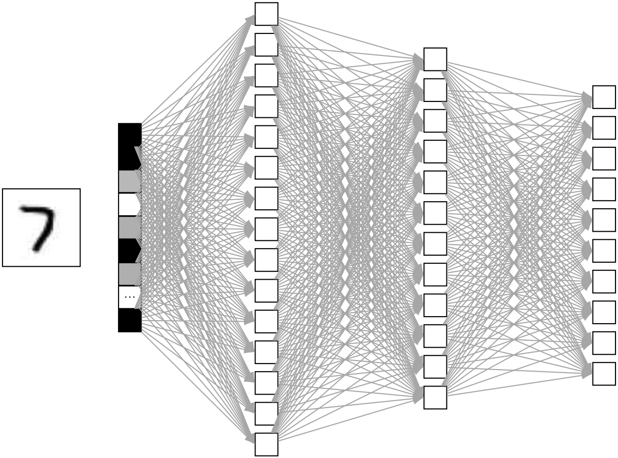
$$\mathbf{Y} = \text{softmax}(\mathbf{X} \cdot \mathbf{W} + \mathbf{b})$$

$$\mathbf{Y} = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} & \cdots & y_{0,9} \\ y_{1,0} & y_{1,1} & y_{1,2} & \cdots & y_{1,9} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{63,0} & y_{63,1} & y_{63,2} & \cdots & y_{63,9} \end{bmatrix}$$

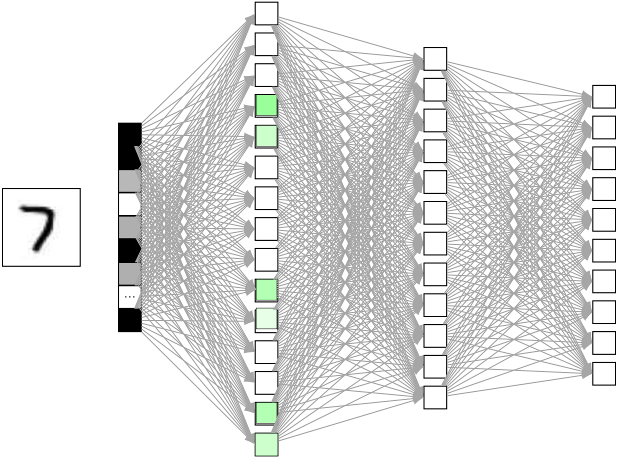
"Deep" NN with two hidden layers



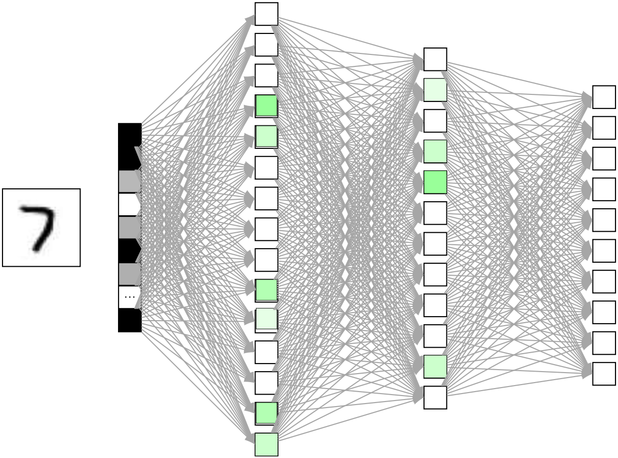
"Deep" NN with two hidden layers : Input



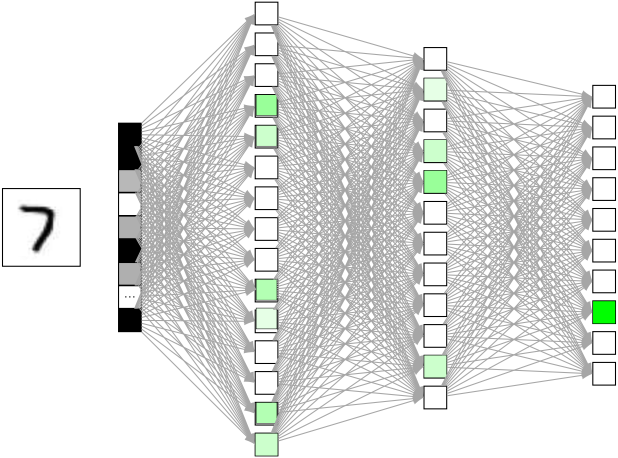
"Deep" NN with two hidden layers : Hidden layer 1



"Deep" NN with two hidden layers : Hidden layer 2



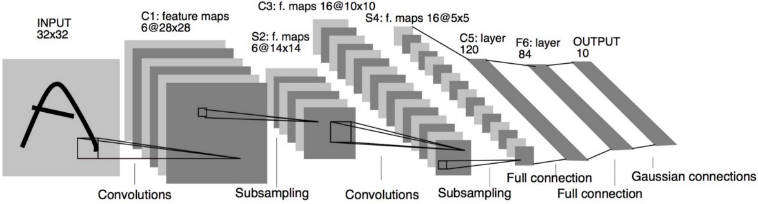
"Deep" NN with two hidden layers : output



Agenda

1. Image classification basics
2. Searching for human-like image classification methods
3. Neural networks: from shallow to deep
   - Motivation and definitions
   - Linear function, loss function, optimization
   - Simple Neural Network
4. Convolutional Neural Networks
   - Current Architectures
   - Guidelines for training
5. How it Works, Limitations and final remarks

Architecture LeNet



New terminology:

- Convolutional layer
- Pooling
- Feature (or Activation) maps
- Fully connected (or Dense) layer

Convolutional layer

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

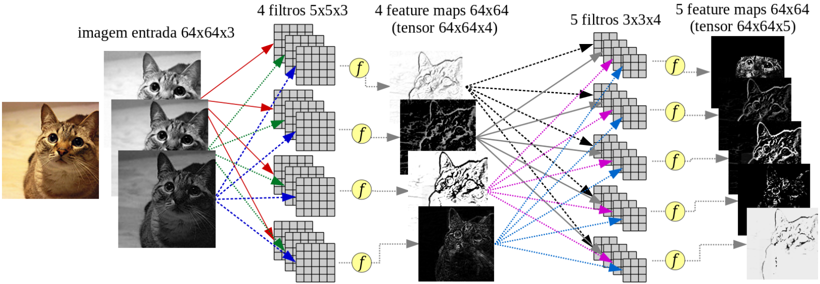
Filter (neuron) w with $P \times Q \times D$, e.g. $5 \times 5 \times 3$ (keeps depth)

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

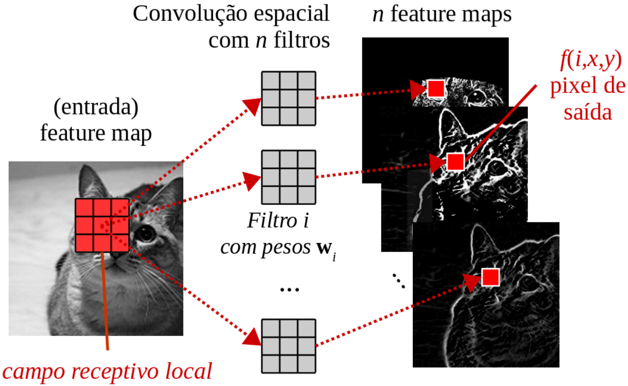
Centred at a specific pixel, we have, mathematically

$$w^T x + b$$

Convolutional layer: feature maps



Convolutional layer: local receptive field



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: 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:

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

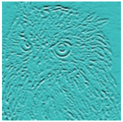
Convolutional layer

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

$(32 \times 32 \times 3)$

convolution layer

10 filters of $5 \times 5 \times 3$ $\rightarrow$

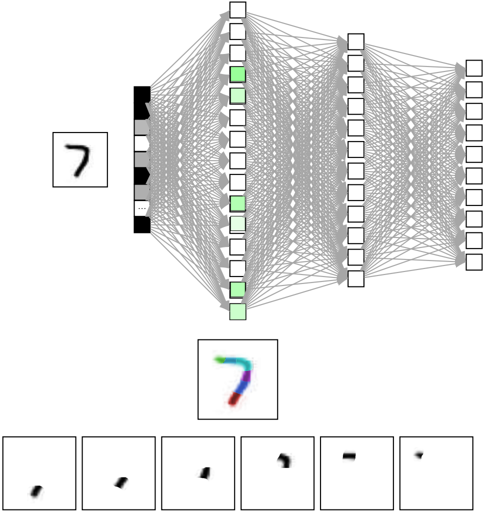
01 

02 

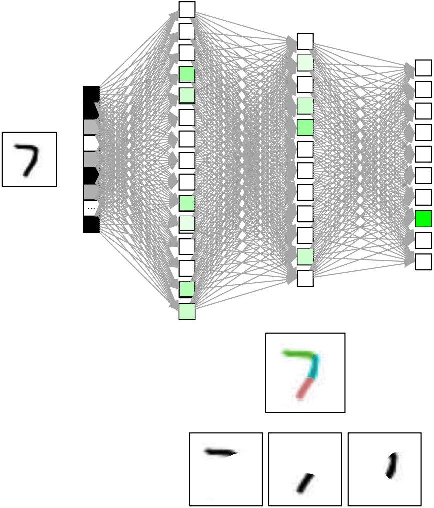
...

10 

The MNIST example: now hidden layers are conv layers



The MNIST example: now hidden layers are conv layers



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: 2
- output:

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: 2
- output: 10×10

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: 2
- 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:
- # parameters Conv layer 2:
- # parameters Conv layer 3:

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

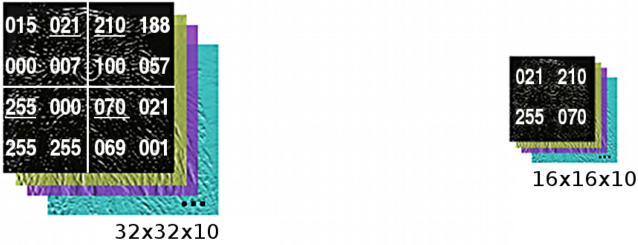
Example, with an image input $32 \times 32 \times 3$:

- 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: $[(1 \times 1 \times 16) + 1] \times 32 = 544$

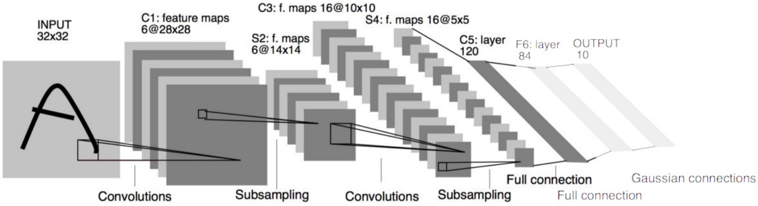
Convolutional layer: pooling

Operates over each feature map, to make the data smaller

Example: max pooling with downsampling factor 2 and stride 2.

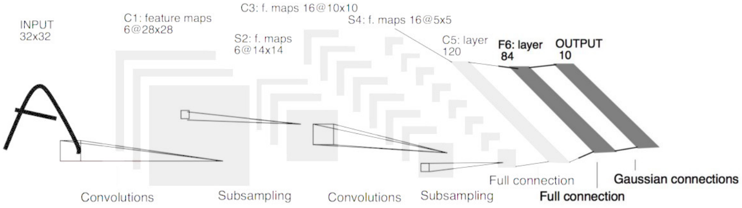


Convolutional layer: convolution + activation + pooling



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

Fully connected layer + Output layer



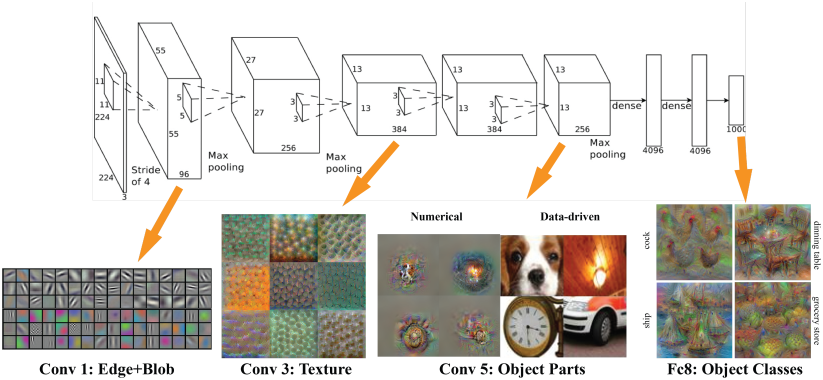
Fully connected (FC) layer:

- 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



Donglai et al. Understanding Intra-Class Knowledge Inside CNN, 2015, Tech Report

Agenda

1. Image classification basics

2. Searching for human-like image classification methods

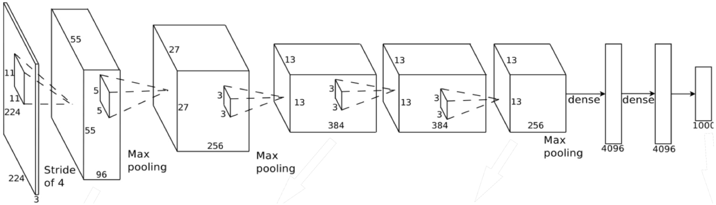
3. Neural networks: from shallow to deep
   - Motivation and definitions
   - Linear function, loss function, optimization
   - Simple Neural Network

4. Convolutional Neural Networks
   - Current Architectures
   - Guidelines for training

5. How it Works, Limitations and final remarks

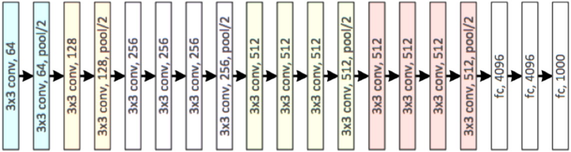
AlexNet (Krizhevsky, 2012)

- 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: $K = 4096$.



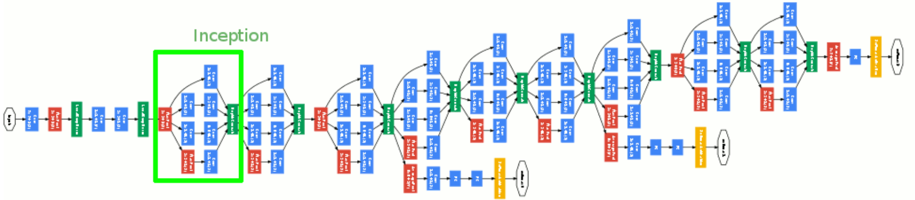
VGG 19 (Simonyan, 2014)

- +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: $K = 4096$

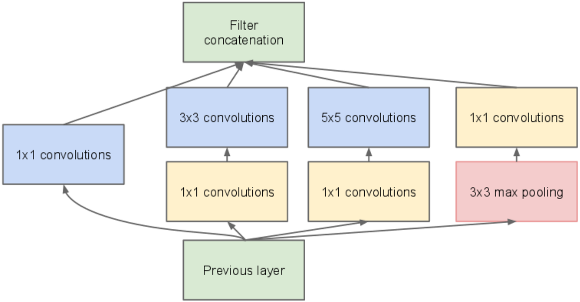


GoogLeNet (Szegedy, 2014)

- 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



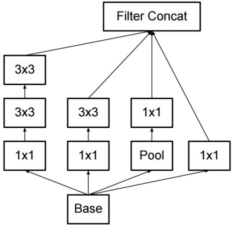
GoogLeNet: inception module



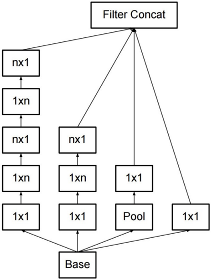
- 1×1 convolution reduces the depth of previous layers by half
- this is needed to reduce complexity (e.g. from 256 to 128 d)
- concatenates 3 filters plus an extra max pooling filter (because).

Inception modules (V2 and V3)

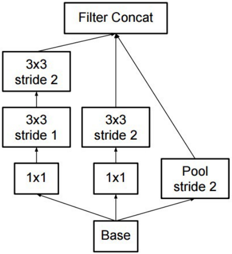
multiple 3×3 convs.



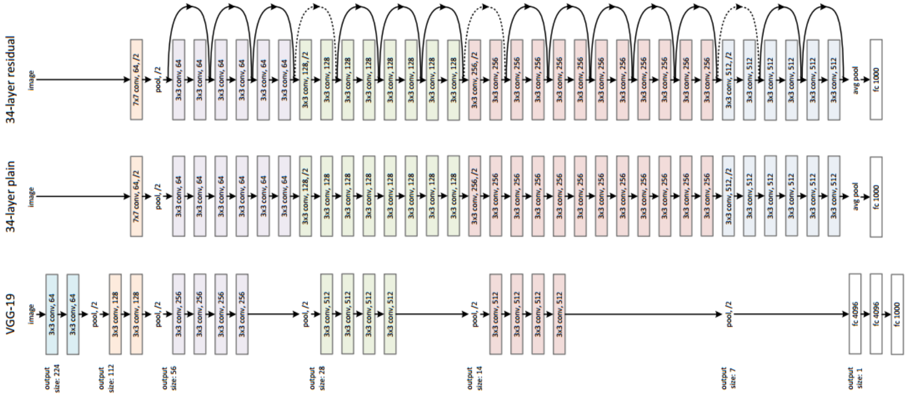
flattened conv.



decrease size

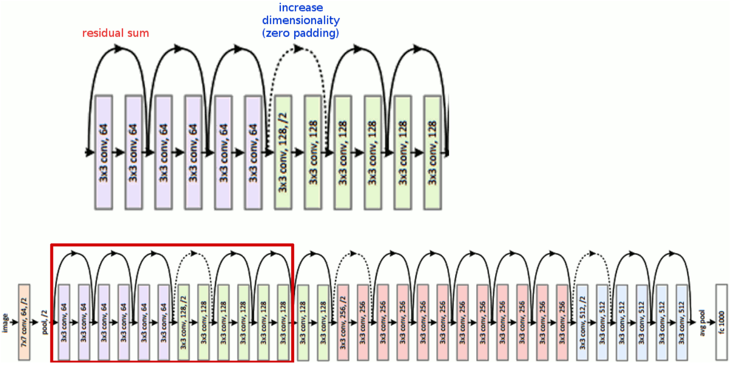


VGG19 vs "VGG34" vs ResNet34

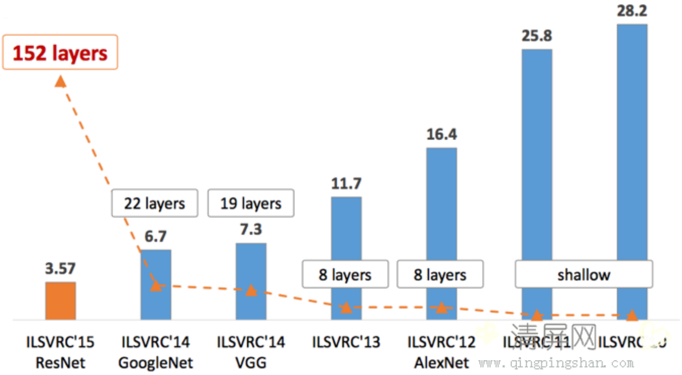


Residual Network — ResNet (He et al, 2015)

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

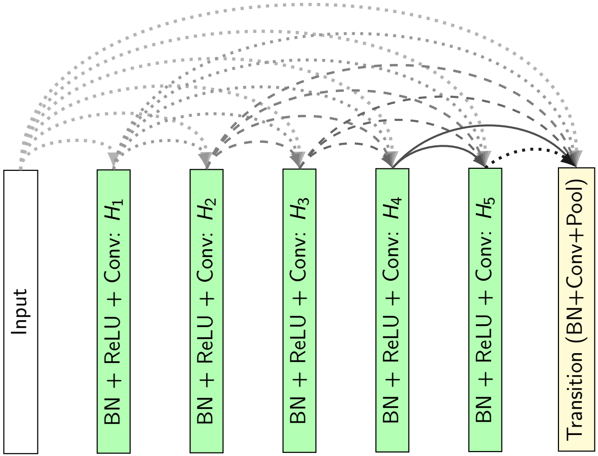


Comparison



Thanks to Qingping Shan `www.qingpingshan.com`

Densenet


Input
BN + ReLU + Conv: H_1
BN + ReLU + Conv: H_2
BN + ReLU + Conv: H_3
BN + ReLU + Conv: H_4
BN + ReLU + Conv: H_5
Transition (BN+Conv+Pool)

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: freeze all Conv Layers, train FC layers

- Data not similar to ImageNet: freeze 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.

How it Works

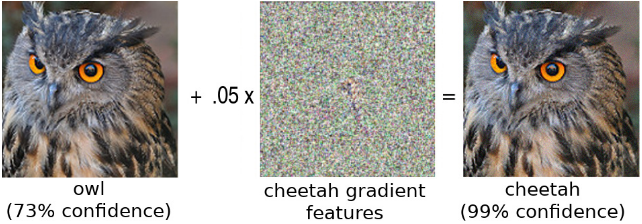
Let an integer $k \geq 1$ for any dimension $d \geq 1$.
Exists a function $f : \mathbb{R}^d \to \mathbb{R}$ computed via a ReLU neural network with $2k^3 + 8$ layers ($3k^3 + 12$ neuros) and $4 + d$ distinct parameters so that

$$\inf_{g \in \mathcal{C}} \int_{[0,1]^d} |f(x) - g(x)| dx \geq \frac{1}{64},$$

$\mathcal{C}$ are:
(1) functions computed by networks (t, α, β)-semi-algebric with $\leq k$ layers and $2^k/(t\alpha\beta)$ neurons (ReLU and maxpool);
(2) functions computed by linearly combined decision trees $\leq t$ with $2^{k^3}/t$ neurons — such as boosted decision trees..

Limitations

CNNs are easily fooled



Concluding remarks

- Deep Learning is not a panacea;
- There are important concerns about generalization of Deep Networks;
- However those methods can be really useful for finding representations;
- Many challenges and research frontiers for more complex tasks.

References

- **Ponti, M.; Paranhos da Costa, G. Como funciona o Deep Learning. Tópicos em Gerenciamento de Dados e Informações. 2017.**
- **Ponti, M.; Ribeiro, L.; Nazare, T.; Bui, T.; Collomosse, J. Everything you wanted to know about Deep Learning for Computer Vision but were afraid to ask. In: SIBGRAPI – Conference on Graphics, Patterns and Images, 2017.** http://sibgrapi.sid.inpe.br/rep/sid.inpe.br/sibgrapi/2017/09.05.22.09
- 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.