

# Deep Learning e Redes Convolutivas

Valner Brusamarello<sup>1</sup>

<sup>1</sup>Universidade Federal do Rio Grande do Sul



## Outline

### CNETs - Introdução

### Definições relacionadas às CNETs

### Encaminhamentos

## Linhas Gerais

1. Precisamos inicialmente de uma estratégia para classificar os objetos: definir classes e esboçar uma banco de imagens (estáticas).
2. Definir e treinar uma rede para classificar os objetos
3. Definir uma estratégia para rastreamento dos objetos (em movies): boas referência nos levam aos *bounding boxes*
4. Vamos iniciar pelas redes neurais convolucionais ou CNETs

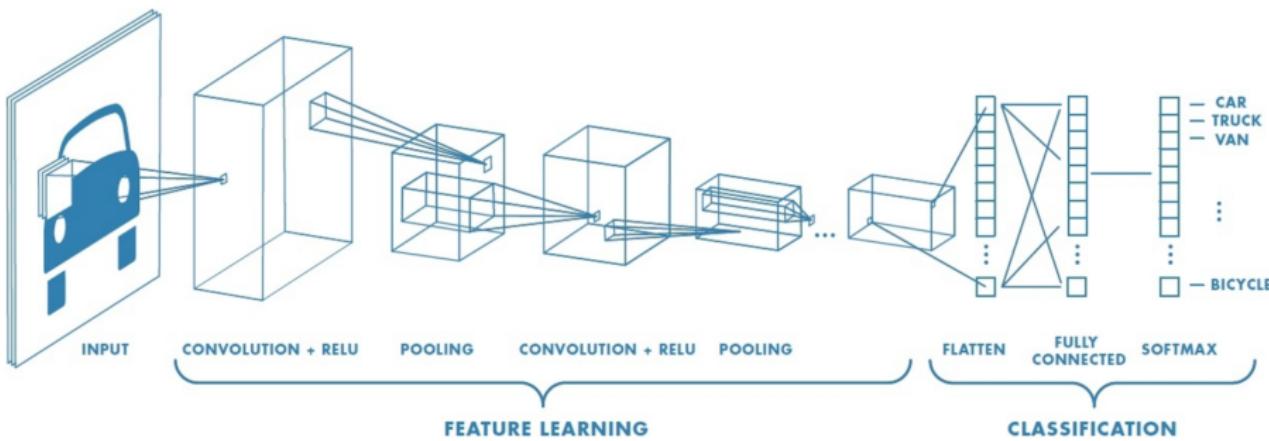
## Convolutional Neural Networks - CNN ou CNETs

- As CNNs são modelos biológicos inspirados nas pesquisas de D. H. Hubel e T. N. Wiesel. Eles propuseram uma explicação para a maneira pela qual os mamíferos percebem visualmente o mundo ao seu redor usando uma arquitetura em camadas de neurônios no cérebro, e isso, por sua vez, inspirou os engenheiros a tentar desenvolver mecanismos de reconhecimento de padrões semelhantes na visão por computador. Em sua hipótese, dentro do córtex visual, respostas funcionais complexas geradas por "células complexas" são construídas a partir de respostas mais simplistas de "células simples". Por exemplo, células simples respondem a arestas orientadas, etc, enquanto células complexas também respondem a arestas orientadas, mas com um certo grau de invariância espacial.

# CNN - Introdução

- Como exemplo, como um humano reconhece que é um carro? Basicamente, procuramos as características exclusivas de um carro: rodas, faróis, portas, porta-malas traseira, janelas de vidro, capô e outros recursos que o diferenciam de outros modos de transporte.
  - Da mesma forma, ao reconhecer uma roda, procuramos objetos de formato circular, comparativamente escuros com uma textura áspera, posicionados abaixo da estrutura principal do carro. Levamos em consideração todos os pequenos detalhes que juntos constituem uma informação básica.
  - Essas pequenas informações são agrupadas para formar uma característica específica que é única para um objeto que estamos reconhecendo.

## CNN - Introdução



## Figure: Convolutional Neural Network

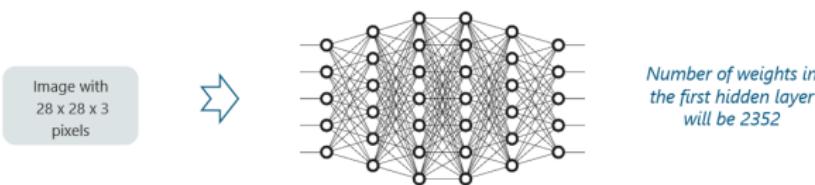
## CNN - Referências

Essa apresentação foi baseada nos seguintes sites:

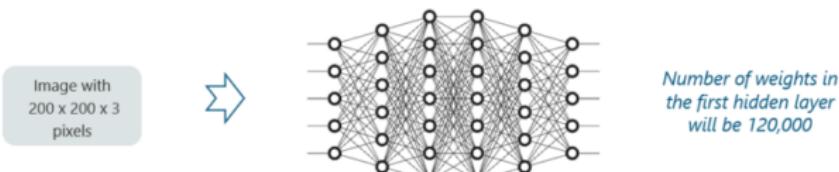
- ► Link
  - ► Link
  - ► Link
  - ► Link
  - ► Link

## Porque CNN?

Por que não redes totalmente conectadas? Consideremos uma entrada de imagens com o tamanho de 28x28x3 pixels. Se inserirmos isso em nossa Rede Neural, teremos cerca de 2352 pesos na primeira camada oculta.



Porém, usualmente uma imagem de entrada genérica terá pelo menos 200x200x3 pixels e o tamanho da primeira camada oculta se torna 120.000. Mas podemos ter várias camadas!



## CNN - Introdução

A CNN é uma combinação de dois elementos básicos:

- O bloco de convolução (Convolutional Block) - consiste na camada de convolução e na camada de pool (Pooling Layer). Essa camada forma o componente essencial da extração de *features*.
- O bloco totalmente conectado (fully connected Block) - consiste em uma arquitetura de rede neural simples totalmente conectada. Essa camada executa a tarefa de classificação com base na entrada do bloco convolucional.

## Convolutional Layer

A CAMADA CONVOLUCIONAL está relacionada à extração de características.

- Filtros: filtros ou "kernels" também são uma imagem que representa uma característica específica. Por exemplo, vamos tirar uma foto dessa curva. Tomamos isso como um recurso de amostra que tentaremos reconhecer, isto é, determinar se ele está presente em uma imagem.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

## CNN - Introdução

Considere a matriz do fundo  $5 \times 5$  e o filtro  $3 \times 3$  com  $x_0 = 0$  e  $x_1 = 1$

1 $\times 1$	1 $\times 0$	1 $\times 1$	0	0
0 $\times 0$	1 $\times 1$	1 $\times 0$	1	0
0 $\times 1$	0 $\times 0$	1 $\times 1$	1	1
0	0	1	1	0
0	1	1	0	0

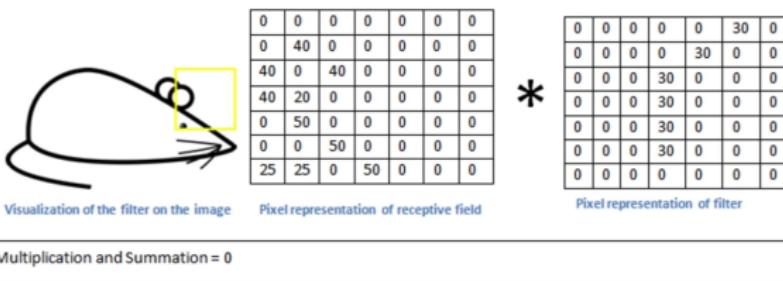
Image

4		

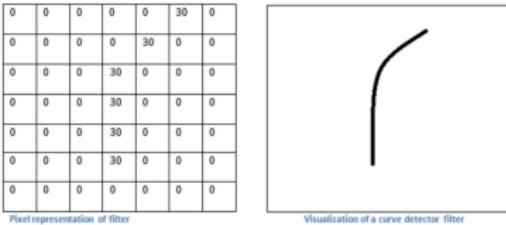
Convolved Feature

## Bloco de Convolução

Agora podemos encontrar uma feature específica com a "matriz de filtro" sobre a matriz da imagem, constituindo outra matriz.

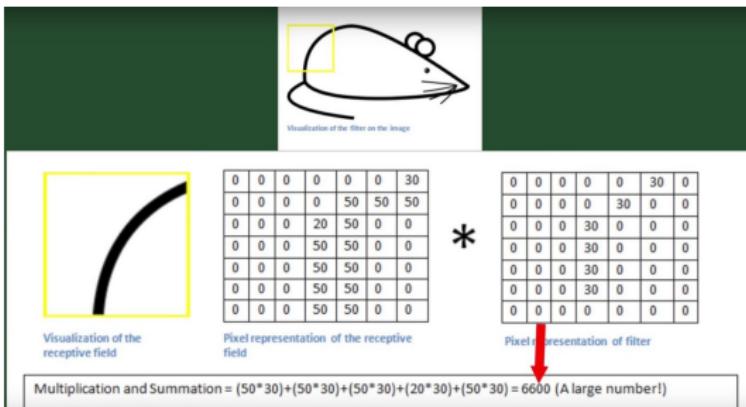


**Figure:** Convolutional Neural Network

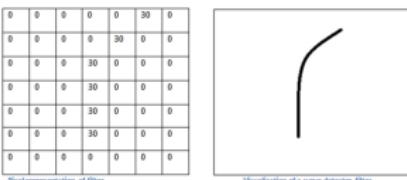


## Bloco de Convolução

Agora podemos encontrar uma feature específica com a "matriz de filtro" sobre a matriz da imagem, constituindo outra matriz.



**Figure:** Convolutional Neural Network



## CNN - Bloco de Convolução

- A parte traseira do mouse tem uma forma semelhante ao filtro e dessa forma o resultado vai ser elevado.
- A parte anterior do mouse tem uma forma muito diferente ao filtro e dessa forma o resultado é muito baixo
- Aplicamos essa técnica na extração de features.
- Considere uma imagem colorida, ou seja, uma matriz 2-D com 3 canais (cores) RGB.
- Cada matriz de filtro é convoluída sobre a matriz de imagens de todos os canais: vermelho, verde e azul e adicionamos os valores de cada canal para formar o valor da célula da matriz de saída.
- Ao fazer isso, estamos essencialmente tentando descobrir se uma característica específica está presente na imagem.

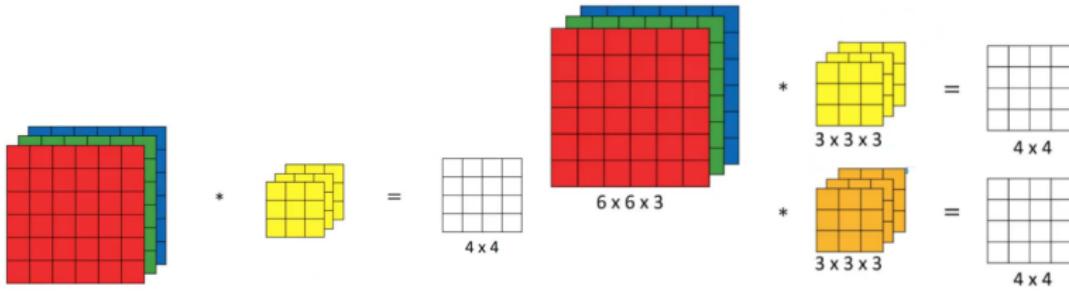
## Filtros

- Ao construir a rede, especificamos aleatoriamente valores para os filtros, que se atualizam continuamente à medida que a rede é treinada.
- É pouco provável que dois filtros iguais sejam produzidos, a menos que o número de filtros escolhidos seja extremamente grande.

$$\text{Edge detection}$$
$$\begin{matrix} * & \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} & = & \text{Kernel} \\ & & & \downarrow \\ & & & \text{Output image showing edges} \end{matrix}$$
$$\text{Sharpen}$$
$$\begin{matrix} * & \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} & = & \text{Output image showing sharpened features} \end{matrix}$$

## Convolução sobre Volume

Supondo uma entrada  $6 \times 6 \times 3$  e um filtro  $3 \times 3 \times 3$  (altura-h, largura-w e canais-c) e depois com 2 filtros  $3 \times 3 \times 3$ :



## 3D Convolution Layer

0	0	0	0	0	0	0	...
0	156	155	156	158	158	...	
0	153	154	157	159	159	...	
0	149	151	155	158	159	...	
0	146	146	149	153	158	...	
0	145	143	143	148	158	...	
...	...	...	...	...	...	...	

Input Channel #1 (Red)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

+

0	0	0	0	0	0	0	...
0	167	166	167	169	169	...	
0	164	165	168	170	170	...	
0	160	162	166	169	170	...	
0	156	156	159	163	168	...	
0	155	153	153	158	168	...	
...	...	...	...	...	...	...	

Input Channel #2 (Green)

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	0	0	0	0	0	0	...
0	163	162	163	165	165	...	
0	160	161	164	166	166	...	
0	156	158	162	165	166	...	
0	155	155	158	162	167	...	
0	154	152	152	157	167	...	
...	...	...	...	...	...	...	

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

$$+ 1 = -25$$

$$\begin{array}{c} \uparrow \\ \text{Bias} = 1 \end{array}$$

Output

-25					...
					...
					...
					...
...	...	...	...	...	...

## Outras Operações Necessárias

- Padding
- os valores das células que estão dentro da matriz são considerados à medida que a matriz de filtro se move por toda a imagem na operação de convolução.
- Às vezes, os valores das células nos cantos ou nas bordas são contabilizados e os valores nos cantos ou nas bordas não recebem uma ponderação igual.
- Para superar isso, adicionamos outra linha e coluna, de apenas 0, em todos os lados da matriz da imagem.
- Essa ideia é conhecida como padding.
- Esses valores sendo '0' não fornecem informação extra, mas ajudam a contabilizar os valores anteriormente menos contabilizados para receberem mais peso.

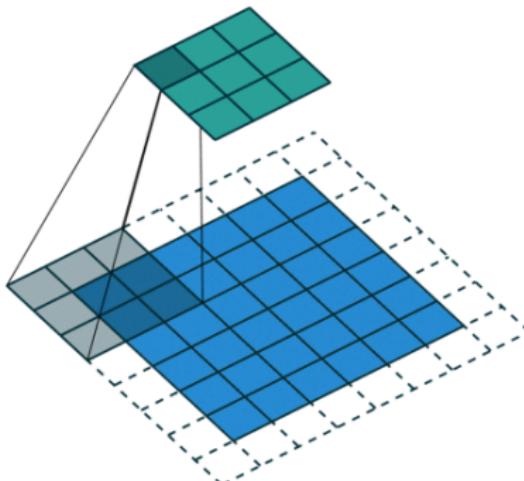
## Padding

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

**Figure:** Padding

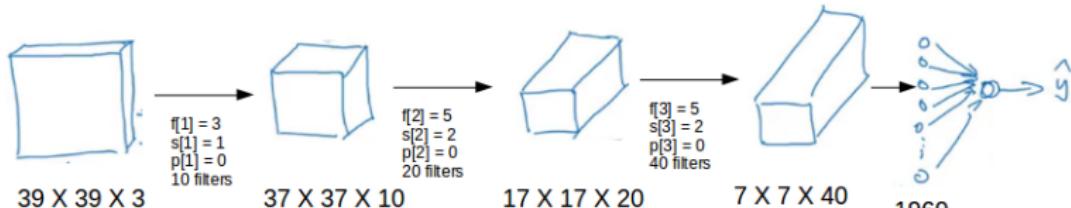
## Striding

Na convolução striding, em vez de mudar o filtro uma linha ou uma coluna de cada vez, podemos alterá-lo, talvez, 2 ou 3 linhas ou colunas, a cada vez. Reduz os cálculos e também o tamanho da matriz de saída. Para imagem grande, isso não resulta em perda de dados, mas reduz o custo de computação em larga escala.

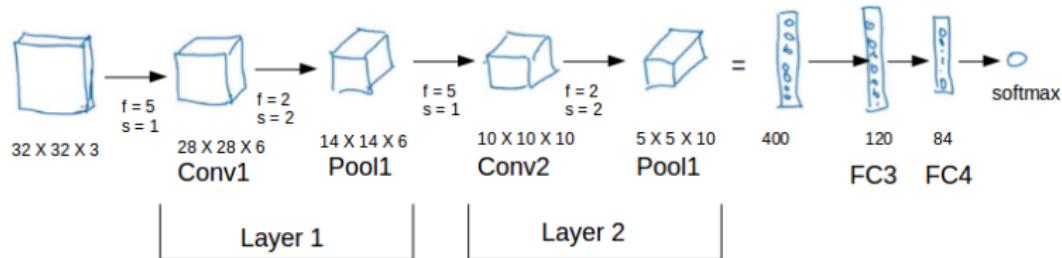


## Simple Convolution Network

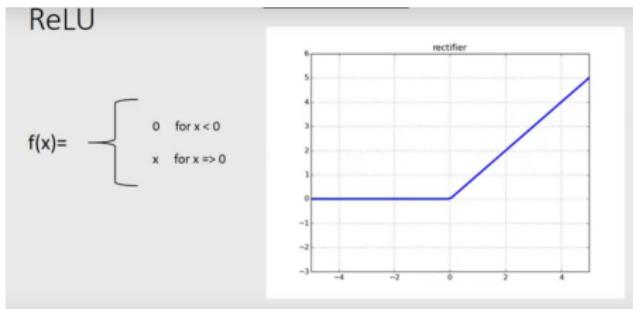
$f$ =filter size,  $s$ =striding,  $p$ =padding  $n$ =number of filters



$$\begin{aligned} nh[1] &= nw[1] = (n + 2p - f) / s + 1 \\ nh[1] &= nw[1] = (39 + 0 - 3) / 1 + 1 \\ nh[1] &= nw[1] = 37 \end{aligned}$$



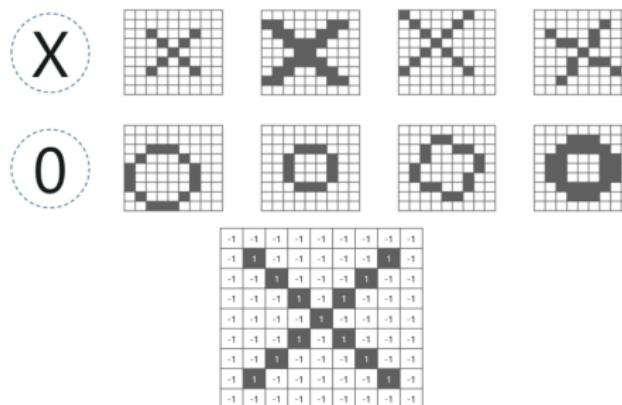
## RELU activation: RELU ou Unidade Linear Retificadora



- Após a convolução, se uma função de convolução específica resultar em 0 ou um valor negativo, então a *feature* não está presente e a saída será 0 (retificador). Para todos os outros casos, mantemos o valor.
- Com essas operações e funções aplicadas na imagem de entrada, formamos a primeira parte do Bloco Convolucional.

## Exemplo de Funcionamento

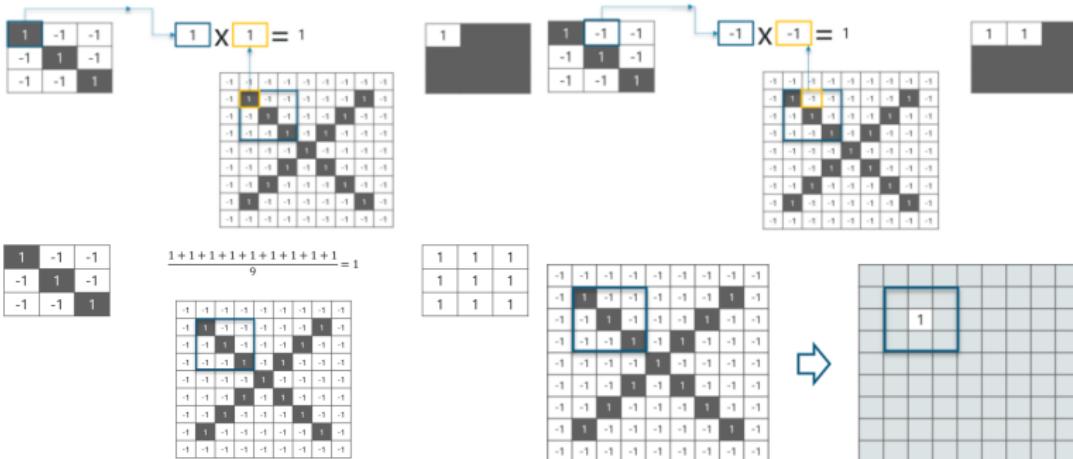
Considere uma matriz que represente execuções de  $X$  e  $O$ . O sinal de referência da "imagem" será convoluído com o sinal de referência.



- Os pixels brancos são  $-1$  e os pretos são  $1$ , para uma classificação binária básica.

## Exemplo

Cada filtro de convolução representa uma característica de interesse e o algoritmo de CNN aprende quais características mais importantes. Considere a imagem com os passos da convolução com o filtro da matriz menor.

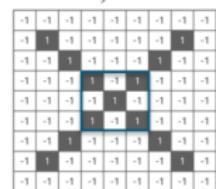


## Exemplo

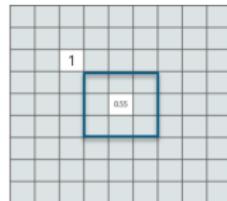
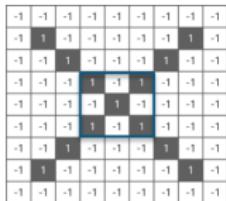
Consideremos o mesmo filtro em outra parte da imagem:

1	-1	-1
-1	1	-1
-1	-1	1

$$\frac{1 + 1 - 1 + 1 + 1 - 1 + 1 + 1}{9} = .55$$



1	1	-1
1	1	1
-1	1	1

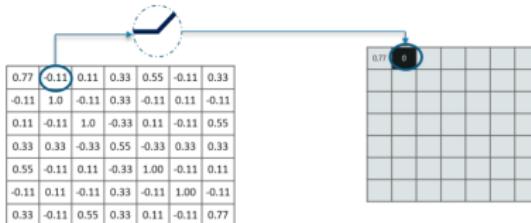
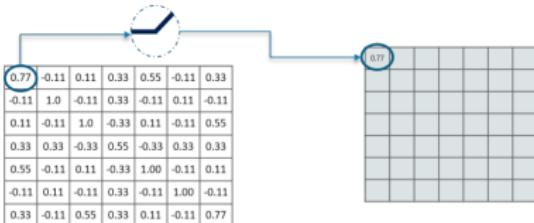


Depois de passar o filtro por toda a imagem, temos:

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

## Exemplo - RELU

O resultado anterior passa pelo bloco de ativação RELU e todos os valores negativos passam a ser zero



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



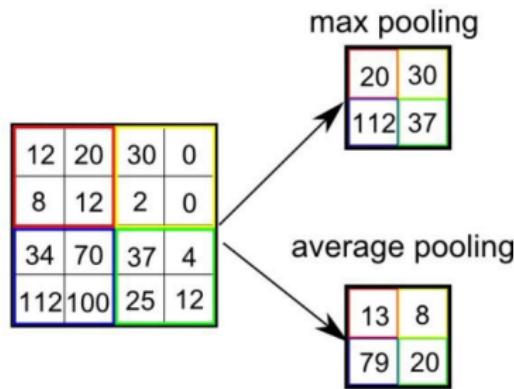
0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77

- As entradas da camada de convolução podem ser "suavizadas" para reduzir a sensibilidade dos filtros a ruídos e variações. Esse processo de suavização é chamado de subamostragem e pode ser alcançado através da média ou do máximo de uma amostra do sinal.

## Pooling Layer

- O Pooling consiste em extrair um valor específico de um conjunto de dados, geralmente o valor máximo ou o valor médio de todos os valores.
- Intuito de reduzir o tamanho da matriz de saída.
- É comum inserir periodicamente uma camada Pooling entre blocos convolucionais sucessivos em uma arquitetura da CNN.
- Sua função é reduzir progressivamente o tamanho espacial da representação para reduzir o número de parâmetros e a computação na rede.

## Pooling Layer



- Escolha um tamanho de janela (geralmente 2 ou 3); escolha um passo (geralmente 2) (striding); passe a janela pelas imagens filtradas; de cada janela, pegue o valor máximo (max pooling)

## Pooling Layer

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

## Pooling

note que iniciamos com uma matrix  $7 \times 7$  e depois do pooling resulta em  $4 \times 4$ .

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.33	0	0.11	0	0.11	0	0.33
0	0.55	0	0.33	0	0.55	0
0.11	0	0.33	0	0.55	0	0.11
0	0.33	0	1.00	0	0.33	0
0.11	0	0.55	0	0.55	0	0.11
0	0.55	0	0.33	0	0.55	0
0.33	0	0.11	0	0.11	0	0.33



0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

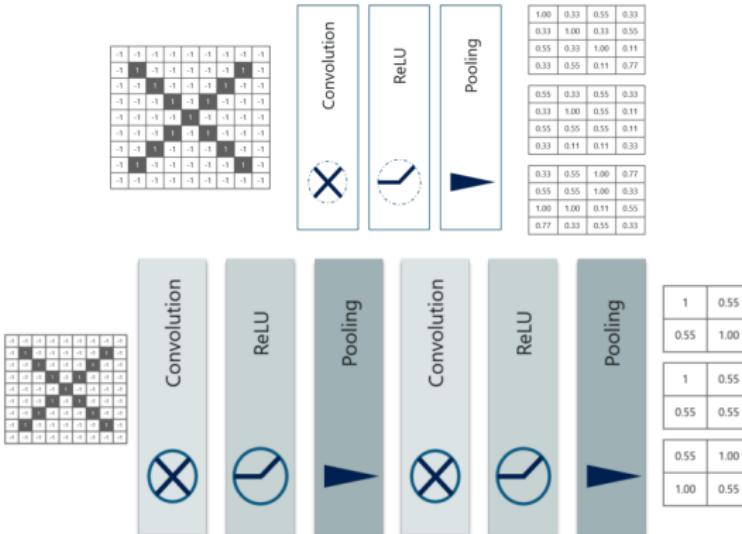
0.88	0	0.08	0.08	0.11	0	0.77
0	0.11	0	0.08	0	0.14	0
0.08	0	0.11	0	0.14	0	0.11
0.08	0.08	0	0.08	0	0.08	0.08
0.11	0	0.14	0	0.11	0	0.08
0	0.14	0	0.08	0	0.11	0
0.08	0	0.11	0	0.08	0	0.08



0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

## Empilhando Camadas

Veja que de uma matriz  $9 \times 9$  terminamos com uma matriz  $4 \times 4$ , vinda de uma matriz  $7 \times 7$ , depois de passar por uma banco de 3 filtros de convolução (nesse exemplo), ReLU e Pooling... e podemos repetir !!!

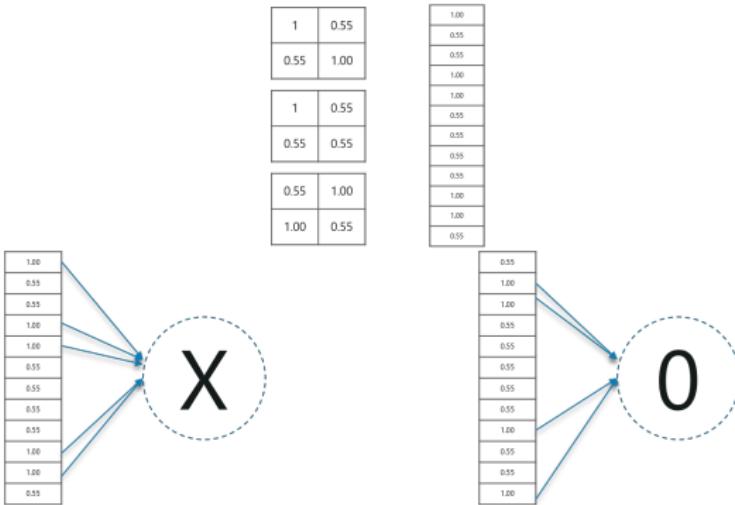


## Fully Connected Layer

- As últimas camadas da rede estão totalmente conectadas, o que significa que os neurônios das camadas anteriores estão conectados a todos os neurônios nas camadas subsequentes.
- Essa camada forma o último bloco da arquitetura da CNN, relacionado à tarefa de classificação.
- Esta é essencialmente uma Rede Neural Simples totalmente conectada, consistindo em duas ou três camadas ocultas e uma camada de saída geralmente implementada usando a 'Regressão Softmax', que executa o trabalho de classificação entre um grande número de categorias.

## Fully Connected Layer

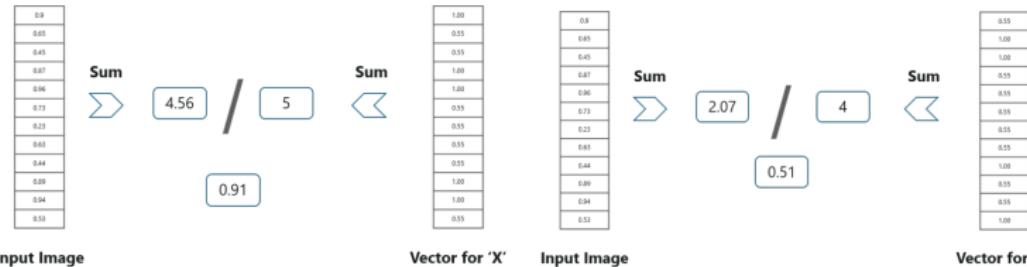
Considere a imagem abaixo, como você pode ver em  $X$ , existem diferentes elementos altos e da mesma forma, para  $O$  temos diferentes elementos altos:



## Previsão de imagem usando redes neurais convolucionais

Assim, terminamos o treinamento da rede e podemos começar a prever e verificar o funcionamento do classificador.

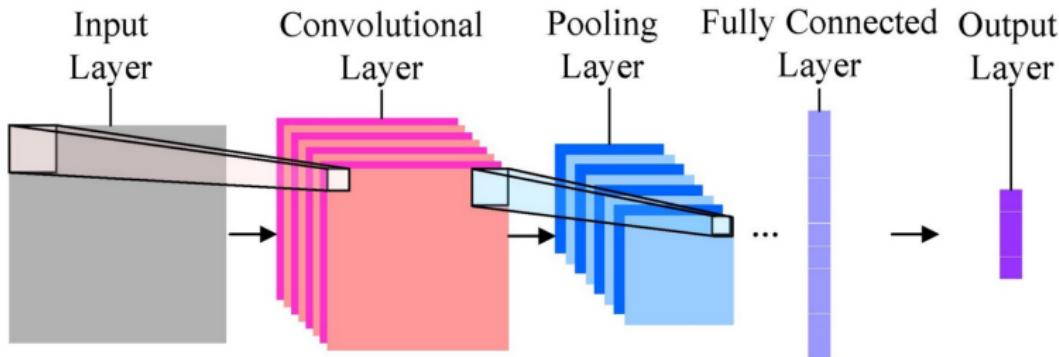
- Nos exemplos, a CNN produz um vetor de 12 elementos.
- Podemos fazer previsões de novas entradas comparando as saídas obtidas com a lista de  $x$  e  $O$  já processadas e que servem de referência.



## Todos os Blocos Juntos

1. Imagem 2D definida para os três canais de cores com dimensão  $n \times n$ , → entrada é uma matriz  $n \times n \times 3$ .
2. Convolução com matrizes  $f \times f$  de  $k$  filtros.
3. PADDING na imagem com com  $p$  linhas e com  $p$  colunas.
4. Realizamos a operação de convolução (striding).
5. POOLING sobre a matriz de saída de cada camada.
6. Repetimos as operações das etapas 3 a 5.
7. Classificação na rede totalmente conectada.

## Todos os Blocos Juntos



## Estruturas de Redes

Existem várias arquiteturas de CNNs disponíveis que foram fundamentais na criação de algoritmos que potencializaram e impulsionaram o desenvolvimento de IA. Alguns deles estão listados abaixo:

- LeNet
- AlexNet
- VGGNet
- GoogLeNet
- ResNet
- ZFNet

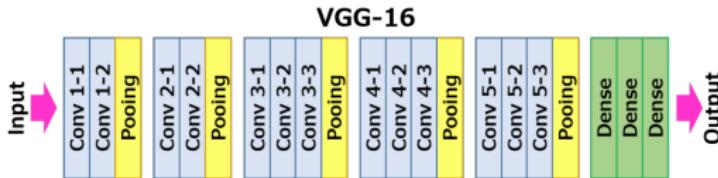
## Estruturas de Redes - Várias Estruturas surgiram nessa competição

Large Scale Visual Recognition Challenge (ILSVRC) - The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale. One high level motivation is to allow researchers to compare progress in detection across a wider variety of objects – taking advantage of the quite expensive labeling effort. Another motivation is to measure the progress of computer vision for large scale image indexing for retrieval and annotation.

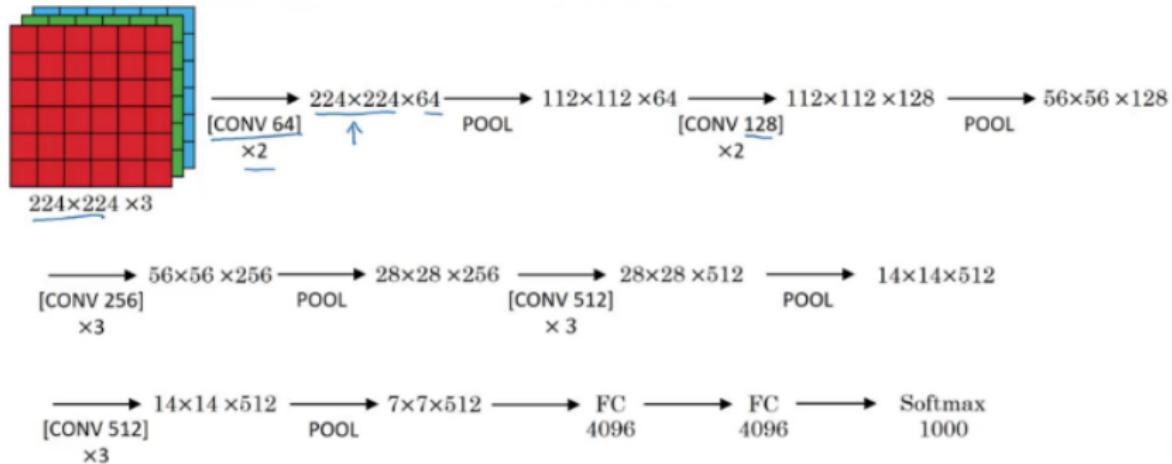
▶ Link

## VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman. The model achieves 92.7% top-5 acc in ImageNet (dataset of over 14 million images from 1000 classes). It was one of the famous model submitted to ILSVRC2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple  $3 \times 3$  kernel filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.

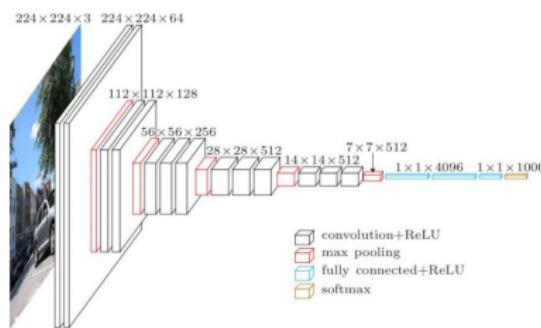


# VGG16



138 milhões de parâmetros

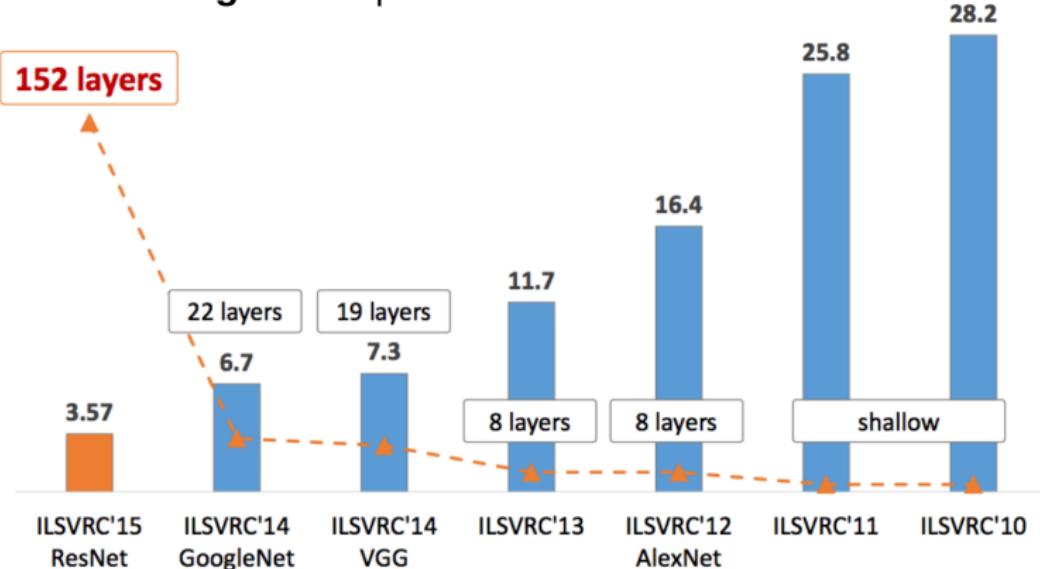
# VGG16



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 LRN	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# RESnet

## Desafio ImageNet: top-5 error



# RESnets

Para resnet50: [Link](#), [Link](#)

## Revolution of Depth

AlexNet, 8 layers  
(ILSVRC 2012)



VGG, 19 layers  
(ILSVRC 2014)

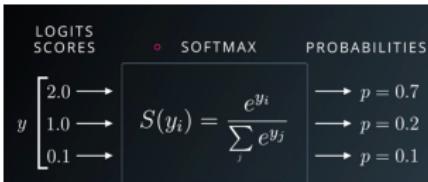


ResNet, **152 layers**  
(ILSVRC 2015)



## Bloco Softmax

- Uma função de ativação que transforma logits decorrentes do processo em probabilidades que somam um. *In deep learning, the term logits layer is popularly used for the last neuron layer of neural network for classification task which produces raw prediction values as real numbers ranging from  $[-\infty, +\infty]$ . — Wikipedia*
- A função Softmax gera um vetor que representa as distribuições de probabilidade de uma lista de possíveis resultados.
- É também um elemento central usado em tarefas de classificação de aprendizado profundo.



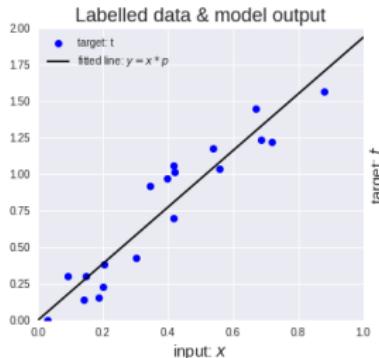
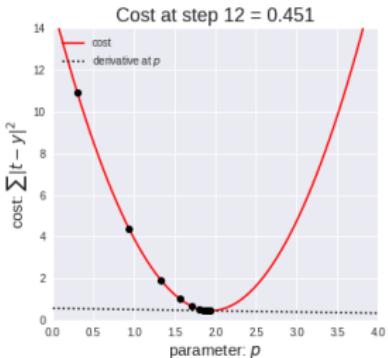
## Loss Function

- Uma CNN aprende a mapear um conjunto de entradas para um conjunto de saídas com dados de treinamento.
- O aprendizado é apresentado como um problema de otimização e um algoritmo é usado no espaço de possíveis conjuntos de pesos que o modelo pode usar para fazer previsões boas o suficiente.
- O algoritmo de otimização de descida de gradiente (descend Gradient) estocástico e os pesos são atualizados usando o backpropagation baseado no erro.
- O algoritmo de descida de gradiente procura alterar os pesos para que na próxima avaliação o erro seja reduzido.
- A função usada para avaliar uma solução candidata é chamada de função objetivo.

## Loss Function

- A função objetivo é frequentemente referida como uma função de custo ou uma função de perda (Loss) e o valor calculado pela função de perda é referido simplesmente como "perda" (Loss).
  - A função de custo ou perda deve destilar fielmente todos os aspectos do modelo em um único número, de forma que as melhorias nesse número sejam um sinal de um modelo melhor.
  - A função de custo reduz todos os vários aspectos bons e ruins de um sistema possivelmente complexo para um único número, um valor escalar, que permite que as soluções candidatas sejam classificadas e comparadas.
  - Ex.: de funções para calcular Loss: Cross-Entropy Loss (or Log Loss) e Mean Squared Error Loss

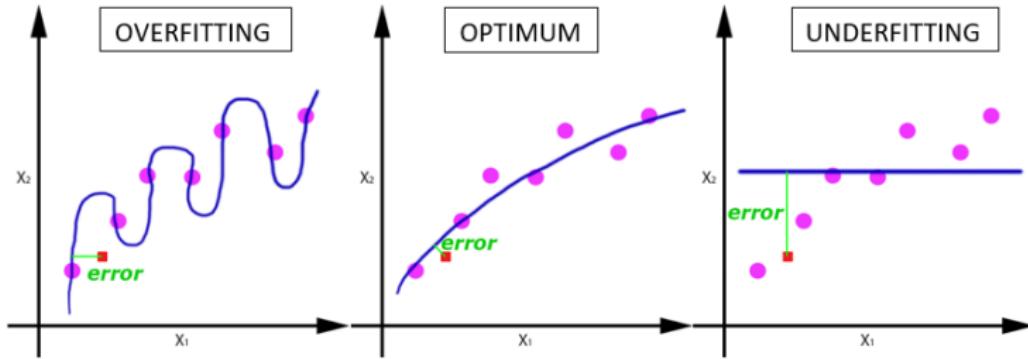
# Gradient Descent



## Gradient Descent

- A descida do gradiente possui um parâmetro chamado taxa de aprendizado.
- Além disso, a função de custo está diminuindo ou o custo ou a perda (Loss) está diminuindo.
- Problemas dessa natureza base dados de referência (Datasets) muito grandes.
- Epochs: uma época representa um conjunto de dados INTEIRO transmitido para frente e para trás através da rede neural apenas UMA VEZ.
- Como uma época é muito grande, a dividimos em vários lotes menores.
- Por que usamos mais de uma época? Usamos um conjunto de dados limitado com aprendizado otimizado com o Gradient Descent, um processo iterativo e limitado.

## Epochs - Underfitting x Overfitting



## Batch Size - Número de exemplos de treinamento presentes em um único lote.

- O tamanho do lote e o número de lotes são duas coisas diferentes.
- Não podemos passar o conjunto de dados inteiro para a rede neural de uma vez, então dividimos o conjunto de dados em Número de lotes.
- Iterações: é o número de lotes necessários para concluir uma época.
- O número de lotes é igual ao número de iterações para uma época.
- Digamos que temos 2000 exemplos de treinamento: Podemos dividir em lotes de 500, sendo necessárias 4 iterações para concluir 1 época.
- O Tamanho do lote é 500 e o de Iterações é 4, para 1 época completa.

## Transferência de Aprendizado

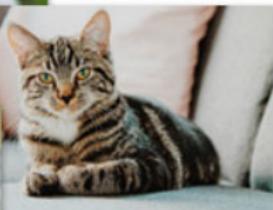
- Adotar uma rede pré-treinada em um conjunto de dados
- Utilizá-la para reconhecer categorias de imagem ou objeto em que não foi treinado
- Podemos utilizar os filtros discriminatórios robustos aprendidos pelas redes estado da arte em conjuntos de dados desafiadores (como ImageNet ou COCO) e, em seguida, aplicar essas redes para reconhecer objetos nos quais o modelo nunca foi treinado.

## Transferência de Aprendizado

- Em geral, existem dois tipos de transferência de aprendizagem no contexto da aprendizagem profunda
- **Transferência de aprendizado por meio da extração de *features*:** tratamos a rede pré-treinada como um extrator de recursos arbitrário, permitindo que a imagem de entrada se propague para a frente, parando na camada pré-especificada e tomando as saídas dessa camada como nossos recursos.
- **Transferência de aprendizado através de ajuste fino:** exige que atualizemos a arquitetura do modelo removendo as cabeças de camada totalmente conectadas anteriores, fornecendo novas e recém inicializadas e treinando as novas camadas FC para prever nossas classes de entrada.

## Transferência de Aprendizado

Uma rede ensinada a classificar dogs e cats pode receber algum aprendizado prévio para classificar ursos.



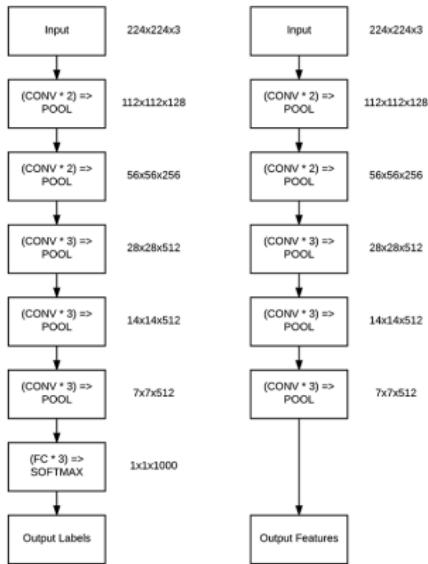
## Transferência de Aprendizado

- Em uma abordagem usual reuniríamos um conjunto de dados rotulados de cães e gatos, seguido pelo treinamento de um modelo.
- Repetiríamos o processo reunindo imagens de raças de ursos e treinando um modelo como conjunto de dados rotulado.
- A transferência de aprendizado propõe utilizar um classificador pré-treinado existente como ponto de partida para uma nova classificação, detecção de objeto ou tarefa de segmentação.
- Proposta: treine uma Rede Neural Convolucional para reconhecer cães versus gatos e em seguida, use a mesma CNN para distinguir entre as classes de ursos, mesmo que nenhum dado tenha sido misturado aos dados de cães e gatos durante o treinamento inicial

## Transferência de Aprendizado - feature extraction

- Redes neurais profundas treinadas em conjuntos de dados em larga escala, como ImageNet e COCO, provaram ser excelentes na tarefa de transferência de aprendizado.
- Essas redes aprendem um conjunto de recursos ricos e discriminatórios capazes de reconhecer de 100 a 1.000 de classes de objetos - faz sentido que esses filtros possam ser reutilizados para tarefas diferentes daquelas em que a CNN foi originalmente treinada.
- Um tipo de transferência de aprendizado trata redes como extratores arbitrários de recursos. A estratégia é remover as camadas totalmente conectadas de uma rede existente, colocando um novo conjunto de camadas FC na CNN e ajustando esses pesos (e opcionalmente as camadas anteriores) para reconhecer as novas classes de objetos.

## Transfer learning via feature extraction



**Figure:** Left: The original VGG16 network architecture that outputs probabilities for each of the 1,000 ImageNet class labels. Right: Removing the FC layers from VGG16 and instead of returning the final POOL layer. This output will serve as our extracted features.

## Transferência de Aprendizado - feature extraction

- Usualmente tratamos uma Rede Neural Convolutional como um classificador de imagem de ponta a ponta.
- A imagem de entrada se propaga pela rede e obtemos nossas probabilidades finais de classificação no final da rede.
- Ao invés, vamos interromper a propagação em uma camada específica e extraia os seus valores.
- Ao tratar as redes como extratores de recursos, nós essencialmente “cortamos” a rede em nossa camada pré-especificada (normalmente antes das camadas totalmente conectadas, mas isso realmente depende do seu conjunto de dados específico).

## Transferência de Aprendizado - feature extraction

- Se interrompermos antes das camadas totalmente conectadas no VGG16, a última camada da rede se tornaria a camada de pool máximo (Figura à direita), que terá uma forma de saída de  $7 \times 7 \times 512 = 25.088$  valores - nosso vetor usado para quantificar a imagem de entrada.
- Repetimos o processo para todo o conjunto de dados de imagens.
- Dado um total de  $N$  imagens em nossa rede, nosso conjunto de dados agora seria representado como uma lista de  $N$  vetores de 25.088.
- Assim que tivermos nossos vetores de recursos, podemos treinar modelos de aprendizado de máquina disponíveis, como SVM linear, regressão logística, árvores de decisão ou florestas aleatórias, sobre esses recursos para obter um classificador que possa reconhecer novas classes de imagens.

## Transferência de Aprendizado - feature extraction

- Os dois modelos mais comuns de aprendizado de máquina para o aprendizado de transferência por meio da extração de recursos são a Regressão logística e SVM linear.
- Precisamos de um modelo rápido que possa ser treinado sobre os recursos aprendidos pela CNN (robustos e discriminatórios), cujos vetores tendem a ser muito grandes e ter alta dimensionalidade.
- Um conjunto de dados de 5.000 imagens, cada um representado por um vetor de característica de 25.088 dim, pode ser treinado em alguns segundos usando um modelo de regressão logística.
- Tenha em mente que a própria CNN não é capaz de reconhecer essas novas classes e estamos usando a CNN como um extrator intermediário de recursos.
- O classificador de aprendizado de máquina a cuidará do aprendizado dos padrões subjacentes dos recursos

## Further Reading about features extraction

- [▶ Link](#)
- [▶ Link](#)

## Transferência de Aprendizado - Fine-tuning

O Fine-Tuning requer que não apenas atualizemos a arquitetura da CNN, mas também a treinemos para aprender novas classes de objetos.

- Remover os nós totalmente conectados no final da rede.
- Substituir os nós totalmente conectados pelos recém-inicializados.
- Congelar camadas CONV anteriores da rede (garantindo que quaisquer recursos robustos aprendidos não sejam destruídos).
- Treinar apenas as cabeças da camada FC.
- Como opção, descongelar algumas camadas CONV da rede e realizar uma segunda passagem de treinamento.

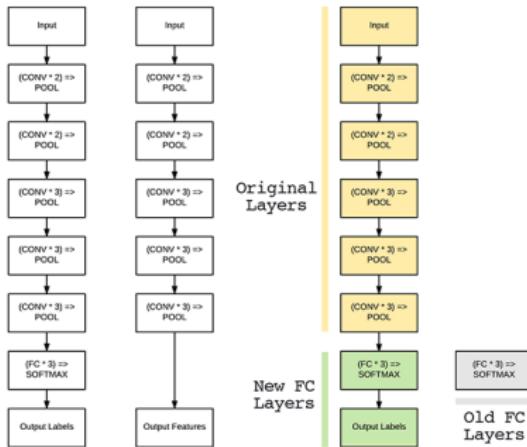
## Transferência de Aprendizado - Fine-tuning

- O conjunto final de camadas totalmente conectadas (CTC) são cortadas de uma CNN pré-treinada (normalmente VGG, ResNet ou Inception).
- Essa etapa é substituída por um novo conjunto de camadas totalmente conectadas por inicializações aleatórias.
- Todas as camadas abaixo da CTC são congeladas (seus pesos não podem ser atualizados) pelo backpropagation.
- A partir daí treinamos a rede usando uma taxa de aprendizado muito pequena para que o novo conjunto de camadas totalmente conectadas possa aprender padrões das camadas CONV aprendidas anteriormente - esse processo é chamado de “warm-up” das CTC.

## Transferência de Aprendizado - Fine-tuning

- Opcionalmente, pode-se descongelar o restante da rede e continuar o treinamento. A aplicação do ajuste fino nos permite utilizar redes pré-treinadas para reconhecer as classes nas quais não foram originalmente treinadas.
- Esse método pode levar a uma precisão mais alta do que a transferência de aprendizado por meio da extração de recursos.

## Transfer learning via fine-tuning

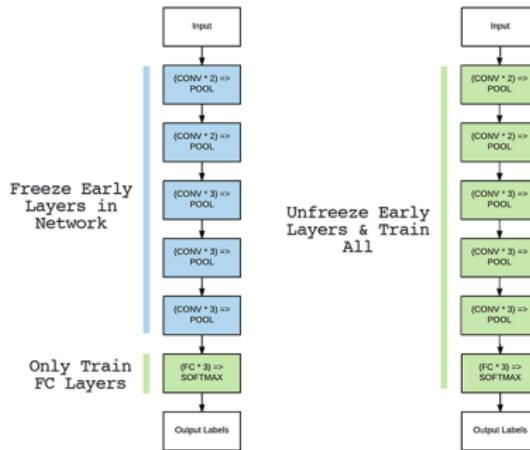


**Figure:** Left: The original VGG16 network architecture. Middle: Removing the FC layers from VGG16 and treating the final POOL layer as a feature extractor. Right: Removing the original FC Layers and replacing them with a brand new FC head. These FC layers can then be fine-tuned to a specific dataset (the old FC Layers are no longer used).

## Transfer learning via fine-tuning

- Diferente da extração de recursos, quando realizamos o ajuste fino, construímos uma nova FCL e o colocamos sobre a arquitetura original.
- A nova camada FC é inicializada aleatoriamente (como qualquer outra camada em uma nova rede) e conectada ao corpo da rede original.
- Deixamos nossa FCL "warm-up" "congelando" todas as camadas no corpo da rede.

## Transfer learning via fine-tuning



**Figure:** When we start the fine-tuning process, we freeze all CONV layers in the network and only allow the gradient to backpropagate through the FC layers. Doing this allows our network to “warm up”. Right: After the FC layers have had a chance to warm up, we may choose to unfreeze all or some of the layers earlier in the network and “rewarm” each of them to be fine-tuned as well.

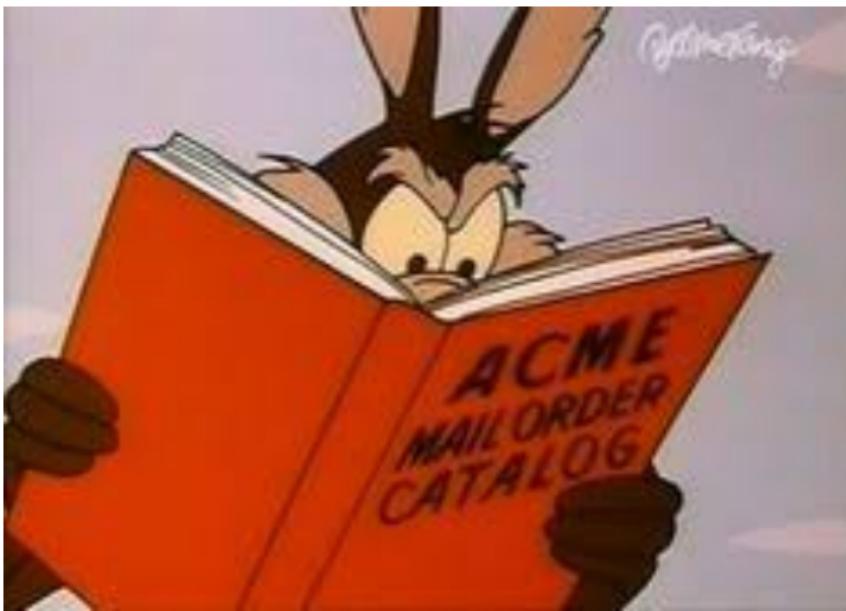
## Transfer learning via fine-tuning

- Os dados de treinamento são propagados para a frente através da rede; no entanto, a retropropagação é interrompida após as camadas FC, o que permite que essas camadas começem a aprender padrões das camadas CONV altamente discriminativas.
- Em alguns casos, podemos decidir nunca descongelar o corpo da rede, pois nossa nova FC pode obter precisão suficiente.
- No entanto, para alguns conjuntos de dados, geralmente é vantajoso permitir que as camadas CONV originais sejam modificadas também durante o processo de ajuste fino (Figura da direita).

## Transfer learning via fine-tuning

- Depois que o chefe do FC começar a aprender padrões em nosso conjunto de dados, podemos pausar o treinamento, descongelar o corpo e continuar o treinamento, mas com uma taxa de aprendizado muito pequena - não queremos alterar drasticamente nossos filtros CONV.
- O ajuste fino é um método super poderoso para obter classificadores de imagem em seus próprios conjuntos de dados personalizados de CNNs pré-treinadas (e é ainda mais poderoso do que transferir o aprendizado por meio da extração de recursos).
- referência: [Link](#)

## Ideias para Implementação do Projeto DAER



## ACME 01-Object Tracking



- O Opencv possui alguns trackers: [▶ Link](#)
- Nesse link é utilizado o *centroid tracking algorithm*
- Mas é necessário uma rede pré-treinada: é aqui que entra a parte de CNN
- Como temos poucas classes acho que a rede pode ser razoavelmente leve (eu já treinei uma smallervgg [▶ Link](#)) - tenho os fontes modificados para rodar no spyder
- Organizar banco de imagens. Podemos usar os nossos dados e ou então usar bancos como kitti, coco, imagenet, etc

## ACME 01



- Junto com a ideia dos *bounding box*, vamos testar outros algoritmos de *object tracking* como o dlib; [Link](#) ou ainda outros (pesquisar!!!)
- inicializa com uma imagem e localiza (dlib bounding box) com o *object detection* (CNN) `tracker.start_track(img, dlib box)`; faz o update com a imagem nova e com `confidence = tracker.update(img)` checa se o confidence é alto o suficiente, senão tem de executar novamente a parte da CNN.