“Fica, vai ter bolo”.

Inicialização de Pesos e Transfer Learning

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

O Elefante na Sala

Considere a seguinte CNN:

```
class custom_3_layer(nn.Module):
   def __init__(self):
       super(custom_3_layer, self).__init__()

       self.conv1 = nn.Conv2d(in_channels=INPUT_SHAPE[0], out_channels=32, kernel_size=3, stride=1, padding=1)
       self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

       self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=0)
       self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

       self.conv3 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=0)
       …
```

O Elefante na Sala

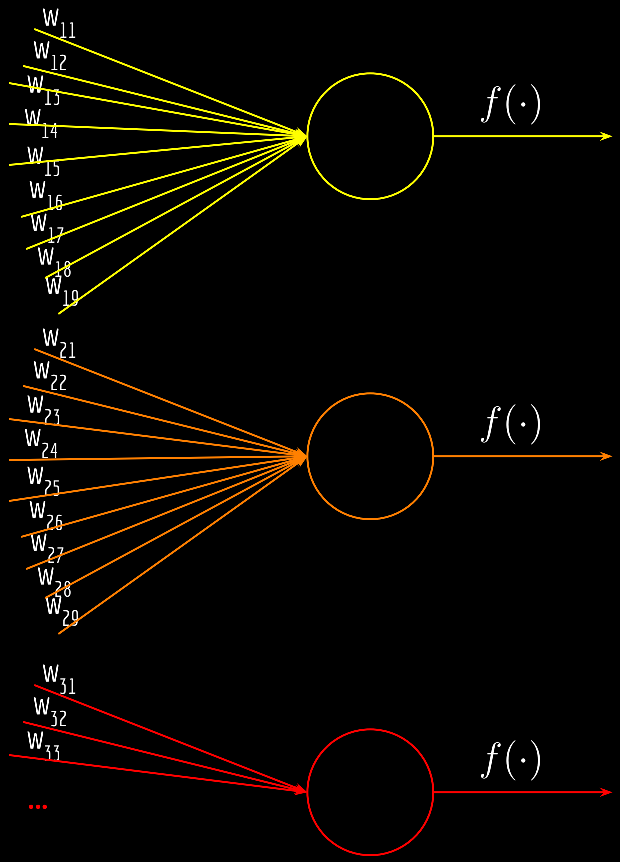
Considere a seguinte CNN:

```
class custom_3_layer(nn.Module):
   def __init__(self):
       super(custom_3_layer, self).__init__()

       self.conv1 = nn.Conv2d(in_channels=INPUT_SHAPE[0], out_channels=32, kernel_size=3, stride=1, padding=1)
       self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

       self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=0)
       self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

       self.conv3 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=0)
       …
```

Quantos filtros de convolução existem na primeira camada, e qual o tamanho dos filtros?

O Elefante na Sala

São 32 filtros de 3 x 3, com 9 parâmetros cada (mais o bias).

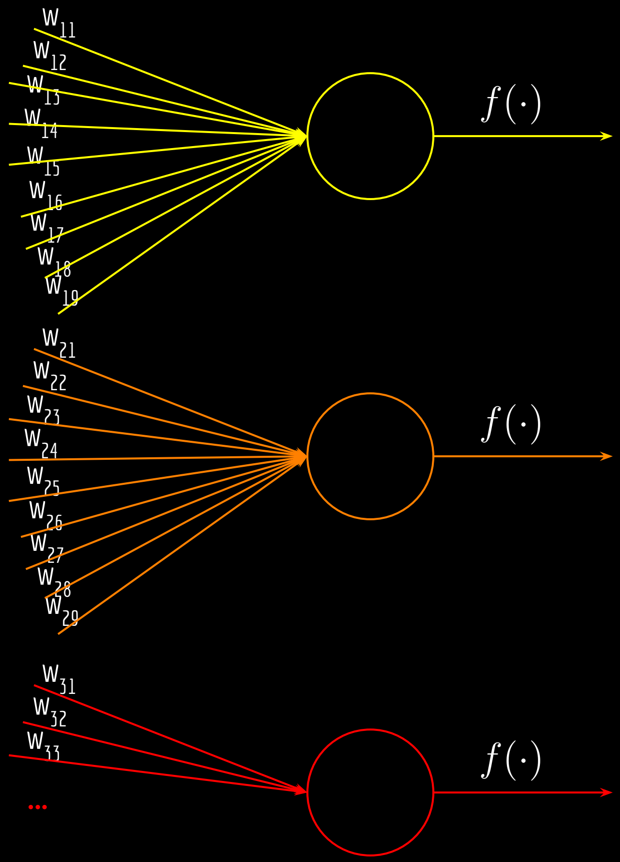


W_{11}	W_{12}	W_{13}
W_{14}	W_{15}	W_{16}
W_{17}	W_{18}	W_{19}

W_{21}	W_{22}	W_{23}
W_{24}	W_{25}	W_{26}
W_{27}	W_{28}	W_{29}

W_{31}	W_{32}	W_{33}
W_{34}	W_{35}	W_{36}
W_{37}	W_{38}	W_{39}

O Elefante na Sala



w_{11}	w_{12}	w_{13}
w_{14}	w_{15}	w_{16}
w_{17}	w_{18}	w_{19}

= ???

w_{21}	w_{22}	w_{23}
w_{24}	w_{25}	w_{26}
w_{27}	w_{28}	w_{29}

= ???

w_{31}	w_{32}	w_{33}
w_{34}	w_{35}	w_{36}
w_{37}	w_{38}	w_{39}

Durante o treinamento, era para todos os filtros convergirem para os mesmos pesos, não? Como filtros diferentes aprendem coisas diferentes?

Faça você mesmo

Execute o exemplo disponibilizado no Google Colab.

É a mesma rede de aulas passadas, mas agora todos os filtros de convolução 3x3 são inicializados com os mesmos valores.

Todos biases inicializados com 1.

Faça você mesmo

É muito provável que todos os kernels de convolução tenham convergido para os mesmos valores.

Ter múltiplos kernels se tornou inútil.

```
... [[[-0.0698,  0.0933, -0.3015],
          [-0.1609, -0.0964, -0.3220],
          [-0.2577, -0.3699, -0.2675]]],


        [[[-0.0698,  0.0933, -0.3015],
          [-0.1609, -0.0964, -0.3220],
          [-0.2577, -0.3699, -0.2675]]],


        [[[-0.0698,  0.0933, -0.3015],
          [-0.1609, -0.0964, -0.3220],
          [-0.2577, -0.3699, -0.2675]]] ...
```

Quebra de simetria

Devemos considerar a quebra de simetria durante a inicialização dos pesos.

Se todos os pesos possuem os mesmos valores inicialmente, a rede é simétrica, eles contribuirão igualmente para o erro, e serão atualizados com os mesmos valores.

Quebra de simetria

É necessário inicializar os pesos aleatoriamente.

Mas qualquer aleatório também não serve.

Dependendo da complexidade da rede, ou da disponibilidade de recursos, podemos ter múltiplas inicializações, e escolher a que converge melhor.

Inicialização

Geralmente os pesos são inicializados através de:

Distribuição Normal no intervalo $[-\epsilon, \epsilon]$.

Gaussiana de média zero $\mathcal{N}(0, \epsilon^2)$.

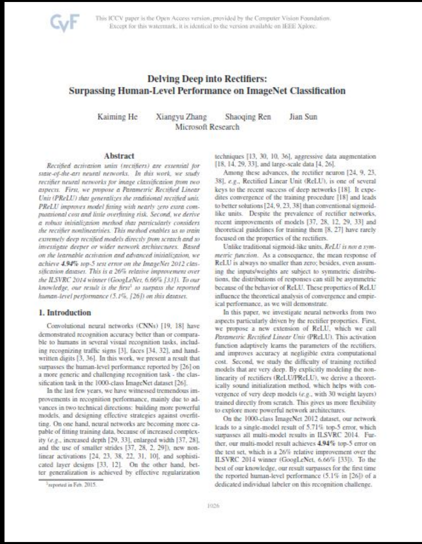
A escolha de ϵ varia de acordo com o método.

Exemplo - Inicialização de He

Em uma rede que utiliza a função de ativação ReLU e similares desejamos que a variância dos dados que entram nos neurônios:

- Não caiam para zero.
- Não subam muito.

He, K., Zhang, X., Ren, S., & Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE international conference on computer vision* , 2015.



**Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification**

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research

Abstract

Rectified activation units (rectifiers) are essential for state-of-the-art neural networks. In this work, we study rectifier neural networks for image classification from two aspects. First, we propose a Parametric Rectified Linear Unit (PReLU) that generalizes the traditional rectified unit. PReLU improves model fitting with nearly zero extra computational cost and little overfitting risk. Second, we derive a robust initialization method that particularly considers the rectifier nonlinearities. This method enables us to train extremely deep rectified models directly from scratch and to investigate deeper or wider network architectures. Based on the learnable activation and advanced initialization, we achieve **4.94%** *top-5 test error on the ImageNet 2012 classification dataset. This is a 26% relative improvement over the ILSVRC 2014 winner (GoogLeNet, 6.66% [33]). To our knowledge, our result is the first*[1] *to surpass the reported human-level performance (5.1%, [26]) on this dataset.*

1. Introduction

Convolutional neural networks (CNNs) [19, 18] have demonstrated recognition accuracy better than or comparable to humans in several visual recognition tasks, including recognizing traffic signs [3], faces [34, 32], and handwritten digits [3, 36]. In this work, we present a result that surpasses the human-level performance reported by [26] on a more generic and challenging recognition task - the classification task in the 1000-class ImageNet dataset [26].

In the last few years, we have witnessed tremendous improvements in recognition performance, mainly due to advances in two technical directions: building more powerful models, and designing effective strategies against overfitting. On one hand, neural networks are becoming more capable of fitting training data, because of increased complexity (*e.g.*, increased depth [29, 33], enlarged width [37, 28], and the use of smaller strides [37, 28, 2, 29]), new nonlinear activations [24, 23, 38, 22, 31, 10], and sophisticated layer designs [33, 12]. On the other hand, better generalization is achieved by effective regularization techniques [13, 30, 10, 36], aggressive data augmentation [18, 14, 29, 33], and large-scale data [4, 26].

Among these advances, the rectifier neuron [24, 9, 23, 38], *e.g.*, Rectified Linear Unit (ReLU), is one of several keys to the recent success of deep networks [18]. It expedites convergence of the training procedure [18] and leads to better solutions [24, 9, 23, 38] than conventional sigmoid-like units. Despite the prevalence of rectifier networks, recent improvements of models [37, 28, 12, 29, 33] and theoretical guidelines for training them [8, 27] have rarely focused on the properties of the rectifiers.

Unlike traditional sigmoid-like units, *ReLU is not a symmetric function*. As a consequence, the mean response of ReLU is always no smaller than zero; besides, even assuming the inputs/weights are subject to symmetric distributions, the distributions of responses can still be asymmetric because of the behavior of ReLU. These properties of ReLU influence the theoretical analysis of convergence and empirical performance, as we will demonstrate.

In this paper, we investigate neural networks from two aspects particularly driven by the rectifier properties. First, we propose a new extension of ReLU, which we call *Parametric Rectified Linear Unit* (PReLU). This activation function adaptively learns the parameters of the rectifiers, and improves accuracy at negligible extra computational cost. Second, we study the difficulty of training rectified models that are very deep. By explicitly modeling the nonlinearity of rectifiers (ReLU/PReLU), we derive a theoretically sound initialization method, which helps with convergence of very deep models (*e.g.*, with 30 weight layers) trained directly from scratch. This gives us more flexibility to explore more powerful network architectures.

On the 1000-class ImageNet 2012 dataset, our network leads to a single-model result of 5.71% top-5 error, which surpasses all multi-model results in ILSVRC 2014. Further, our multi-model result achieves **4.94%** top-5 error on the test set, which is a 26% relative improvement over the ILSVRC 2014 winner (GoogLeNet, 6.66% [33]). To the best of our knowledge, our result surpasses for the first time the reported human-level performance (5.1% in [26]) of a dedicated individual labeler on this recognition challenge.

[1] reported in Feb. 2015.

Exemplo - Inicialização de He

Considerando que temos M conexões com a camada anterior da camada atual, o valor ϵ é dado por:

$$\epsilon = \sqrt{\frac{2}{M}}$$

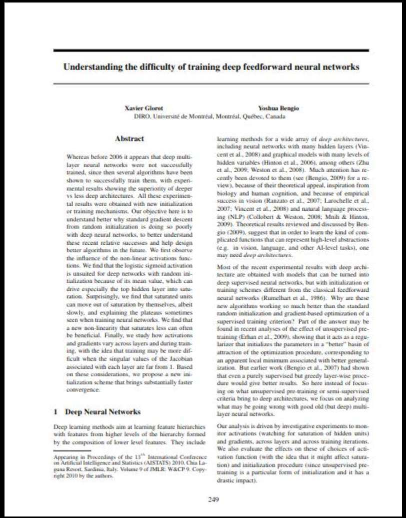
Outro exemplo

Inicialização de Xavier.

$$\epsilon = \sqrt{\frac{2}{M+N}}$$

Onde *N* é o número de conexões de saída.

Glorot, X., & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.



Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

Abstract

Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future. We first observe the influence of the non-linear activations functions. We find that the logistic sigmoid activation is unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. Surprisingly, we find that saturated units can move out of saturation by themselves, albeit slowly, and explaining the plateaus sometimes seen when training neural networks. We find that a new non-linearity that saturates less can often be beneficial. Finally, we study how activations and gradients vary across layers and during training, with the idea that training may be more difficult when the singular values of the Jacobian associated with each layer are far from 1. Based on these considerations, we propose a new initialization scheme that brings substantially faster convergence.

1 Deep Neural Networks

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. They include learning methods for a wide array of *deep architectures*, including neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Much attention has recently been devoted to them (see (Bengio, 2009) for a review), because of their theoretical appeal, inspiration from biology and human cognition, and because of empirical success in vision (Ranzato et al., 2007; Larochelle et al., 2007; Vincent et al., 2008) and natural language processing (NLP) (Collobert & Weston, 2008; Mnih & Hinton, 2009). Theoretical results reviewed and discussed by Bengio (2009), suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one may need *deep architectures*.

Most of the recent experimental results with deep architecture are obtained with models that can be turned into deep supervised neural networks, but with initialization or training schemes different from the classical feedforward neural networks (Rumelhart et al., 1986). Why are these new algorithms working so much better than the standard random initialization and gradient-based optimization of a supervised training criterion? Part of the answer may be found in recent analyses of the effect of unsupervised pre-training (Erhan et al., 2009), showing that it acts as a regularizer that initializes the parameters in a "better" basin of attraction of the optimization procedure, corresponding to an apparent local minimum associated with better generalization. But earlier work (Bengio et al., 2007) had shown that even a purely supervised but greedy layer-wise procedure would give better results. So here instead of focusing on what unsupervised pre-training or semi-supervised criteria bring to deep architectures, we focus on analyzing what may be going wrong with good old (but deep) multi-layer neural networks.

Our analysis is driven by investigative experiments to monitor activations (watching for saturation of hidden units) and gradients, across layers and across training iterations. We also evaluate the effects on these of choices of activation function (with the idea that it might affect saturation) and initialization procedure (since unsupervised pre-training is a particular form of initialization and it has a drastic impact).

Appearing in Proceedings of the 13[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9. Copyright 2010 by the authors.

Transfer Learning

Ideia: usar algum dataset grande para treinar os filtros de convolução da rede.

- O dataset não precisa conter os mesmos dados da tarefa alvo.
  - Exemplo: treinamos uma rede para reconhecer cães, e depois a ajustamos para reconhecer gatos.

Transfer Learning

Ideia: usar algum dataset grande para treinar os filtros de convolução da rede.

- O dataset não precisa conter os mesmos dados da tarefa alvo.
  - Exemplo: treinamos uma rede para reconhecer cães, e depois a ajustamos para reconhecer gatos.

Assume-se que os filtros de convolução podem ser usados para outras tarefas.

- Retreinamos a camada de classificação da rede, que usa como entrada os filtros de convolução já treinados.

Transfer Learning

Ideia: usar algum dataset grande para treinar os filtros de convolução da rede.

- O dataset não precisa conter os mesmos dados da tarefa alvo.
  - Exemplo: treinamos uma rede para reconhecer cães, e depois a ajustamos para reconhecer gatos.

Assume-se que os filtros de convolução podem ser usados para outras tarefas.

- Retreinamos a camada de classificação da rede, que usa como entrada os filtros de convolução já treinados.

Pode ser visto como uma técnica de **regularização**.

- Estamos usando dados de outras tarefas para treinar um modelo para a tarefa alvo.

Exemplo

No exemplo, vamos usar a MobileNetV3 small.

Veja como a rede funciona nos artigos.

Especial atenção para as camadas de *Bottleneck*.

Misturam convoluções com expansões sem uso de não linearidades para dimensões maiores.

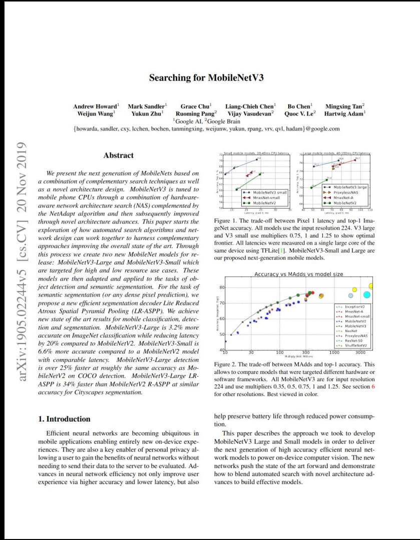
Sandler, M., et al. Mobilenetv2: Inverted residuals and linear bottlenecks. *IEEE conference on computer vision and pattern recognition.* 2018.

MobileNetV2: Inverted Residuals and Linear Bottlenecks

Mark Sandler Andrew Howard Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen
Google Inc.
{sandler, howarda, menglong, azhmogin, lcchen}@google.com

arXiv:1801.04381v4 [cs.CV] 21 Mar 2019

Howard, Andrew, et al. Searching for mobilenetv3. IEEE/CVF international conference on computer vision. 2019.



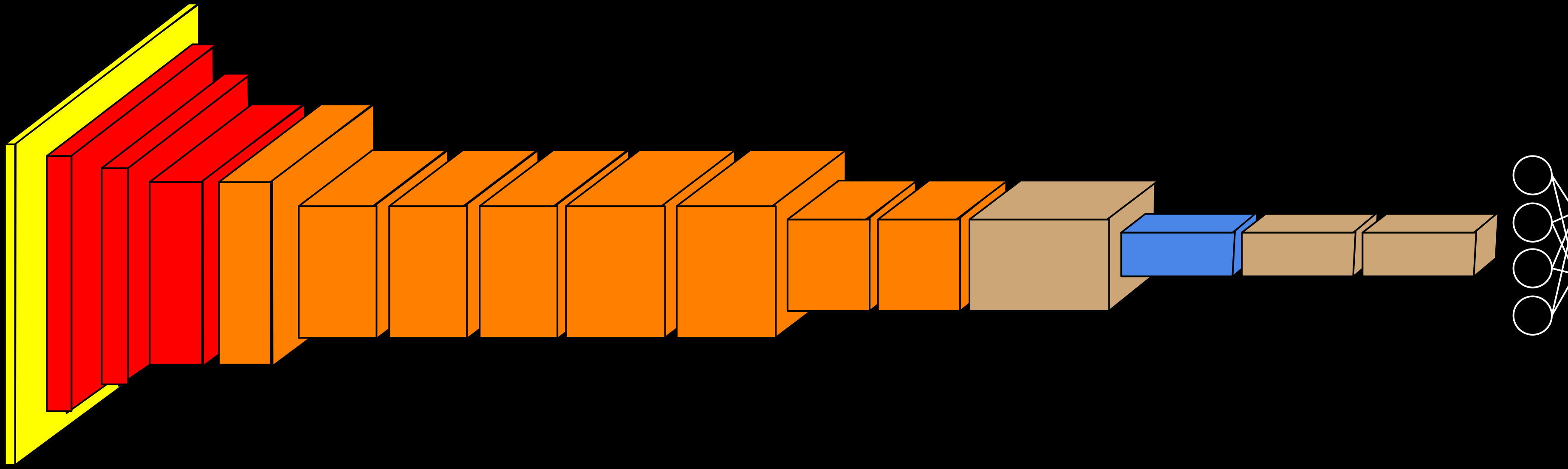
Searching for MobileNetV3

arXiv:1905.02244v5 [cs.CV] 20 Nov 2019

MobileNetV3 Small

Convolucional 3x3
Bottleneck 3x3
Bottleneck 5x5
Pooling
Convolucional 1x1

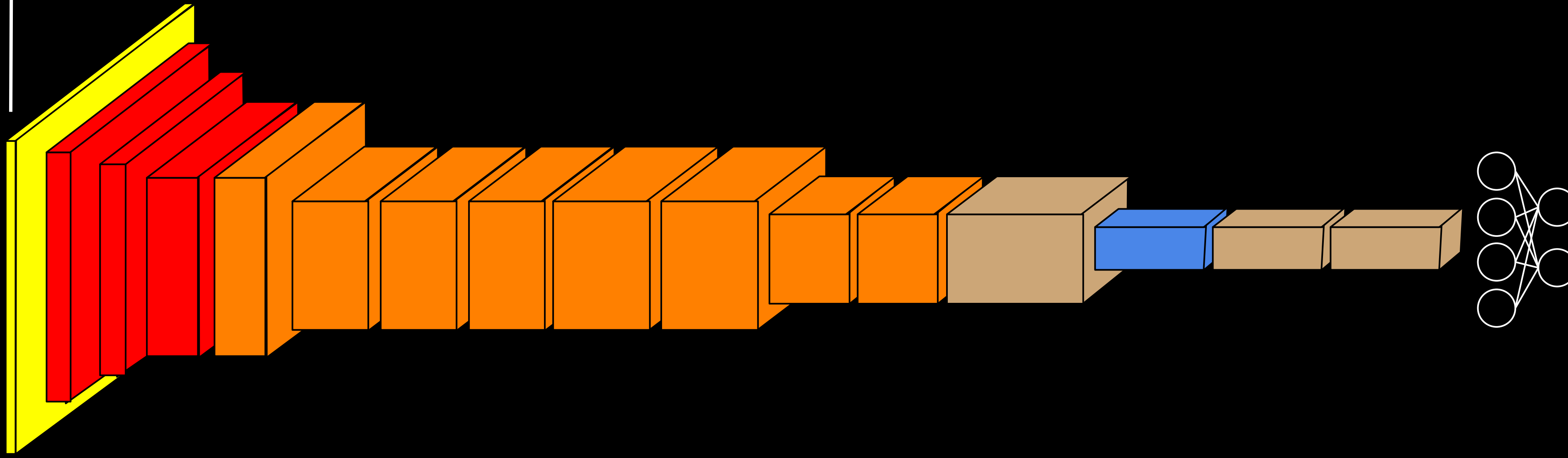
Tensores de entrada 224x224x3.



MobileNetV3 Small

Convolucional 3x3
Bottleneck 3x3
Bottleneck 5x5
Pooling
Convolucional 1x1

Treinar a rede completa na ImageNet – aprox. 14 milhões de imagens.



MobileNetV3 Small

Convolucional 3x3

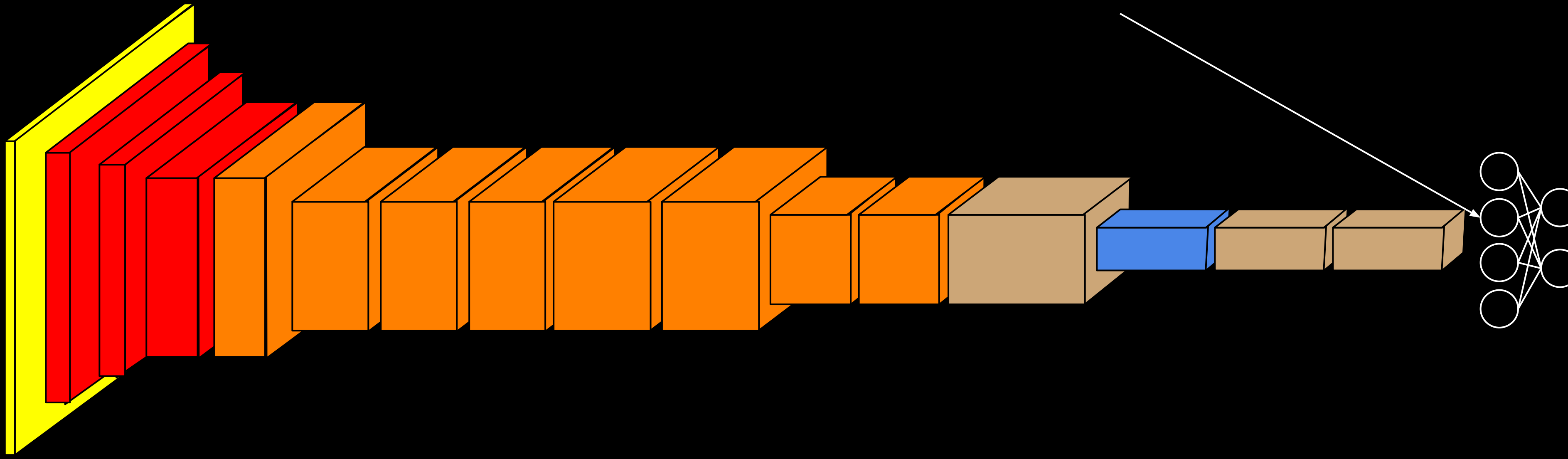
Bottleneck 3x3

Bottleneck 5x5

Pooling

Convolucional 1x1

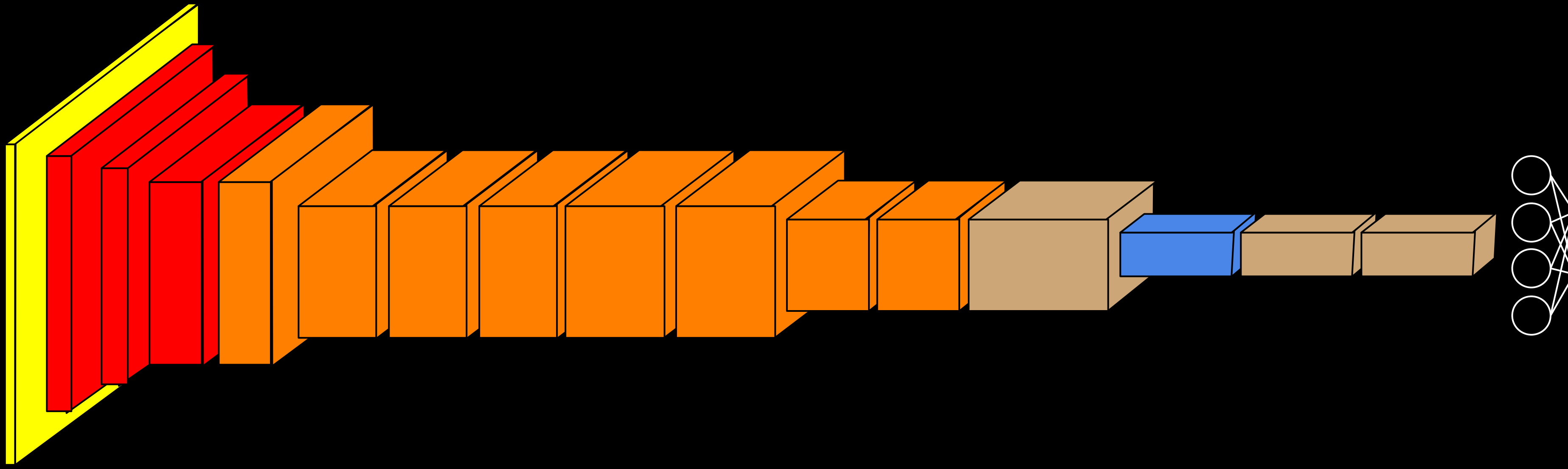
Criar uma nova camada fully-connected, ou retreinar a camada existente nos dados alvo. Não alterar os pesos das demais camadas.



MobileNetV3 Small

Convolucional 3x3
Bottleneck 3x3
Bottleneck 5x5
Pooling
Convolucional 1x1

Camadas que não alteramos os pesos geralmente são chamadas de **camadas congeladas**.



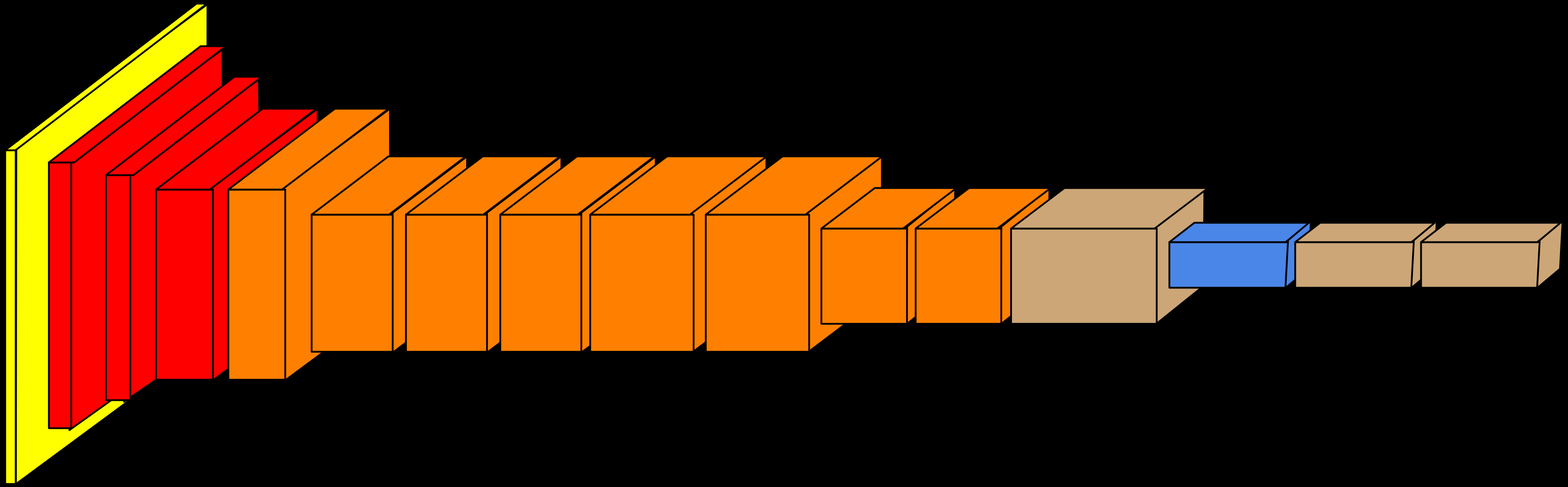
Faça você mesmo

Execute o exemplo de Transfer Learning no Google Colab disponibilizado.

Fine-tuning

Podemos descongelar mais camadas da rede (de trás para frente).

Nesse caso, comumente chamamos de *Fine-tuning* da rede.



Fine-tuning

Um fine-tuning de todas as camadas da rede pode ser considerado uma espécie de inicialização da rede.

A rede foi inicializada em uma região do espaço “boa para outro problema”.

Esperamos que essa região seja, pelo menos, próxima de uma boa região para o problema em questão.

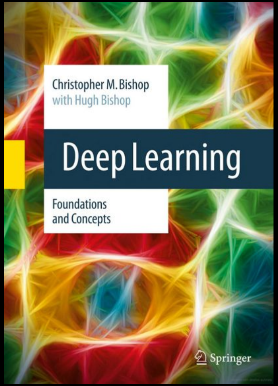
Geralmente usamos fatores de aprendizagem baixos para realizar o fine-tuning.

Exercícios

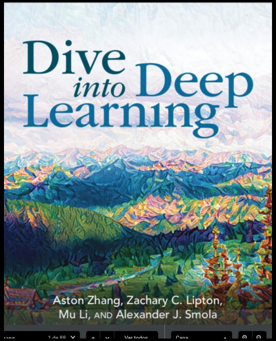
1. Baseado no exemplo de transfer learning.
    a. Faça um Fine-tuning em todas as camadas da rede.
    b. Treine, e compare os resultados com os obtidos com o transfer learning.

Referências

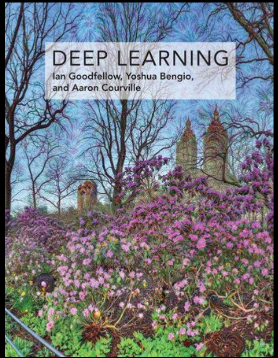
Bishop, C. M., Bishop, H. Deep Learning: Foundations and Concepts. 2023.



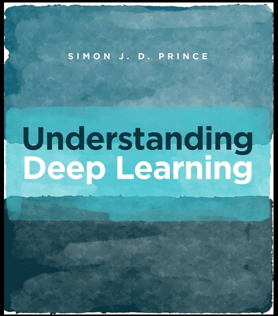
Zhang, A., Lipton, Z. C., Li, M., Smola, A. J. Dive Into Deep Learning. Índia. 2023.



Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. 2016.



Prince, S. J. Understanding Deep Learning. 2023.



Licença