Redes Neurais Convolucionais

André Gustavo Hochuli

Orientador: Prof. Dr. Luiz Eduardo Soares de Oliveira

Programa de Pós-Graduação em Informática

Departamento de Informática

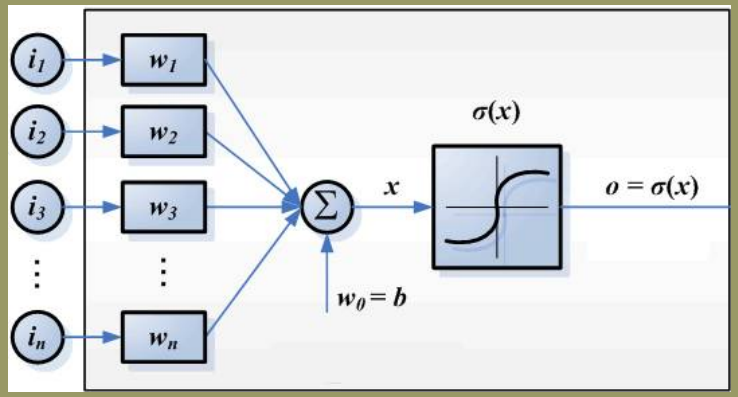
UFPR

http://www.inf.ufpr.br/aghochuli/caffe/

Redes Neurais

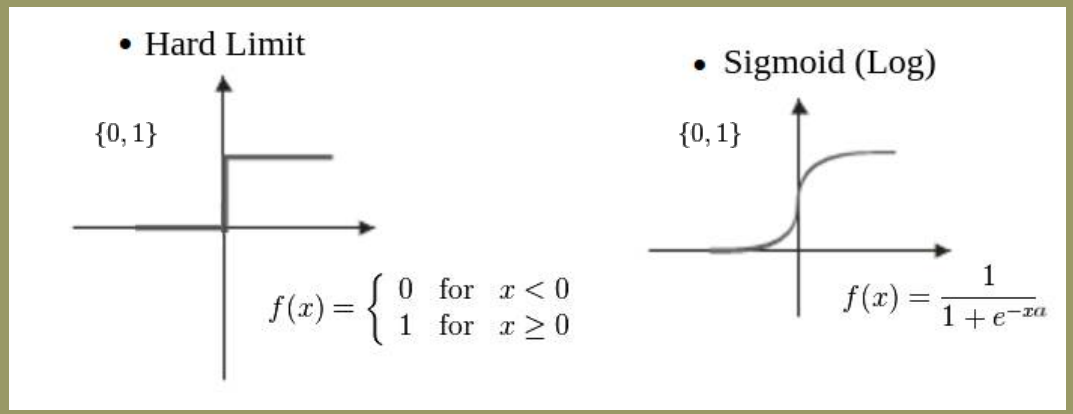
- Neurônio Artificial

  - Vetor de Caracteristícas
  - Pesos
  - Bias
  - Função de Ativação



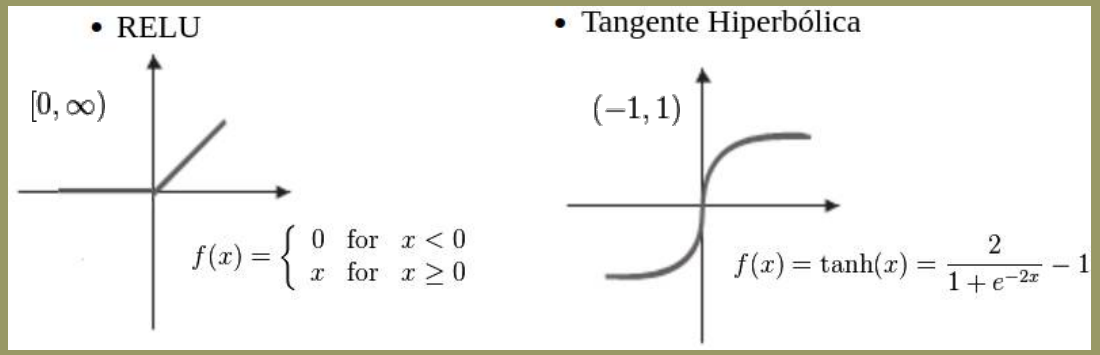
- Funções de Ativação

  - Dado uma entrada, define um estado para o neurônio.

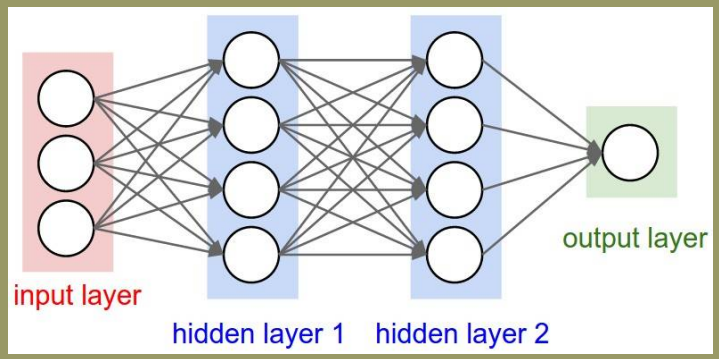


Redes Neurais

- Funções de Ativação

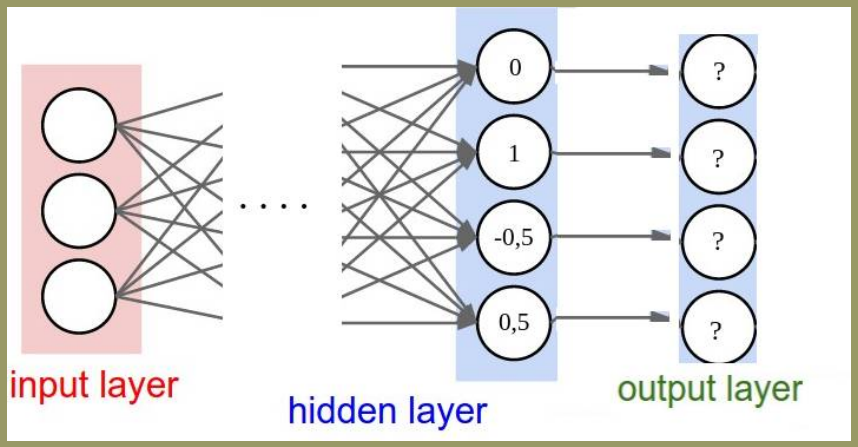


- Rede Neural Artificial (MLP)



Redes Neurais

- Problema: Como determinar a probabilidade de cada classe na camada de saída da rede neural ?



Redes Neurais

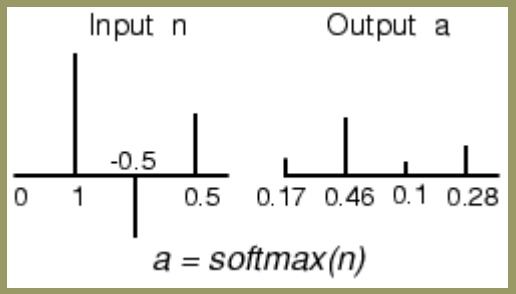
- Função SoftMax

  - Normaliza entre 0 e 1 as saídas, em um problema multi-classes, obtidas por um classificador linear (Ex: Neurônios).

  - Aplicada na camada de saída da rede (output layer).

  - Objetivo: Definir a probabilidade de uma classe dentro de um problema multi-classes

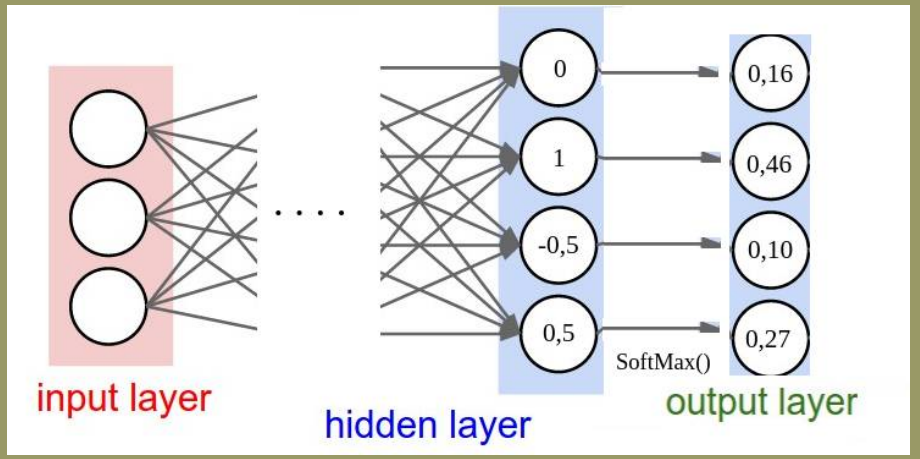
$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$



Redes Neurais

- Função SoftMax - Exemplo

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$



$$e^{0} = 1.0000$$
$$e^{1} = 2.7182$$
$$e^{-0.5} = 0.6065$$
$$e^{0.5} = 1.6487$$

$$\sum_{j \in group} e^{z_j} = \sim 5.9734$$

$$y_0 = \frac{1.0000}{\sim 5.9734} = 0.16$$

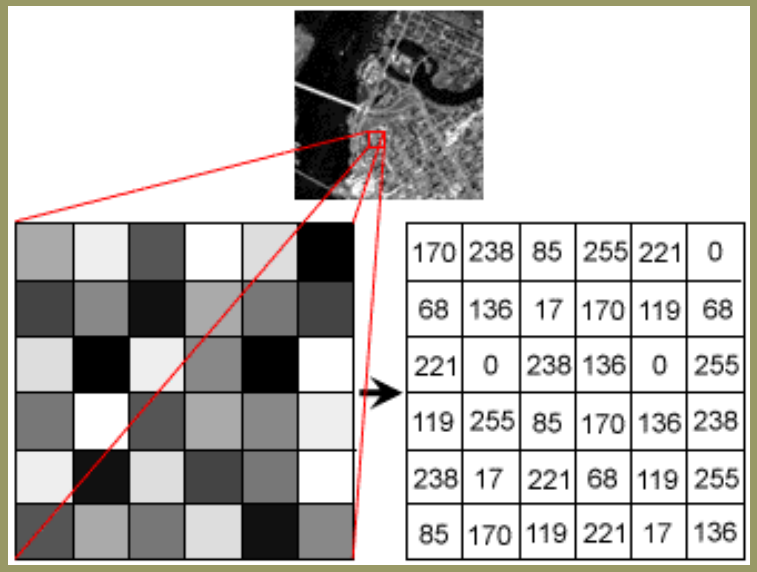
$$y_1 = \frac{2.7182}{\sim 5.9734} = 0.46$$

$$y_2 = \frac{0.6065}{\sim 5.9734} = 0.10$$

$$y_3 = \frac{1.6487}{\sim 5.9734} = 0.27$$

Imagem Digital

- Representação



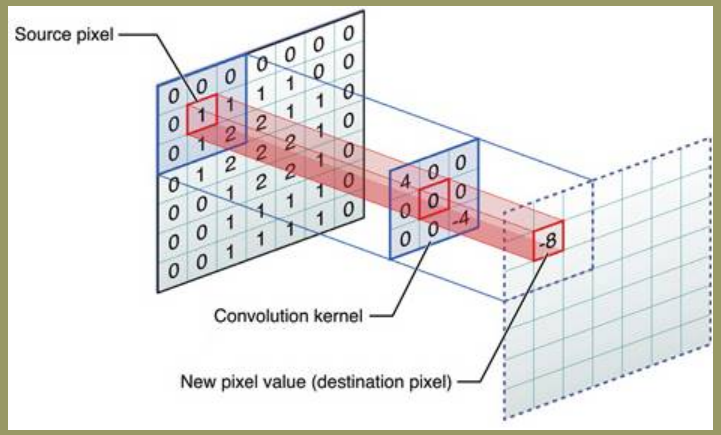
- Escala de Intensidade

Imagem	Variação
8 Bit (Nivel de Cinza)	2^8 (0 – 255)
24 Bit (Colorida 3 Canais)	2^24 (0 - 16777215)

Convoluções

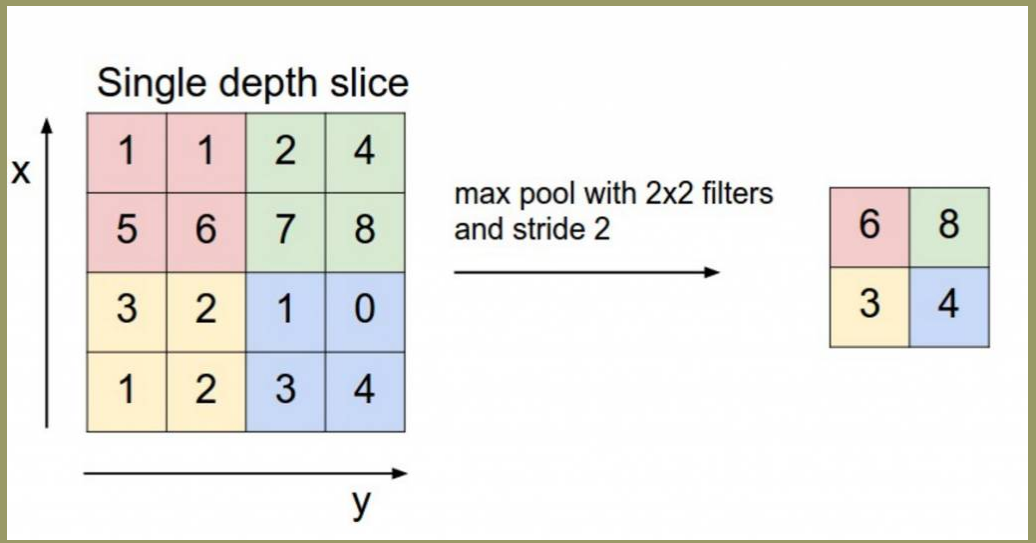
- Operações entre Matrizes (Imagem X Kernel)
  Ex: Filtros, Bordas, Etc

- Parâmetros do Kernel
  - Tamanho
  - Passo / Salto
  - Pesos
  - “Padding”

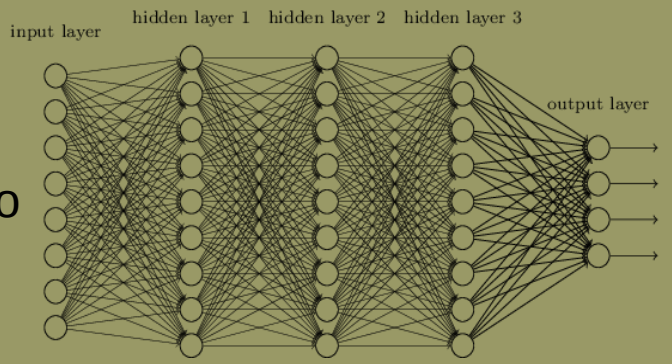


Pooling (Agregação)

- Convolução para reduzir escala (downsampling)
- Max, Avg, Mediana, etc.
- Max é o mais comum

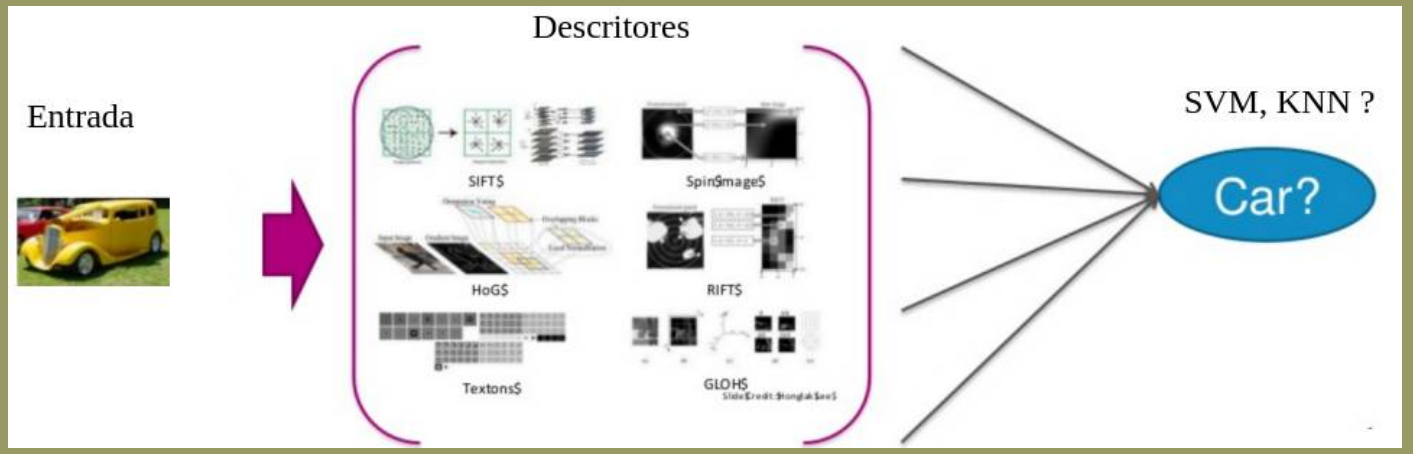


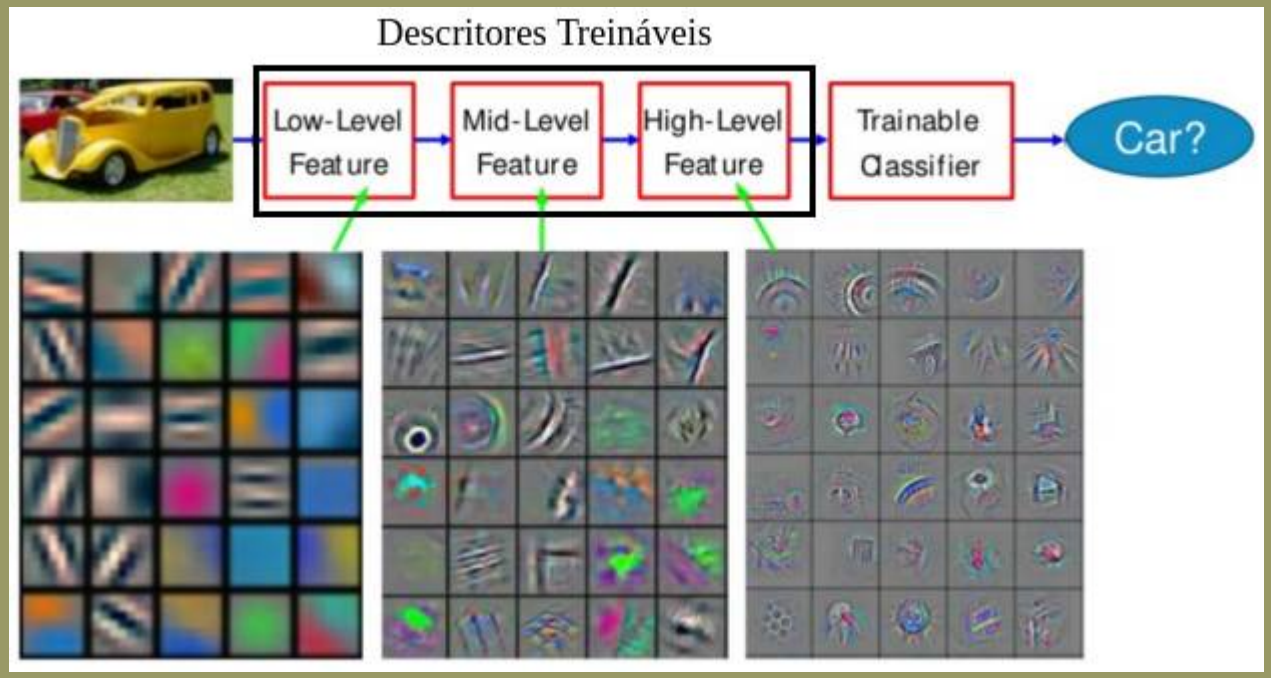
Deep Learning



- Aprendizado baseado em Multi-Camadas (RN Densas)
- Representação dos dados em diferentes níveis de abstração
- Extração de Características é Implícita
- Elevada quantidade de parâmetros (Ajuste por BackProgation)
- Necessita de um grande número de exemplos para o aprendizado eficiente
- O aprendizado é voltado para compreensão pela máquina, ou seja, os dados são de díficil compreensão/visualização humana
- Vasta aplicação na área de Visão Computacional (processamento de áudio, imagens, vídeo, etc).

Feature Extraction X Deep Learning



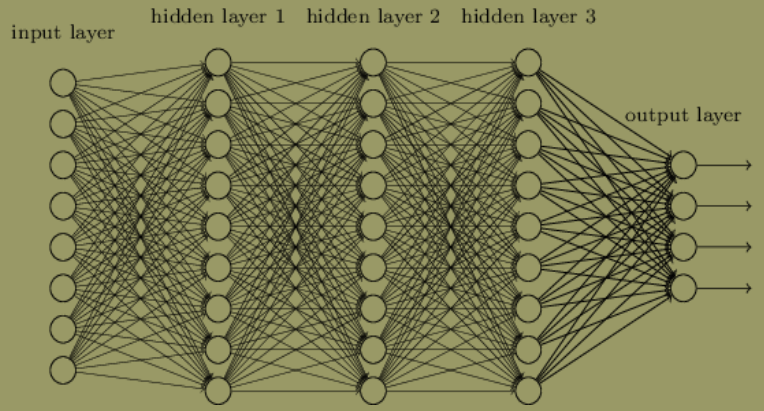


Redes Neurais Convolucionais

Deep Learning – Vanish Gradient Problem

Problema do desaparecimento do gradiente e convergência da rede
- → Valores de entrada mapeados em gradientes pequenos
- → Ativação por Sigmoid [0 →1] ou Tangh [-1 → 1]
- → Variação “pequena”

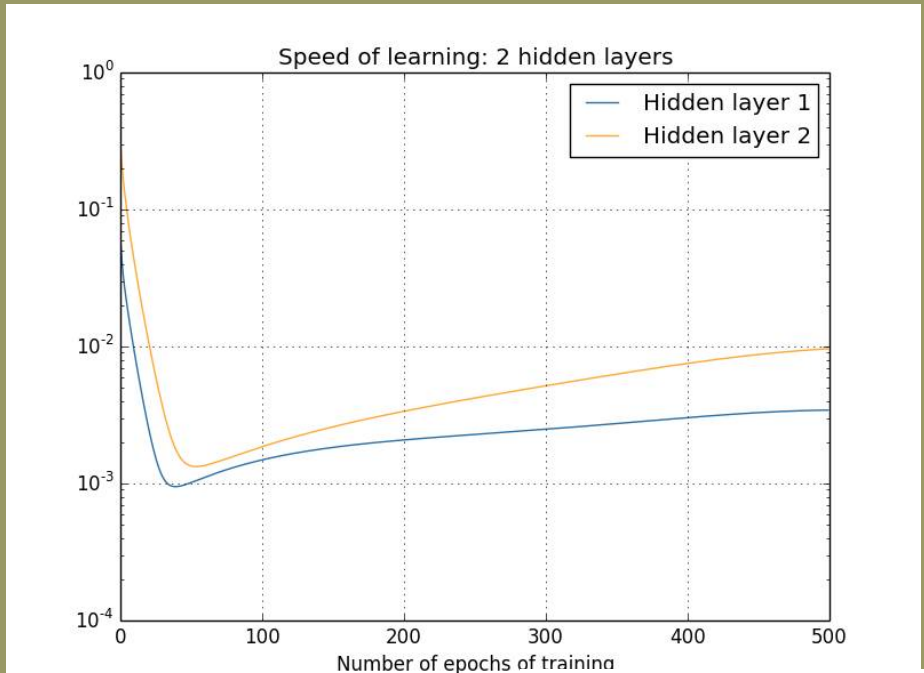
- Agrava ao longo de uma rede múltiplas camadas
  → Uma mudança de valores na entrada, pouco altera as camadas mais distantes

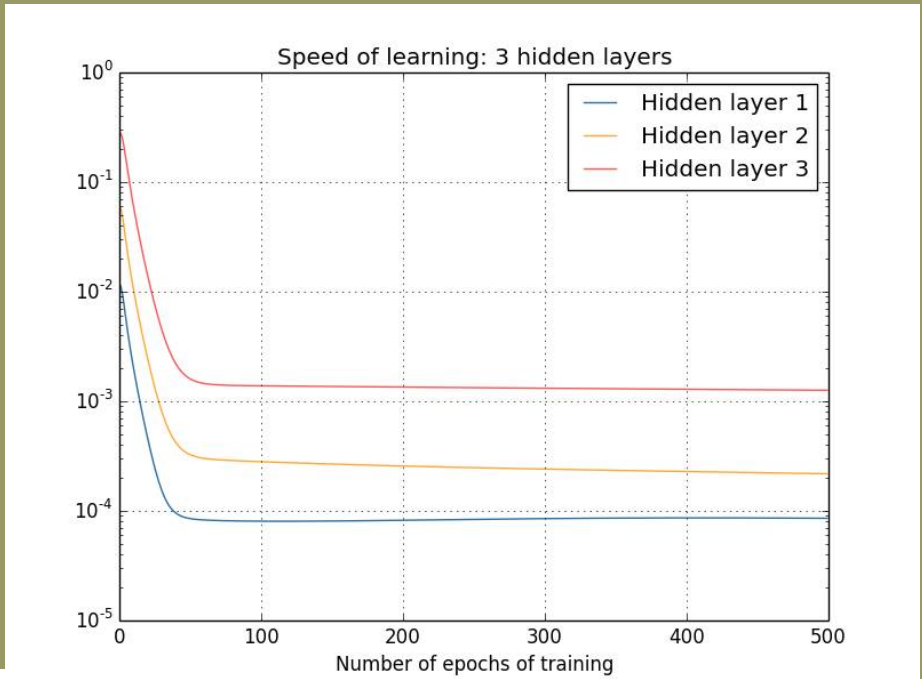


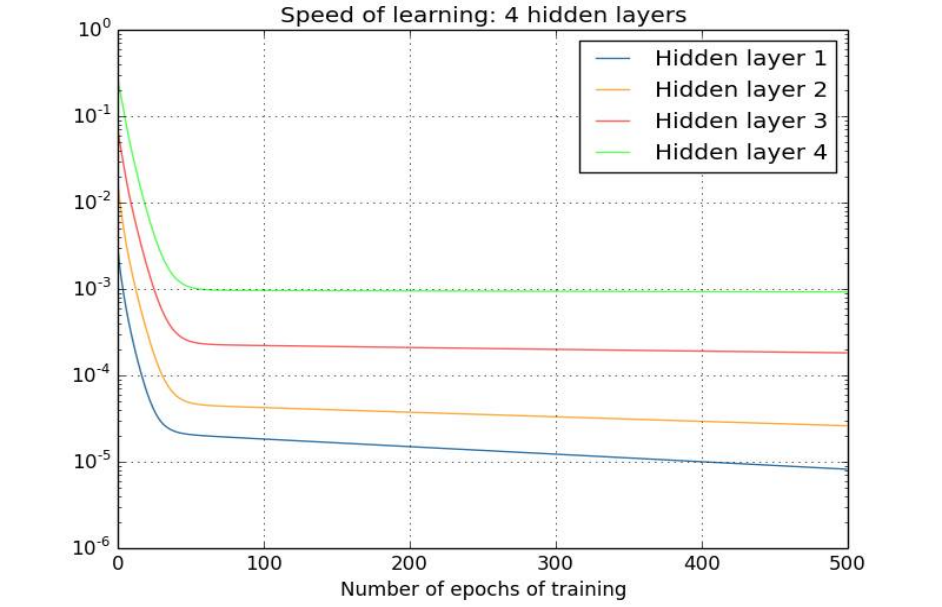
- Aprendizado mais se torna mais lento

- Solução encontrada: RELU [f(x) = max(0,x)]
  → Mapeamento [0,∞]

Deep Learning – Vanish Gradient Problem



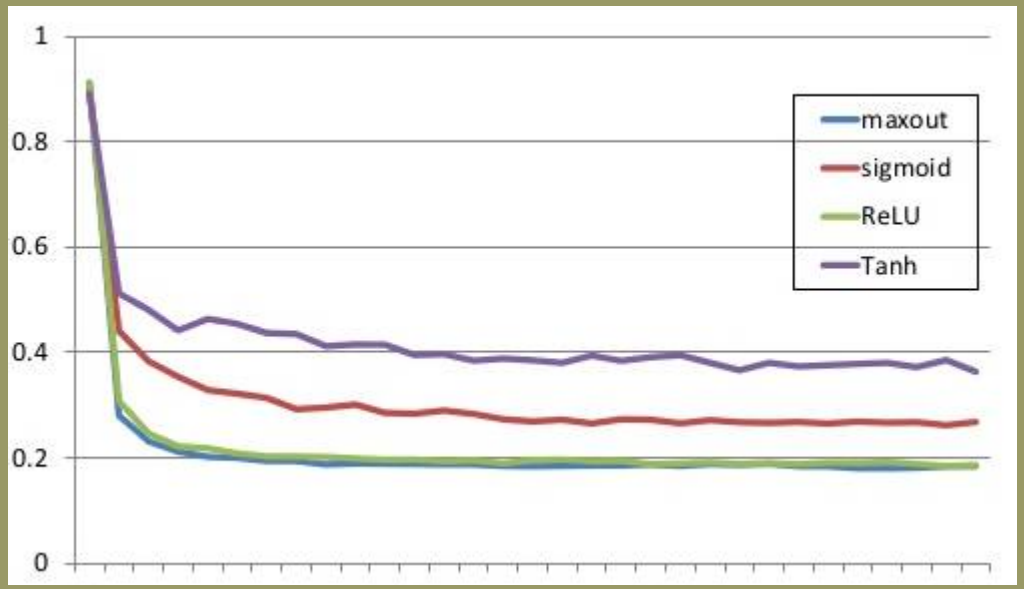


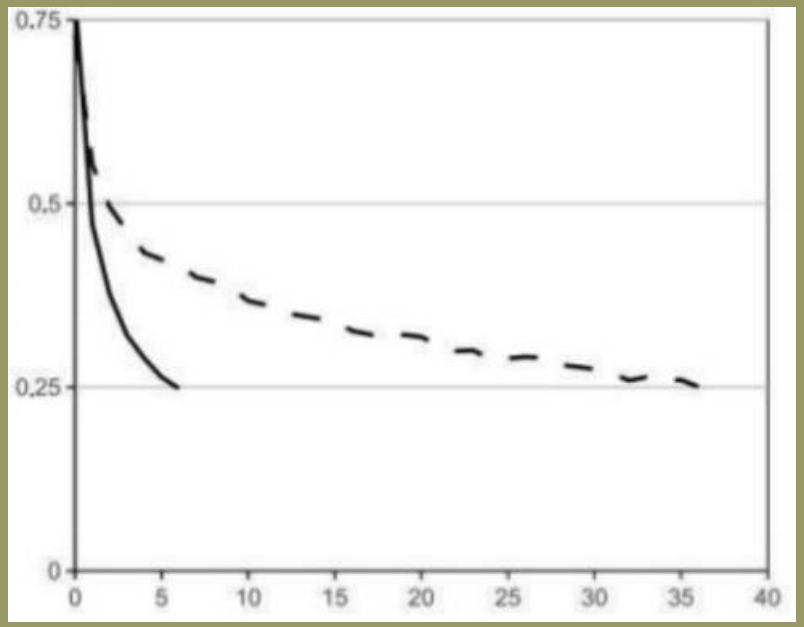


Deep Learning – Vanish Gradient Problem

→ Sigmoid/Tanh [-1,1] = Aprendizado Lento / Custo computacional mais alto (expoentes e multiplicações)

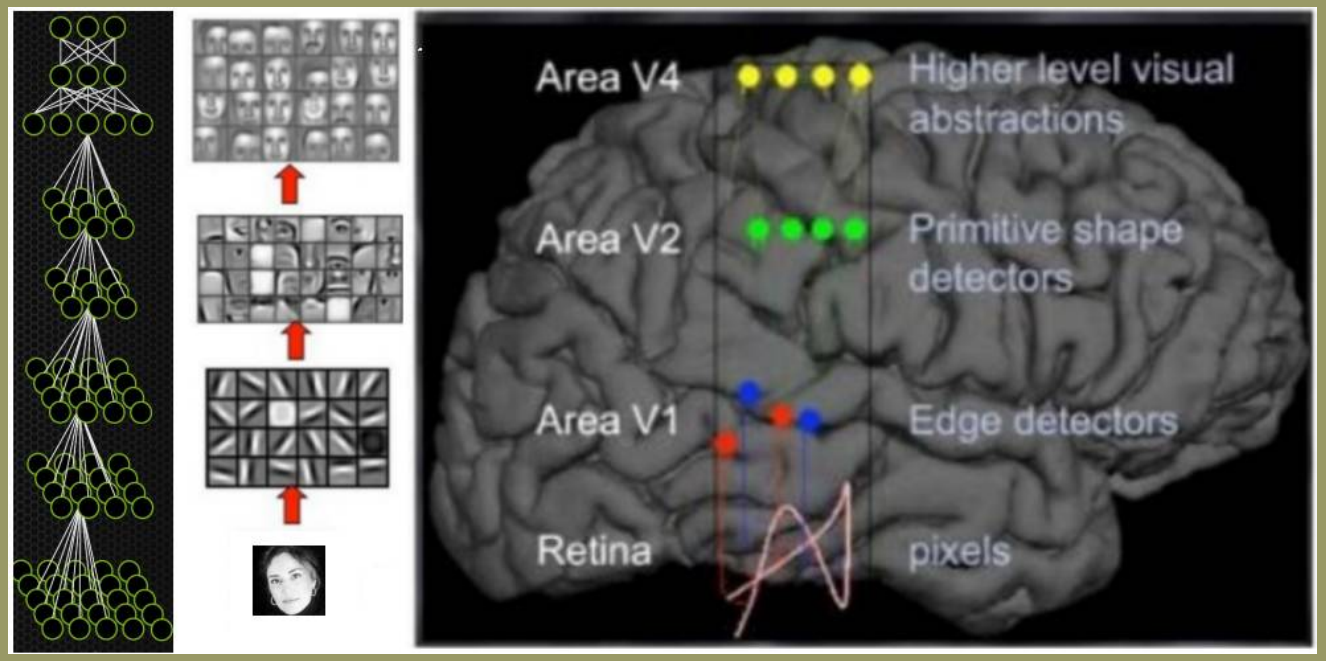
→ RELU [0,∞] = Aprendizado Rápido / Custo computacional mais baixo (threshold)





Redes Neurais Convolucionais

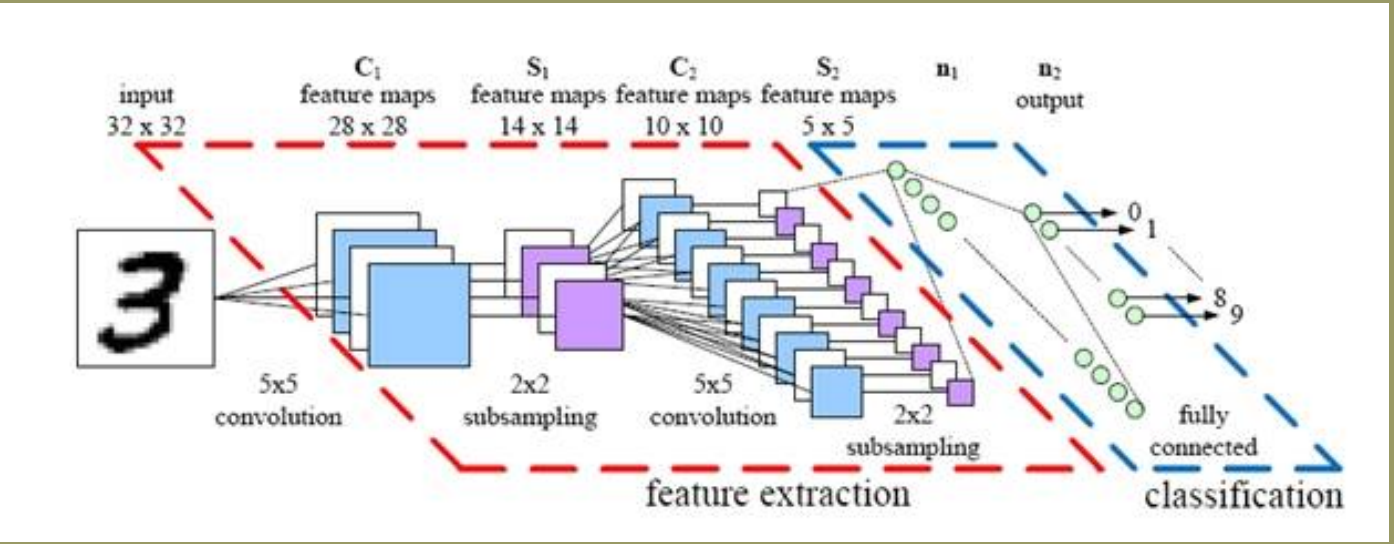
- Inspiradas no modelo biológico da visão
- Conceito de Deep Learning (Multi-Camadas)
- Idealizada no ínicio do anos 90 [Lecun], e vasta aplicação após 2006 devido a “popularização” de GPU's (Custo ~$ 3000,00)
- Treinamento requer alto custo computacional e numerosa base de dados



Redes Neurais Convolucionais

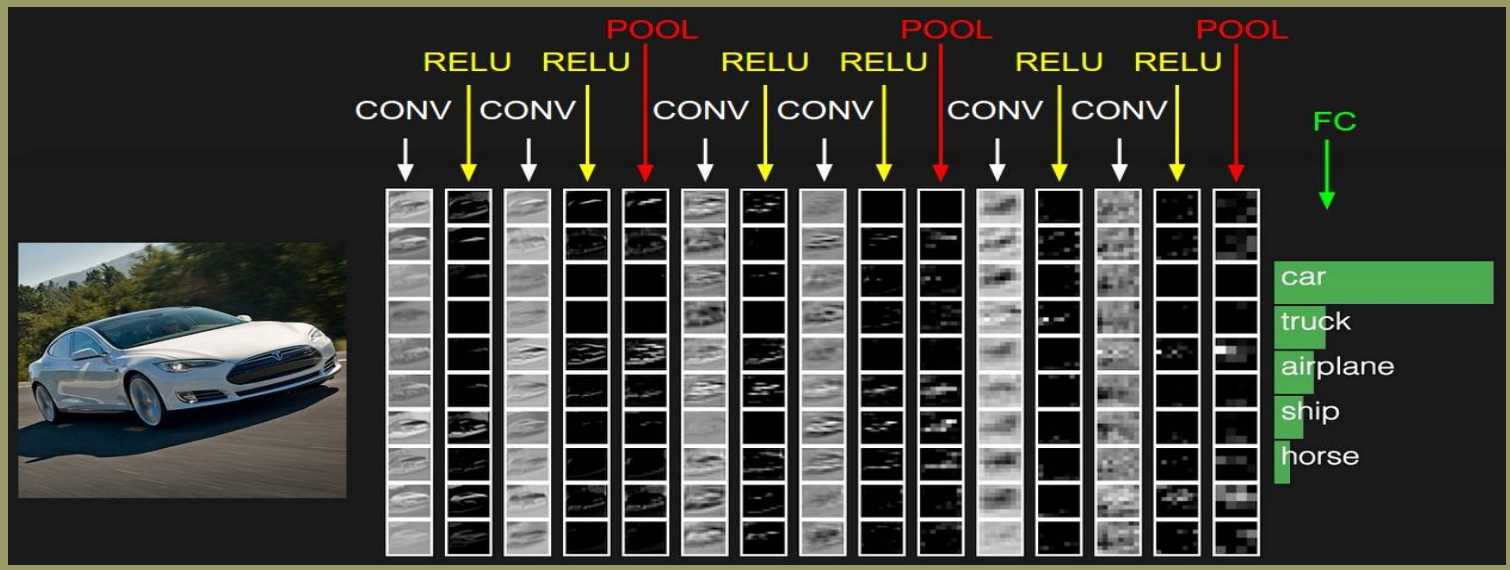
- Compostas de duas grandes etapas:

  → Extração de Características pelas Camadas Convolucionais
  → Classificação



Tipos de Camadas (Layers)

- Convolucional : Definem os filtros (Aprendizado / BackPropagation)
- Ativação: Neurônios (Relu / Sigmoid / TangH)
- Pooling : Reduzem as escalas (Max, Median, etc..)
- Fully-Connected (FC): Camada que determina as classes (Classificador)



Extração de Características

- Transforma uma imagem em um vetor de características
  - → Sequência de Convoluções, Ativação e Pooling's
  - → Convoluções: altera a representação dos dados e aprende os filtros
  - → Pooling: reduz a escala para o próximo layer

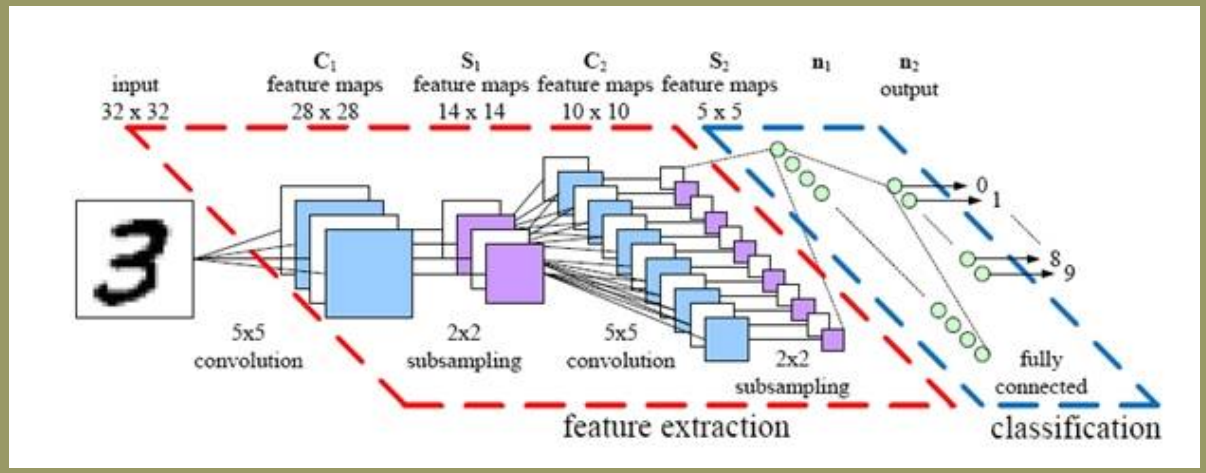
  Como ?

- Pesos dos filtros são calculados automaticamente (BackPropagation)

- Cada filtro aprendido é um extrator de características

- Cada imagem resultante de um filtro é um mapa de características

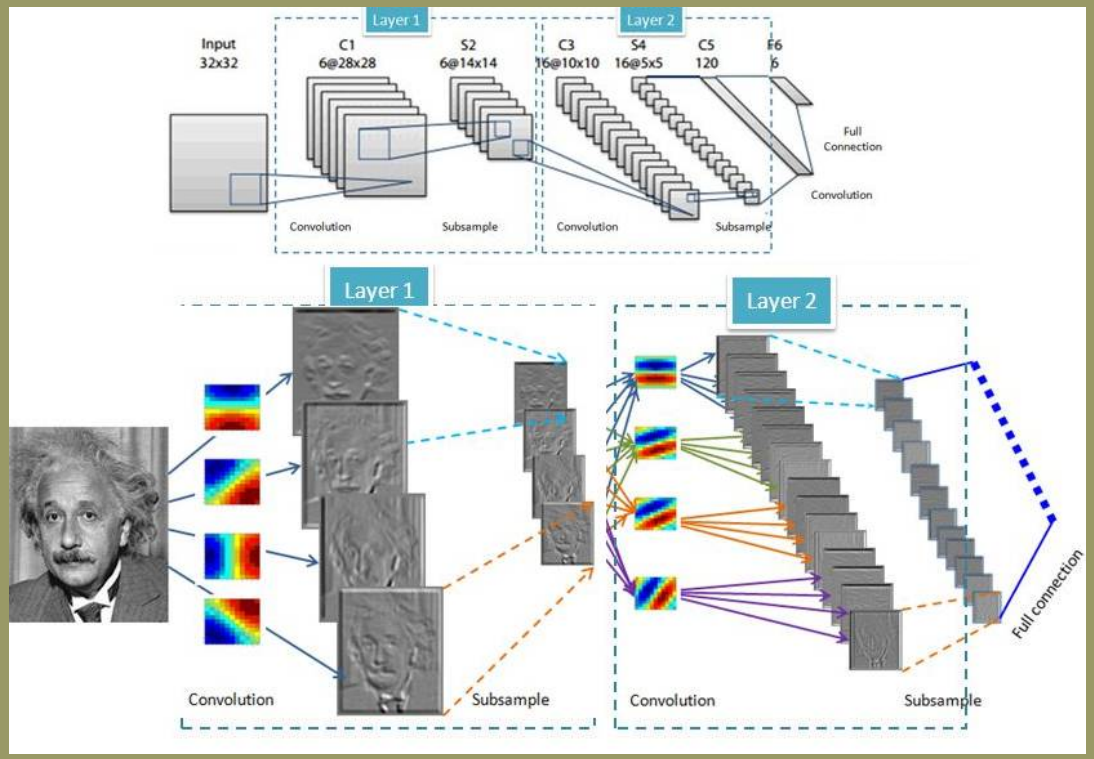
- Os pooling's são necessários para reduzir a quantidade de características por filtro (redução de escala)



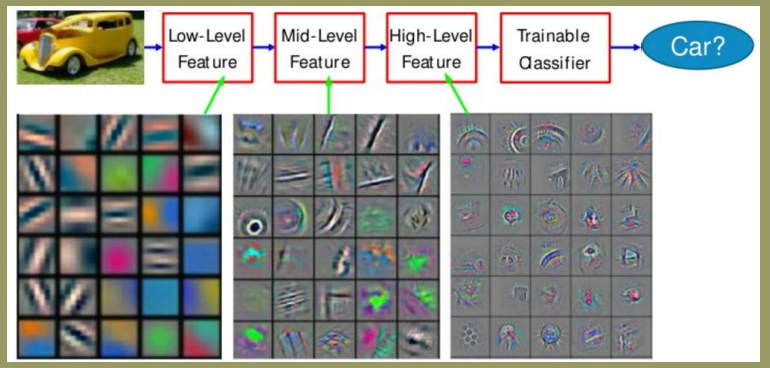
Redes Neurais Convolucionais

Extração de Características

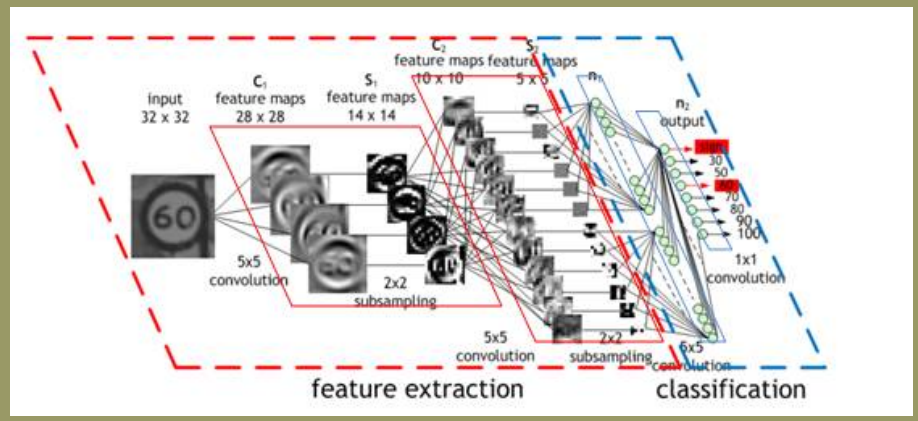
- Exemplos:



→ Filtros + Conv + Pool



→ Filtros Treinados (Aprendidos)



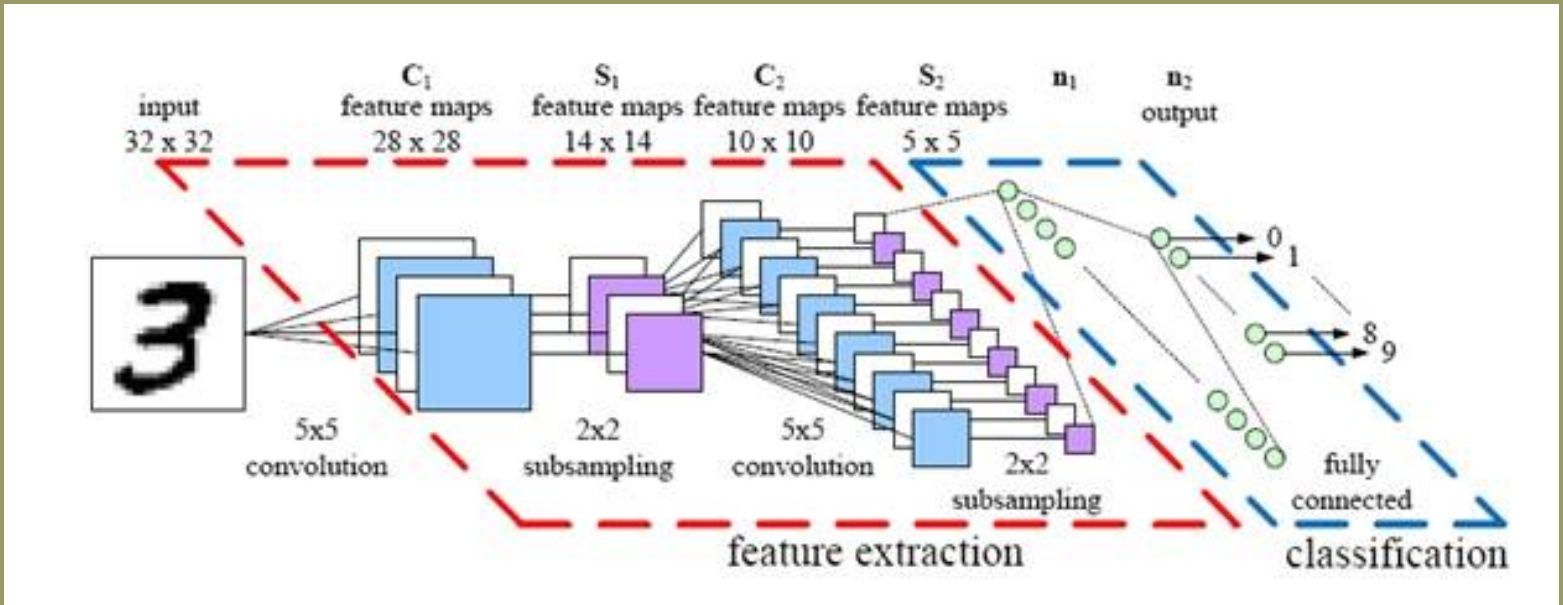
→ Representação dos dados

Extração de Características

- Exemplos:

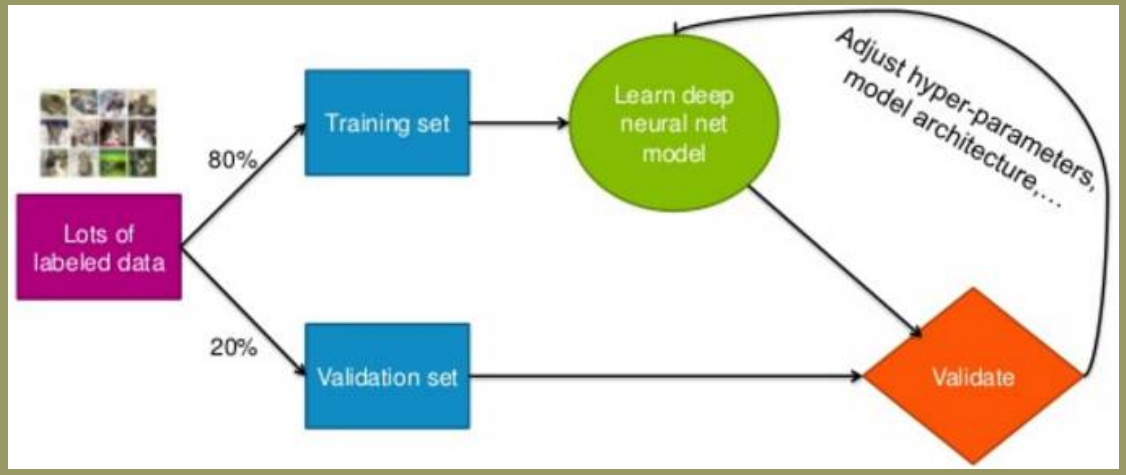
Extração de Características

- Números de neurônios por convolução



→ C1: 3136 (28*28*4)
→ S1: 1568 (C1 / 2)
→ C2: 1200 (10*10*12)
→ S2: 600 (C2 / 2)

Treinamento X Teste



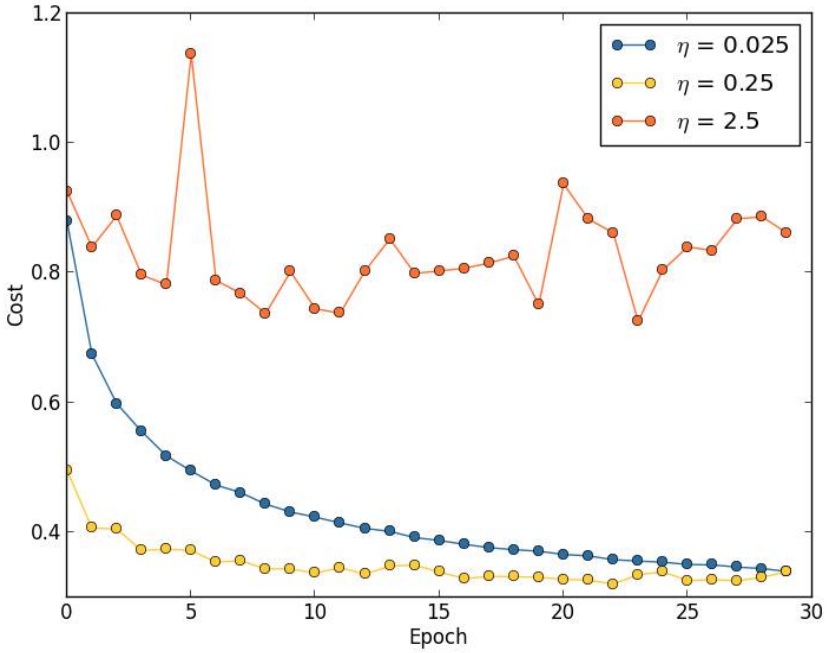
- Treinamento

  → Inicialização dos filtros e FC (Random – Const – Gaussian – Xavier)
  → Fase de Aprendizado (Pode demorar horas ou dias dependendo da configuração da rede)
  → Validação e Backpropagation
  → Armazenamento dos pesos e filtros ao longo do processo

- Teste da CNN

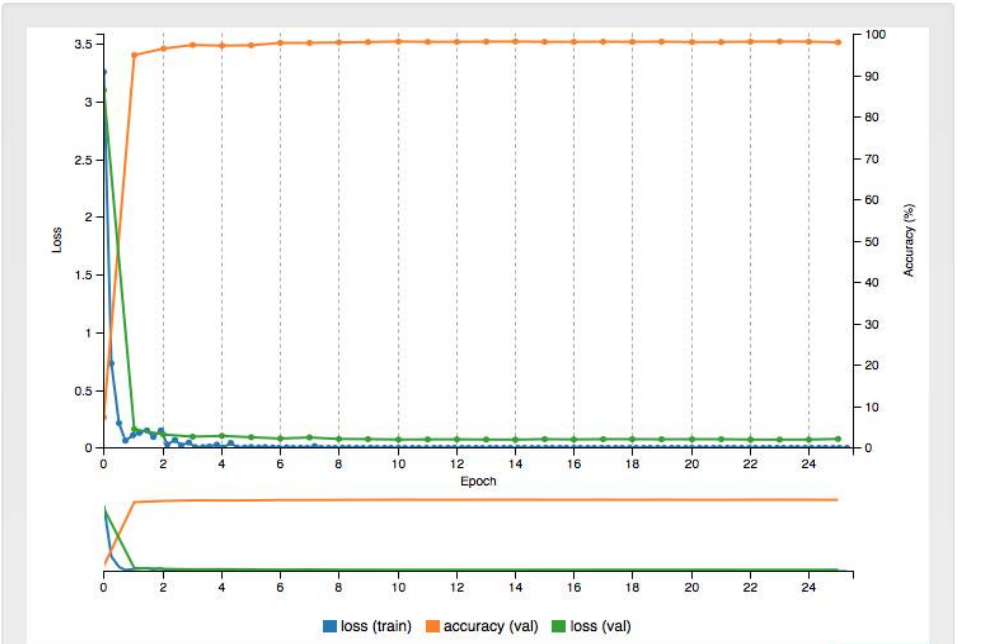
  → Utiliza os filtros e pesos da época definida

Treinamento



Desempenho do Aprendizado para diferentes “Learning Rates”

Rede “IDEAL”
Apredizado em poucas épocas



Data Augmentation

- O que fazer se houver poucos exemplos na base ? Ou pouca variabilidade de uma mesma classe ?

Amostra →

Data Augmentation →

Ampliação do número de amostras modificando a amostra original:

→ Translações
→ Ruídos
→ Brilho / Contraste
→ Flip's
→ Smooth's

Deep Learning / CNN - Análise

Pros

- Enables learning of features rather than hand tuning
- Impressive performance gains on
  - Computer vision
  - Speech recognition
  - Some text analysis
- Potential for much more impact

Cons

- Computationally really expensive
- Requires a lot of data for high accuracy
- Extremely hard to tune
  - Choice of architecture
  - Parameter types
  - Hyperparameters
  - Learning algorithm
  - …
- Computational + so many choices = incredibly hard to tune

Algumas Aplicações

- Faces

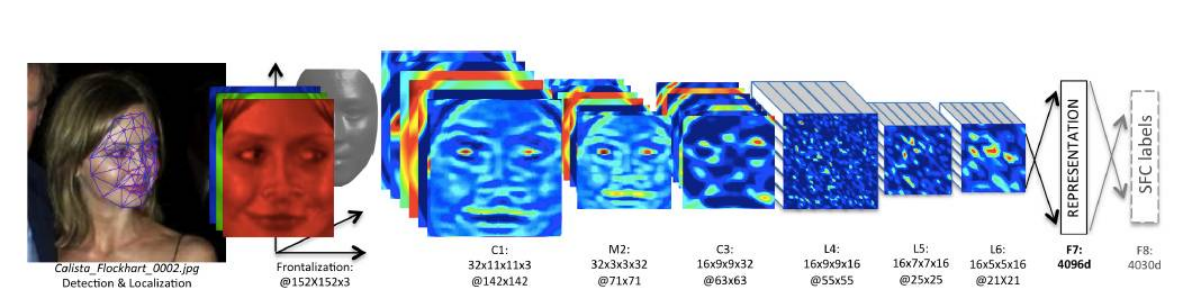
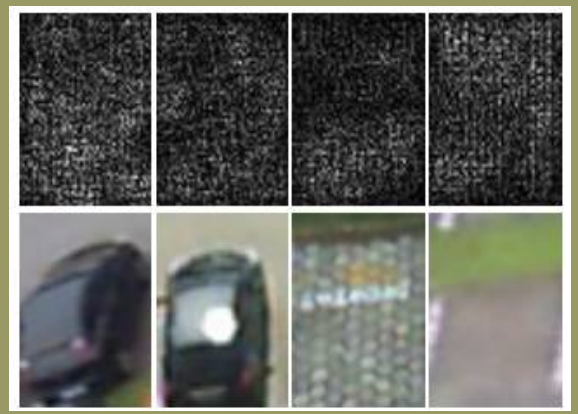


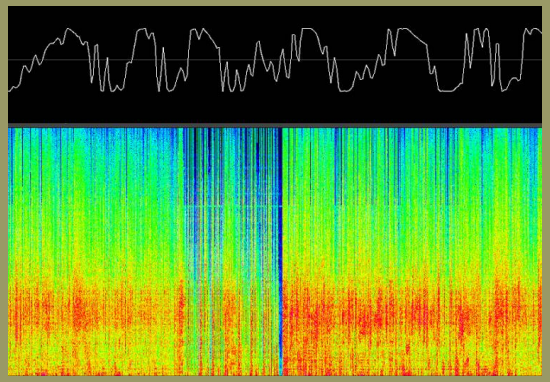
Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate outputs for each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

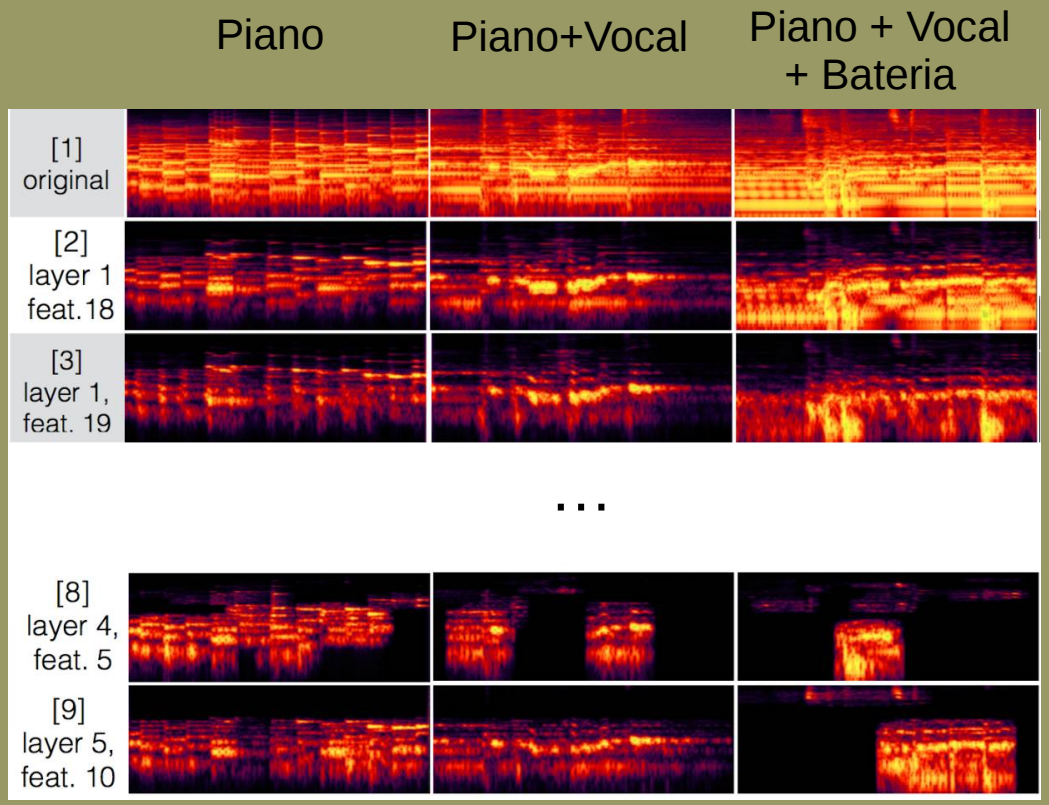
- PKLot



Algumas Aplicações

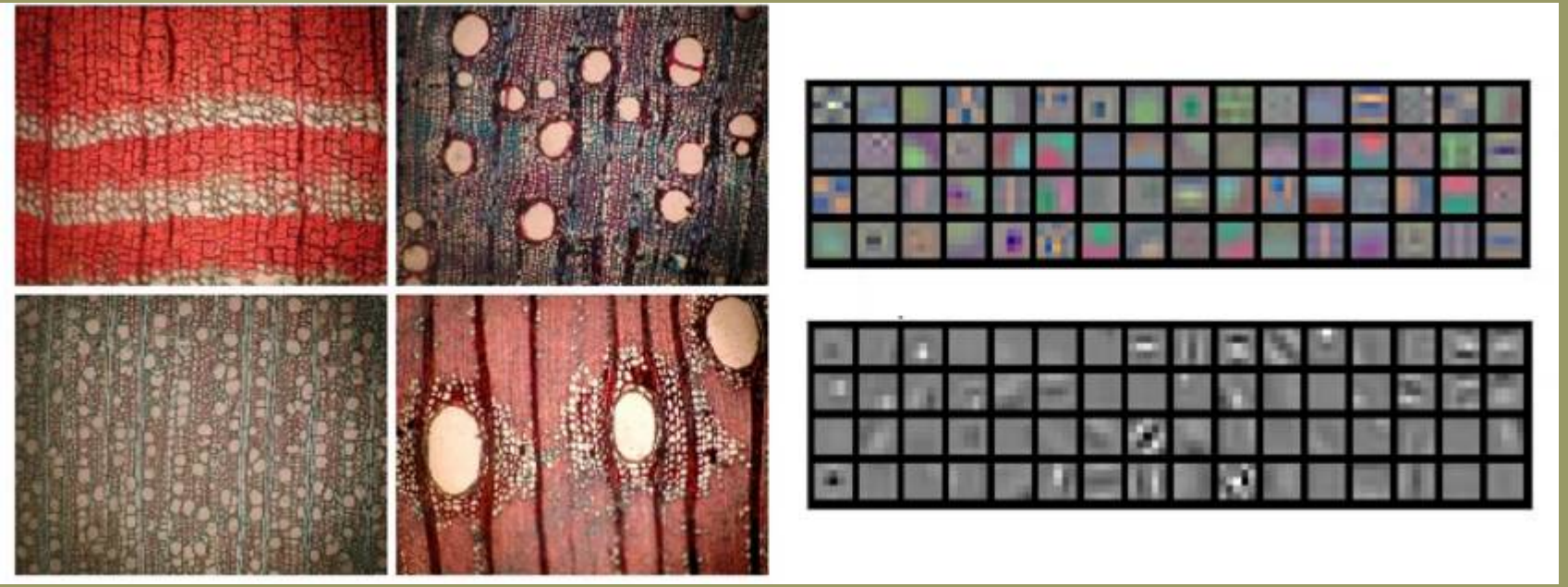
- Processamento de Aúdio (Vocal / Genêro Musical)





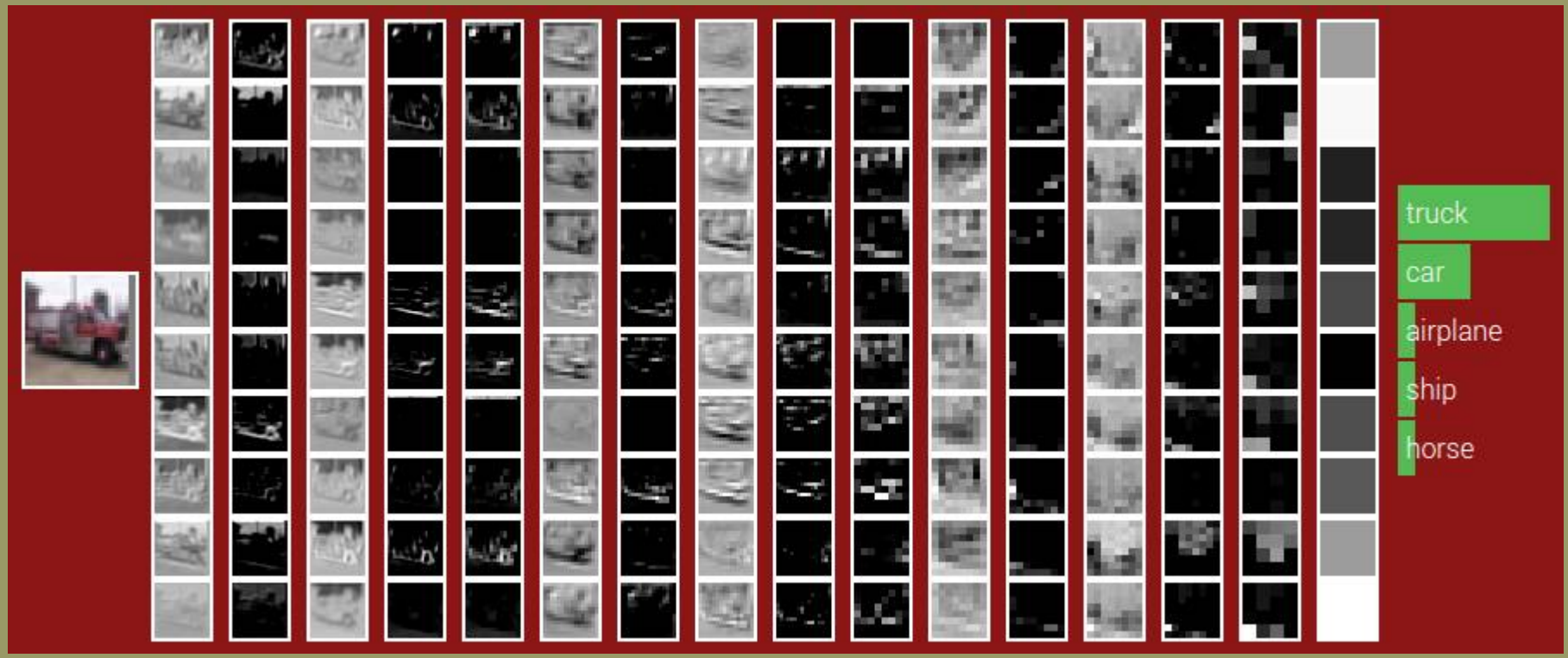
Algumas Aplicações

- Texturas
  - → Tecido Celular
  - → Células Cancerígenas
  - → Espécie Florestal



Algumas Aplicações

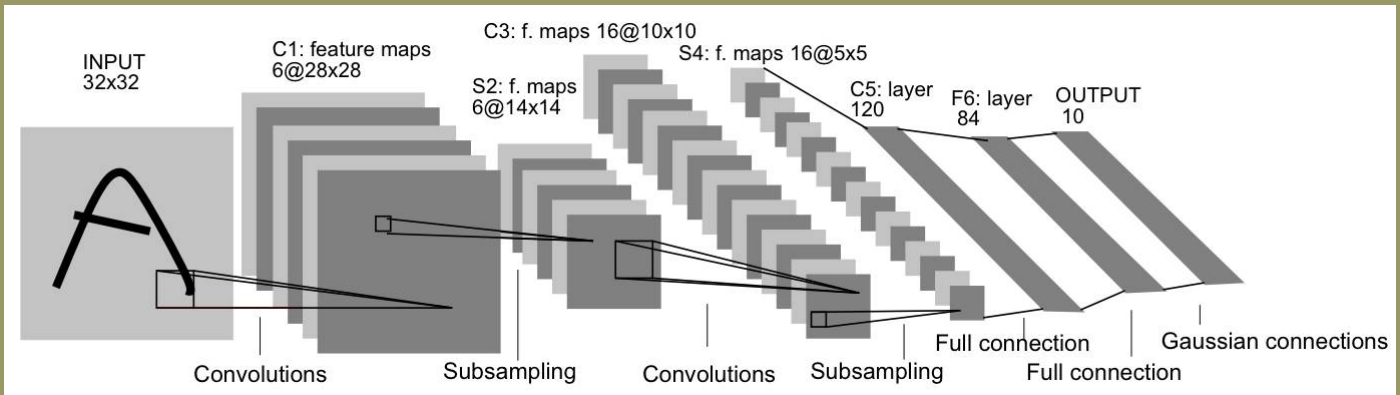
- Contexto de Imagens



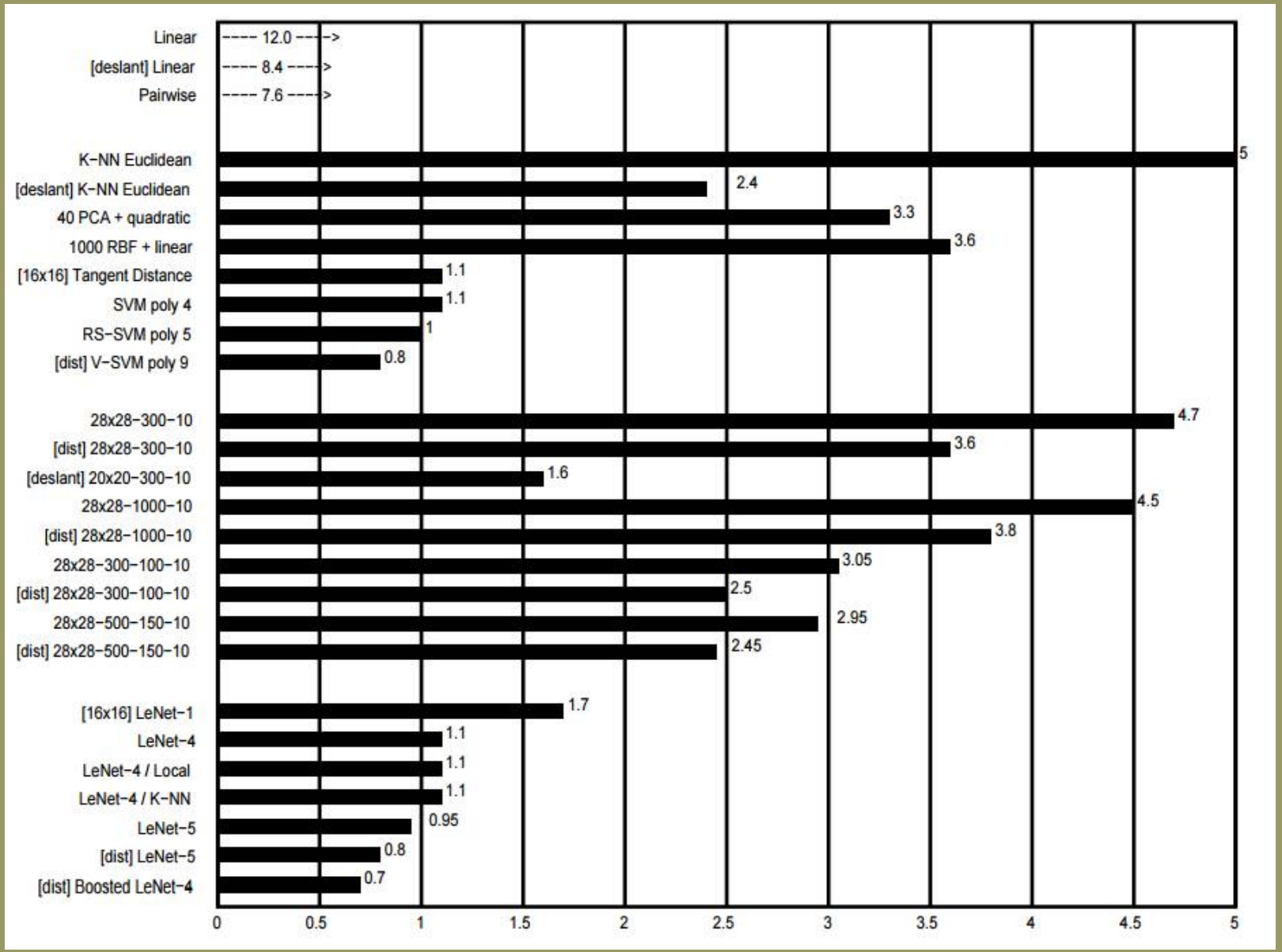
LENET-5

- Yan Lecun – 1998
- Primeira CNN implementada e testada com sucesso (Bell Labs)
- Reconhecimento de Dígitos Manuscritos
  - → MNIST DATASET (10 Classes [#0....#9])
  - → 60 K Treinamento
  - → 10 K Teste

  - 0.95% (erro)
  - ~ 345 K de conexões
  - ~ 60 K parâmetros

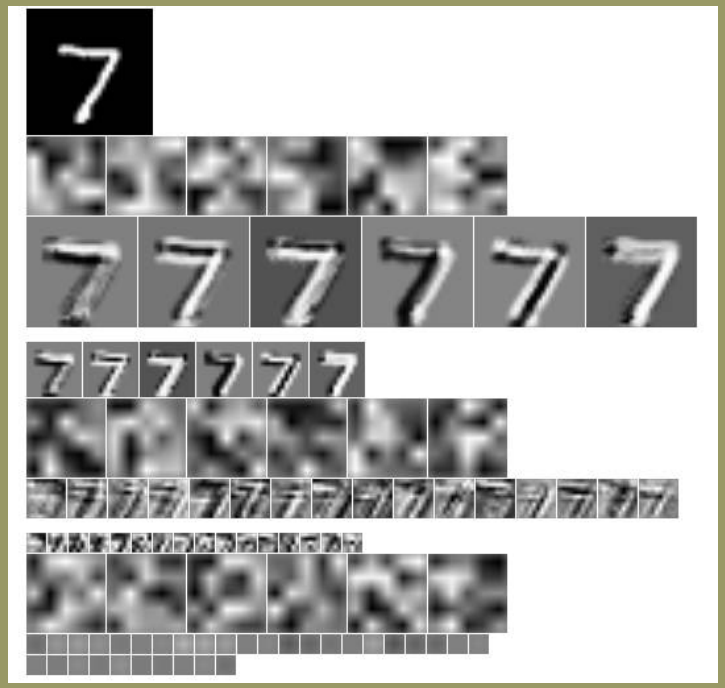


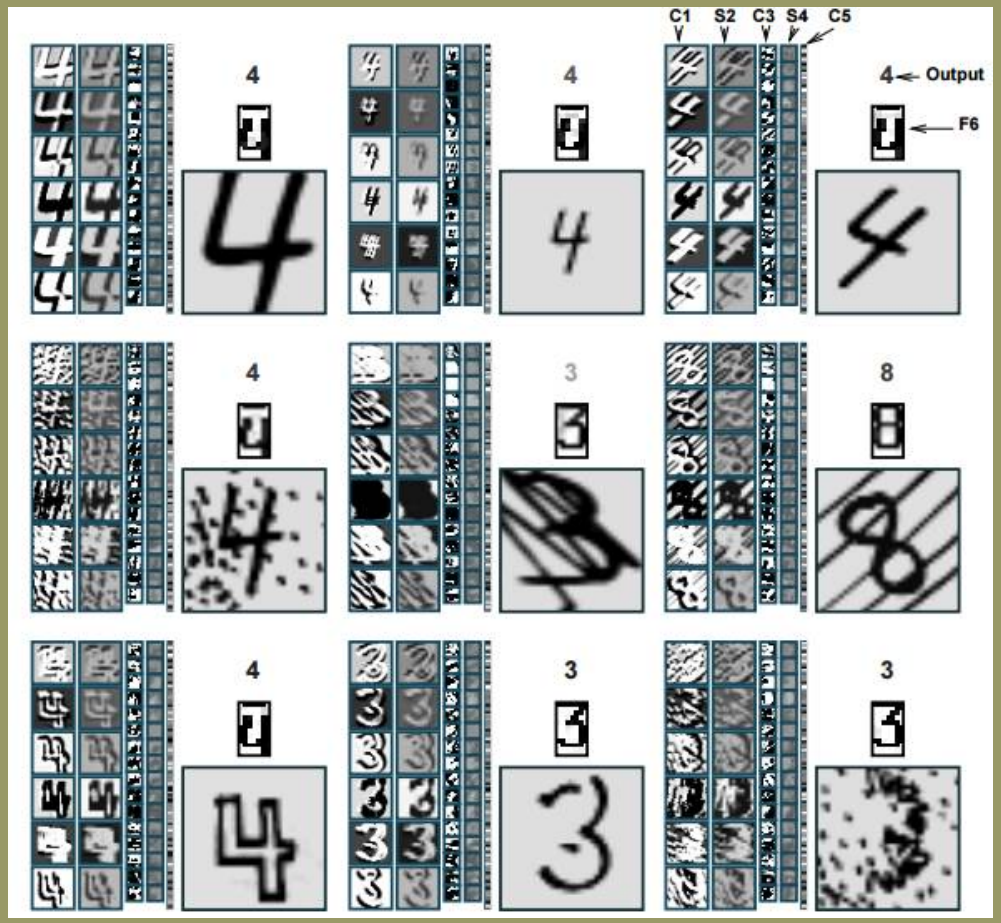
LENET X ESTADO DA ARTE (1998)



LENET

Filtros por Camada

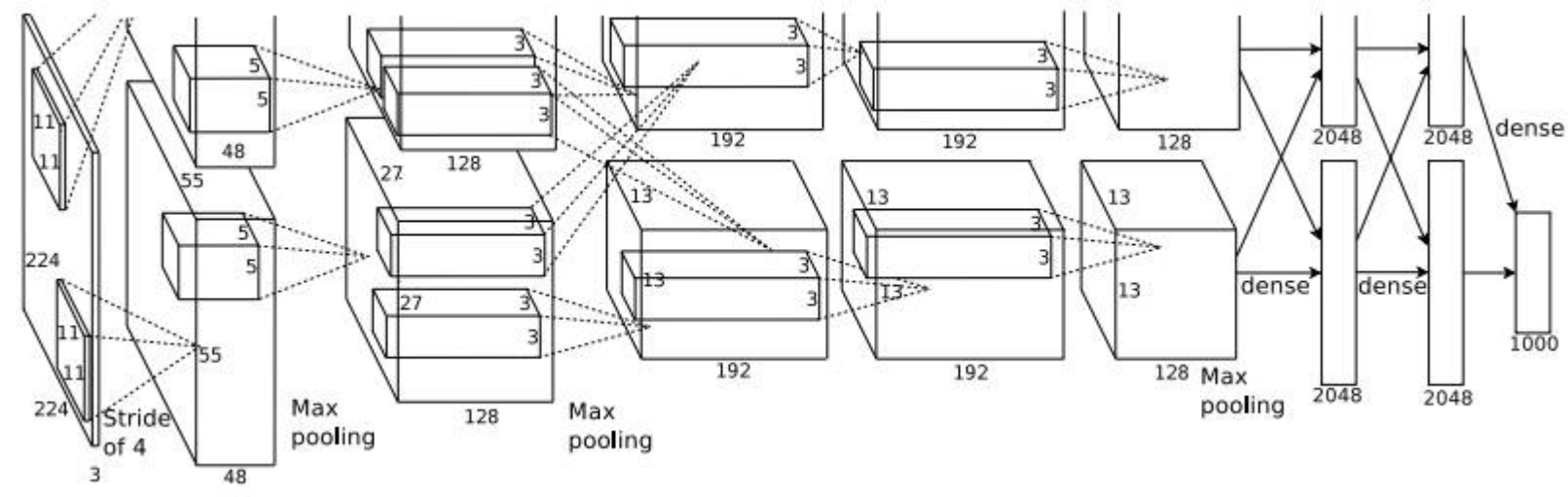




Resistência a ruídos, inclinação e formatação

Redes Neurais Convolucionais

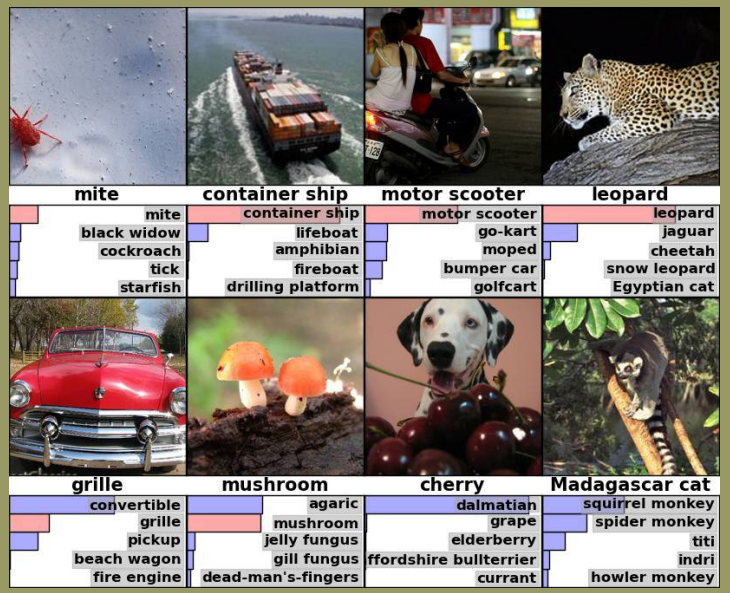
ALEXNET



- Alex Krizhevsky – 2012 (Krizhevsky Net)

- Imagenet 2012 Challenge (1000 classes)
  - → 1.2 M Treinamento
  - → 50 K Validação
  - → 150 K Teste

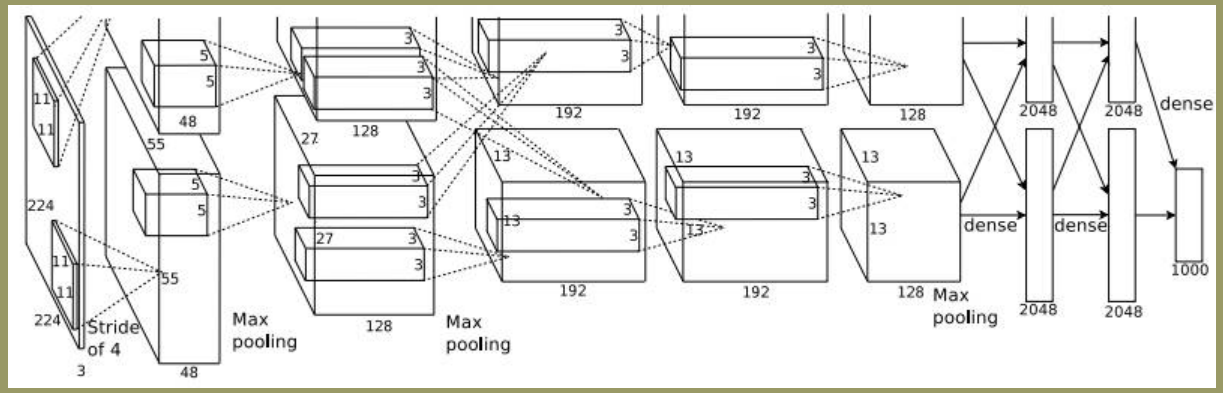
- Vencedor (erro 15.3% - Top 5)
  - → 2° SIFT Based (26.2%)



Redes Neurais Convolucionais

- http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

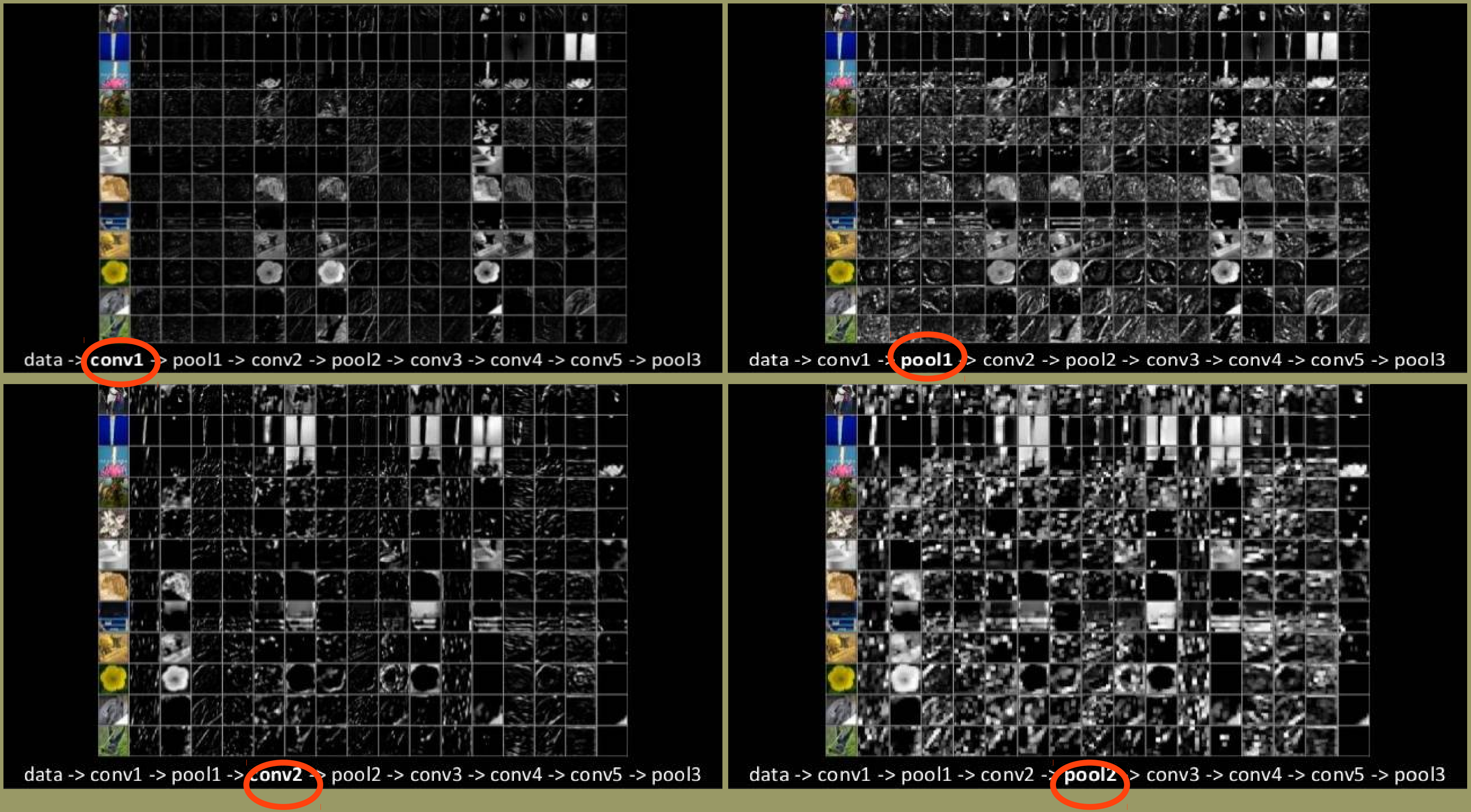
ALEXNET



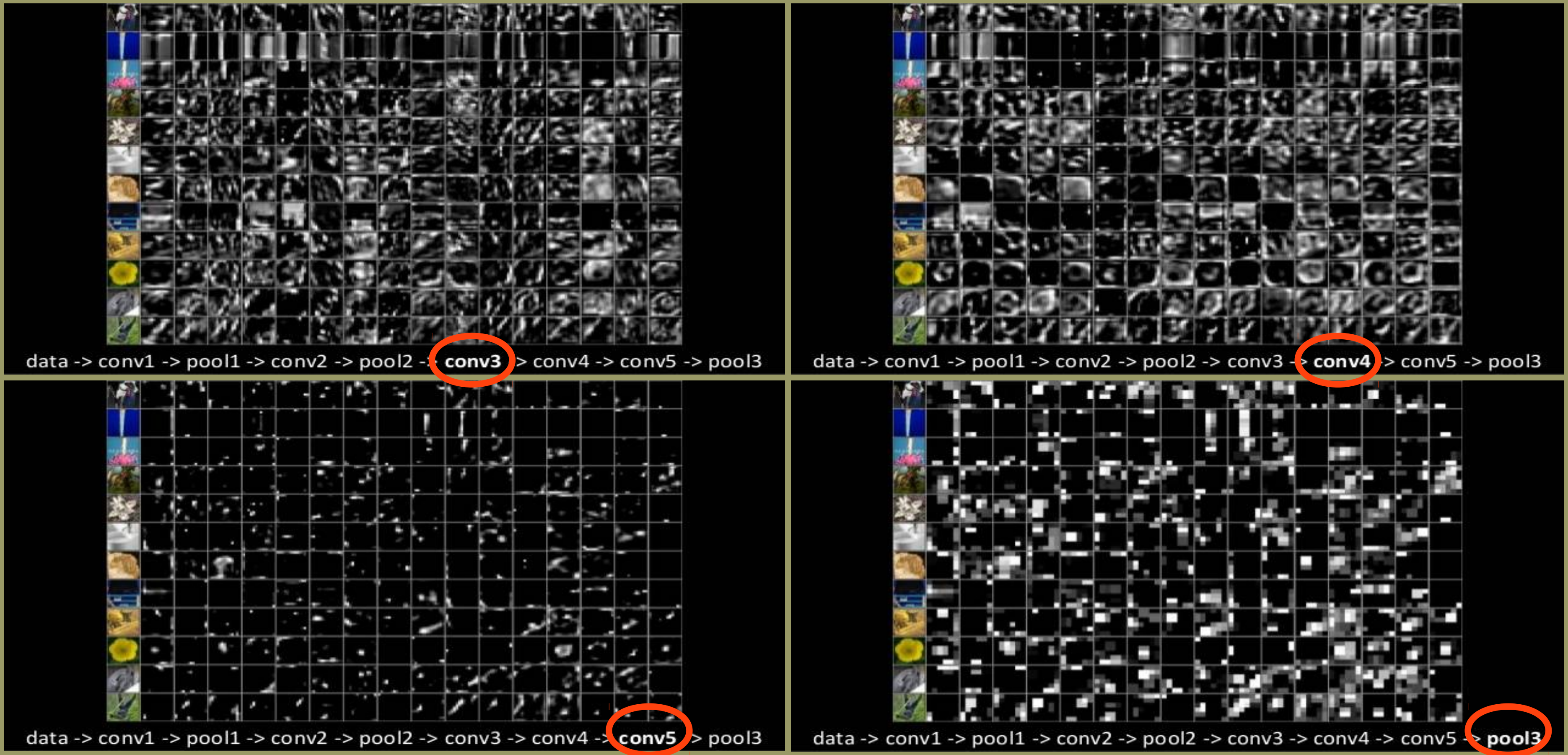
- Neurônios por Camada: ~253k, ~186k, ~64k , ~64k, ~43k, ~4k, ~4k, ~1k
- 60 M de Parâmetros
- Treinamento
  - → 6 dias
  - → 02 PLACAS GPU's (NVIDIA GTX 580 3GB)

- http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

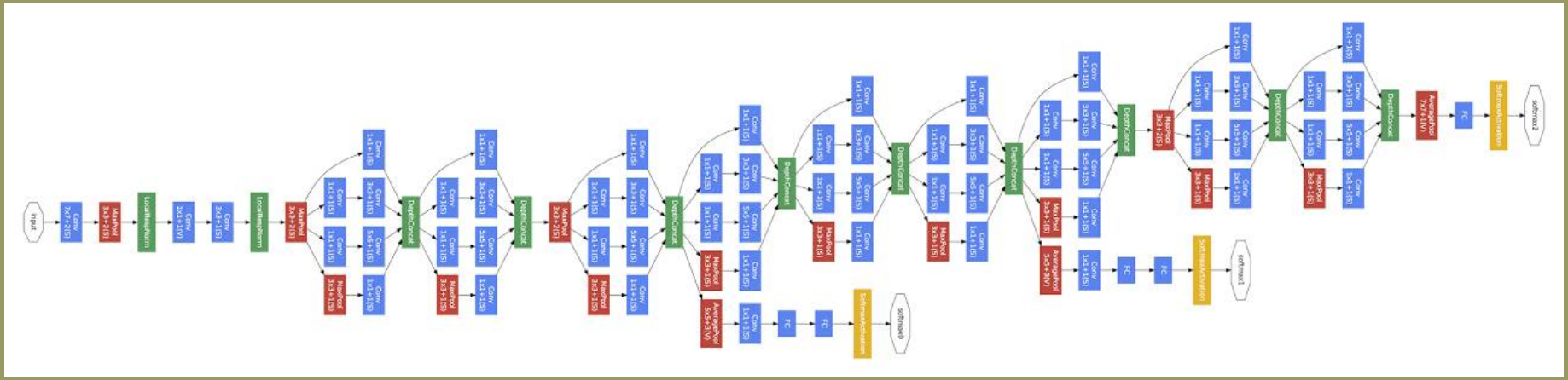
ALEXNET


data -> conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> conv4 -> conv5 -> pool3
data -> conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> conv4 -> conv5 -> pool3
data -> conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> conv4 -> conv5 -> pool3
data -> conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> conv4 -> conv5 -> pool3

ALEXNET


data -> conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> conv4 -> conv5 -> pool3
data -> conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> conv4 -> conv5 -> pool3
data -> conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> conv4 -> conv5 -> pool3
data -> conv1 -> pool1 -> conv2 -> pool2 -> conv3 -> conv4 -> conv5 -> pool3

GOOGLE NET



- Imagenet 2014 Challenge (1000 classes)
  - → 1.2 M Treinamento
  - → 50 K Validação
  - → 100 K Teste

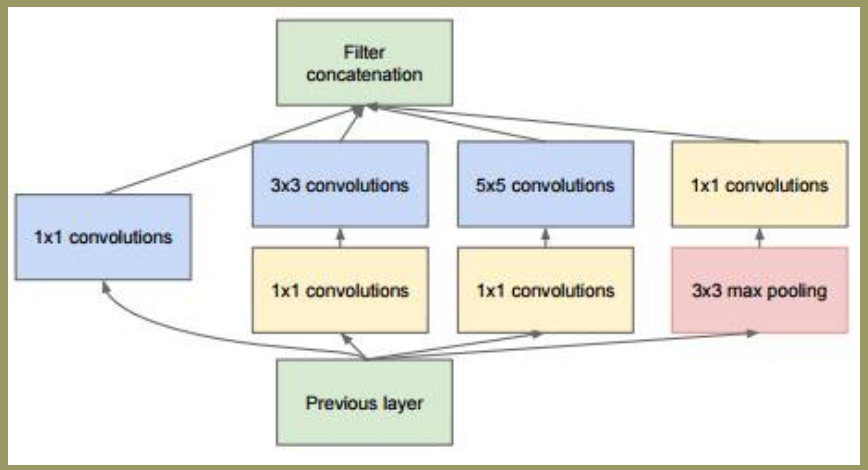
- Vencedor (erro 6.67% - Top 5)
  - → 2° CNN Based (7.32%) e 3° CNN Based (7.35%)

http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

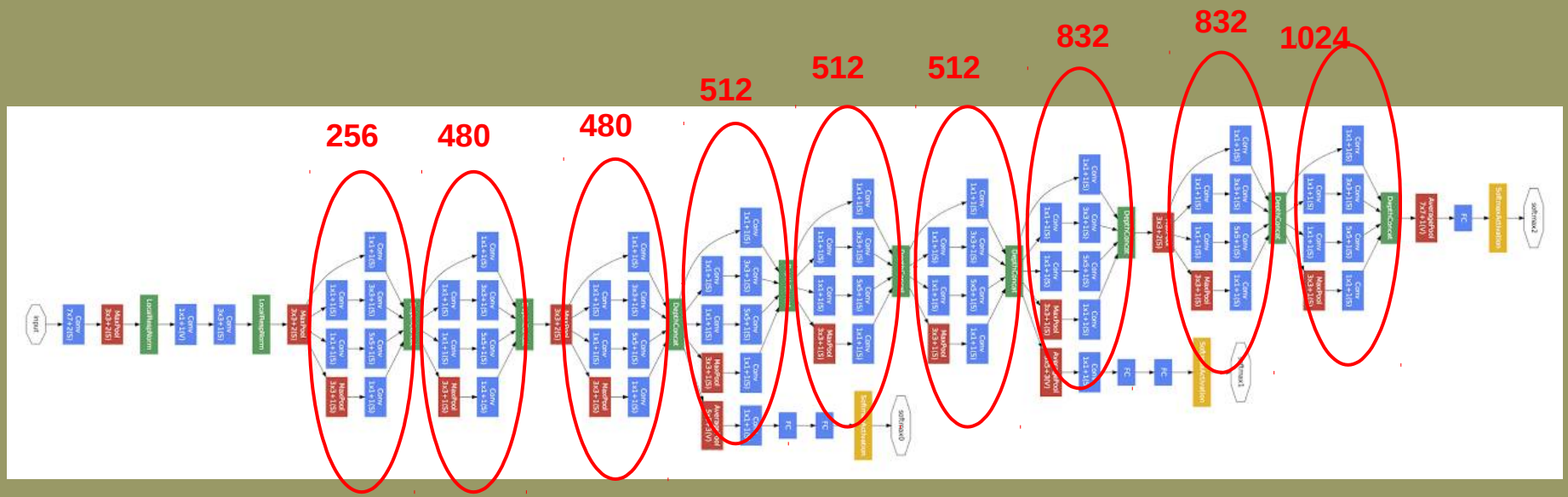
GOOGLE NET

- Inceptions: 256 filtros até 1024 filtros
- Redes dentro de Redes !

Inceptions



- Custo computacional “apenas” 2x mais que AlexNet



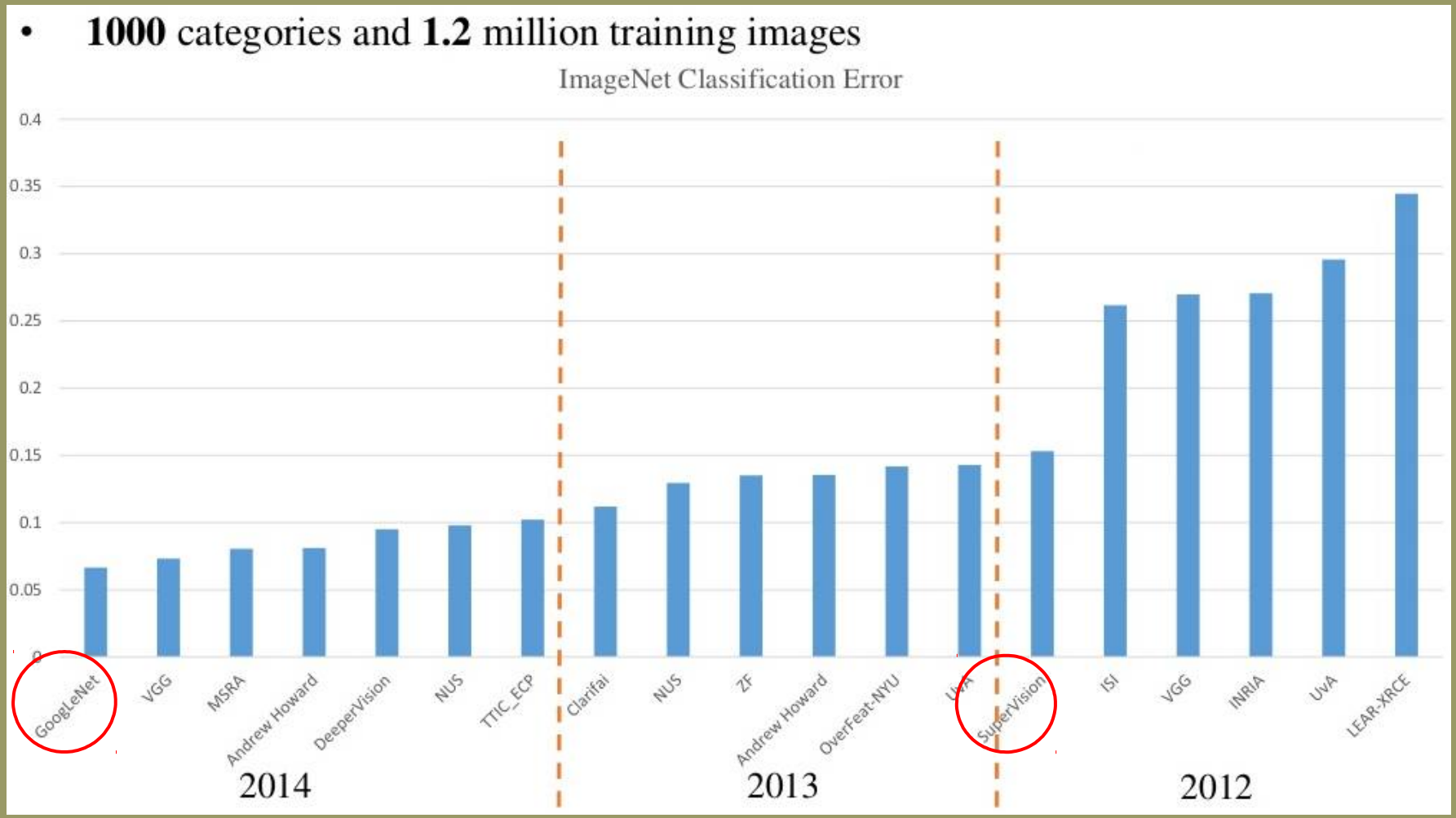
Redes Neurais Convolucionais

GOOGLE NET

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

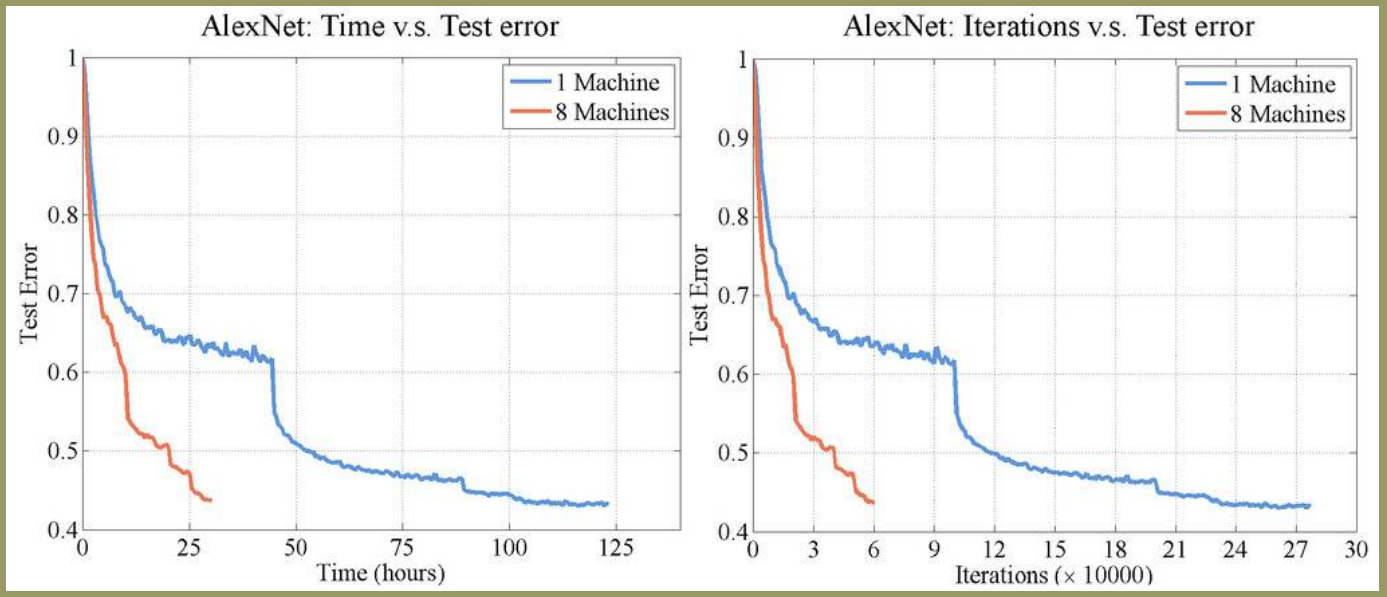
Imagenet Challenge

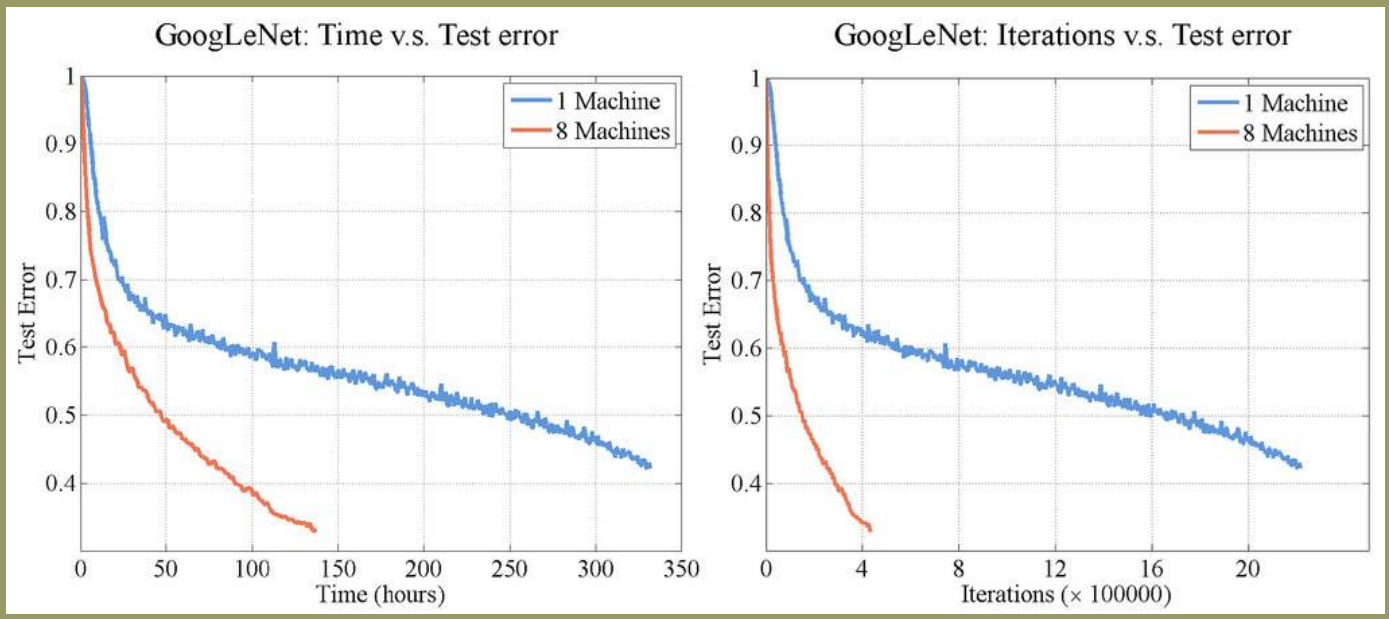
- Evolução (redução de erro) em casos complexos em apenas 2 anos !!



Imagenet Challenge

- AlexNet x GoogleNet (Poseidon FrameWork)





GOOGLE NET “DREAMS”

Imagem Entrada

GOOGLE NET “DREAMS”

Porque apareçem cães nas imagens ?

Em 2012, o google utilizou uma parte da Imagenet, para treinar sua rede. Esse “subset” era composto por imagens de cães apenas. Então a rede aprendeu a extrair características com base em cães !

TRANSFER LEARNING !!

FrameWork CAFFE DEEP LEARNING

- Implementação C++ / CUDA (GPU)
- Python Interface
- SITE: http://caffe.berkeleyvision.org/
- Tutorial de Instalação:

  http://www.inf.ufpr.br/aghochuli/caffe/

  http://caffe.berkeleyvision.org/installation.html
- Tutorial Configuração da CNN (layers):
  http://caffe.berkeleyvision.org/tutorial/

FrameWork CAFFE DEEP LEARNING

- lenet_solver.prototxt

```
# The train/test net protocol buffer definition
net: "dummy/models/lenet/lenet_train_val.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 2000
snapshot_prefix: "dummy/models/lenet/lenet"
# solver mode: CPU or GPU
solver_mode: GPU
```

FrameWork CAFFE DEEP LEARNING

- lenet_train_val.prototxt : Data Sources

```
name: "LeNet"
layer {
  name: "script"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "dummy/data/dummy_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
```

1 / 256

```
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "dummy/data/dummy_val_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
```

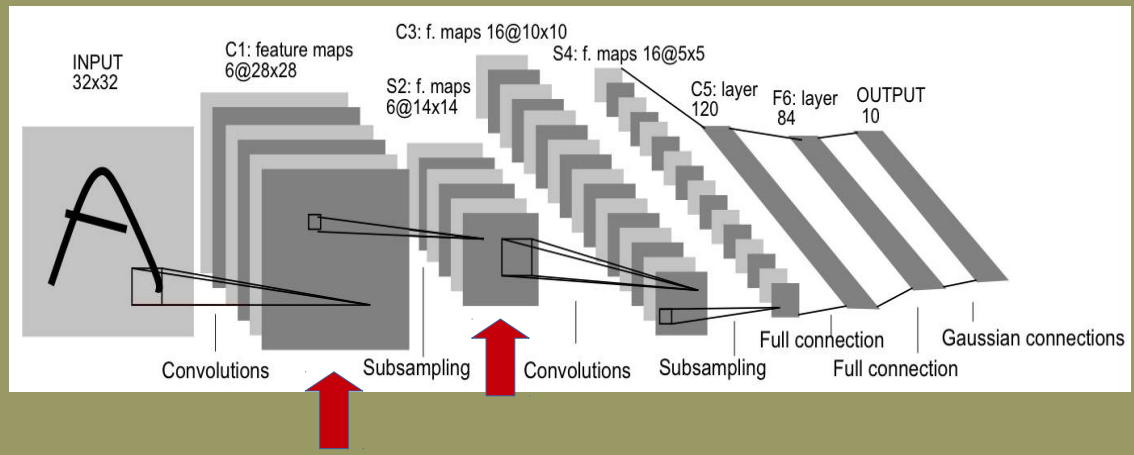
1 / 256

FrameWork CAFFE DEEP LEARNING

- lenet_train_val.prototxt : Layers

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 6
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

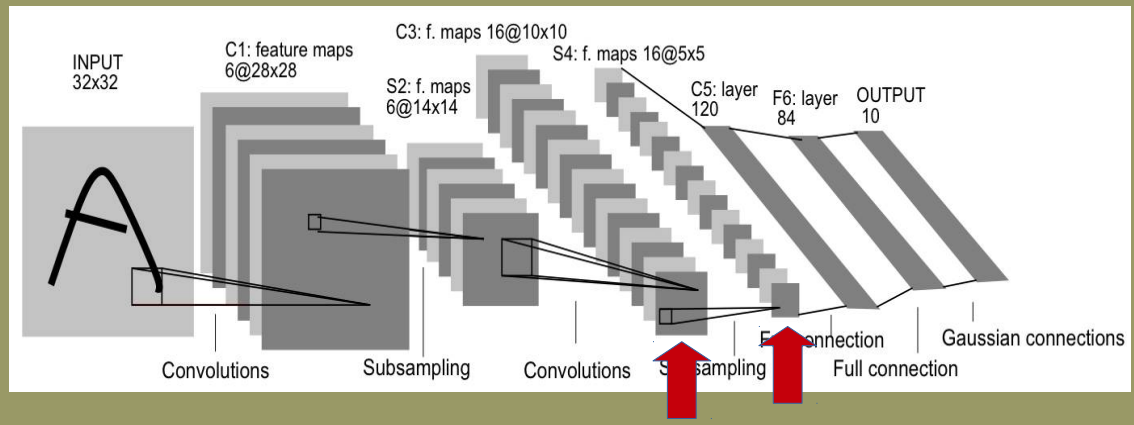


FrameWork CAFFE DEEP LEARNING

- lenet_train_val.prototxt : Layers

```
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 16
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

```
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```



FrameWork CAFFE DEEP LEARNING

- lenet_train_val.prototxt : Layers

```
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 120
    kernel_size: 1
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

```
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "conv3"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 84
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
```

```
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```