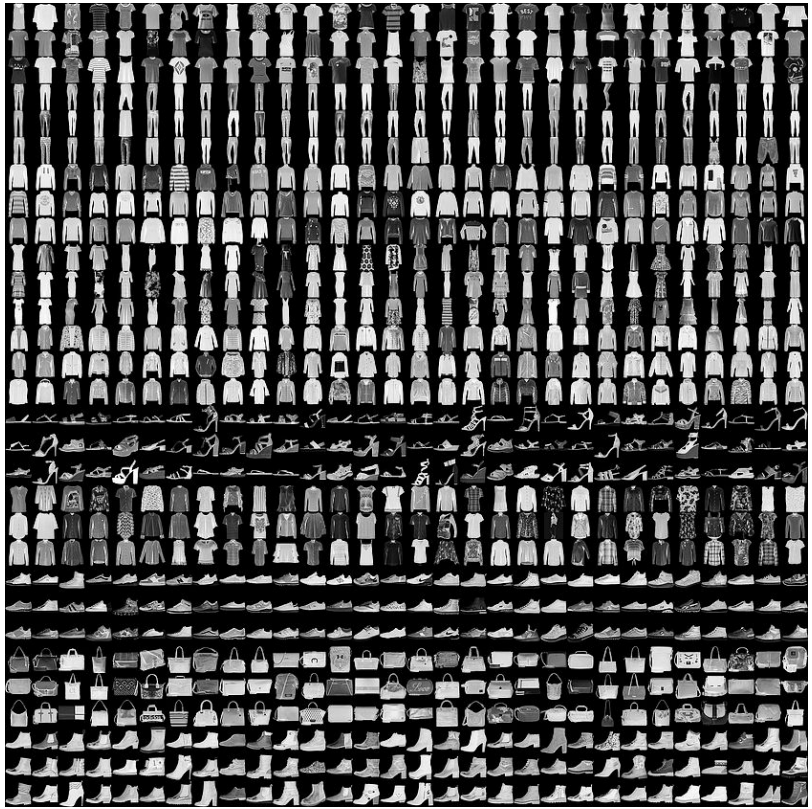
Convolutional Neural Networks

Prof.: Leandro Bezerra Marinho

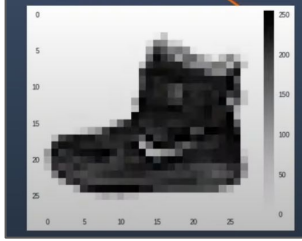
Um classificador para aprender imagens de roupa

Um classificador para aprender imagens de roupa

- 70k imagens
- 10 classes
- Imagens são 28x28
- Pode usar uma rede neural artificial
- Link do colab https://colab.research.google.com/drive/1HfgVZ2WKhV1GVkQRdK79KWA-Hsw-bFTm?usp=sharing

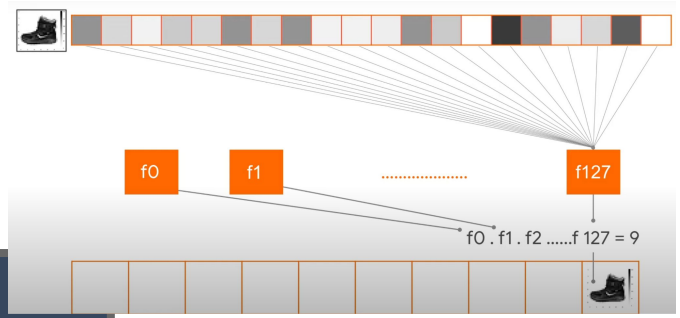


```python
import tensorflow as tf
from tensorflow import keras

fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```



09

```python
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy')
```



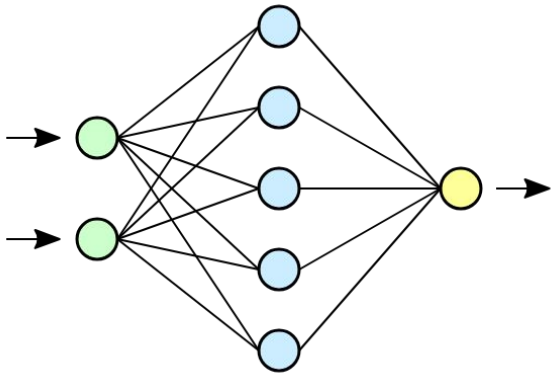

```python
model.fit(train_images, train_labels, epochs=5)

test_loss, test_acc = model.evaluate(test_images, test_labels)

predictions = model.predict(my_images)
```

Problema!

- Se o item não estiver posicionado no meio?
- Se tiver outros objetos na imagem?

Diferença entre RNAs e CNNs

- Semelhantes às **Redes Neurais Artificiais**
  - **Neurônios** com **pesos** e **vieses (bias)** que podem ser aprendidos.
  - Cada **neurônio** recebe algumas **entradas**, executa um produto escalar e opcionalmente o segue com uma não linearidade.
  - **Função de pontuação diferenciável**: desde os pixels brutos da imagem em uma extremidade até as pontuações das classes na outra.
  - **Função de perda** (por exemplo, **softmax**) na última camada (totalmente conectada)
  - **Técnicas** para **aprender** se aplicam.
- **Então, o que muda?**
  - **ConvNet** fazem a suposição explícita de que as **entradas são imagens**
  - Isso torna a **função forward** mais eficiente para implementar e reduz enormemente a quantidade de parâmetros na redes

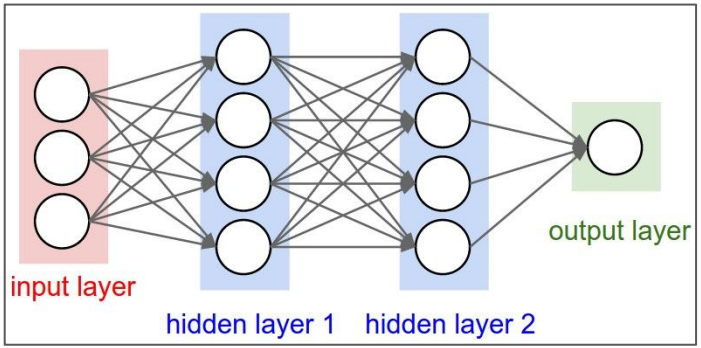


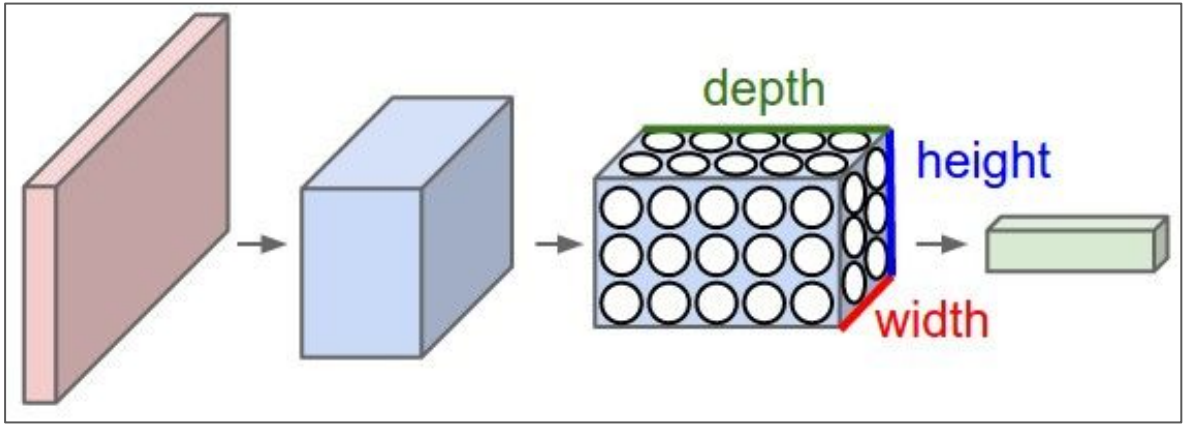
Diferença entre RNAs e CNNs

- **Redes Neurais Artificiais** (RNAs)
  - **Entrada** (um único vetor) e a transformam através de uma série de camadas ocultas.
  - **Camada oculta**: conjunto de neurônios conectado a todos os outros da camada anterior
  - **Camada de saída:** totalmente conectada e indicam a classe
- RNAs não se adaptam bem a imagens completas
  - **CIFAR-10**: imagens de tamanho **32x32x3**, (um **único neurônio totalmente conectado** em uma primeira camada oculta de uma rede neural regular teria 32 * 32 * 3 = 3.072 pesos).
  - Esta estrutura totalmente conectada **não** se adapta a **imagens maiores**.
  - Por exemplo, imagem de 200x200x3 levaria a neurônios com pesos 200*200*3 = **120.000**.
  - Claramente precisamos de mais neurônios, o que levaria a um **overfitting**.

Visão geral da arquitetura da CNN

- As camadas de uma ConvNet possuem neurônios organizados em 3 dimensões: largura, altura, profundidade.
- Cada camada de um ConvNet transforma o **volume de entrada 3D** em um volume de **saída 3D de ativações** de neurônios.
    - Neste exemplo, a camada de entrada vermelha contém a imagem, portanto sua largura e altura seriam as dimensões da imagem e a profundidade seria 3





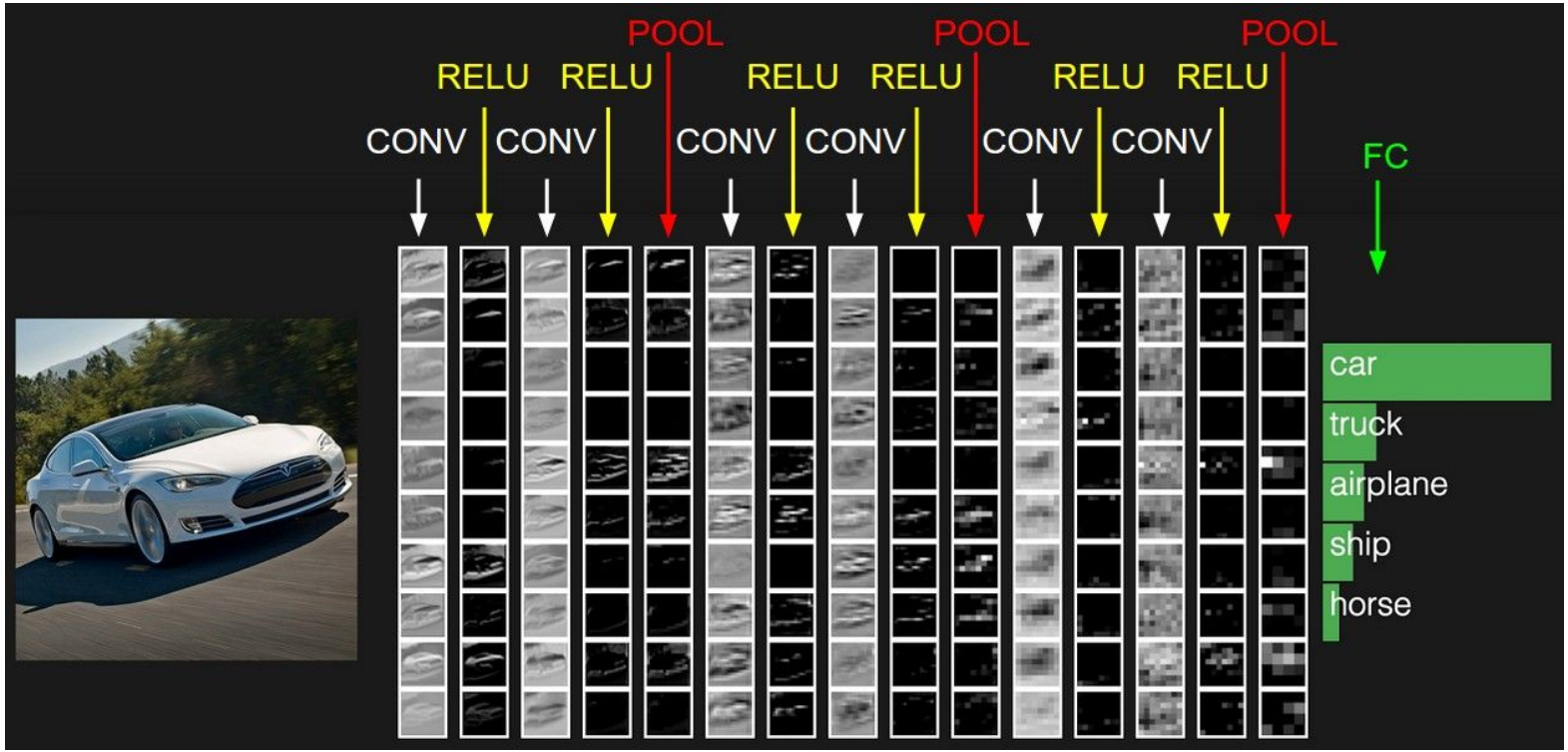
Camadas usadas para construir ConvNets

- **Camada Convolucional**, **Camada Pooling** e **Camada Totalmente Conectada**.
- Um ConvNet simples para classificação **CIFAR-10** poderia ter a arquitetura [INPUT - CONV - RELU - POOL - FC]. Em mais detalhes:
  - **INPUT [32x32x3]** valores brutos de **pixel** da imagem
  - **CONV (filtros):** calculará a saída dos neurônios que estão conectados às regiões locais na entrada, cada um computando um produto escalar entre seus pesos e uma pequena região à qual estão conectados no volume de entrada. Isso pode resultar em volumes como [32x32x12] se decidirmos usar **12 filtros**.
  - **RELU**: função de ativação elemento a elemento, como max(0,x). Deixa o tamanho do volume inalterado ([32x32x12]).
  - **POOL**: reduz a resolução ao longo das dimensões espaciais (largura, altura), resultando em um volume como [16x16x12].
  - **FC**: calcula as **classe**, resultando em um volume de tamanho [1x1x10], onde cada um dos 10 números corresponde a uma pontuação da classe, como entre as 10 categorias do CIFAR-10.

Camadas usadas para construir ConvNets

- Lista de camadas que **transformam o volume da imagem** em um **volume de saída** (por exemplo, contendo os scores das classes)
- Camadas mais populares: CONV/FC/RELU/POOL
- Cada camada aceita um **volume 3D de entrada** e o **transforma** em um **volume 3D de saída** através de uma **função diferenciável**.
- Cada camada pode ou não ter **parâmetros** (por exemplo, CONV/FC sim, RELU/POOL não)
- Cada camada pode ou não ter **hiperparâmetros adicionais** (por exemplo, CONV/FC/POOL sim, RELU não)

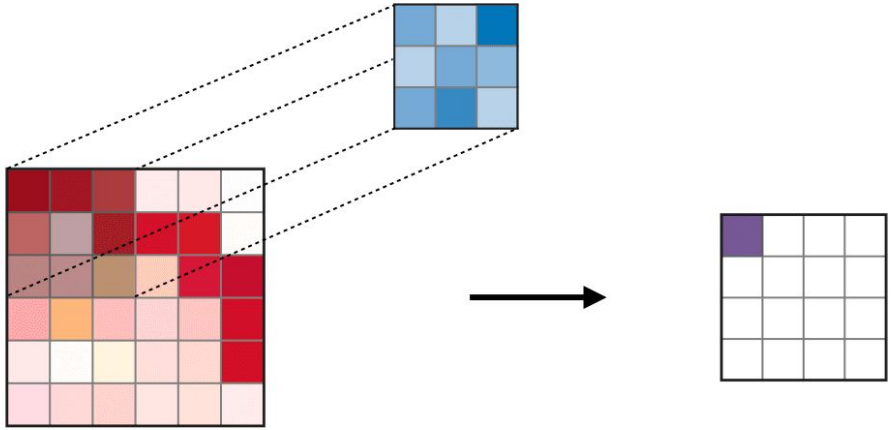
As ativações de um exemplo de arquitetura ConvNet (VGG)



Demonstração

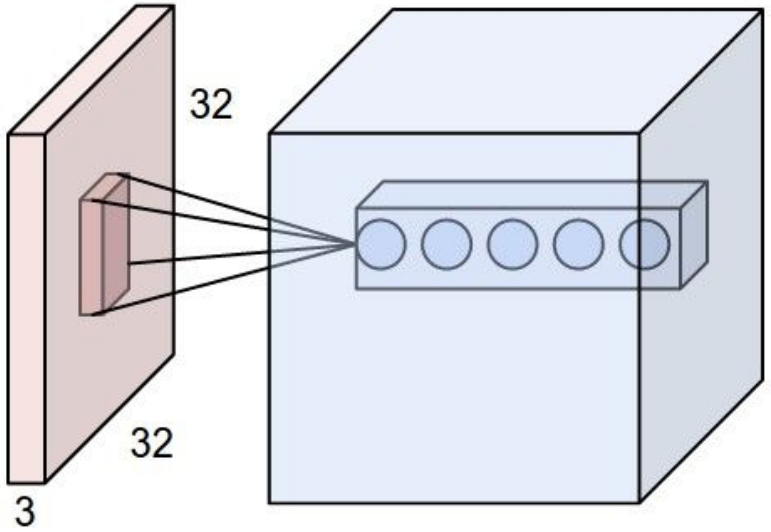
Camada convolucional

- **Bloco de construção** central de uma ConvNet que faz a maior parte do trabalho pesado computacional.
- Usa **filtros** que podem ser aprendidos e realizam operações de **convolução** enquanto varre a entrada I em relação às suas dimensões.
- Seus hiperparâmetros incluem o **tamanho do filtro F** e o **passo S.** A saída resultante O é chamada de mapa de características ou **mapa de ativação**.
- **Analogia ao cérebro**: semelhante à maneira como os **neurônios corticais** identificam **características visuais** em nossa percepção.

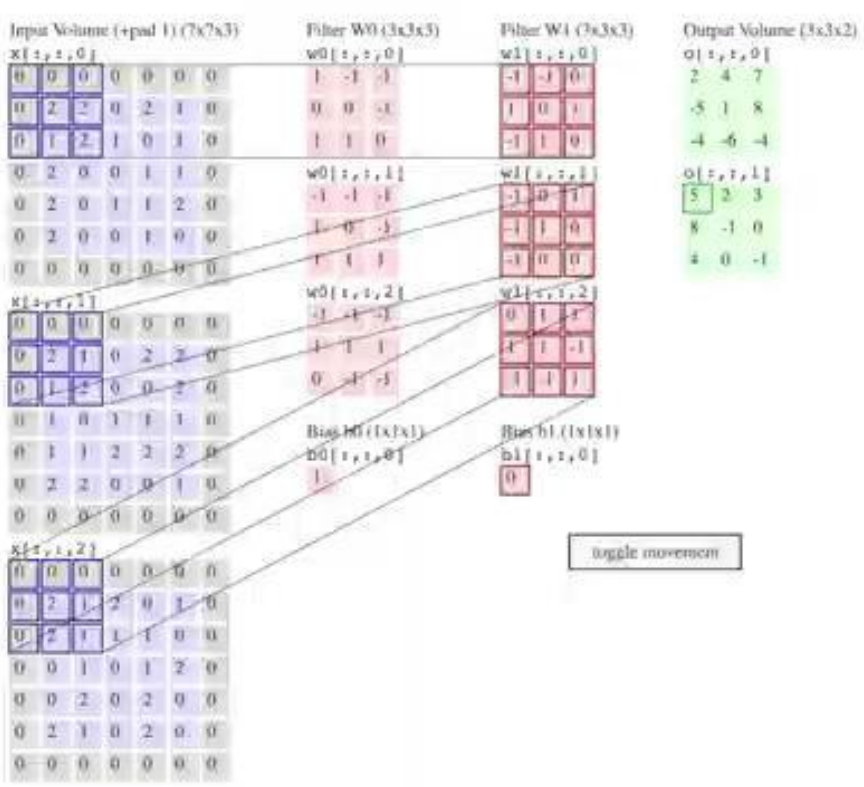


Camada convolucional

- Por exemplo, suponha que o volume de entrada tenha tamanho **[32x32x3]**, (por exemplo, uma imagem RGB CIFAR-10).
- Se o **campo receptivo** (ou tamanho do filtro) for **5x5**, então cada neurônio na camada Conv terá pesos para uma região **[5x5x3]** no volume de entrada, para um total de **5*5*3 = 75** pesos (e +1 parâmetro de bias).
- Há vários neurônios (5 neste exemplo) ao longo da profundidade, todos olhando para a mesma região na entrada
- Esses 5 neurônios **não compartilham o mesmo pesos**, mas compartilham o mesmo campo receptivo.

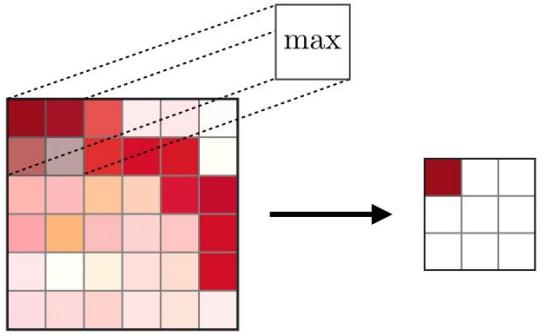


Camada convolucional

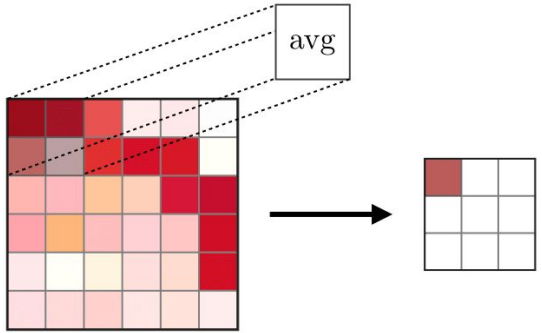


Camada de Pooling

- É uma **operação de redução da resolução**, normalmente aplicada após uma camada de convolução, que faz alguma invariância espacial.
- Sua função é reduzir a **quantidade de parâmetros** e computação na rede e, portanto, controlar também o overfitting.

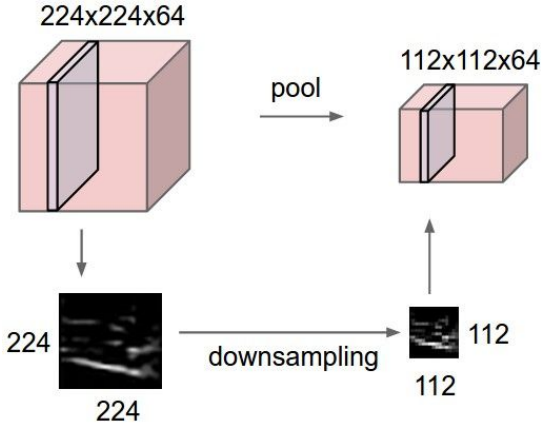


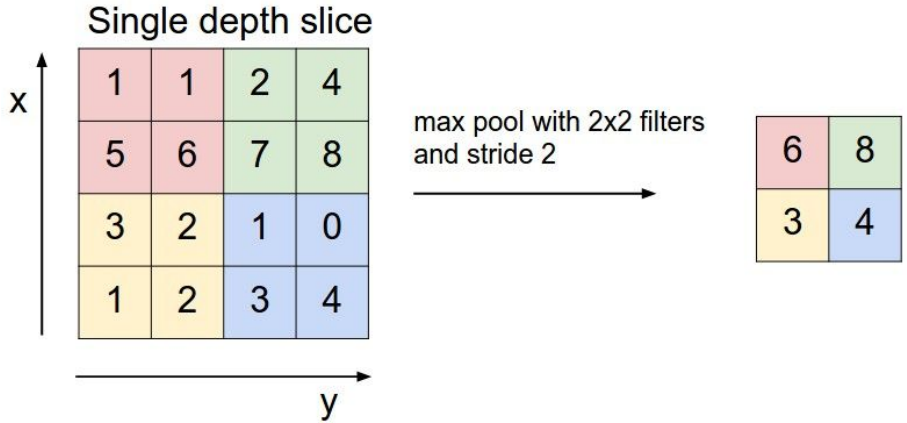
• Preserva as características detectados
• Mais comumente usado



• Downsamples do mapa de ativação
• Usado no LeNet

Camada de Pooling

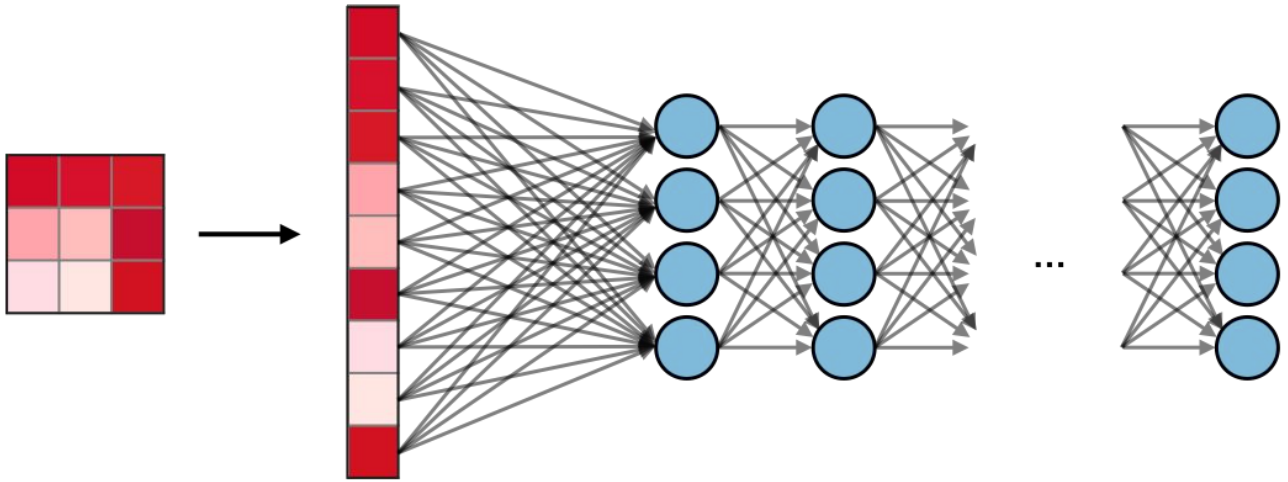




- **Esquerda**: o volume de entrada de tamanho [224 x 224 x 64] é agrupado com filtro de tamanho 2, passo 2 no volume de saída de tamanho [112 x 112 x 64]. A profundidade do volume é preservada.
- **À direita**: A operação de redução da resolução mais comum é max, dando origem ao pooling máximo, aqui mostrado com um passo de 2.

Camada Totalmente Conectada

- Opera em uma entrada achatada onde cada entrada está conectada a todos os neurônios.
- Se presentes, as camadas FC são geralmente encontradas no **final** das arquiteturas CNN e podem ser usadas para indicar a classe.

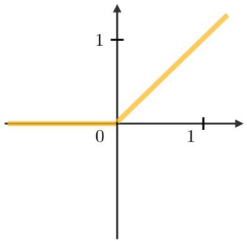


Funções de ativação comumente usadas

- **Unidade linear retificada (ReLU)**: (ReLU) é usada em todos os elementos do volume. Tem como objetivo introduzir não linearidades na rede. Suas variantes estão resumidas na tabela:
- **Softmax**: pode ser vista como uma função logística generalizada que toma como entrada um **vetor de scores** gera um **vetor de probabilidade de saída** no final da arquitetura.

ReLU

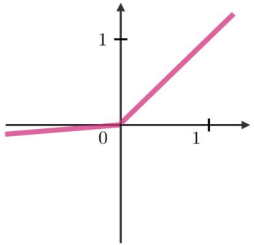
$g(z) = \max(0, z)$



• Complexidades não lineares biologicamente interpretáveis

Leaky ReLU

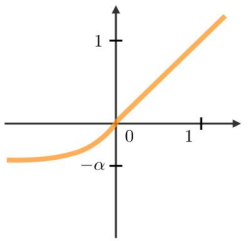
$g(z) = \max(\epsilon z, z)$
with $\epsilon \ll 1$



• Resolve o problema ReLU para valores negativos

ELU

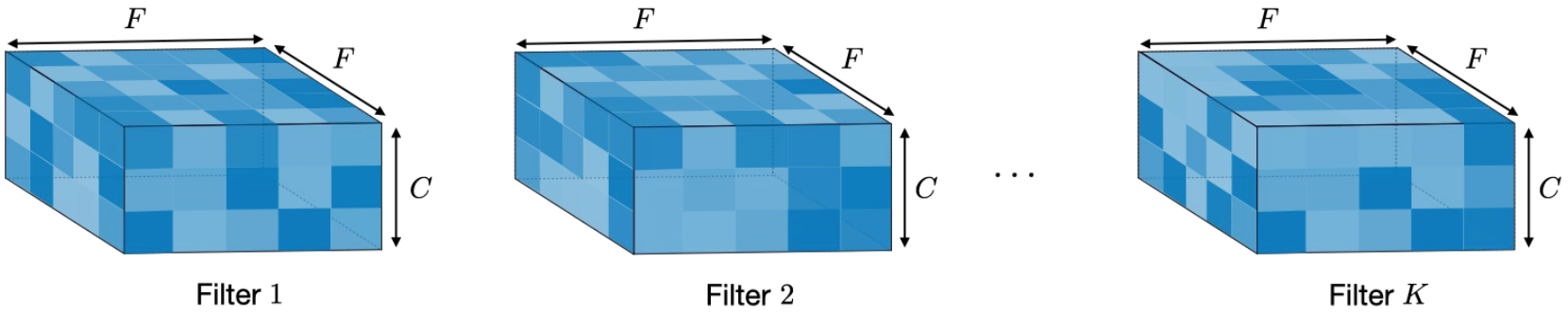
$g(z) = \max(\alpha(e^z - 1), z)$
with $\alpha \ll 1$



• Diferenciável em qualquer lugar

Hiperparâmetros dos filtros

- **Dimensões de um filtro, passo e preenchimento de zeros.**
- **Dimensões de um filtro**: um filtro de tamanho FxF aplicado a uma entrada contendo canais C é um **volume F×F×C** que executa convoluções em uma entrada de tamanho I×I×C e produz um mapa de características de saída (ou de **mapa de ativação**) de tamanho O×O×1.
- A aplicação de K filtros de tamanho F×F resulta em um mapa de ativação de saída de tamanho **O×O×K**.

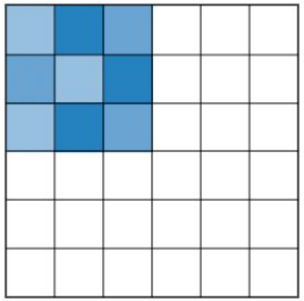


Hiperparâmetros dos filtros

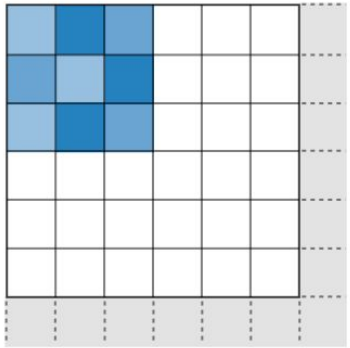
- **Passo**: para uma operação convolucional ou de pooling, o passo S denota o número de pixels pelos quais a janela se move após cada operação.

Hiperparâmetros dos filtros

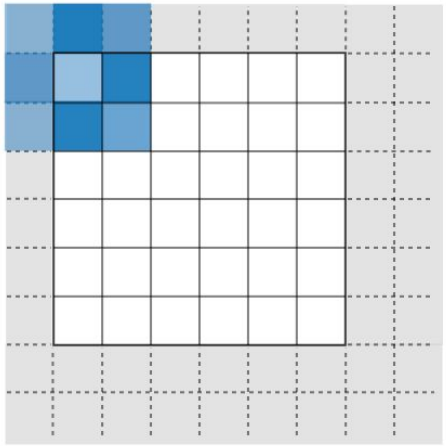
- **Preenchimento de zeros**: denota o processo de adição de **P** zeros a cada lado dos limites da entrada. Este valor pode ser especificado manualmente ou definido automaticamente através de um dos três modos detalhados abaixo



• Sem preenchimento
• Elimina a última convolução se as dimensões não corresponderem



• Preenchimento de forma que o tamanho do mapa de ativação tenha tamanho [I/S]
• O tamanho da saída é matematicamente conveniente
• Também chamado de 'meio' preenchimento



• Preenchimento máximo de modo que convoluções finais sejam aplicadas nos limites da entrada
• O filtro 'vê' a entrada de ponta a ponta

Construir convoluções e realizar pooling

Build convolutions and perform pooling

33 mins remaining

1. Before you begin
2. **What are convolutions?**
3. Start coding
4. Create the convolution matrix
5. Examine the results
6. Understanding Pooling
7. Write code for pooling
8. Congratulations

Report a mistake

Back

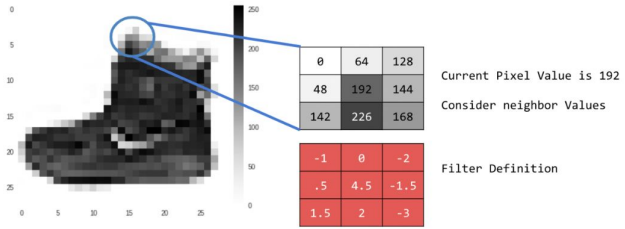
2. What are convolutions?

A convolution is a filter that passes over an image, processes it, and extracts the important features.

Let's say you have an image of a person wearing a sneaker. How would you detect that a sneaker is present in the image? In order for your program to "see" the image as a sneaker, you'll have to extract the important features, and blur the inessential features. This is called *feature mapping*.

The feature mapping process is theoretically simple. You'll scan every pixel in the image and then look at its neighboring pixels. You multiply the values of those pixels by the equivalent weights in a filter.

For example:




```
CURRENT_PIXEL_VALUE = 192
NEW_PIXEL_VALUE = (-1 * 0) + (0 * 64) + (-2 * 128) +
                  (.5 * 48) + (4.5 * 192) + (-1.5 * 144) +
                  (1.5 * 42) + (2 * 226) + (-3 * 168)
```

In this case, a 3x3 convolution matrix, or image *kernel*, is specified.

https://developers.google.com/codelabs/tensorflow-3-convolutions#0

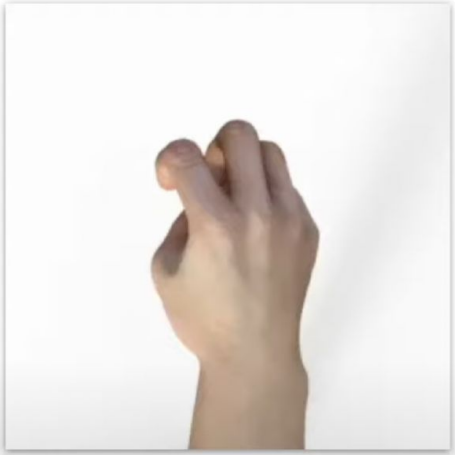
Como melhorar a visão computacional e a precisão com convoluções com Keras

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

```
import tensorflow as tf
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
                         input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

https://developers.google.com/codelabs/tensorflow-4-cnns#7

Classificador de imagem para pedra, papel e tesoura



```
01010010101001010101
00101010100101110101
00101010010101001010
1001010100101010
```

Rótulo = **Pedra**

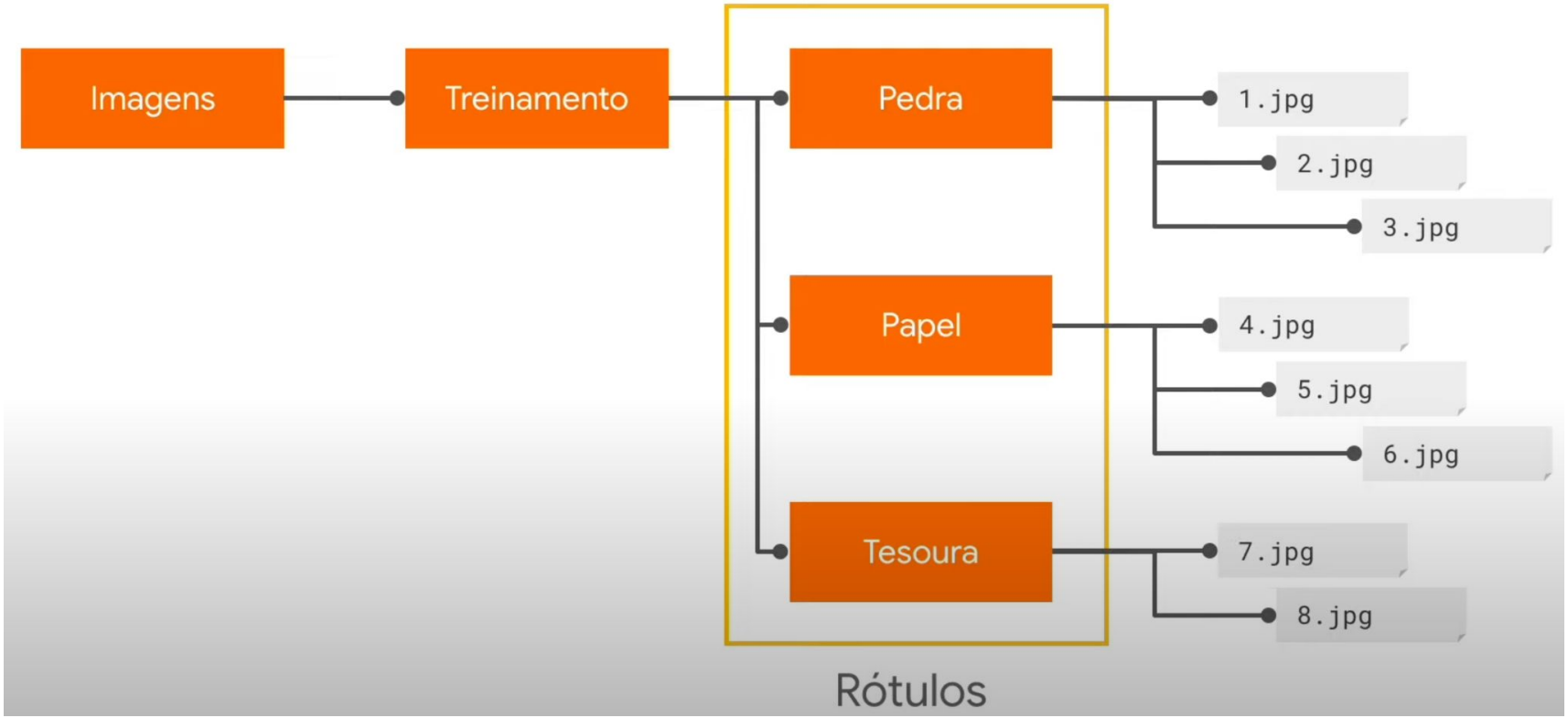
```
10010100111110101011
10101011101010111010
10101111010101011111
1110001111010101
```

Rótulo = **Tesoura**

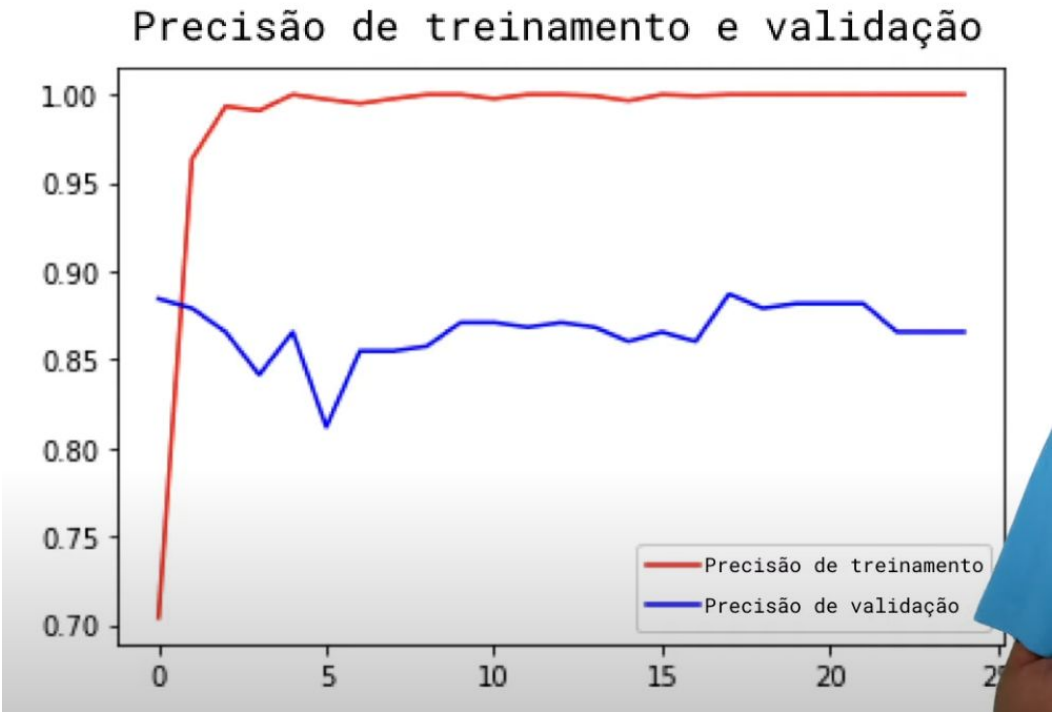
```
10101001010010101010
10101001001001000100
10011111010101111101
0100100111101011
```

Rótulo = **Papel**

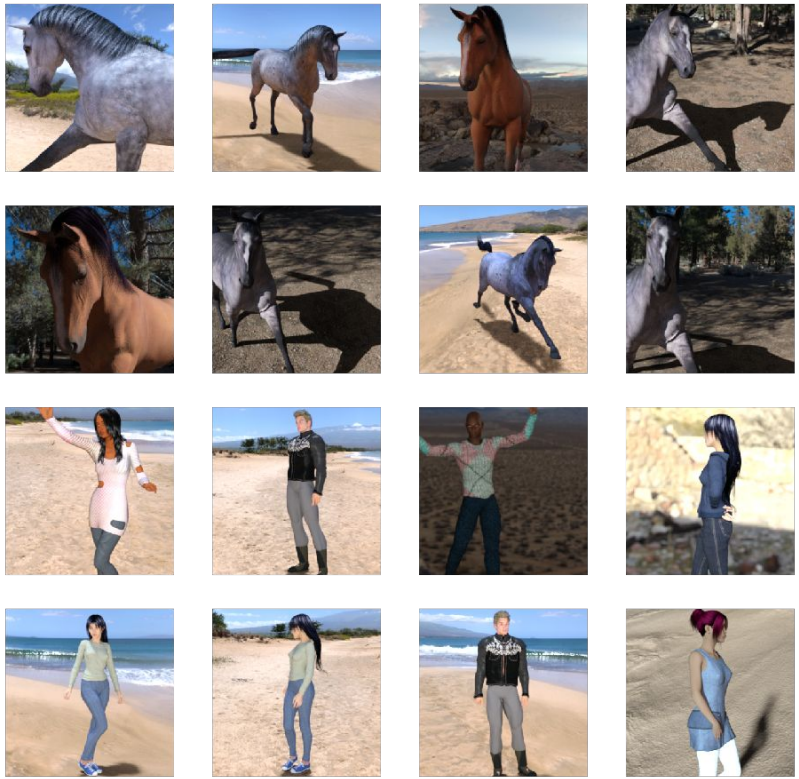
Classificador de imagem para pedra, papel e tesoura



Classificador de imagem para pedra, papel e tesoura (Keras)

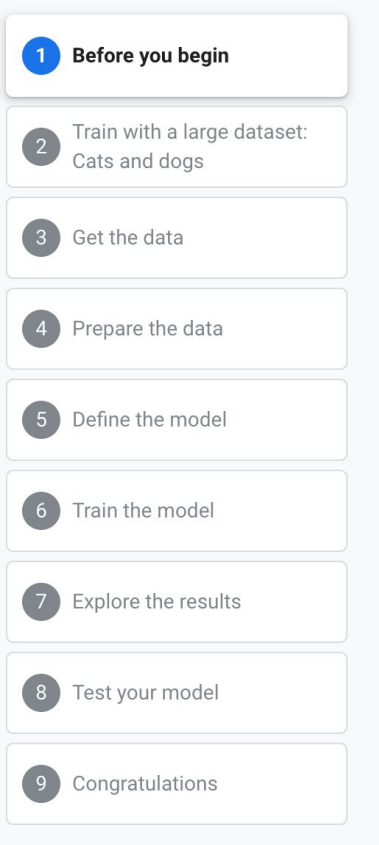


Como treinar computadores para reconhecer atributos em uma imagem na qual o contexto não está claro (Keras)



https://developers.google.com/codelabs/tensorflow-5-complexeimages#0

Como evitar overfitting (Keras)



https://developers.google.com/codelabs/tensorflow-6-largecnns#0

Material adaptado de

Introdução às redes neurais convolucionais (TensorFlow)

e

Convolutional Neural Networks (Stanford)