(3) Arquiteturas de CNNs e treinamento de redes profundas

Redes Neurais e Arquiteturas Profundas

Moacir Ponti
CeMEAI/ICMC, Universidade de São Paulo
MBA em Ciência de Dados

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

São Carlos-SP/Brasil – 2020

Agenda

Estratégias de treinamento
- Função de custo e gradiente
- Otimizadores

Arquiteturas típicas de CNNs

Visualização das camadas

Agenda

Estratégias de treinamento
- Função de custo e gradiente
- Otimizadores

Arquiteturas típicas de CNNs

Visualização das camadas

Como treinar? Otimização

Machine Learning e Deep Learning depende de entender otimização e conhecer bem:

- Função de custo/perda e intuição de seus valores
- (Intuição) do gradiente da função
- Inicialização
- Algoritmo de otimização
- Taxa de aprendizado
- Tamanho do batch
- Convergência ao longo do treinamento

Função de custo/perda

Métrica que indique o custo de escolher o modelo atual

- Idealmente deve ser convexa e produzir um gradiente com boa magnitude
- Difícil, considerando todas as direções do hiper-espaço de parâmetros

Função de custo/perda

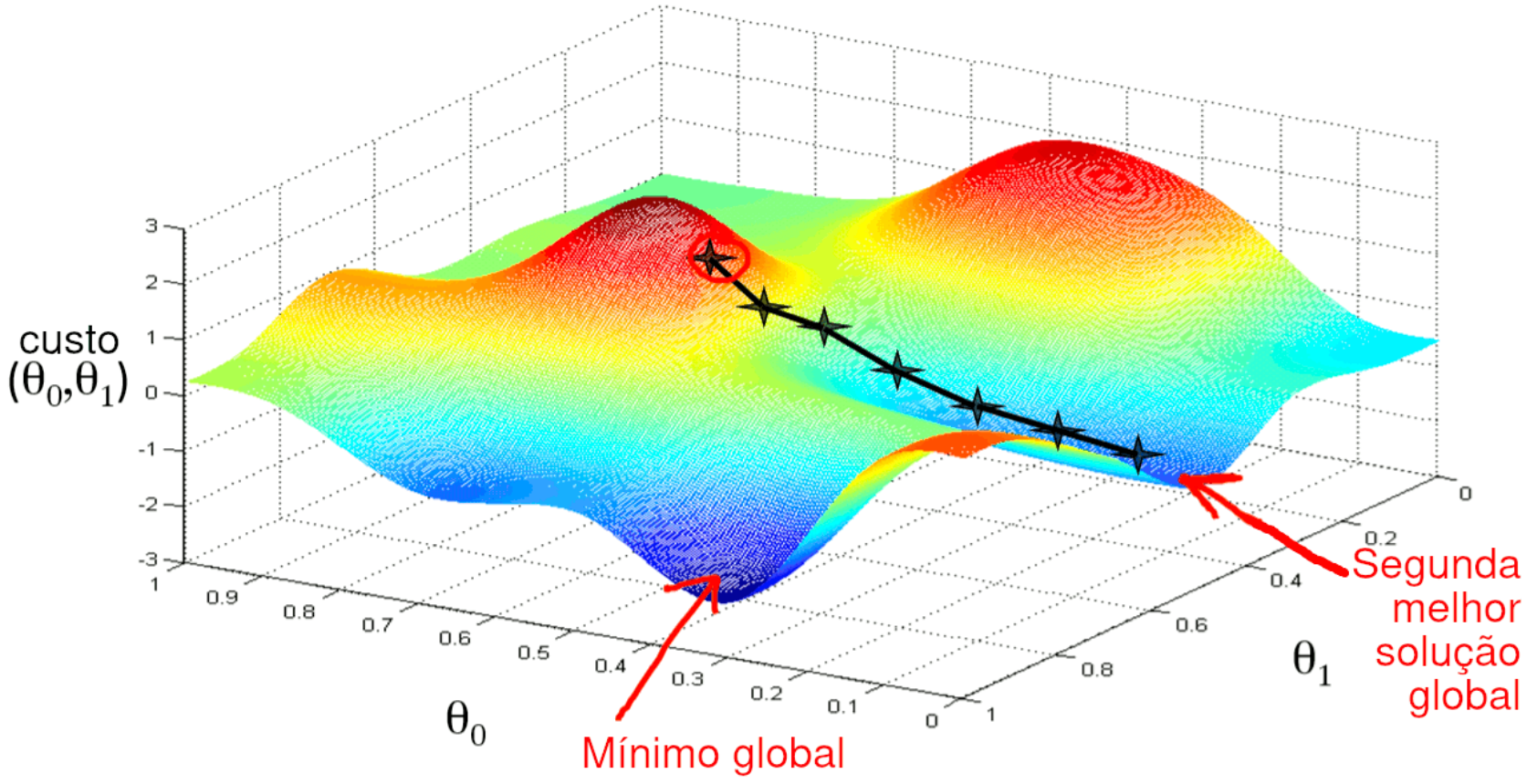
Destaques

- **Mean-squared-error**: erro médio quadrático/perda quadrática
  - utilizada para valores contínuos,
  - mede a divergência quadrática de cada valor de entrada com relação à saída
- **Cross-entropy**: entropia cruzada
  - mais comum e recomendada para probabilidades
  - teoria da informação
  - intuição: o numero de bits adicionais necessários para representar o evento de referência ao invés do predito.

O gradiente

Codifica as taxas de alteração no espaço de parâmetros

- queremos andar na direção do vale, em busca do mínimo global



O papel do gradiente no treinamento

Backpropagation

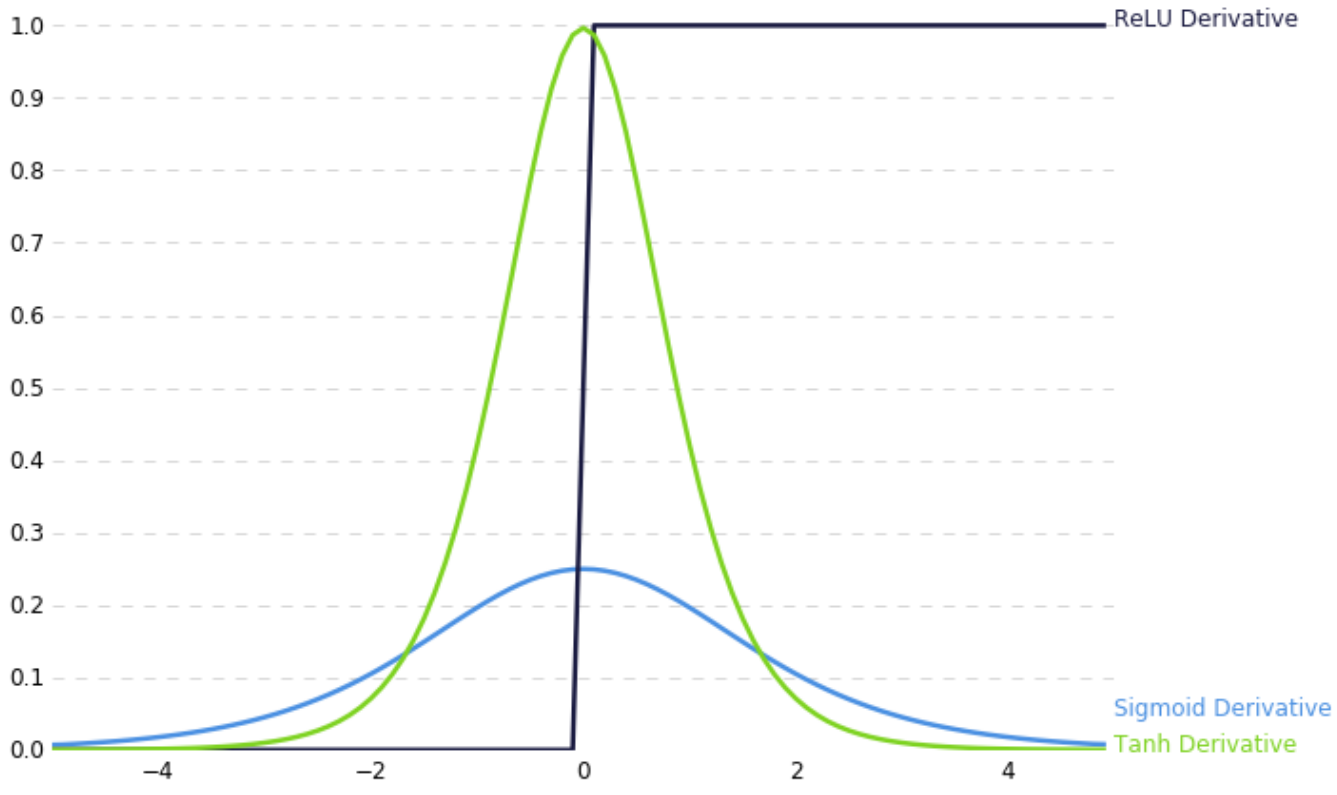
- utiliza a derivada ao longo das camadas para adaptar os pesos
- as funções de custo e de ativação tem que produzir derivada útil

Vanishing gradient

- se ativações geram valores muito baixos não é possível adaptar
- usar precisão dupla (double) e escalar as funções é uma possibilidade
- esse é um dos motivadores do uso de ReLU ao invés de Sigmóides como função de ativação

Uso de funções diferenciáveis

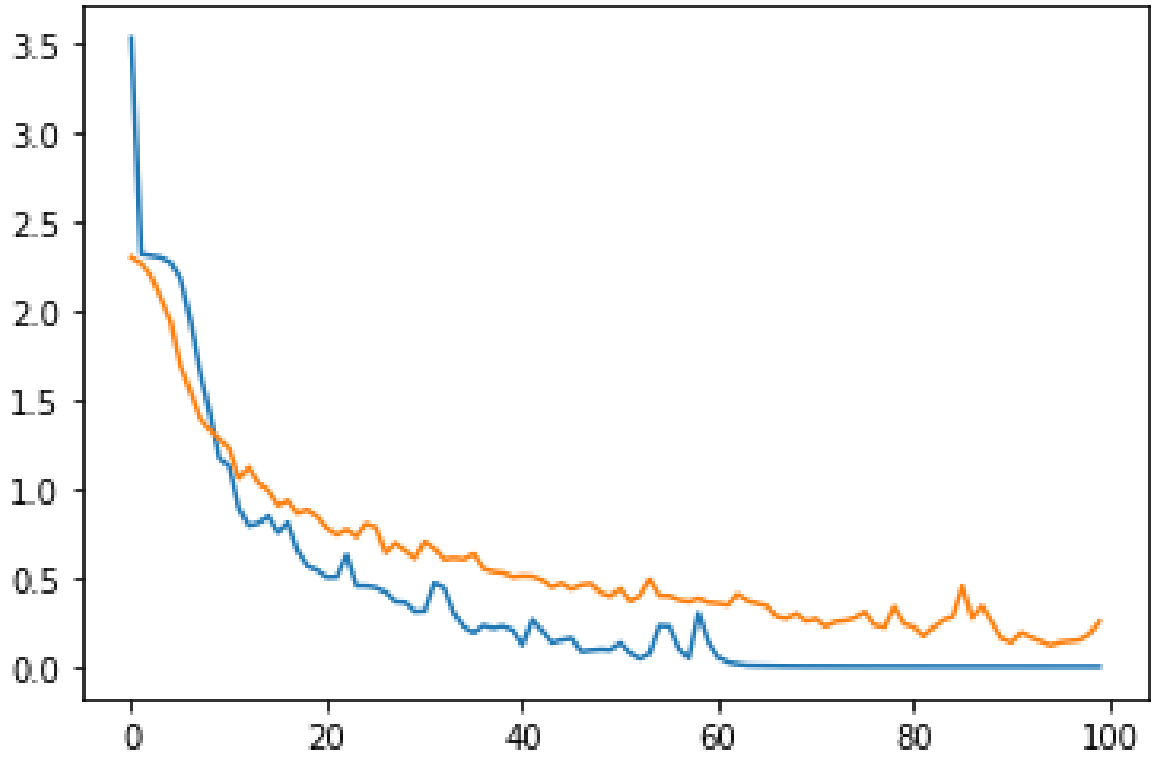
Derivada da sigmóide, ReLU e tangente hiperbólica



Agradecimentos a Harini suresh (`http://harinisuresh.com`) pelos gráficos

Valor da função de custo x gradiente

Ao longo do treinamento a rede adapta os pesos cada vez mais devagar, convergindo para uma solução



Regularização da função de custo

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

regularização

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

Dificulta esparsidade nos parâmetros e pesos com valores muito grandes. Exemplos:

- $L2 : R(\Theta) = \sum_k \theta_k^2$
- $L1 : R(\Theta) = \sum_k |\theta_k|$

Inicialização

Aleatória portanto o resultado é diferente a cada execução.

Escolhas comuns

- Pesos: valor aleatório pela distribuição normal entre 0-1
- Bias: 0 (zero)

A complexidade do treinamento dificulta múltiplas execuções

- Importante fazer experimentos piloto em pequenos subconjuntos de dados

Check-list 1

- O valor da função de custo nos pesos aleatórios faz sentido?
- Ex. num problema de classificação com 10 classes com entropia cruzada calculada na saída softmax:
  - $-\ln(0.1) = 2.3026$

Agenda

Estratégias de treinamento
- Função de custo e gradiente
- Otimizadores

Arquiteturas típicas de CNNs

Visualização das camadas

Otimizadores

Stochastic Gradient Descent (SGD)

Formulação original (atualização por instância)

$$\begin{aligned}\theta_{k+1} &= \theta_k - \alpha_k \nabla_\theta \ell(y, f(x; \theta_k)), \\ &= \theta_k - \alpha_k g(x, \theta_k),\end{aligned}$$

α_k é a taxa de aprendizado (learning rate) na iteração k

Batch Stochastic Gradient Descent (SGD)

computando o gradiente da função de custo de um lote de instâncias X_k na iteração k

$$\theta_{k+1} = \theta_k - \alpha_k g(X_k, \theta_k),$$

Tamanho de batch e Taxa de aprendizado

Há uma relação entre tamanho de batch e taxa de aprendizado.

Batch

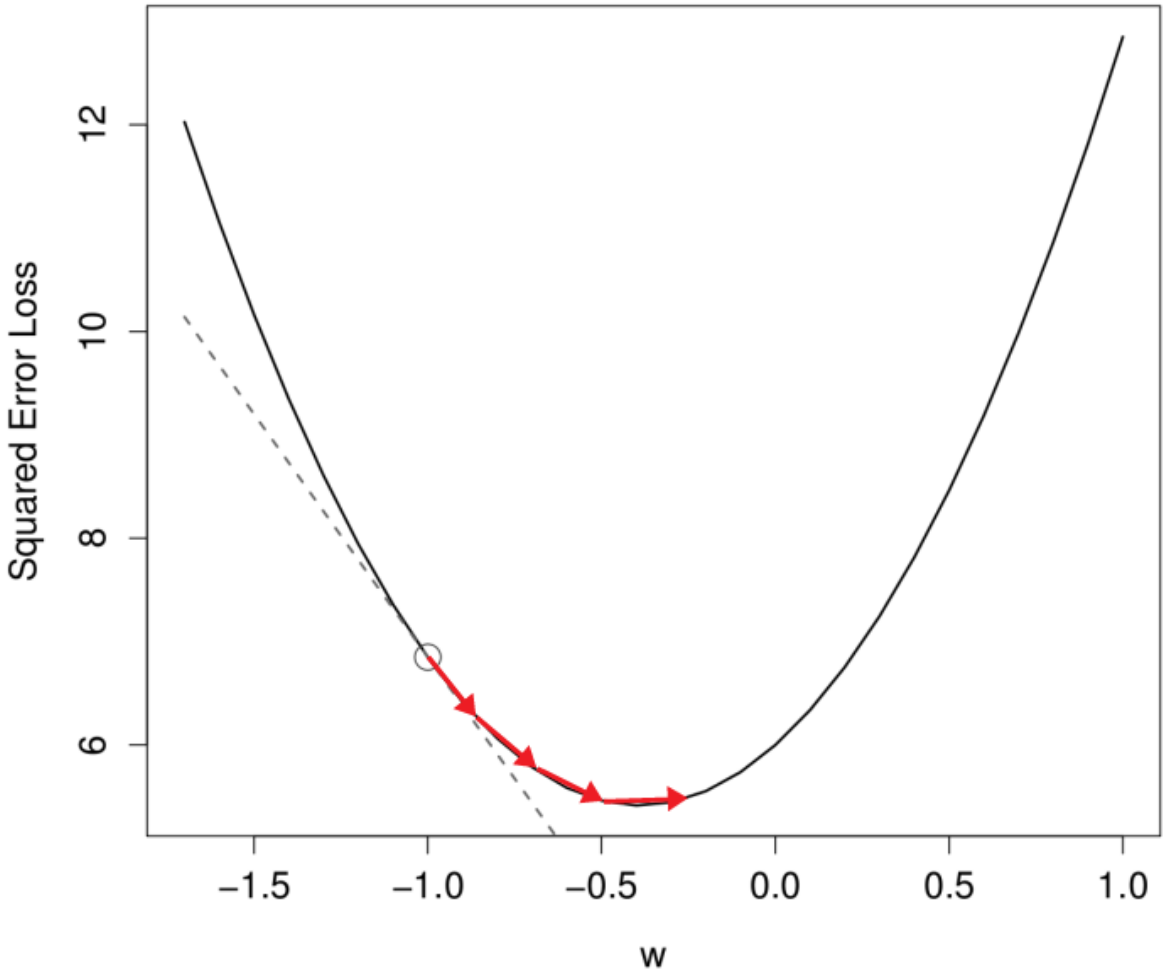
- Padrão é 32
  - batches maiores: estimativas mais suaves, difícil manter na memória, exige ajustar bem a taxa de aprendizado,
  - batches menores: estimativas mais ruidosas, mas que mostraram vantagens em encontrar melhores mínimos.

Tamanho de batch e Taxa de aprendizado

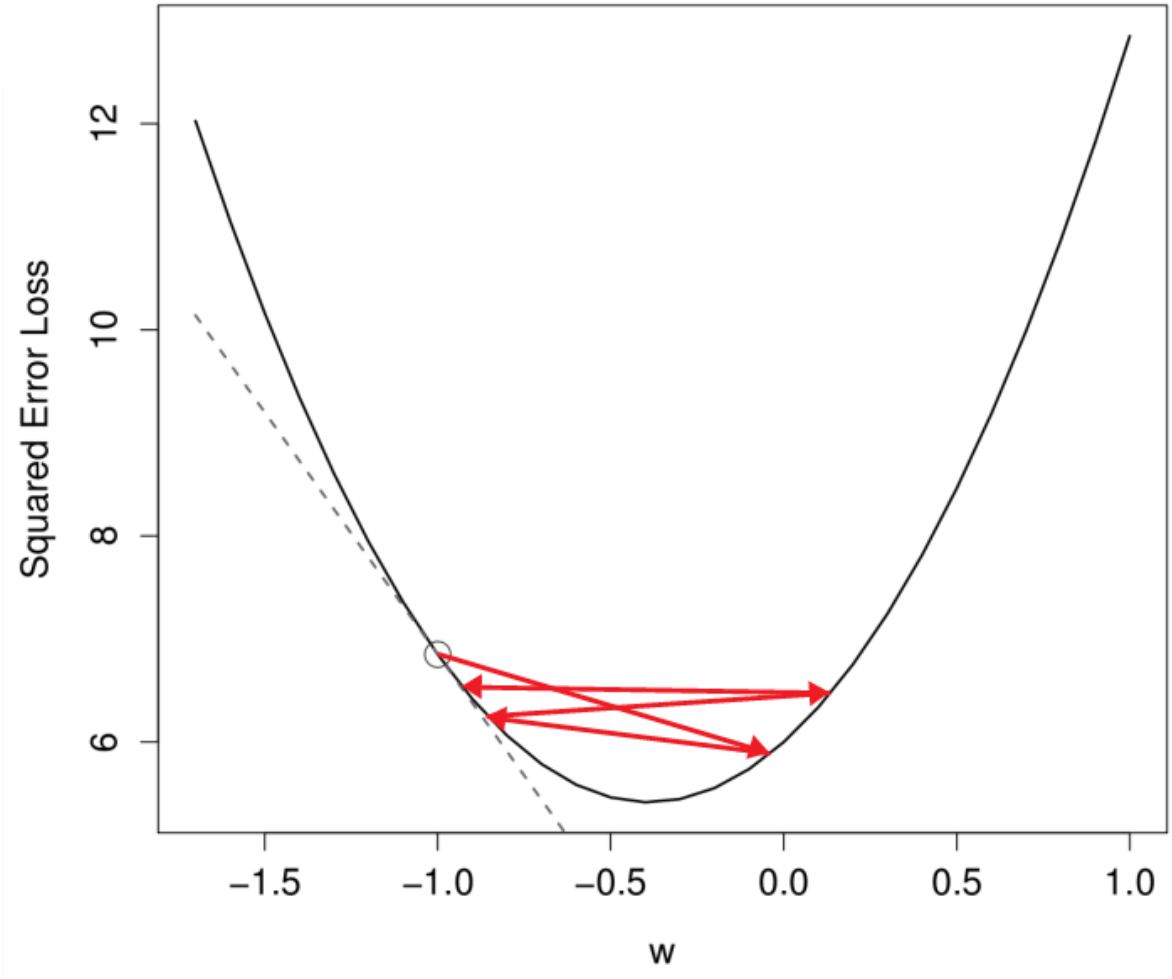
Taxa de aprendizado

- Padrão é 0.01
  - pode ser pouco adequado para alguns otimizadores
  - pode ser pouco adequado para batchs maiores (ou muito pequenos)
- É recomendado iniciar com um valor maior, e reduzir a taxa progressivamente (learning rate scheduling).

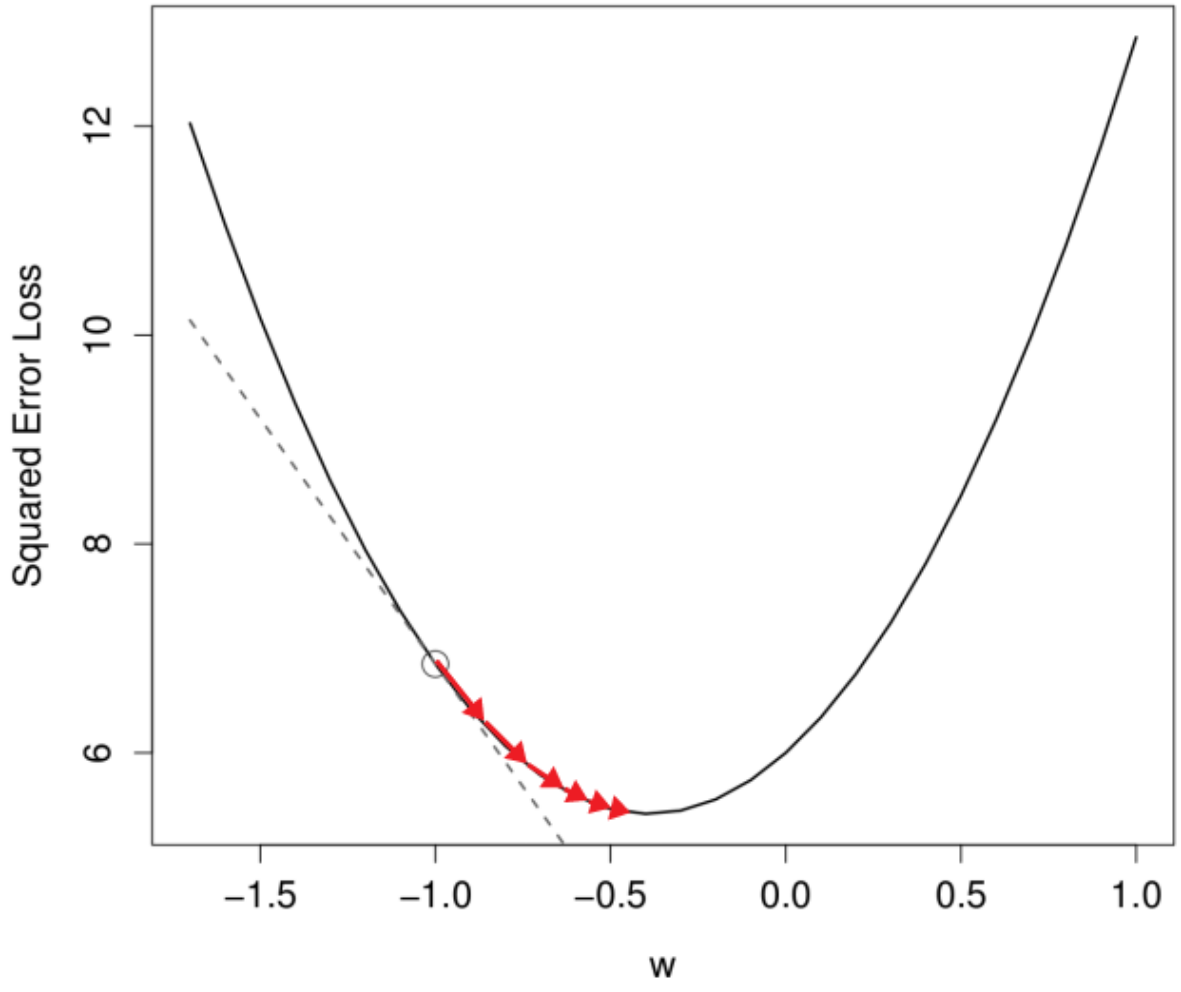
Taxa de aprendizado: intuição com valor pequeno



Taxa de aprendizado: intuição com valor excessivamente grande



Taxa de aprendizado: intuição com decaimento



Taxa de aprendizado

Check-list 2

- Utilize decaimento de taxa de aprendizado
- ... de forma fixa ou de acordo com métricas computadas no treinamento ou validação

Otimizadores

Momentum

Interpreta o custo como um terreno montanhoso.

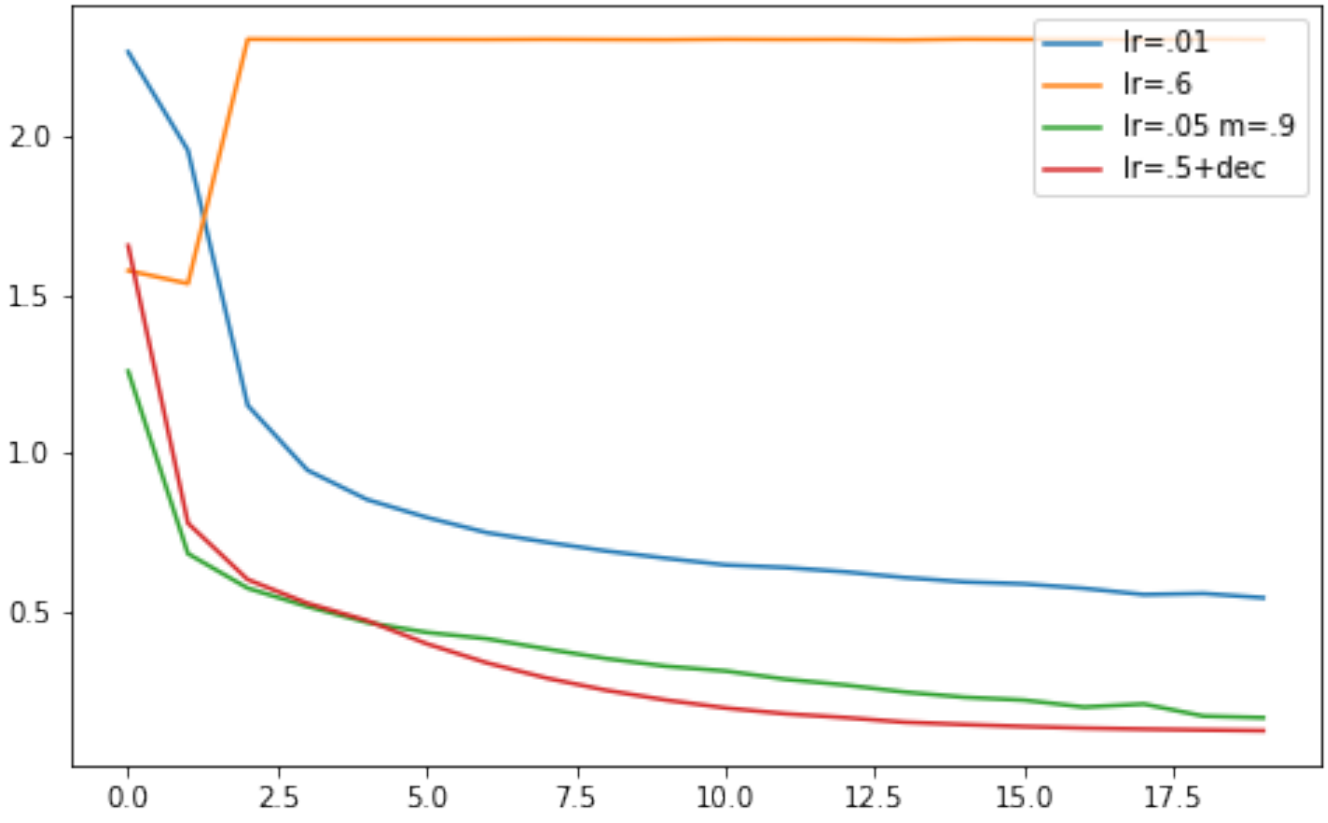
- Inicializar: posicionar partícula com velocidade zero no terreno
- Otimização: rolar partícula, considerando a aceleração.
- Consequência: a velocidade é ajustada considerando a magnitude de atualizações anteriores

$$\theta_{k+1} = \theta_k + m \cdot v - \alpha_k g(x, \theta_k),$$

v é o momentum, inicialmente 0; m peso: menor funciona como atrito que reduz a energia cinética do sistema (hiperparâmetro)

- Investigar $m \in [0.5, 0.9, 0.95, 0.99]$
- ou iniciar com valor menor e aumentar ao longo das épocas

Taxa de aprendizado: diferentes abordagens, caso real



Outros otimizadores

Adam

Utiliza momentos do gradiente: o segundo momento é usado para normalizar o primeiro, evitando outliers/pontos de inflexão

$$\theta_{k+1} = \theta_k - \alpha_k \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}$$

$\hat{m}$ e $\hat{v}$ são estimativas corrigidas do primeiro e segundo momentos do gradiente.

- $\hat{m}_k$ é a soma do gradiente atual com o acúmulo de gradientes anteriores $\hat{m}_{k-1}$ (similar a momentum).
- $\hat{v}_k$ é a soma do quadrado do gradiente em k com o acúmulo de valores anteriores $\hat{v}_{k-1}$ (taxa de aprendizado adaptativa).
- Funciona melhor com taxa de aprendizado **menor**, quando comparado ao SGD

Otimizadores

Check-list 3

Utilizar:

- SGD (+ Momentum), ou
- Adam

Convergência ao longo do treinamento

O gráfico do custo diz **muito** sobre o aprendizado

Check-list 4

- Acompanhe o custo ao longo de épocas, se possível com conjunto de validação (idealmente não deve ser o teste!)
- Inicie com experimentos com poucos exemplos
  - explore os hiperparâmetros tentando obter "overfitting" para um subconjunto de exemplos, obtendo custo próximo a zero, e depois refine a busca num conjunto maior.

Agenda

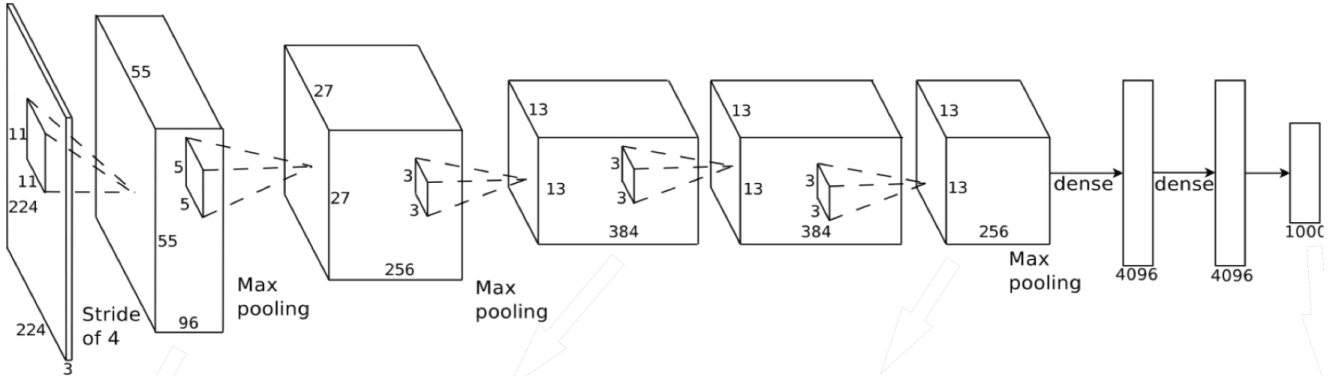
Estratégias de treinamento
- Função de custo e gradiente
- Otimizadores

Arquiteturas típicas de CNNs

Visualização das camadas

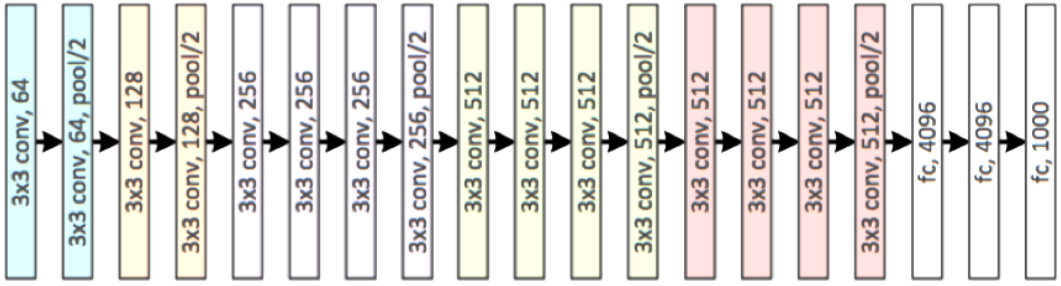
AlexNet (Krizhevsky, 2012)

- entrada 224×224
- conv1: $K = 96$ filters with $11 \times 11 \times 3$, stride 4,
- conv2: $K = 256$ filters with $5 \times 5 \times 96$,
- conv3: $K = 384$ filters with $3 \times 3 \times 256$,
- conv4: $K = 384$ filters with $3 \times 3 \times 384$,
- conv5: $K = 256$ filters with $3 \times 3 \times 384$,
- densas1, 2: $K = 4096$.



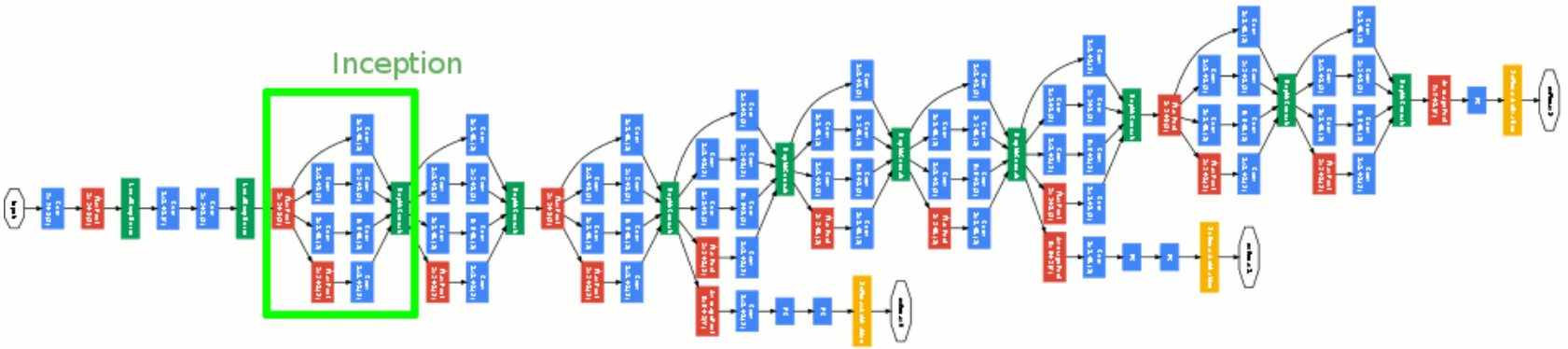
VGGNet (Simonyan, 2014)

- entrada 224×224,
- filtros: todos 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
- densas1, 2: $K = 4096$

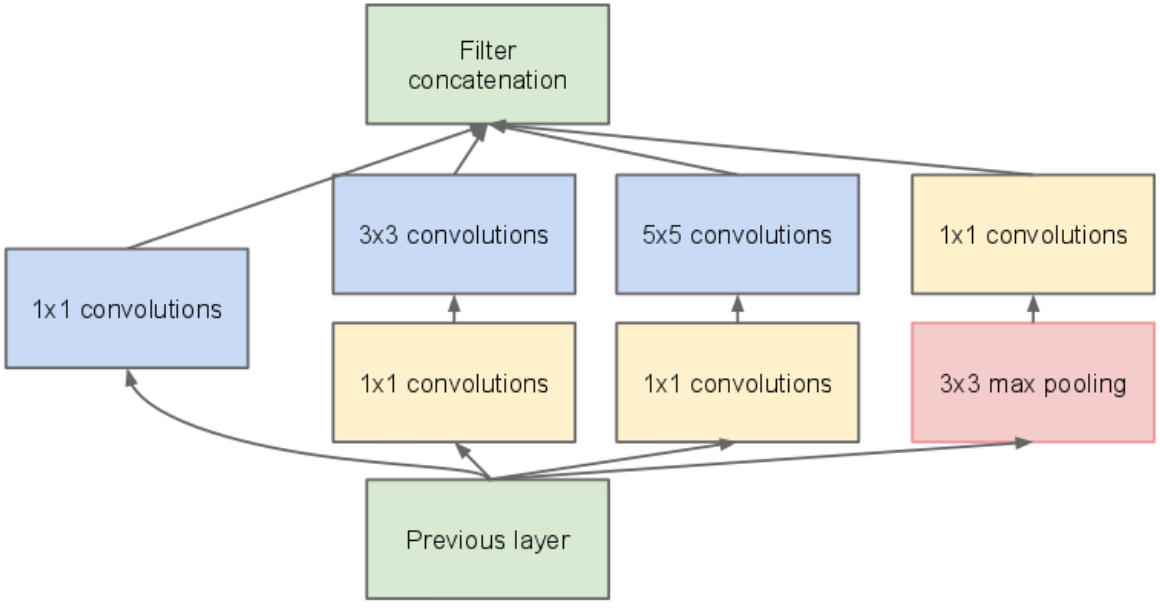


GoogLeNet / Inception (Szegedy, 2014)

- 22 layers (v1)
- Começa com duas camadas convolucionais
- *Inception layer* (banco de filtros):
  - filtros 1×1, 3×3, 5×5 + max pooling 3×3;
  - controla dimensionalidade usando filtros 1×1.
  - 3 classificadores (não sequenciais)
- Azul = conv.,
- Vermelho = pool.,
- Amarelo = densa+softmax,
- Verde = concatenação.

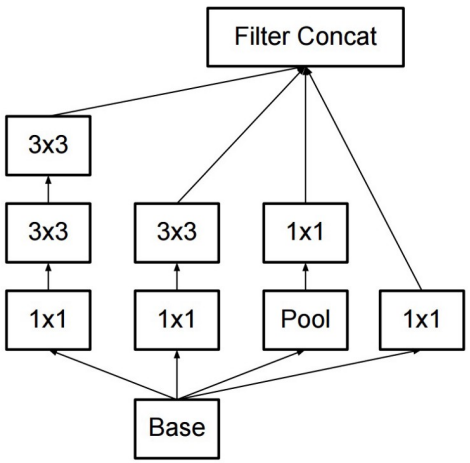


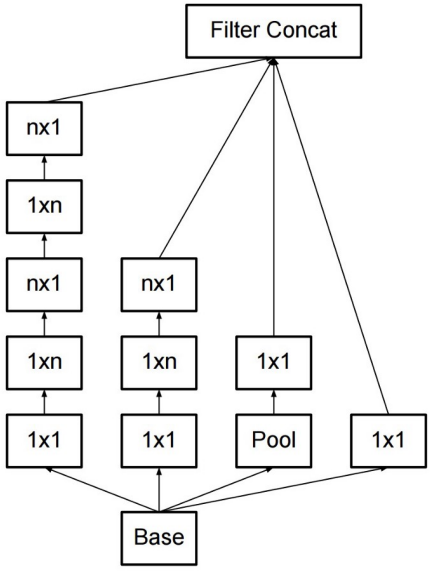
GoogLeNet: módulo inception v1

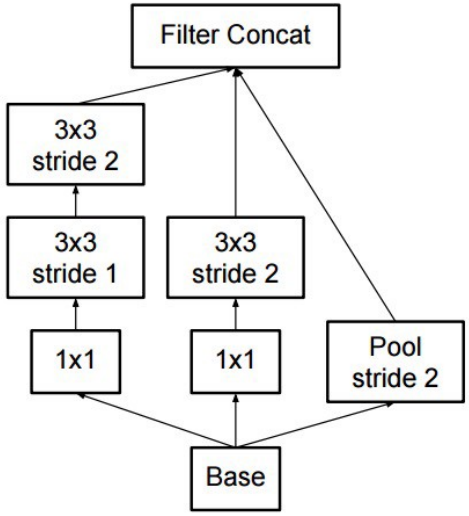


- filtro 1×1 reduz a profundidade da entrada
- concatena mapas de ativação de 3 filtros + maxpooling

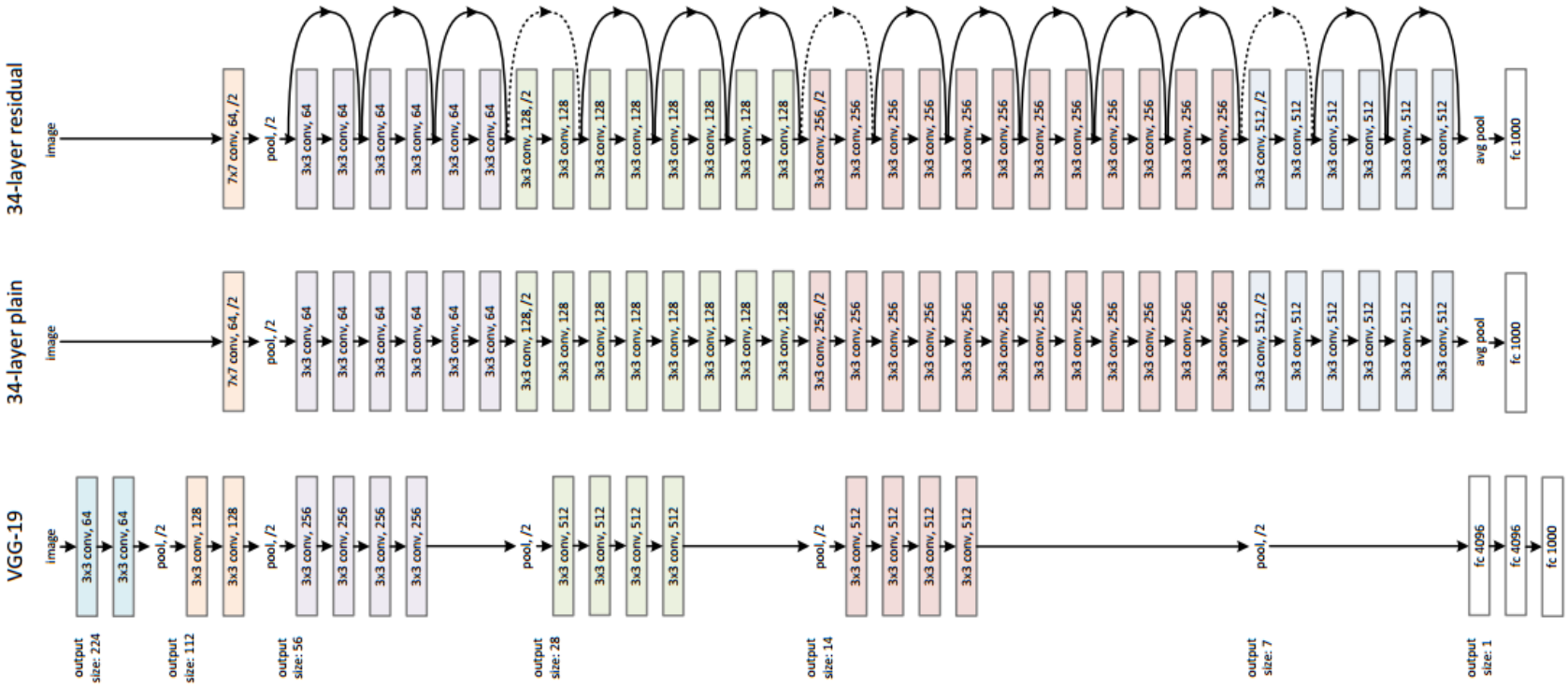
Módulos inception (V2 and V3)





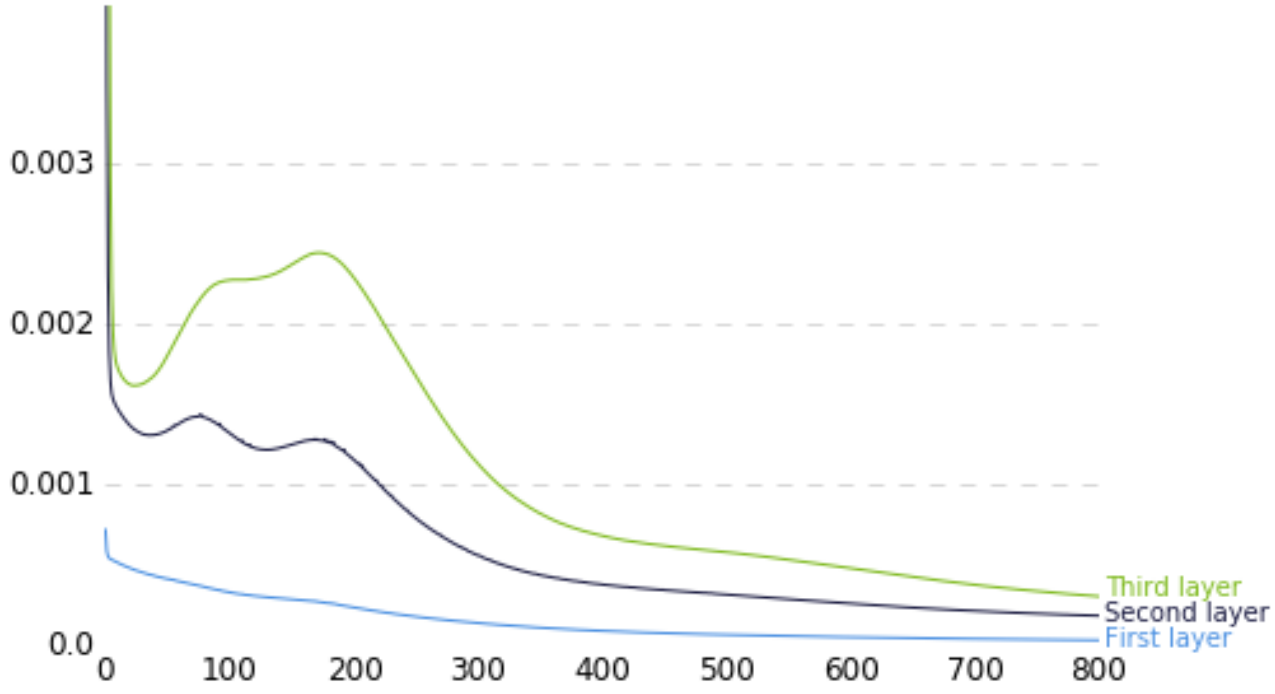


Residual Network (He et al, 2015)



O gradiente não se comporta igual em todas as camadas

Gradiente medido em cada camada

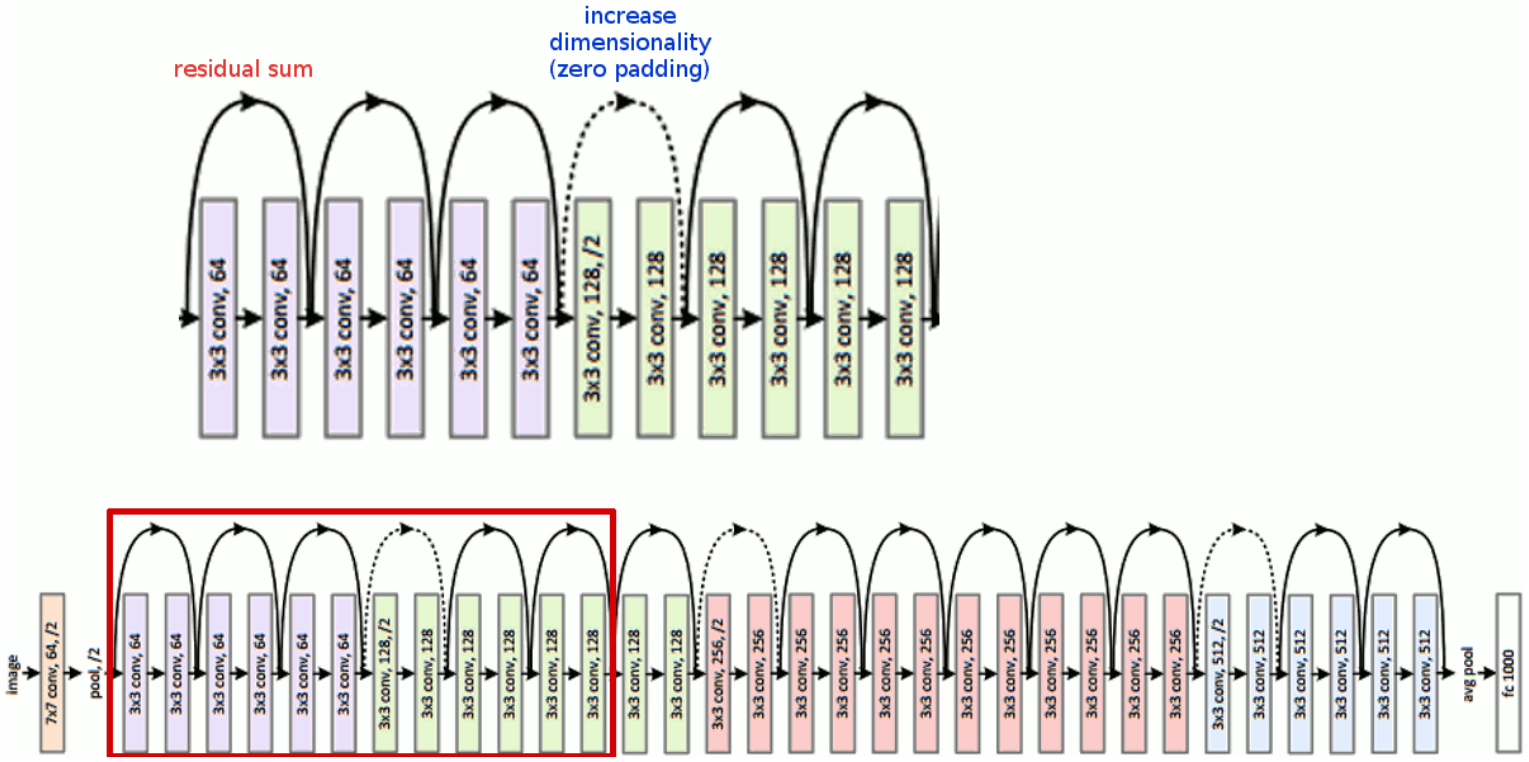


Agradecimentos a Harini suresh (`http://harinisuresh.com`) pelos gráficos

Residual Network — ResNet (He et al, 2015)

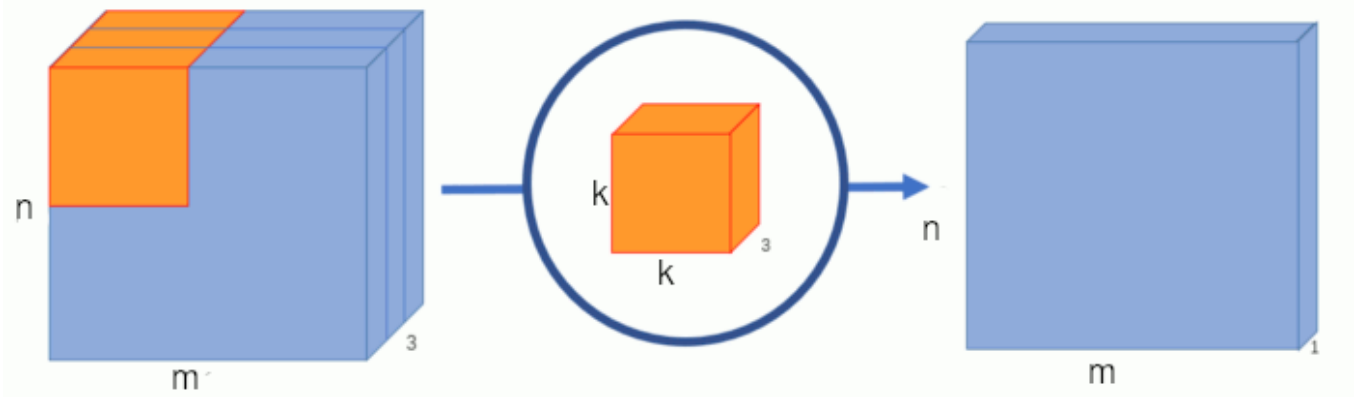
Pular camadas (skip layers) permite empilhar mais camadas (de 34 a $\sim$ 1000).

Arquitetura residual: adiciona resultado anterior preservando gradiente.



Depthwise separable convolutions - Xception, Mobilenet

Convolução convencional

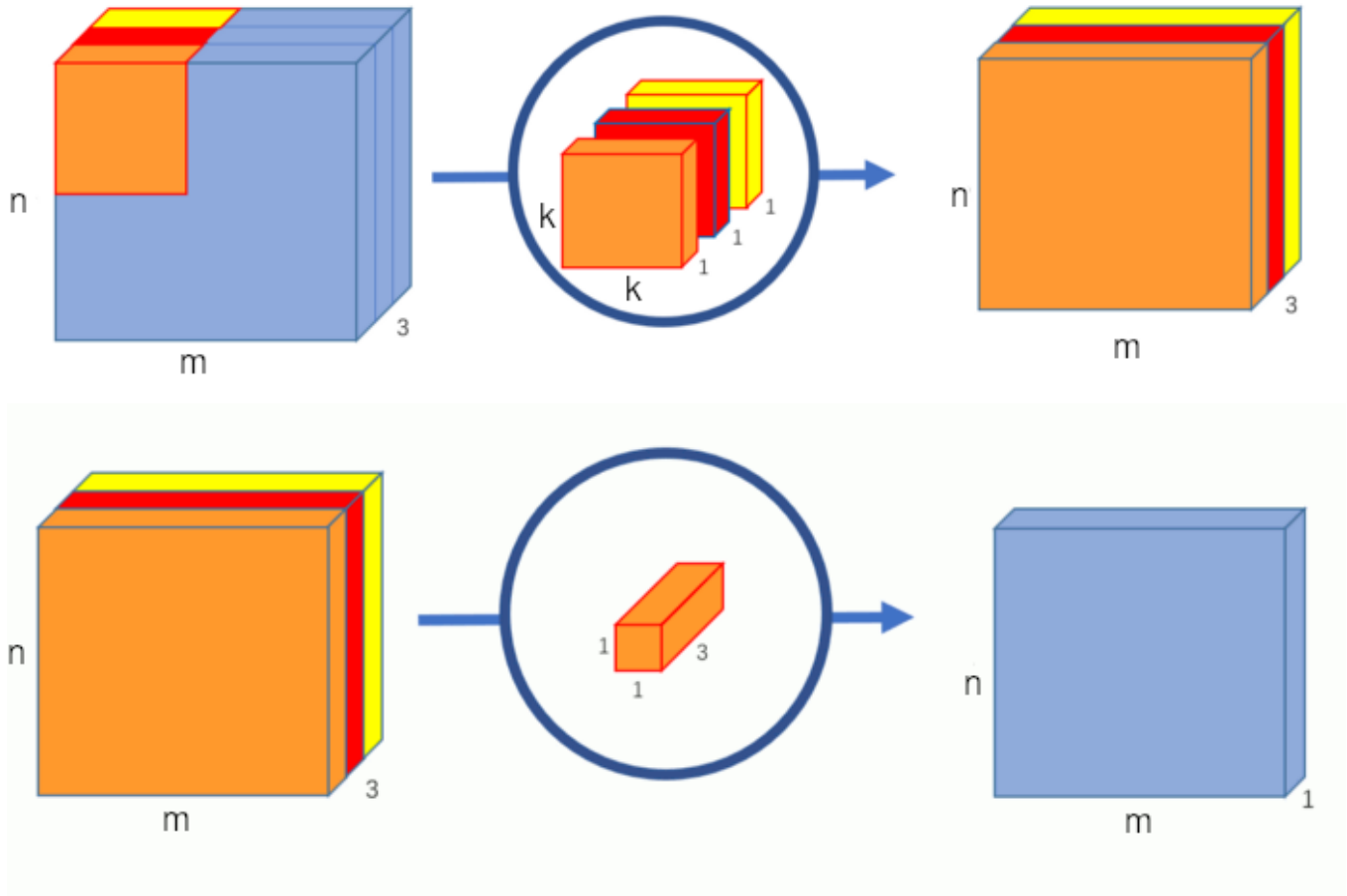


Para obter 128 mapas de características com filtros 3×3 temos:

- $3 \times 3 \times 3 \times 128 = 1638$ parâmetros.
- em imagem com $100 \times 100 \times 3$ pixels, movemos cada filtro 10000 vezes para multiplicar os valores locais, total em multiplicações: $10000 \times 1638 = 16.380.000$

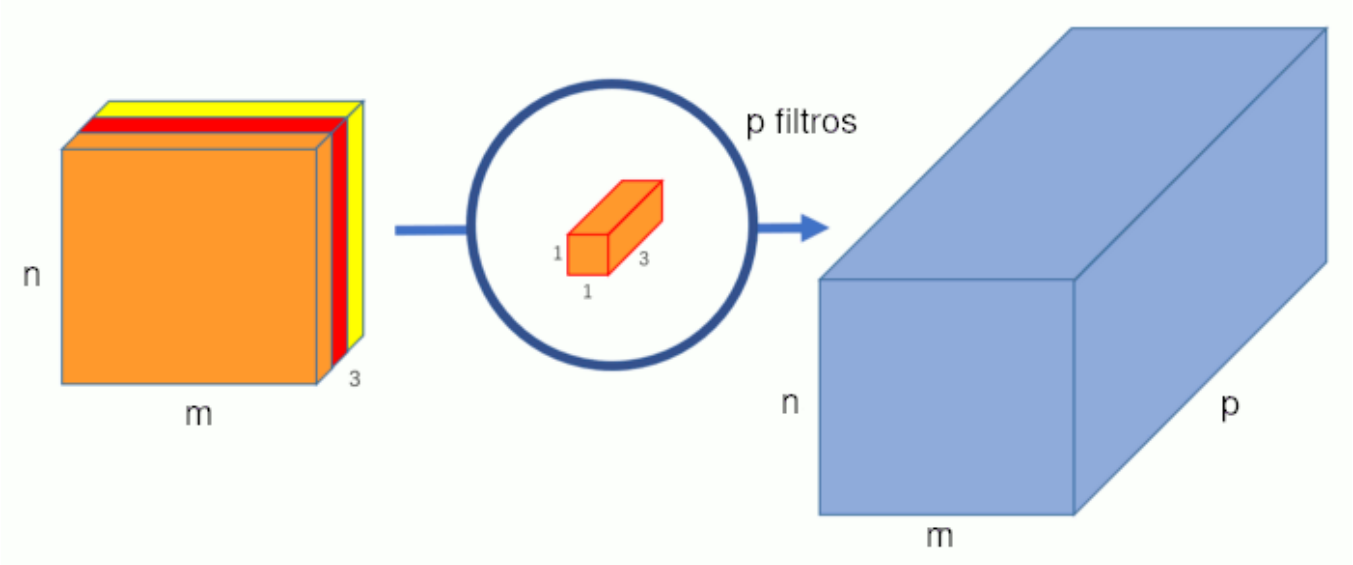
Depthwise separable convolutions - Xception, Mobilenet

Separável: primeiro lateral, depois em profundidade

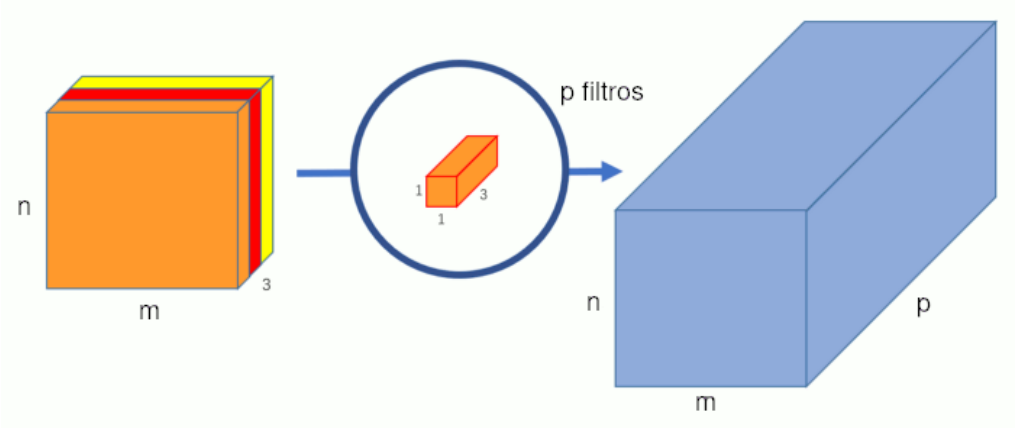


Depthwise separable convolutions - Xception, Mobilenet

Lateral executada uma única vez, produzindo um único volume, depois p filtros $1 \times 1 \times 3$ produzirão p mapas de ativação



Depthwise separable convolutions - Xception, Mobilenet



Para obter 128 mapas de características com filtros 3×3 temos:

- lateral: $3 \times 3 \times (3) = 9$ parâmetros.
- profundidade: $1 \times 1 \times 3 \times (128) = 384$ parâmetros
- na imagem $100 \times 100 \times 3$, são 10000 posições,
  - fase 1: $1000 \times 9 = 9.000$ multiplicações
  - fase 2: $1000 \times 384 = 384.000$ multiplicações
  - total = 393.000 contra **16 milhões** da convencional

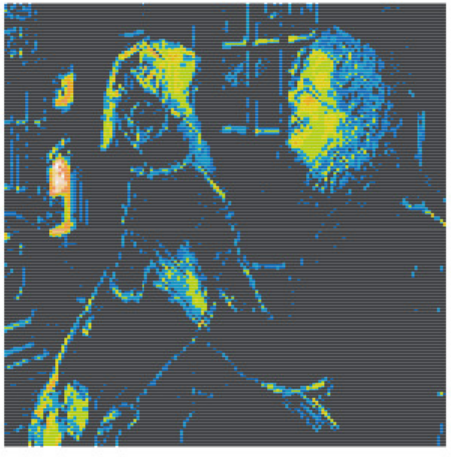
Agradecimentos a Chi-Feng Wang (@reina.wang) pelas belas ilustrações

Agenda

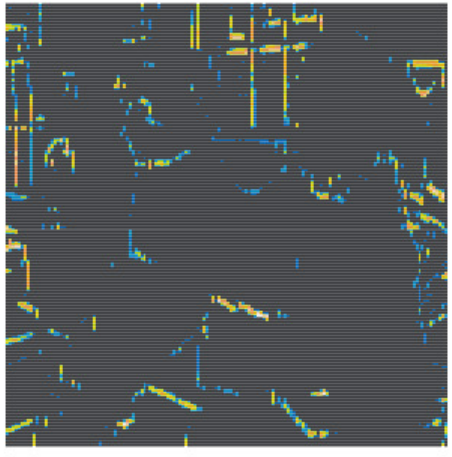
Visualização

Imagem de entrada

Mapas de ativação - 1.a camada conv.



#20

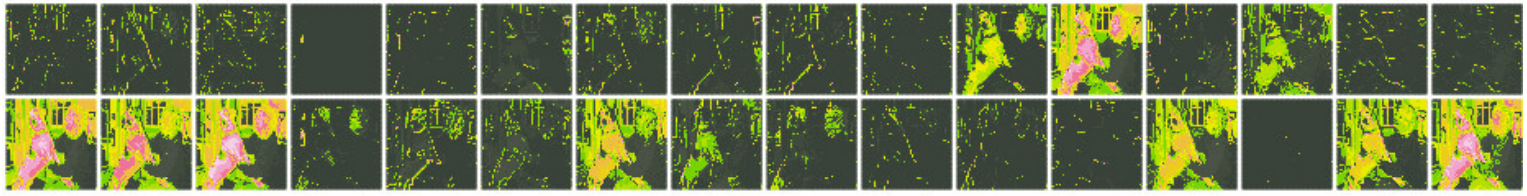


#28

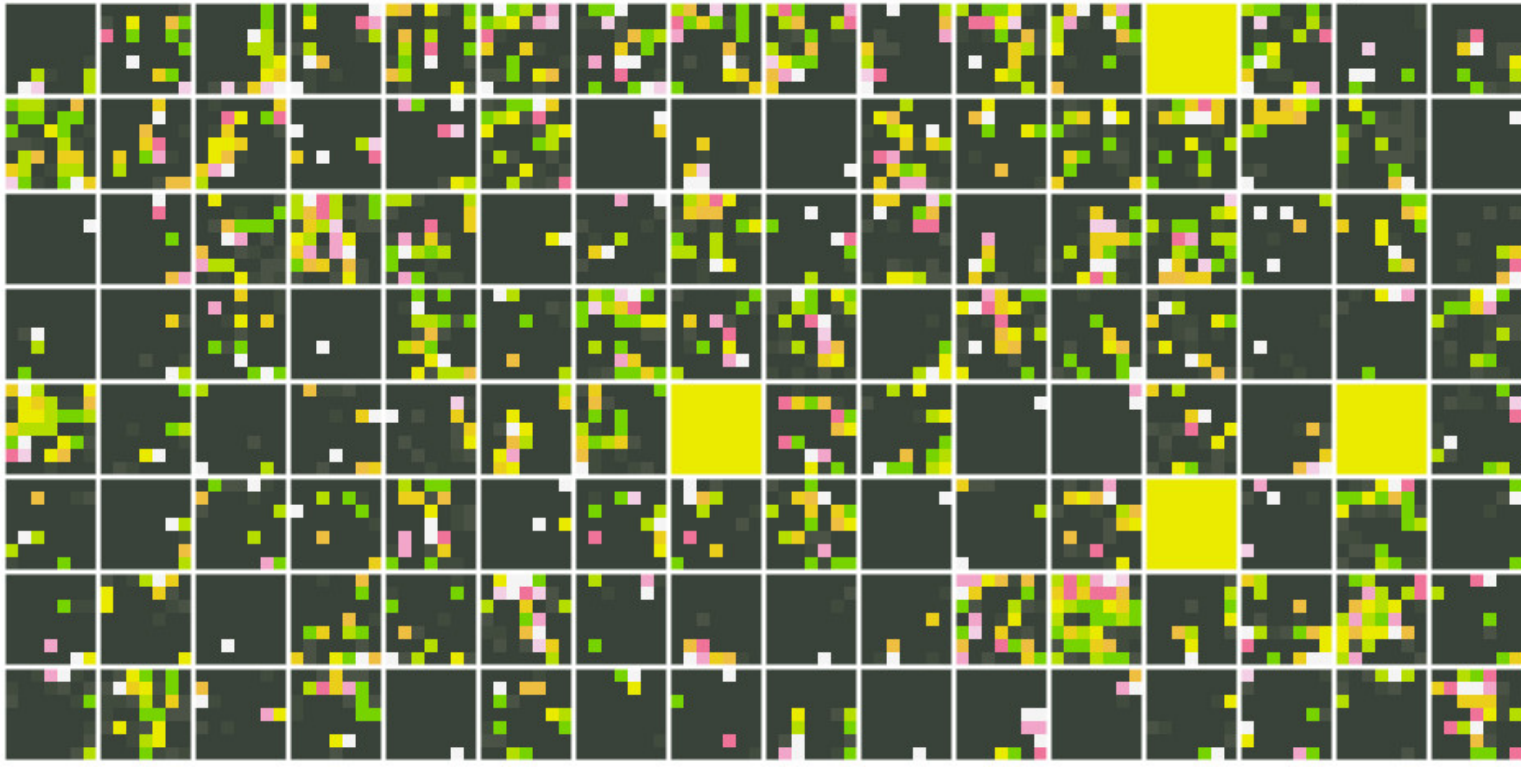
https://rstudio-conf-2020.github.io/dl-keras-tf/notebooks/visualizing-what-cnns-learn.nb.html

Visualização

32 filtros da 1.a camada convolucional



128 filtros da última camada (antes da densa)



Bibliography I

Moacir A. Ponti, Gabriel Paranhos da Costa. **Como funciona o Deep Learning**
SBC, 2017. Book chapter.
https://arxiv.org/abs/1806.07908

Moacir A. Ponti, Leo Ribeiro, Tiago Nazaré, Tu Bui, John Collomosse. **Everything You Wanted to Know About Deep Learning for Computer Vision but were Afraid to Ask**.
SIBGRAPI-T, 2017. Tutorial.

Moacir A. Ponti, **Introduction to Deep Learning** (Code).
Github Repository:
https://github.com/maponti/deeplearning_intro_datascience
CNN notebook: https://colab.research.google.com/drive/1EnNjtzdw8ftIO7I9xCUhb-ovq1iNy4pf