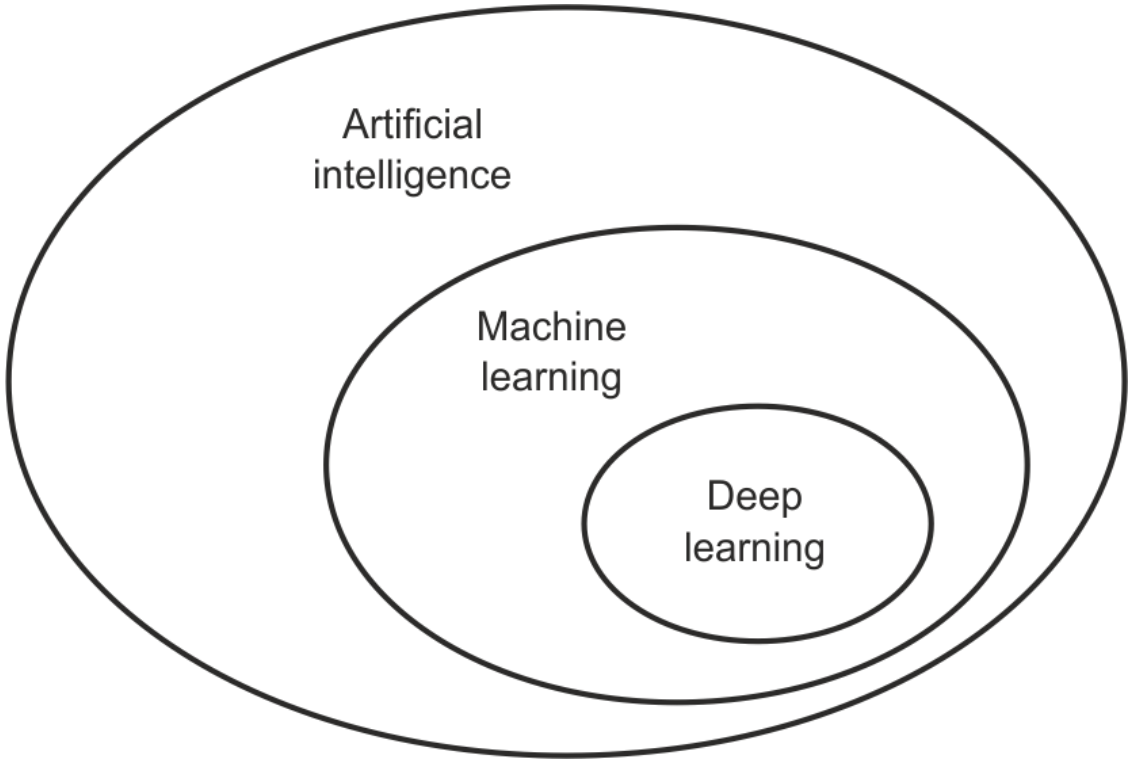
Análise de Imagens e Visão Computacional

Prof. Henrique Batista da Silva

PUC Minas

Introdução a redes neurais

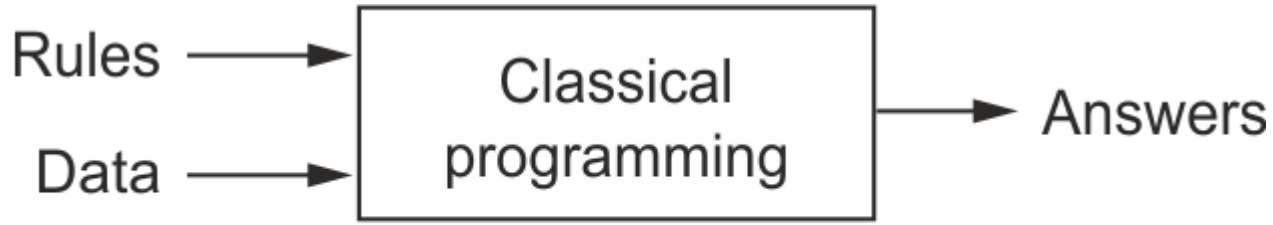
Introdução

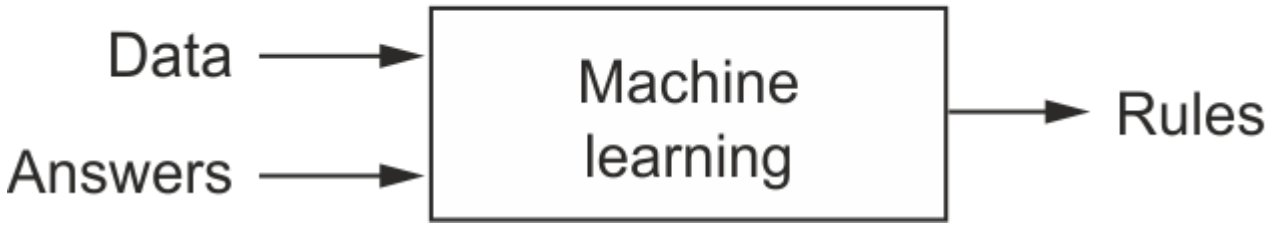


Fonte: François Chollet. **Deep Learning with Python.** November 2017

Artificial intelligence, machine learning, and deep learning

Introdução

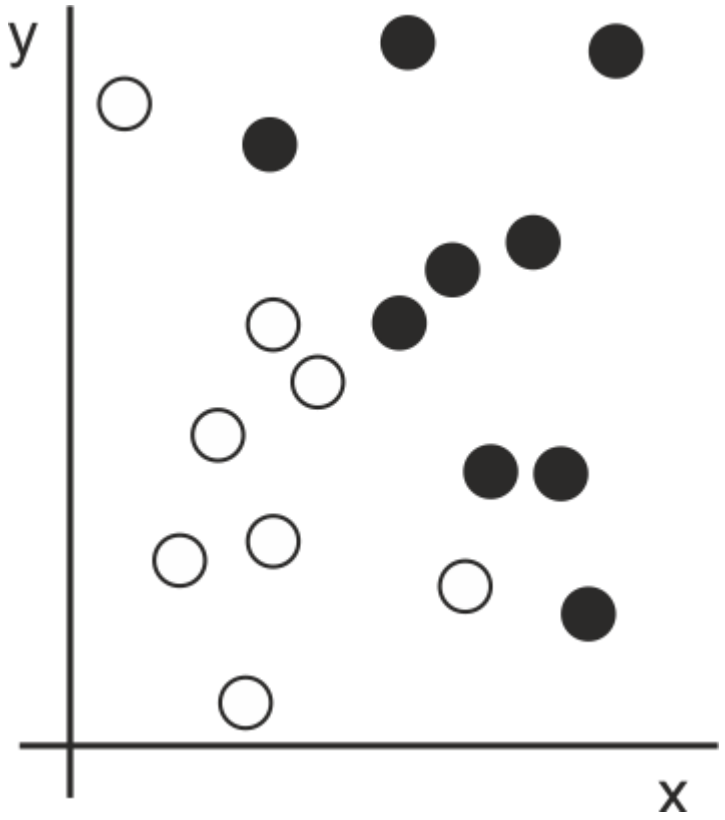




Machine learning: a new programming paradigm

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Introdução

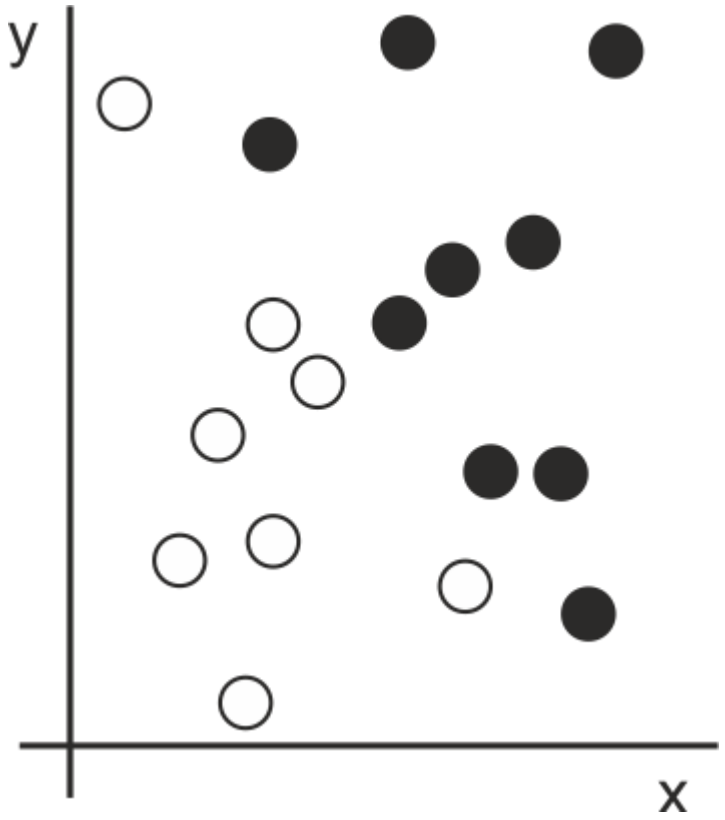


Fonte: François Chollet. **Deep Learning with Python.** November 2017

Learning representations from data:

As entradas do modelo são as coordenadas dos pontos (imagens, por exemplo)

Introdução



Fonte: François Chollet. **Deep Learning with Python.** November 2017

A saída esperada é a cor dos pontos.

Medimos o algoritmo pela porcentagem de pontos classificados corretamente.

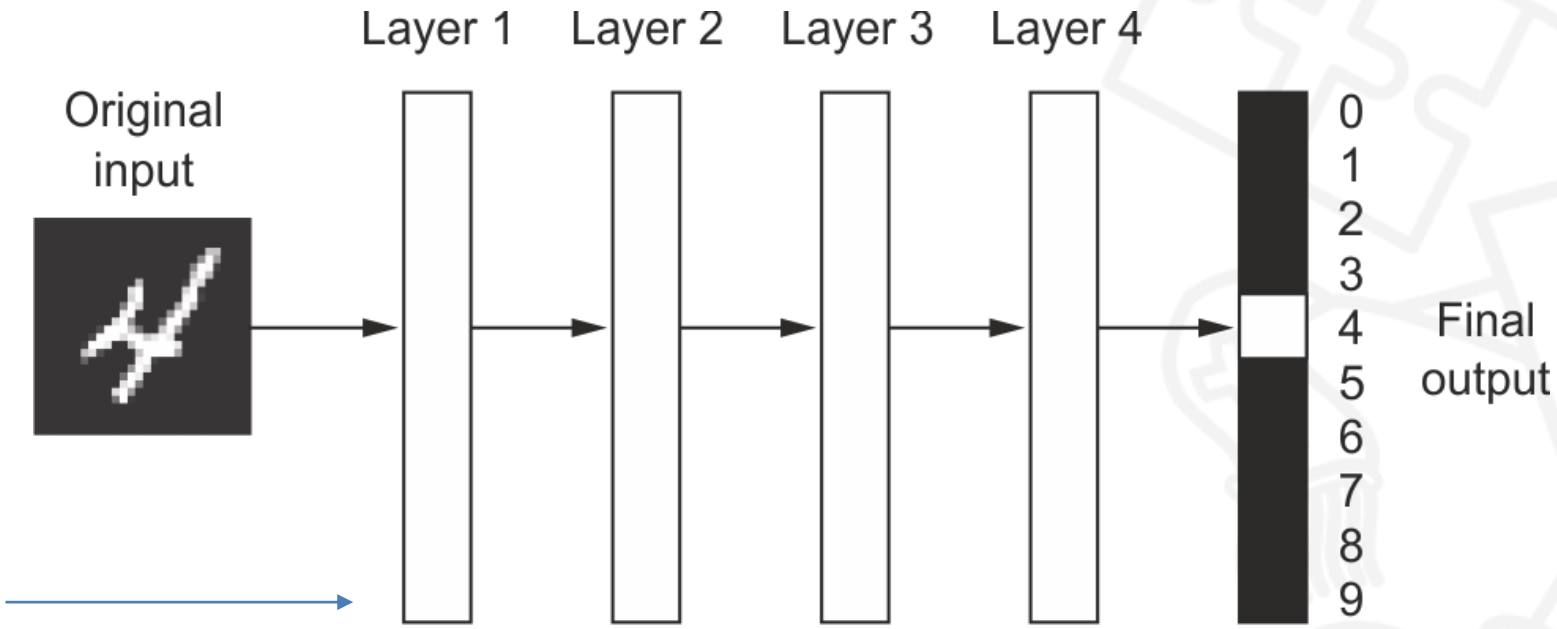
Deep Learning

A ideia do aprendizado profundo (deep) está relacionada sucessivas camadas (layer) de representação.

Estas camadas de representação são construídas por meio de modelos de redes neurais. Assim, **deep learning é um framework para aprendizado de representações a partir de dados** (input).

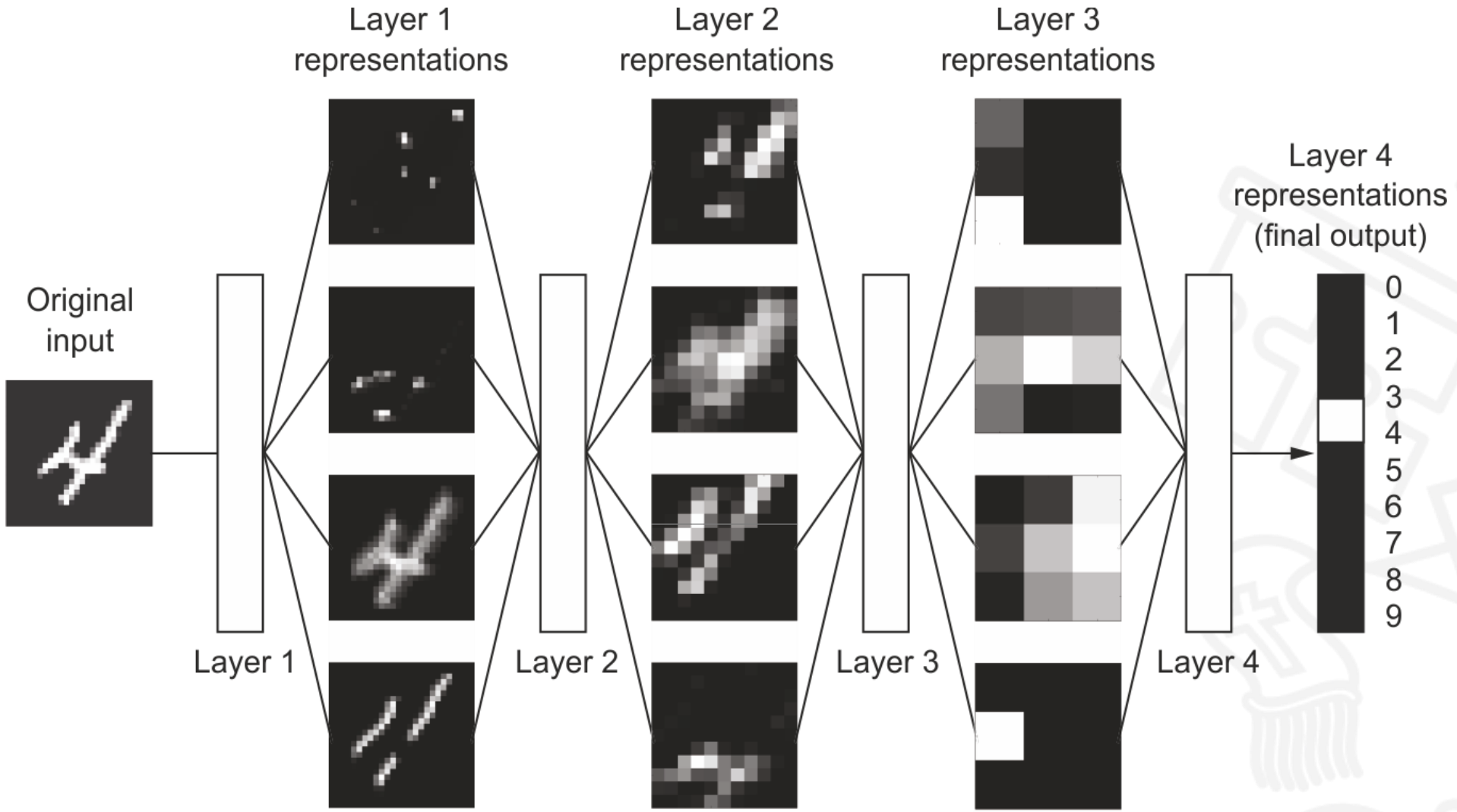
Deep Learning

Rede neural para classificação de dígitos



Transformação do dado de entrada

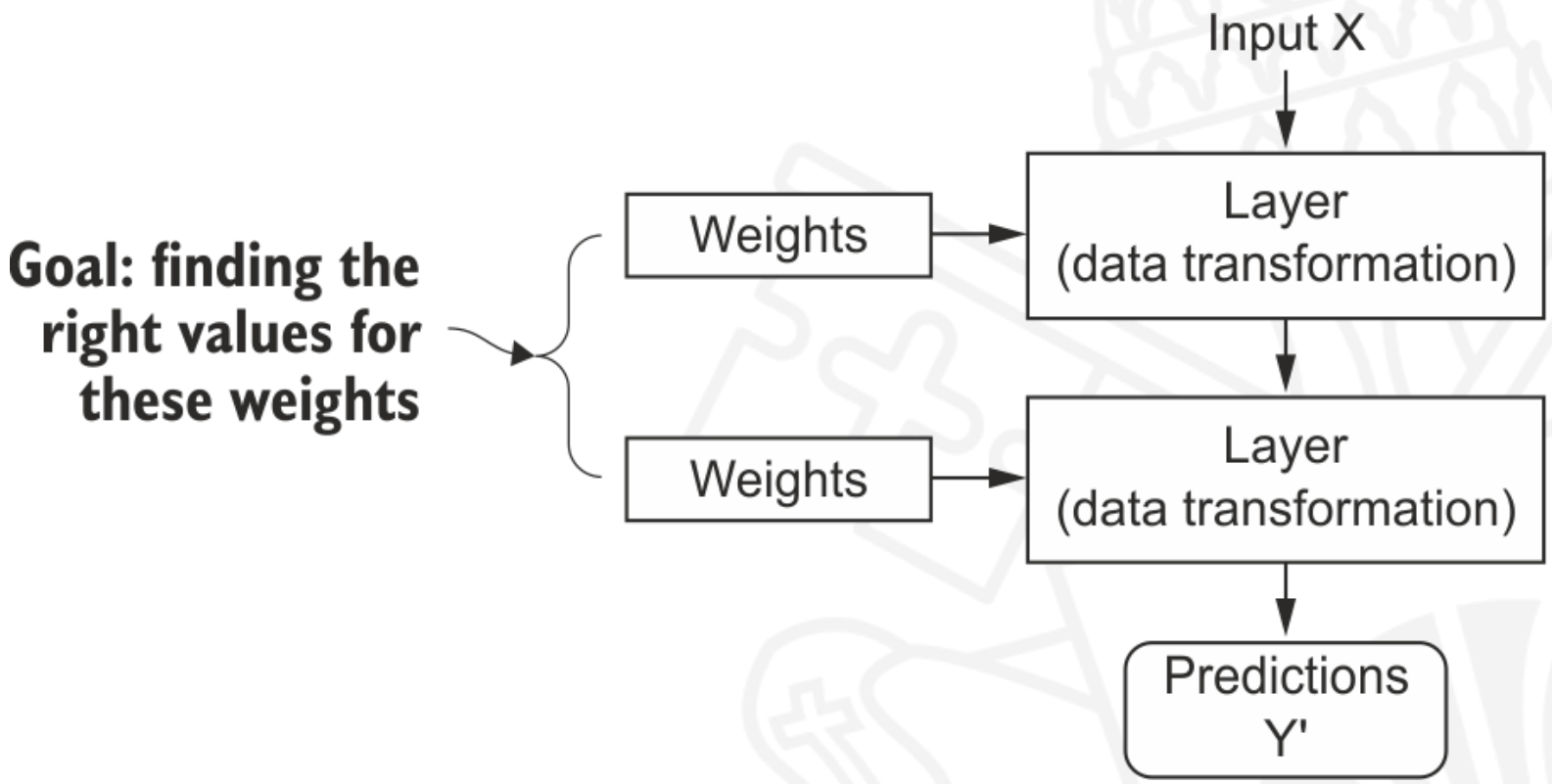
Fonte: François Chollet. **Deep Learning with Python.** November 2017



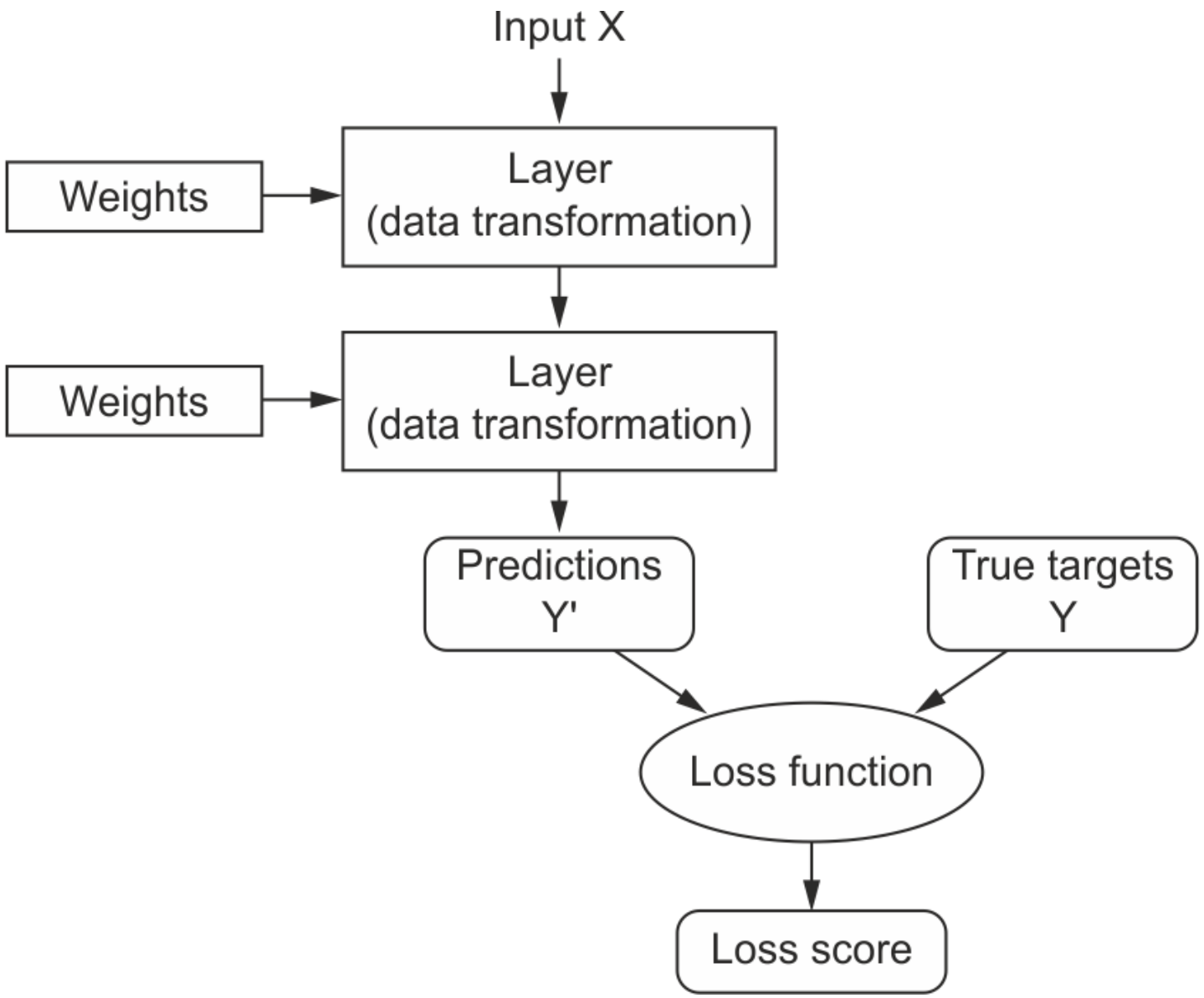
Fonte: François Chollet. **Deep Learning with Python.** November 2017

A transformação implementada pela camada é parametrizada pelos seus pesos (parâmetros de entrada da camada), que são iniciados aleatoriamente.

“Aprender” significa encontrar um conjunto de valores para os pesos de todas as camadas da rede, de forma que a rede irá mapear corretamente exemplos de entrada para a sua saída.



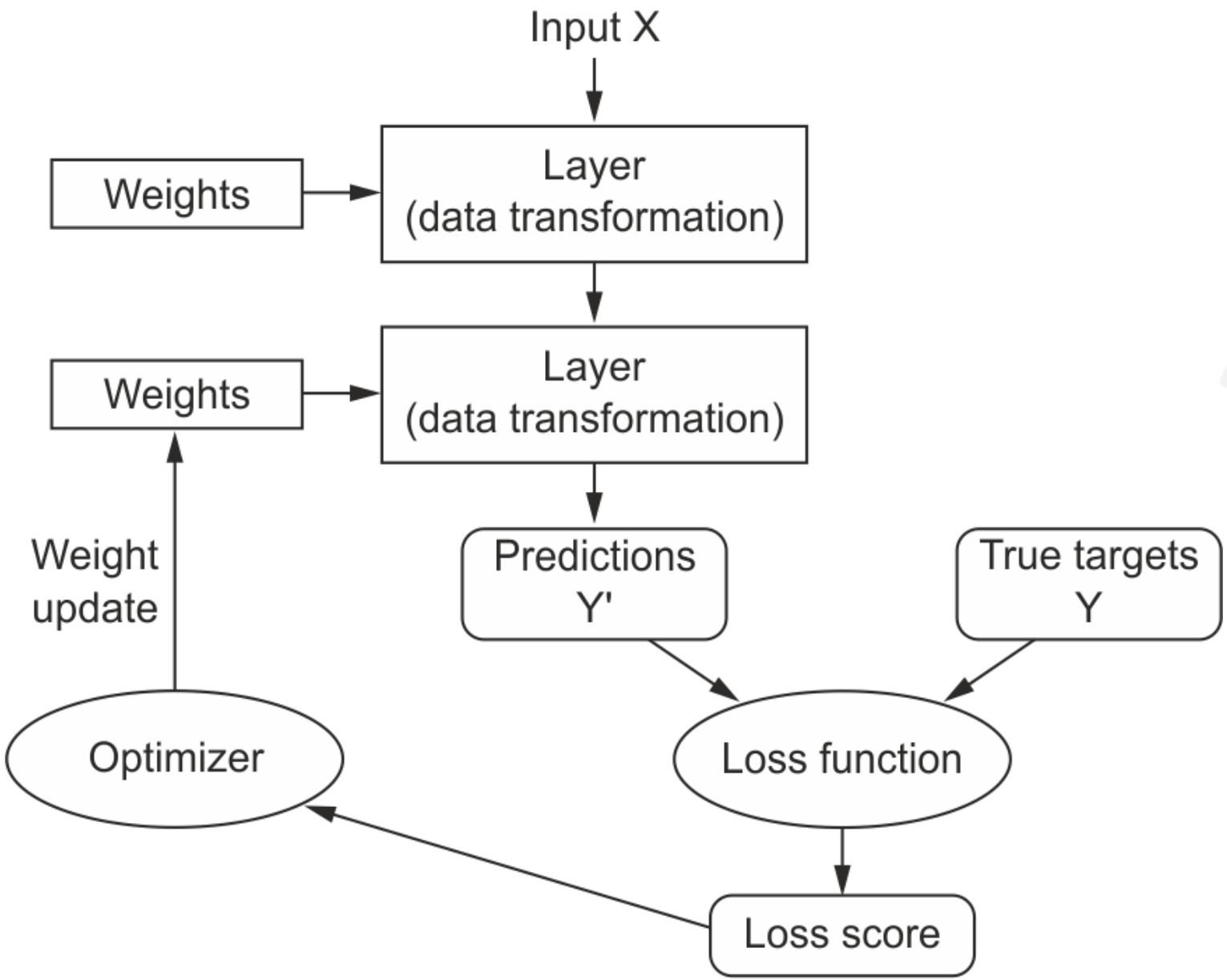
Fonte: François Chollet. **Deep Learning with Python.** November 2017



Para medir a saída, devemos saber o quão distante a saída está do resultado esperado (loss function)

A loss function calcula o quão distante a predição está do resultado esperado.

Fonte: François Chollet. **Deep Learning with Python.** November 2017



Assim, o score é obtido como o feedback para ajustar o valor dos pesos da rede, com o intuito de diminuir o valor obtido pela loss function (optimizer).

O processo é chamado de backpropagation.

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Deep Learning

No início, como os pesos são iniciados aleatoriamente, o valor da loss é muito alto.

Com os ajustes dos pesos de pouco em pouco, a loss tende a cair.

Iniciando com redes Neurais

Anatomia de uma rede neural

Layers (camadas): são combinadas dentro de uma rede (modelo)

Loss function, que definie o valor de feedback usando para o aprendizado

Optimizer, que determina como o aprendizado ocorre

Como uma rede manipula um vetor?

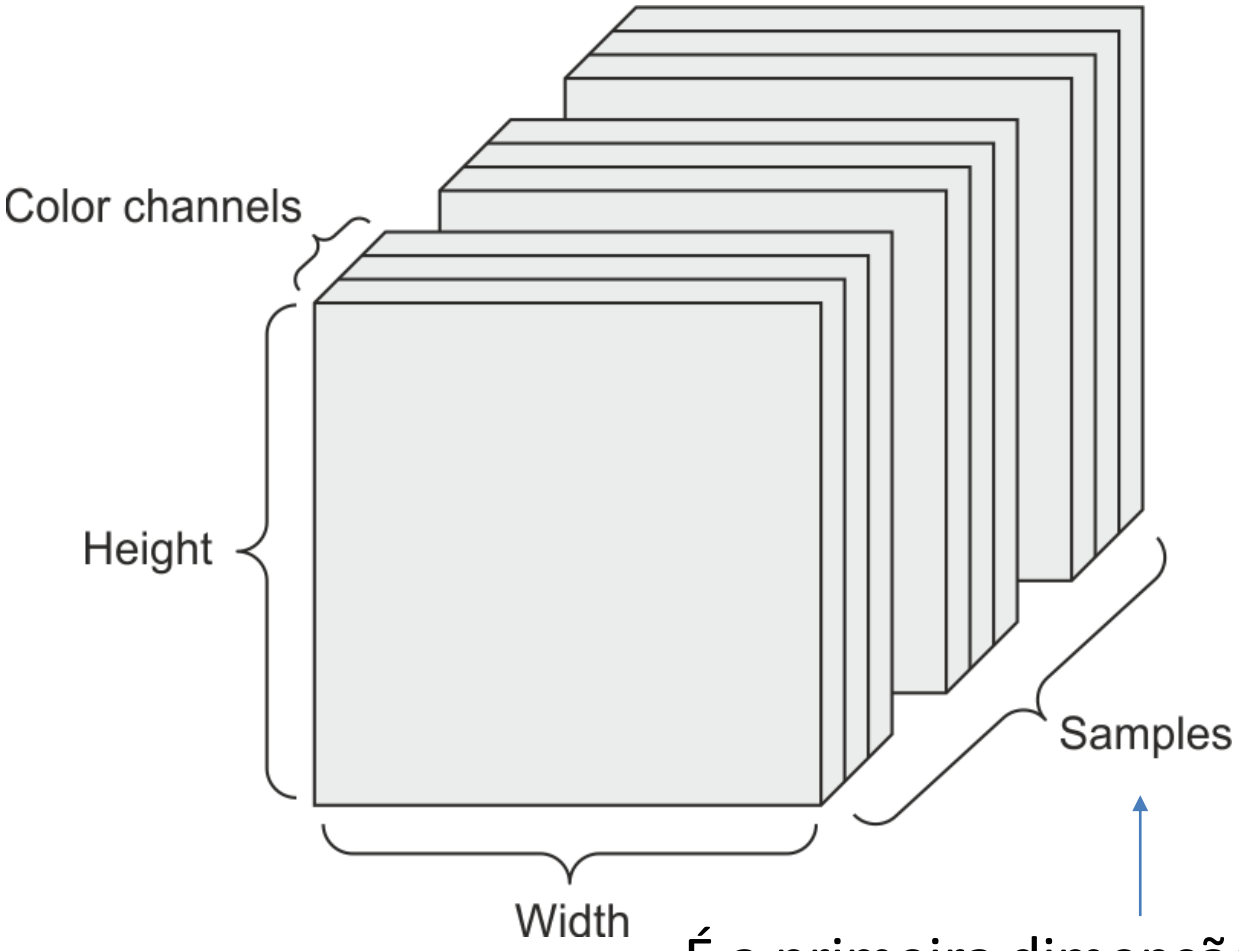
Por meio de um tensor 2D (um array de vetores) onde o primeiro axis (nome da primeira dimensão de um tensor) é chamado de *sample axis* e o segundo axis é chamado de *feature axis.*

Por exemplo, um dataset de pessoas com idade, cpf e nome. Cada pessoa seria representada como um vetor de 3 valores. Para um dataset de 100.000 pessoas, um tensor 2D teria a seguinte forma (100000, 3)

É a primeira dimensão de um tensor, também chamado de batch.

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Como uma rede manipula a imagem?



Por meio de um tensor 4D (samples, height, width, channels)

Batch de 128 imagens (grayscale) de dimensões 256 x 256: tensor shape (128, 256, 256, 1)

Para o mesmo batch de imagem colorida : tensor shape (128, 256, 256, 3)

É a primeira dimensão de um tensor, também chamado de batch.

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Camadas e construção dos blocos

Layers (camadas) é a mais fundamental estrutura de dados de uma rede.

Recebe como entrada um ou mais tensors e retorna um ou mais tensors.

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Camadas e construção dos blocos

Diferentes tipos de camadas são usadas para diferentes tipos de operações.

Para um vetor, armazenado em um tensor 2D, normalmente utiliza-se uma **fully connected** or **dense layers** (`Dense` no `Keras`).

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Camadas e construção dos blocos

Imagens normalmente são processadas por uma “2D convolution layers” (Conv2D no Keras)

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Camadas e construção dos blocos

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(32, input_shape=(784,)))
model.add(layers.Dense(32))
```

Layer que recebe um tensor 2D com a primeira dimensão de 784. a camada irá retornar um tensor com a primeira dimensão transformada para 32

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Camadas e construção dos blocos

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(32, input_shape=(784,)))
model.add(layers.Dense(32))
```

Receberá como entrada a saída da camada anterior

No Keras basta adicionar um linha após a outra para construir a entrada e saída de uma camada.

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Introdução ao Keras

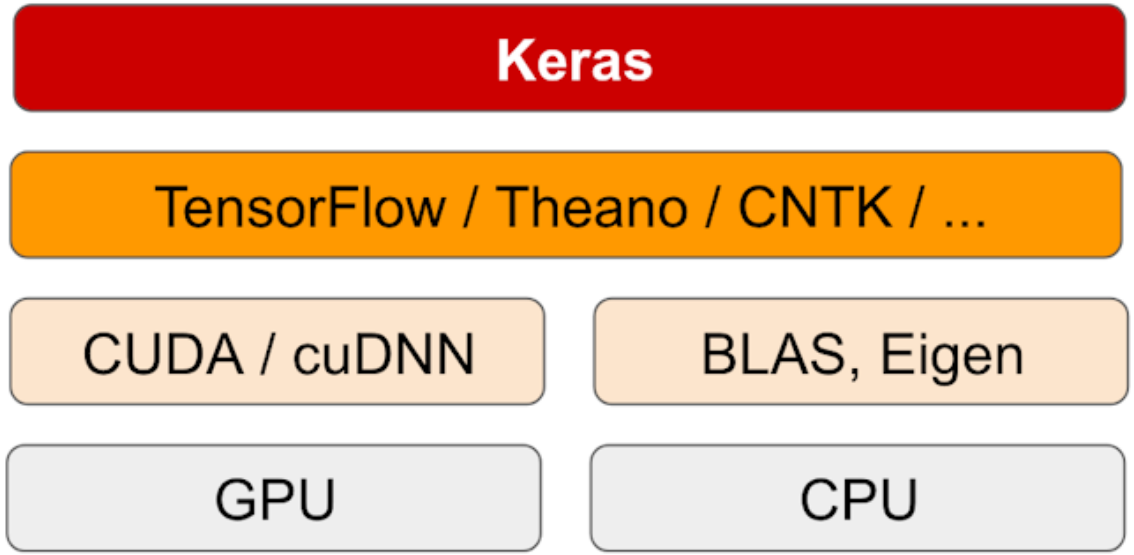
Keras é um framework para deep learning em Python.

Keras é uma biblioteca alto nível. Keras não manipula operação baixo nível como operações sobre tensor. Para isso, utiliza um backend engine: TensorFlow backend, the Theano backend, and the Microsoft Cognitive Toolkit (CNTK) backend

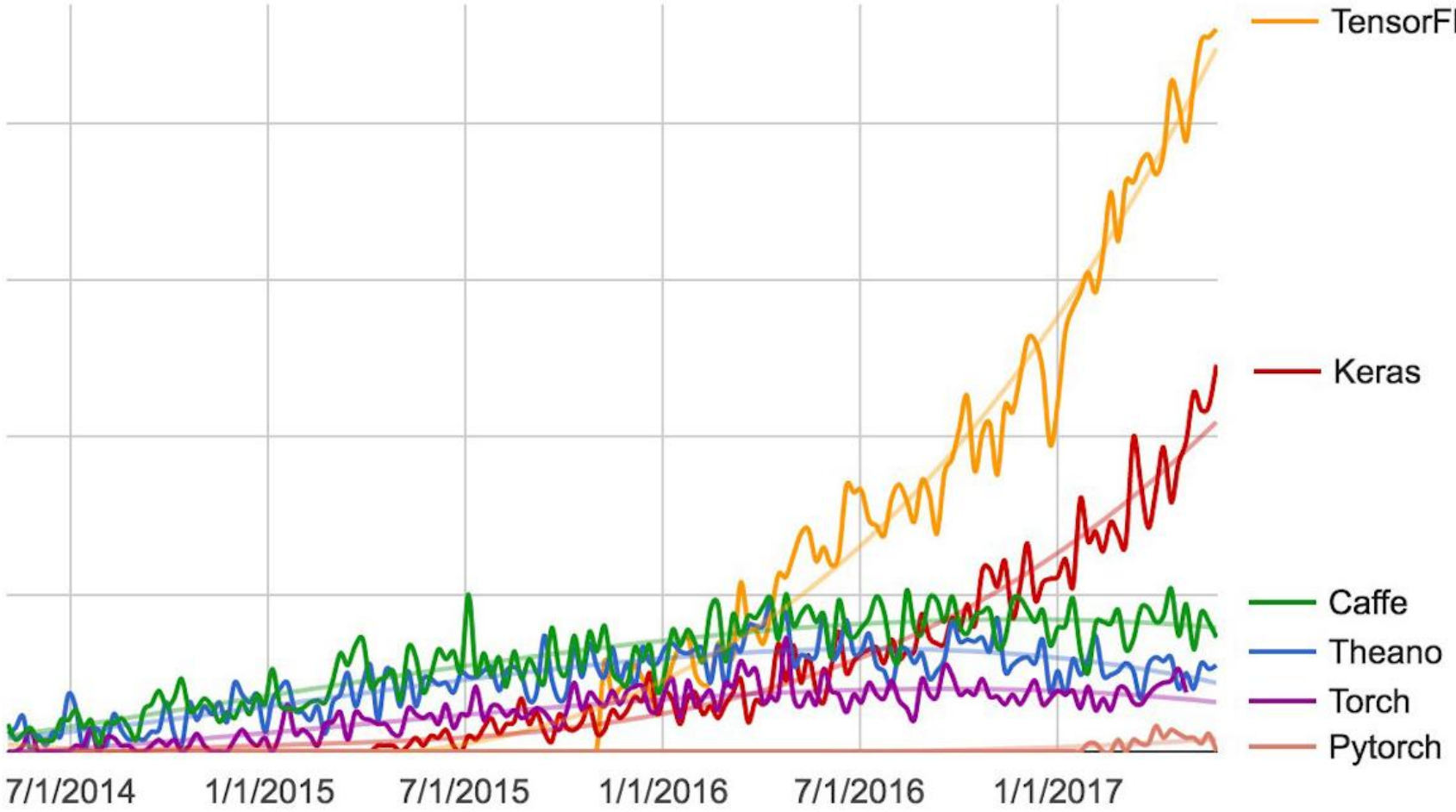
Fonte: François Chollet. **Deep Learning with Python.** November 2017

Introdução ao Keras

Deep learning software stack.



Fonte: François Chollet. **Deep Learning with Python.** November 2017



Fonte: François Chollet. **Deep Learning with Python.** November 2017

Desenvolvendo com Keras

Típico Keras workflow:

1. Define dataset de treinamento: input tensor e target tensor
2. Define a network layer (model) que mapeia os inputs para os targets
3. Configura o processo de aprendizado escolhendo a loss function, um optimizer, e algumas métricas para monitoramento

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Desenvolvendo com Keras

Típico Keras workflow:

4. Iterações com o dataset de treinamento chamando o método fit() do seu modelo

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Desenvolvendo com Keras

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))
```

Definição de um modelo

ReLU: função de ativação que transforma os pesos de entrada e retorna uma saída. ReLU retorna a entrada diretamente, se esta for positiva, caso contrário, retorna zero.

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Desenvolvendo com Keras

No processo de aprendizado especificamos na etapa de compilação a loss function e o optimizer.

Divide a learning rate (lr) pelo peso

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
loss='mse',
metrics=['accuracy'])
```

Mean Squared Error (retorna um não negativo floating point – melhor valor é zero)

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Desenvolvendo com Keras

Finalmente, passamos um array de input para o modelo via fit()

```
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

Fonte: François Chollet. **Deep Learning with Python.** November 2017

Classificação de reviews de filmes

PUC Minas

The IMDB dataset

Neste exemplo você irá construir um classificador binário

"IMDB dataset: a set of 50,000 reviews from the Internet Movie Database. They're split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting of 50% negative and 50% positive reviews"

Fonte: François Chollet. **Deep Learning with Python.** November 2017

The IMDB dataset

O imdb dataset já vem com o Keras e já é pré-processado.

```
from keras.datasets import imdb

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)
```


As 10000 palavras mais frequentes serão mantidas no dataset

Fonte: François Chollet. **Deep Learning with Python.** November 2017

The IMDB dataset

O imdb dataset já vem com o Keras e já é pré-processado.

```
from keras.datasets import imdb

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)
```

Train_data e test_data são as listas de reviews, cada review é uma lista de índices de palavras

```
print(train_data[0])
```

The IMDB dataset

O imdb dataset já vem com o Keras e já é pré-processado.

```
from keras.datasets import imdb

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)
```

Train_labels e test_labels são as listas de 0's e 1's (0 = negativo review e 1 = positivo review)

```
print(train_labels[0])
```

The IMDB dataset

O imdb dataset já vem com o Keras e já é pré-processado.

```
from keras.datasets import imdb

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)
```

Veja que nenhum índice de palavra irá exceder 10000

```
max([max(sequence) for sequence in train_data])
```

Preparando o dataset

Não podemos inserir um lista de inteiros na rede. Temos que converter a listar para um tensor no formato (samples, word_indices)

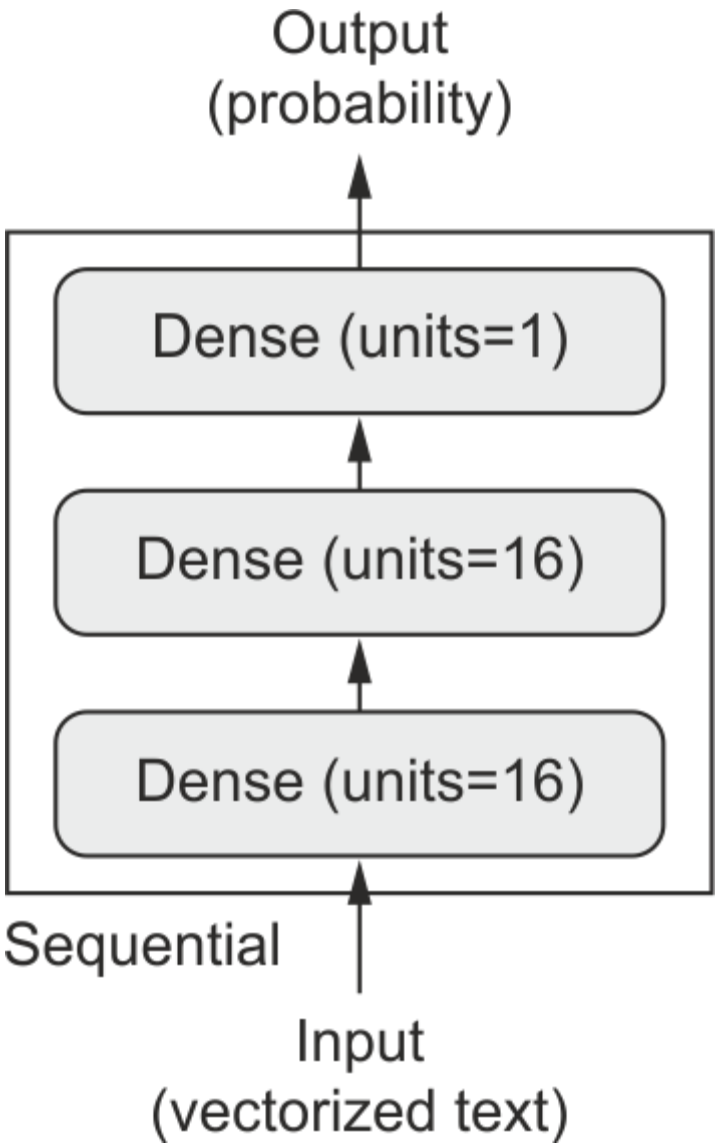
```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
results = np.zeros((len(sequences), dimension))
for i, sequence in enumerate(sequences):
results[i, sequence] = 1.
return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Preparando o dataset

O mesmo para os vetores de labels

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Construindo a rede



O input é um vetor e os labels são escalares (1's e 0's). Vamos usar um fully connected (Dense) com relu como activation function.

Dense(16, activation='relu')

16: é o argumento para cada dense layer. Ou seja, é o número de hidden units da camada (é a dimensão da representação da camada).

Construindo a rede

Configurando a rede

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Construindo a rede

Escolhendo a loss function e o optimizer (como strings por já fazerem parte do Keras)

```
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
loss=losses.binary_crossentropy,
metrics=[metrics.binary_accuracy])
```

Validando

Para monitorar o treinamento, criamos o conjunto de validação

separando 10000 amostras do conjunto original

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Construindo a rede

No entanto, você pode deixar tudo como padrão:

```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
```

Construindo a rede

Iremos treinar o modelo por 20 épocas (20 iterações sobre todas as amostras do treinamento) em um batch de 512 amostras.

```
model.fit(partial_x_train,
partial_y_train,
epochs=20,
batch_size=512,
validation_data=(x_val, y_val))
```

Treinando o modelo do zero

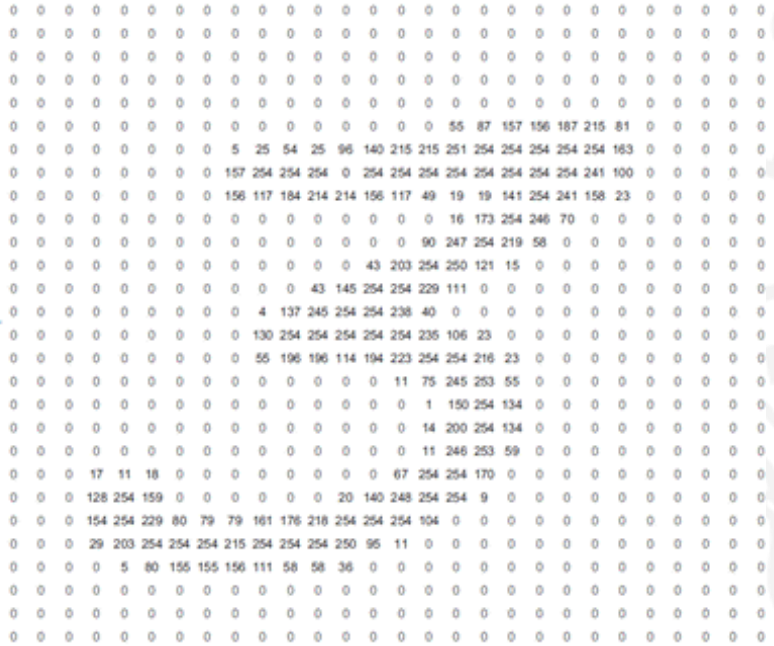
```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

Redes Neurais Convolucionais

Como computador vê uma imagem

Imagem 28 x 28:

28 x 28
= 784 pixels

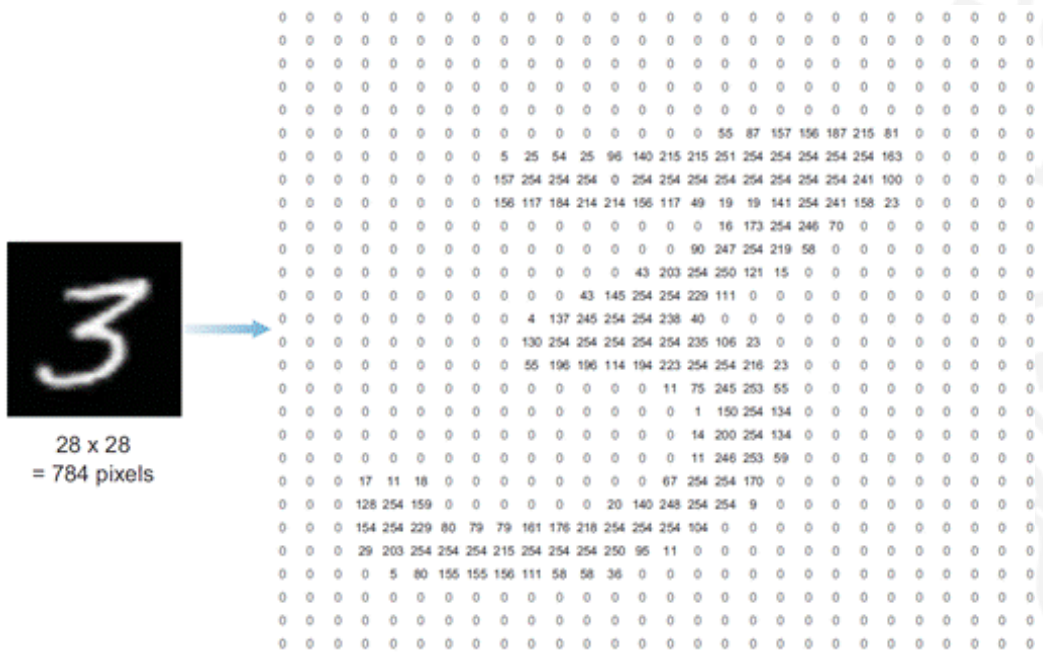


Essa imagem é vista pelo computador como uma matriz 28x28 com valores de pixel que variam de 0 a 255

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Como computador vê uma imagem

Imagem 28 x 28:



28 x 28
= 784 pixels

Uma simples rede neural aceita apenas a entrada como um vetor 1D com dimensões (1, n) (flatten image), ela não podem assumir a matriz de imagem 2D bruta com dimensões (x, y). Sem representação 2D, perdemos informação espacial

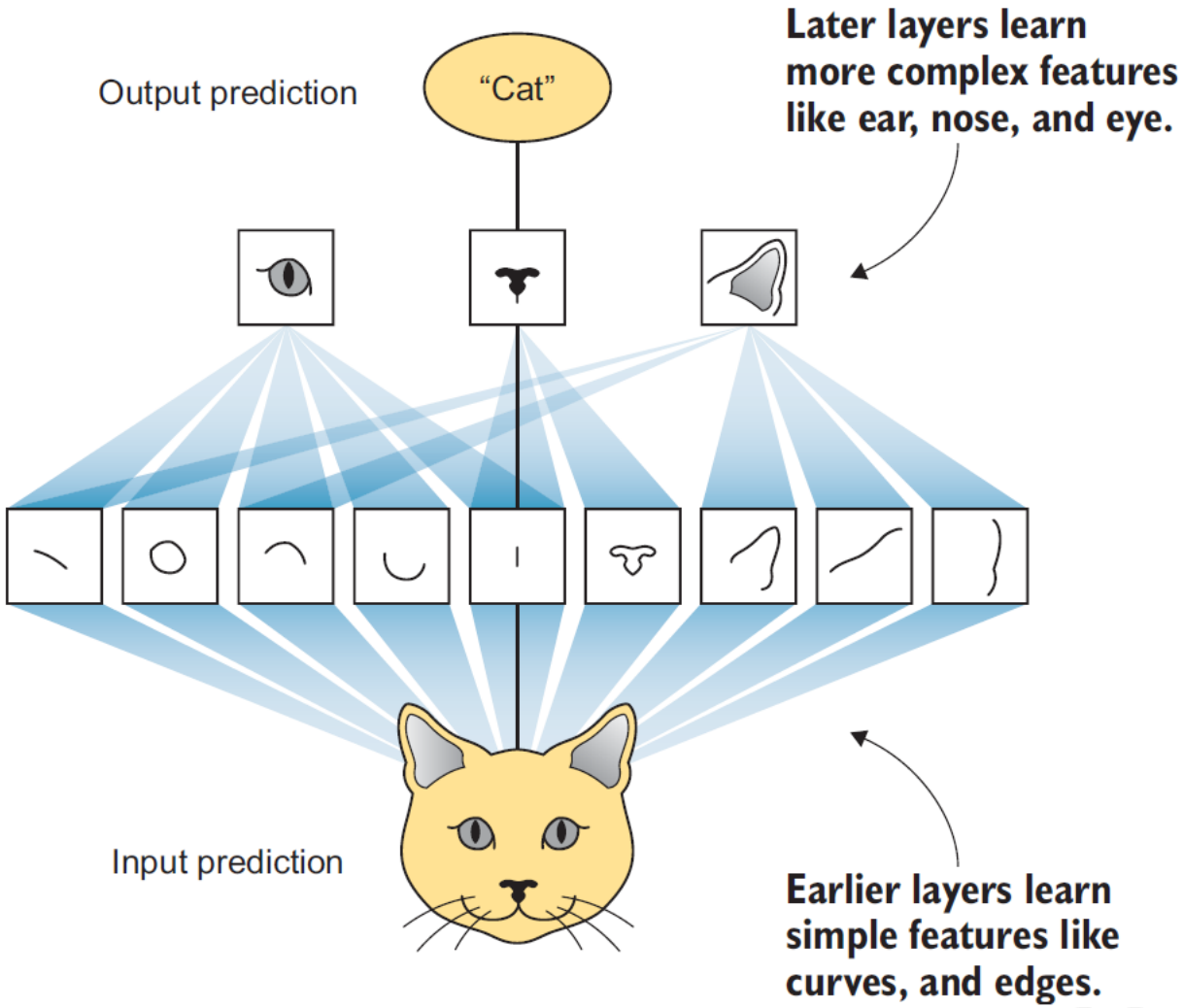
Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Como computador vê uma imagem

Para ensinar a rede a reconhecer gatos, o ideal é que a rede neural aprenda a forma completa das features do gato, independentemente de onde eles apareçam na imagem (orelhas, nariz, olhos etc.). Isso só acontece quando a rede olha para a imagem como um conjunto de pixels que, quando próximos um dos outros, estão fortemente relacionados.

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Como computador vê uma imagem



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolution Neural Networks

As redes neurais são compostas de camadas densas (Fully Connected – FC: todos os nós de uma camada estão conectados a TODOS os nós da camada anterior e a todos os nós da próxima camada).

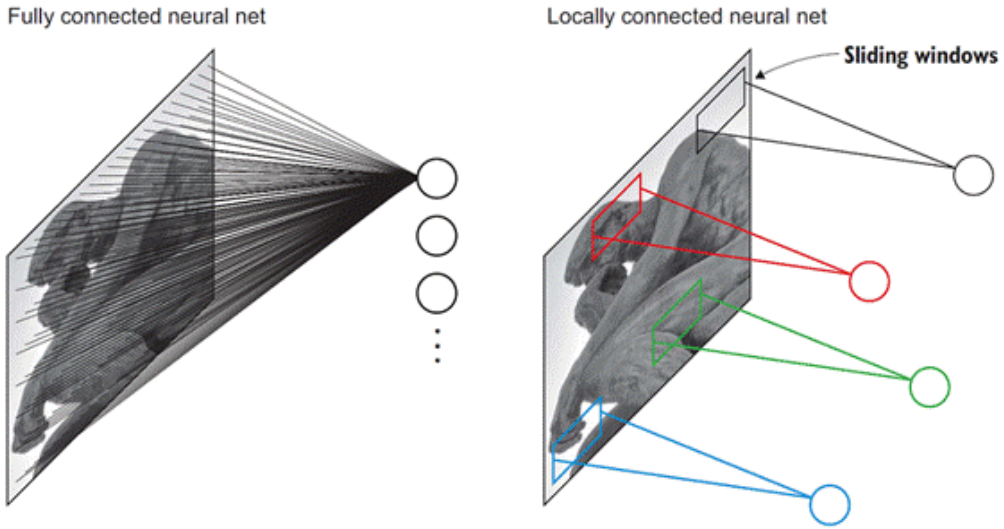
Assim, se tivermos uma imagem com dimensões = (1.000 x 1.000), ela produzirá um milhão de parâmetros para cada nó único na primeira camada oculta.

Portanto, se a primeira camada oculta tiver 1.000 neurônios, isso produzirá 1 bilhão de parâmetros em uma rede tão pequena.

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolution Neural Networks

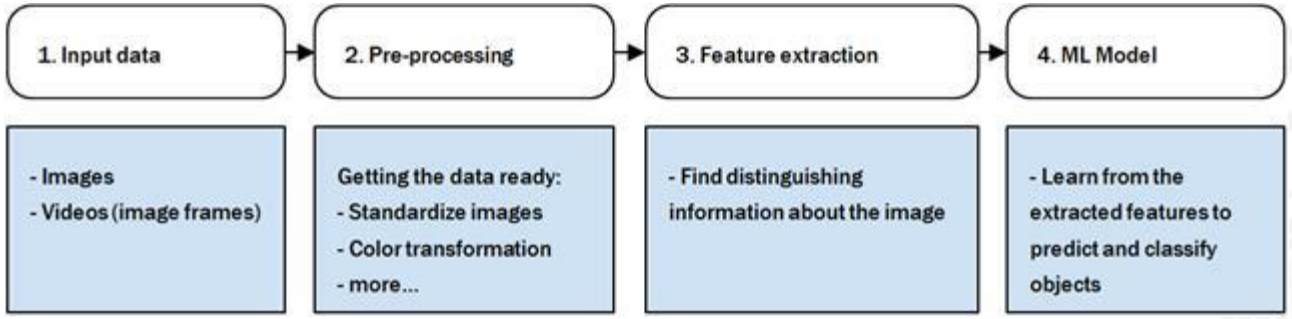
As CNNs (Convolution Neural Networks), por outro lado, são camadas conectadas localmente. Onde há nós conectados a apenas um pequeno subconjunto dos nós das camadas anteriores. Camadas conectadas localmente usam muito menos parâmetros do que uma camada densamente conectada.



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolution Neural Networks

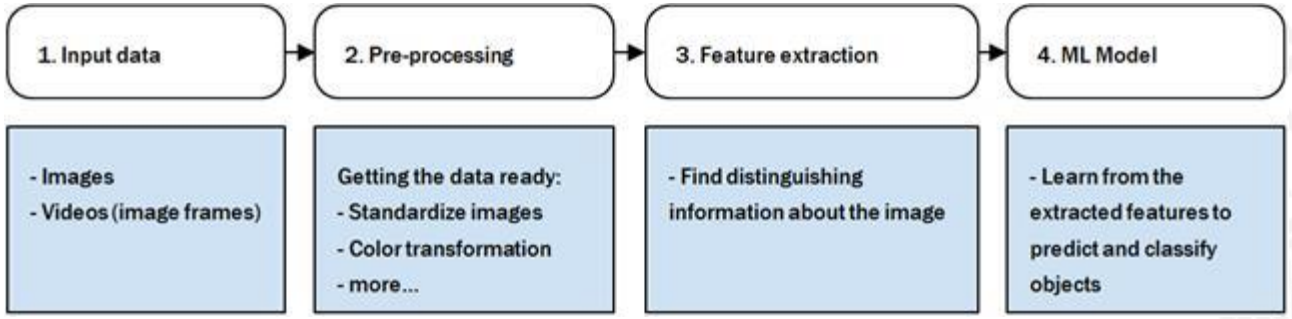
Antes de Deep Learning, a extração de features das imagens era feita manualmente para depois serem usadas em um classificador classificá-las. Com as redes neurais, substituiremos o trabalho manual da etapa 3 (figura abaixo) por uma rede neural que fará o aprendizado e a classificação de features (etapas 3 e 4).



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolution Neural Networks

Fully Connected são ótimas para etapa de classificação. Mas não são muito práticas para as etapas de extração de features (etapa 3). Nesta etapa, podemos usar camadas conectadas localmente (locally connected layers - camadas convolucionais), mantendo as FC na etapa de classificação.



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolution Neural Networks

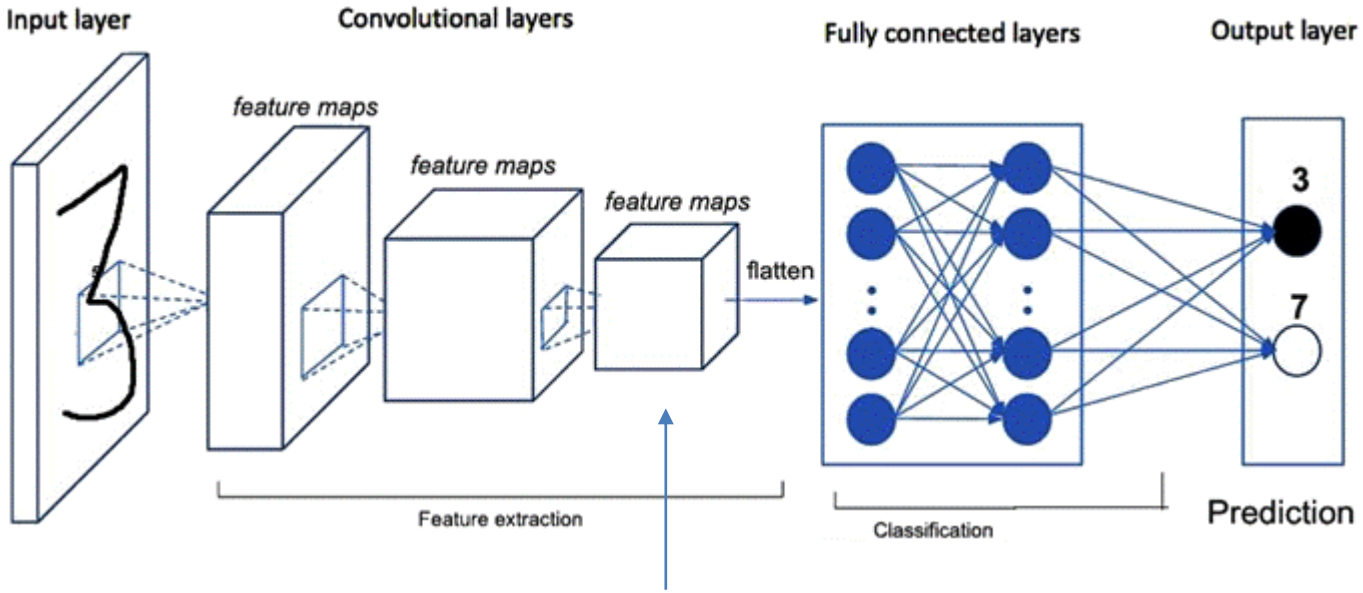
Dito isto, a arquitetura de alto nível das CNNs ficará assim:

- Input layer
- Convolutional layers para extração de features
- Fully connected layers para classificação
- Output prediction (predição de saída)

Veja imagem no a seguir

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolution Neural Networks



as dimensões da imagem diminuem após cada camada

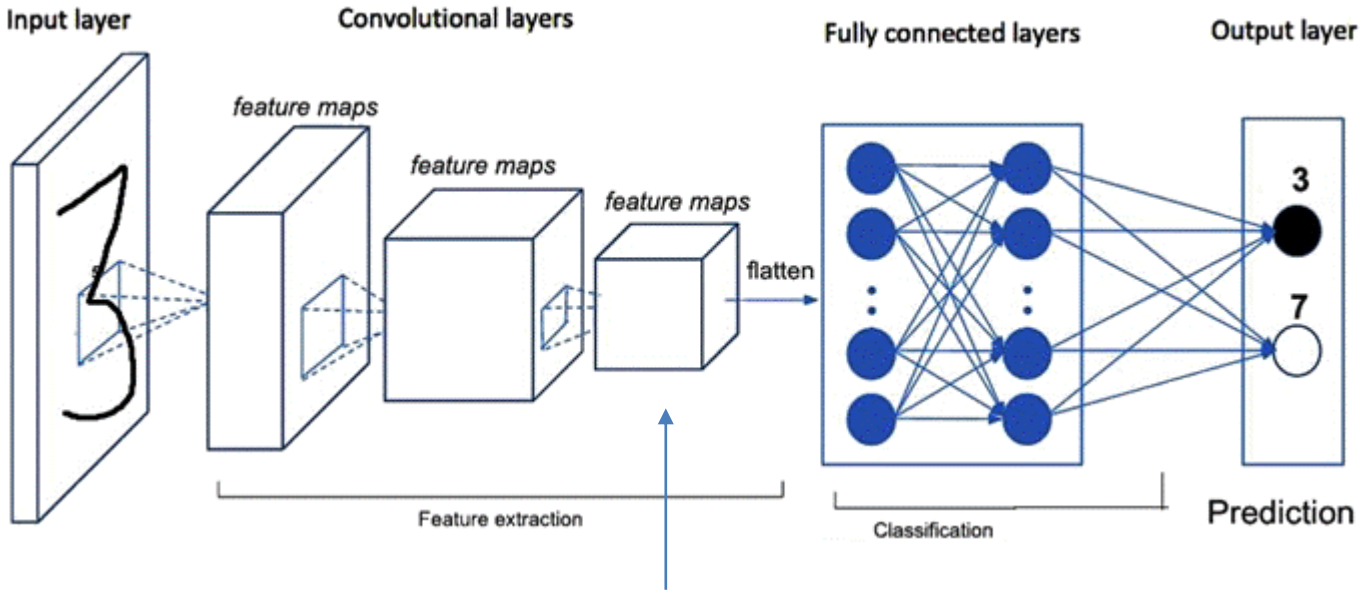
Suponha que estamos construindo uma CNN para classificar entre imagens de duas classes: o número 3 e 7.

1) Primeiro passamos uma imagem como entrada para as camadas convolucionais.

2) A imagem passa pelas camadas da CNN para detectar padrões e extrair features chamados **feature maps**. A saída desta etapa é então transformada em um vetor de features aprendidos da imagem.

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolution Neural Networks



as dimensões da imagem diminuem após cada camada

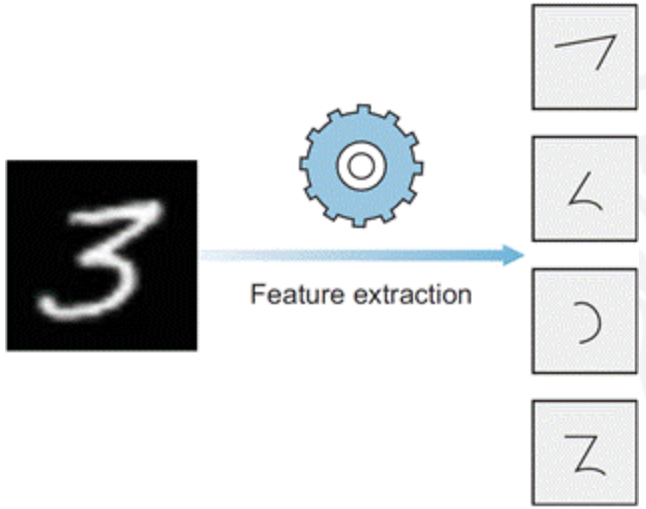
3) O vetor de features é passado como entrada para as camadas Fully Connected (FC) para classificar as features extraídos da imagem.

4) Observe que neste exemplo estamos classificando duas classes (3 e 7), então a camada de saída terá dois nós. Um para representar o dígito 3 e um para 7.

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Extração de features

Por exemplo, uma imagem do dígito “3” é uma imagem (profundidade = 1) será dividida em imagens menores que contêm features específicas do dígito 3. Se estiver dividida em 4 features, a profundidade = 4. Portanto, à medida em que a imagem passa pelas camadas da CNN, ela diminui de dimensão e fica mais profunda porque contém mais "imagens" de pequenos features.



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Classificação

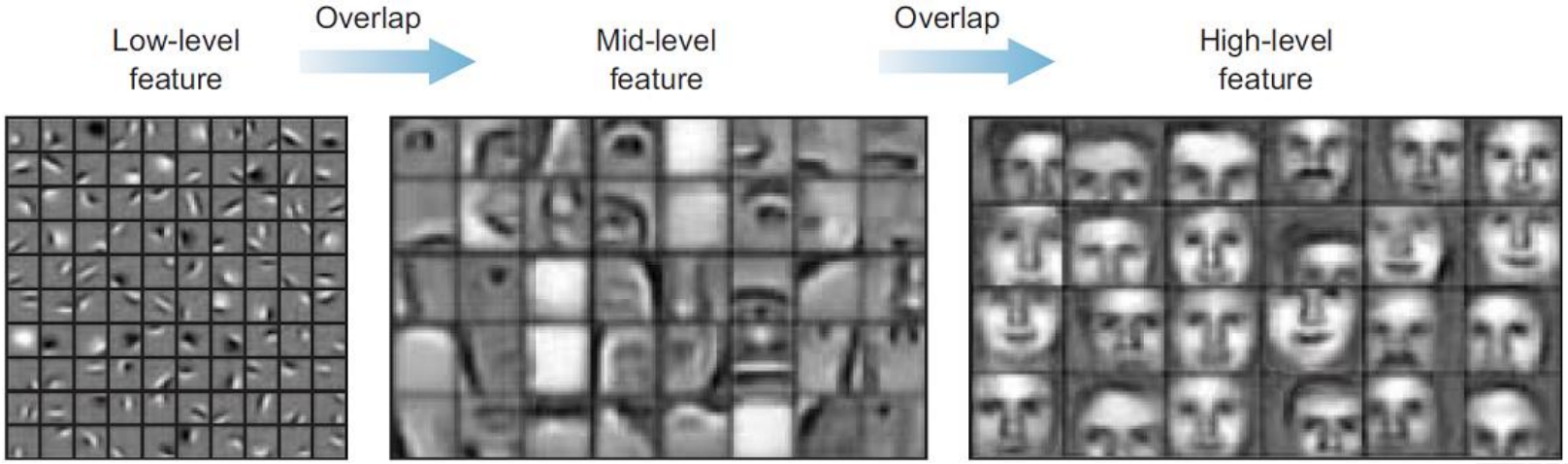
Nas CNNs, a primeira camada geralmente aprende recursos muito básicos, como linhas e arestas, e a segunda reúne essas linhas para reconhecer formas, cantos e círculos.

Em seguida, nas camadas mais profundas da rede, aprende formas mais complexas, como mãos, olhos, ouvidos, etc. Por exemplo, aqui está uma versão simplificada de como as CNNs aprendem as faces:

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Classificação

Por exemplo, uma versão simplificada de como as CNNs aprendem as faces:



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Componentes básicos de uma CNN

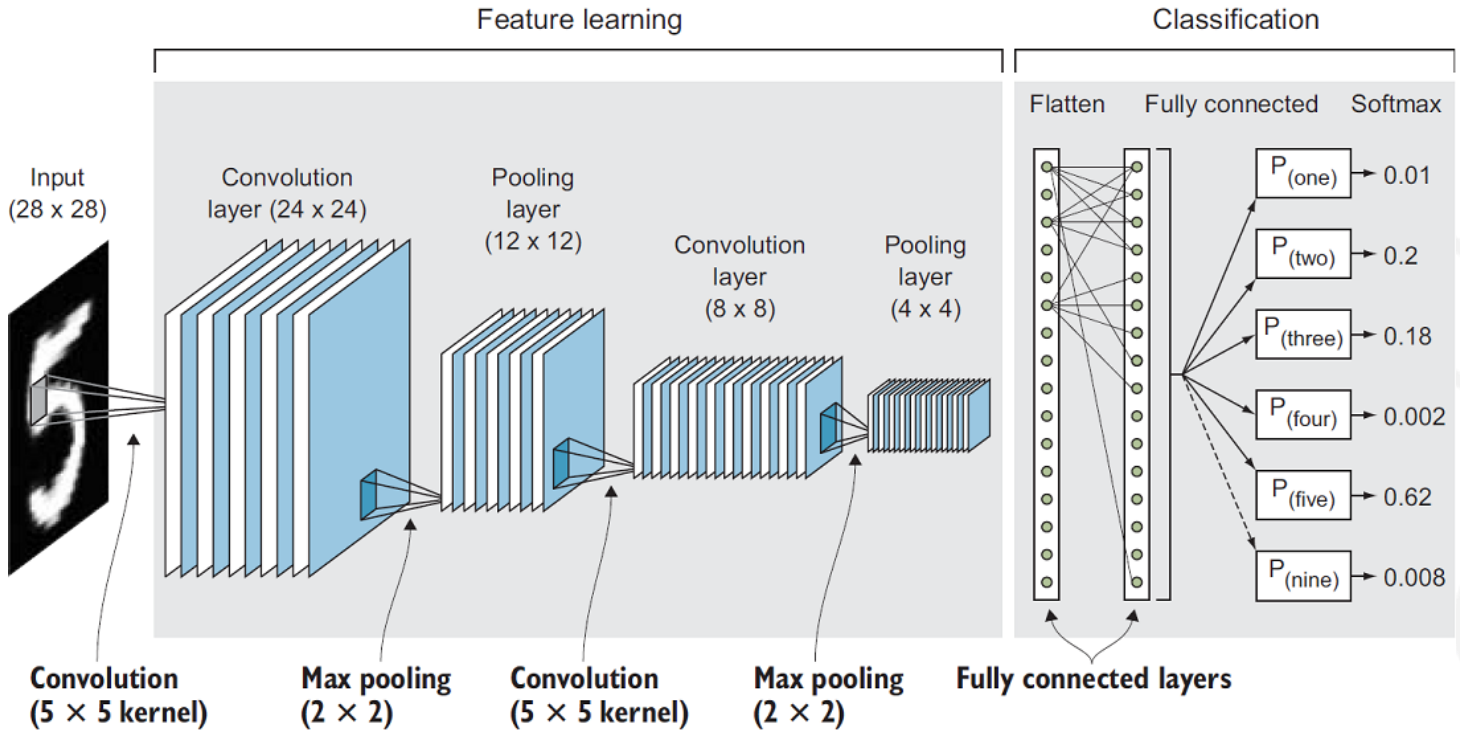
Componentes básicos de uma CNN

Existem três tipos principais de camadas que você verá em quase todas as redes convolucionais:

1. Convolutional layer [CONV]
2. Pooling layer [POOL]
3. Fully connected layer [FC]

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Componentes básicos de uma CNN

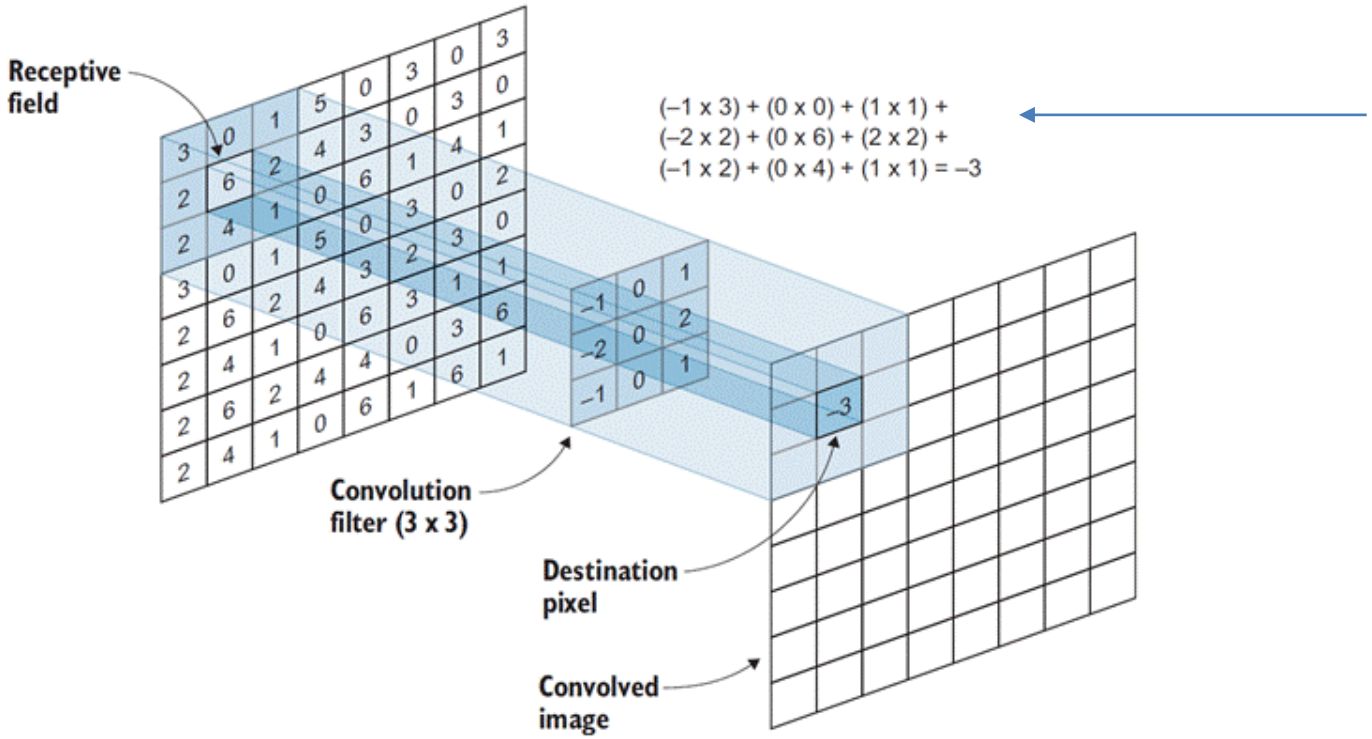


Vamos ampliar a primeira camada convolucional para ver como ela processa uma imagem (próximo slide)

PUC Minas

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolutional layer



O filtro convolucional é aplicado em uma janela da imagem, produzindo uma saída na imagem convolucionada.

PUC Minas

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolutional layer

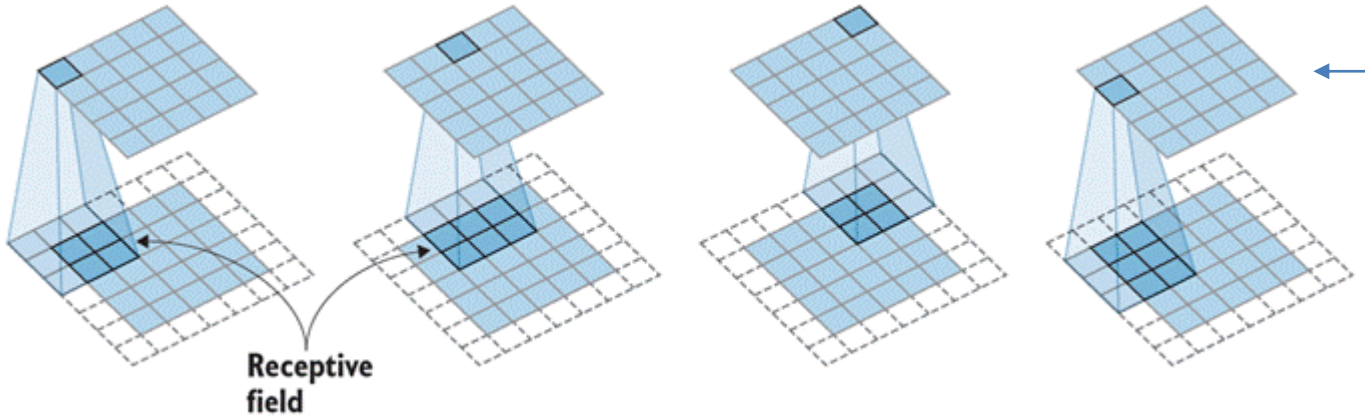


Imagem convolucionada

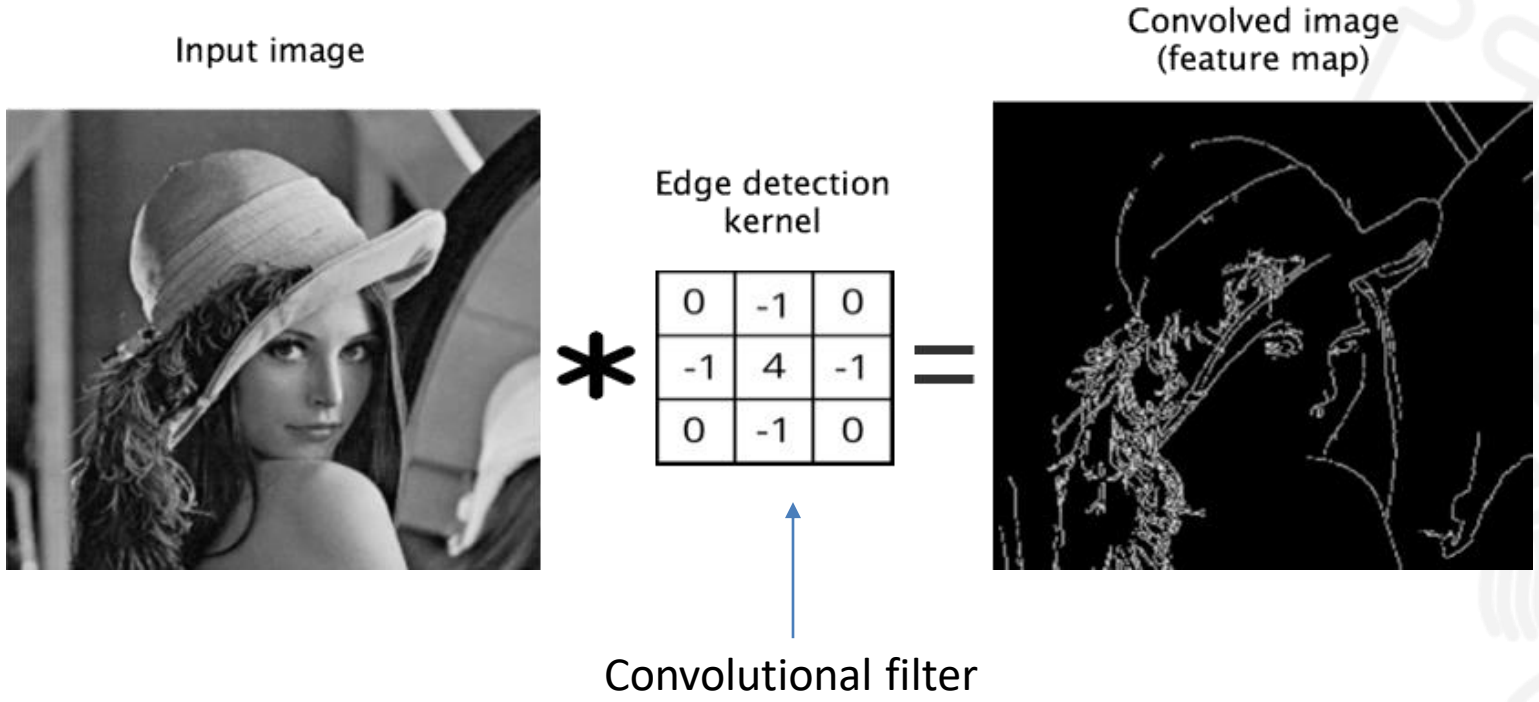
Imagem de entrada

Processo de janela deslizante na imagem para processar toda a imagem.

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Convolutional layer

Aplicando filtros para aprender features



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Pooling Layer

Após adicionar várias camadas CONV (dezenas ou centenas), isso produzirá um grande número de parâmetros (pesos).

Esse aumento na dimensionalidade da rede aumentará a complexidade de tempo e espaço das operações matemáticas. É quando as camadas de pool entram.

A etapa de pooling ajuda a reduzir o tamanho da rede, reduzindo o número de parâmetros passados para a próxima camada.

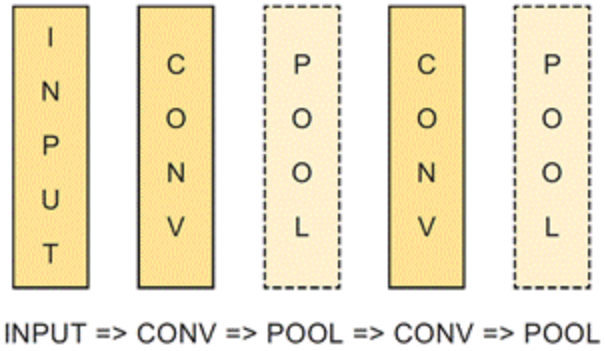
A operação de pool redimensiona sua entrada aplicando uma função estatística (max ou média).

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Pooling Layer

O objetivo da camada de pool é reduzir a amostragem dos features maps produzidos pela camada CONV em um número menor de parâmetros.

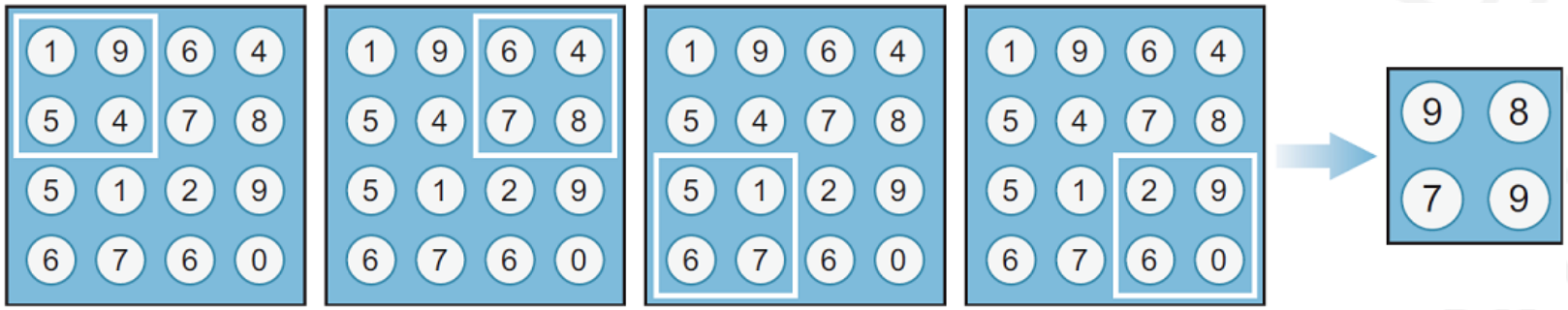
É uma prática comum adicionar camadas POOL após cada uma ou duas camadas CONV na arquitetura CNN



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Pooling Layer

Exemplo: Max pooling



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

CONV layer

```
model.add(Conv2D(filters=16, kernel_size=2, strides='1', padding='same', activation='relu'))
```

filters = número de filtros em cada camada (a profundidade da hidden layer)
kernel_size = o tamanho de cada filtro (também conhecido como kernel). Normalmente 2, 3, ou 5
stride = é a quantidade pela qual o filtro desliza sobre a imagem. (geralmente 1 ou 2)
padding = adiciona colunas e linhas de valores zero ao redor da borda da imagem para reservar o tamanho da imagem na próxima camada
activation = é altamente recomendável usar a função de ativação "relu" nas hidden layers

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

POOL layer

Configuração da camada de pooling

```
model.add(MaxPooling2D(pool_size=(2, 2), strides = 2))
```

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Fully connected layers (FC)

Ótimas para a etapa de classificação (quando já temos os vetores de features calculados pelas CONV layers)

Vetor de entrada: vetor com as dimensões (1, n). Por exemplo, os vetores de featuires tiverem as dimensões de 5x5x40, o vetor de entrada da FC será = (1, 1000).

Camada oculta (FC): uma ou mais camadas FC e cada camada possui 1 ou mais neurônios

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

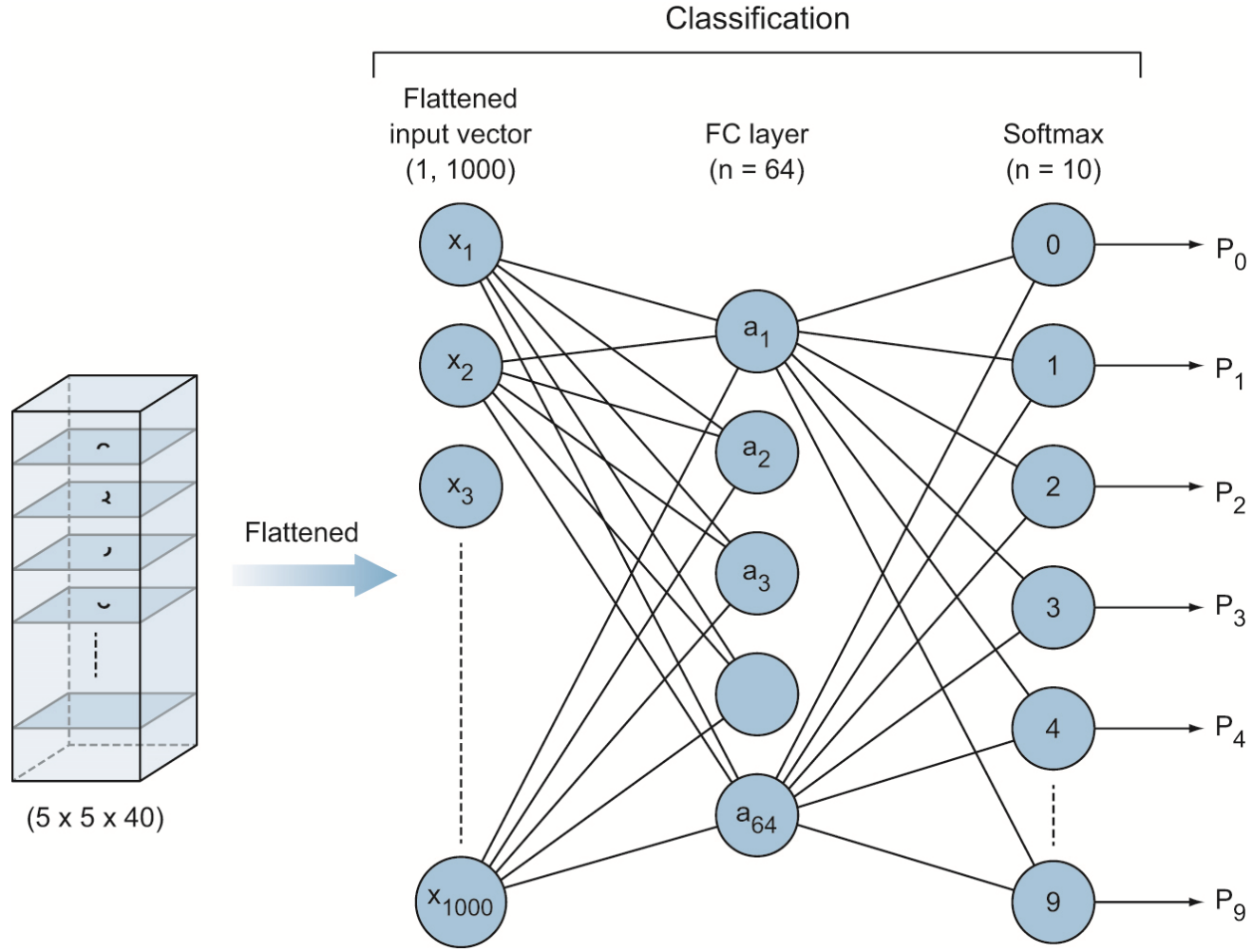
Fully connected layers (FC)

Camada de saída: é recomendável usar a "função de ativação softmax" para problemas de classificação com mais de 2 classes.

Neste exemplo, estamos tentando classificar dígitos de 0 a 9 => 10 classes. E o número de neurônios nas camadas de saída é igual ao número de classes. Então a camada de saída terá 10 nós (veja imagem a seguir).

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Fully connected layers (FC)



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

MNIST Dataset

MNIST Dataset

O MNIST dataset é um dataset de imagens de dígitos escritos a mão. O objetivo é classificar cada dígito.

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Modelo

CONV_1: formato da entrada é (28x28x1). A saída de conv2d_1 é (28x28x32). Com stride = 1 e padding como “same”, as dimensões da imagem de entrada não foram alteradas. A profundidade aumentou para 32 (adicionamos 32 filtros nessa camada). Cada filtro produz um feature map

POOL_1: a entrada é a saída da camada anterior = (28x28x32). Após POOL, as dimensões da imagem diminuem e a profundidade permanece a mesma. Com um pool 2x2, o formato da saída é (14x14x32)

CONV_2: A entrada da camada anterior é (14x14x32). Como os filtros nesta camada estão definidos como 64, a saída = (14x14x64)

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Modelo

POOL_2: 2x2, mantém a profundidade e reduz as dimensões. Saída = (7x7x64)

Flatten: entrada conjunto de features = (7x7x64). Converte em um vetor de dimensões (1, 3136)

Dense_1: camada FC para ter 64 neurônios, então a saída é 64

Dense_2: camada de saída definida para 10 neurônios, pois são 10 classes

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

Dropout layer

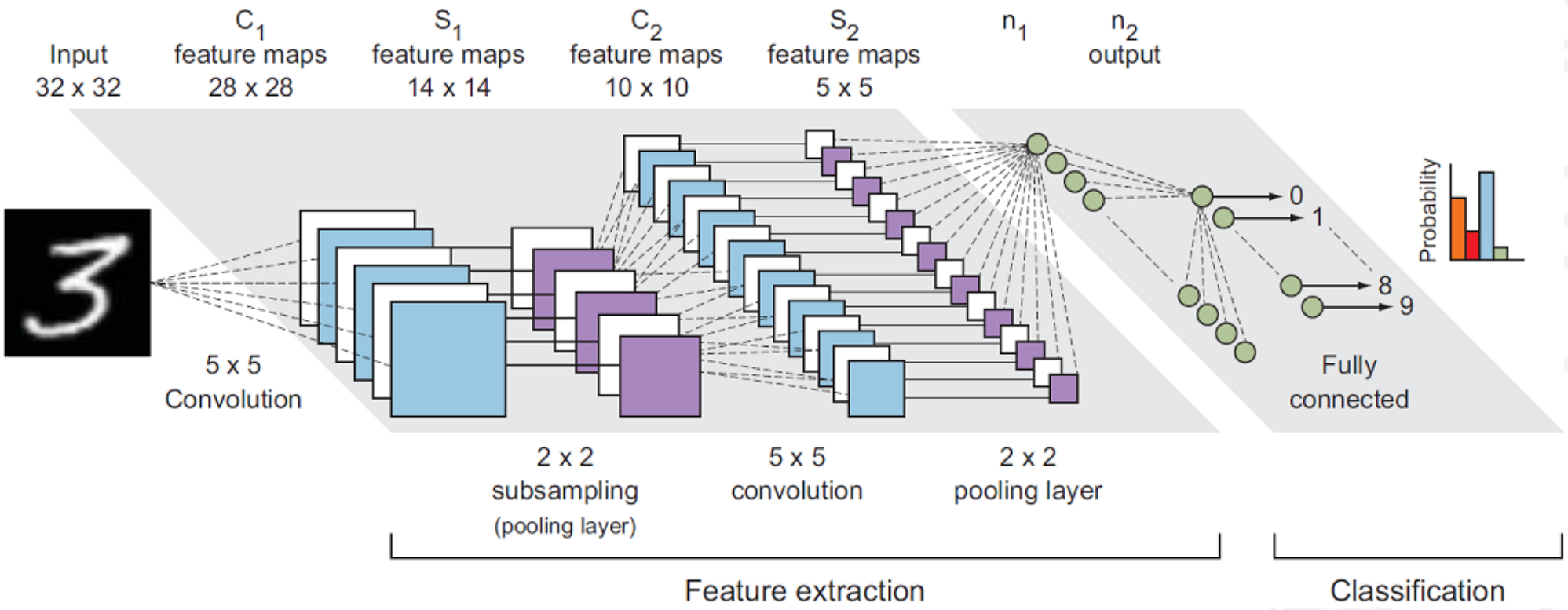
Underfitting: o modelo falha ao ajustar os dados de treinamento. Isso acontece quando o modelo é muito simples.

Overfitting: acontece quando construímos uma super rede que se encaixa perfeitamente no conjunto de dados de, mas falha em generalizar para outras amostras de dados que ele nunca viu antes.

Dropout layer: usada para evitar Overfitting. Ela desativa uma porcentagem de neurônios (nós) que compõem a camada. Essa porcentagem é identificada como um hiperparâmetro que podemos ajustar da rede. Dropout torna a rede resiliente; garantindo que nenhum nó seja muito fraco ou muito forte.

Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

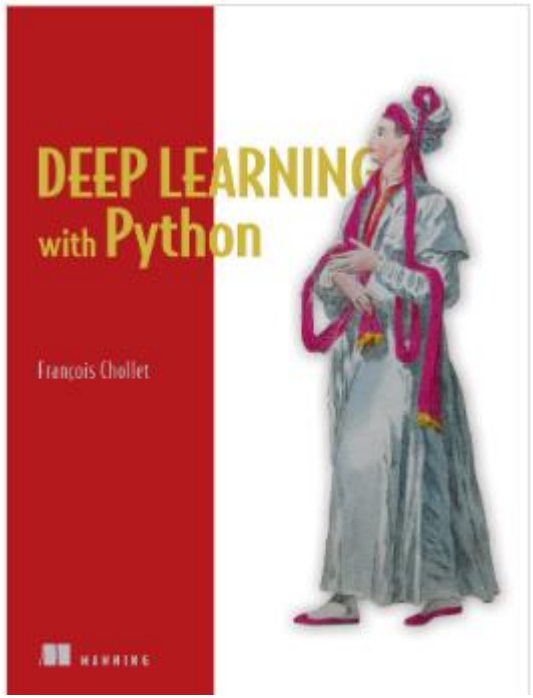
Modelo



Ref.: Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020

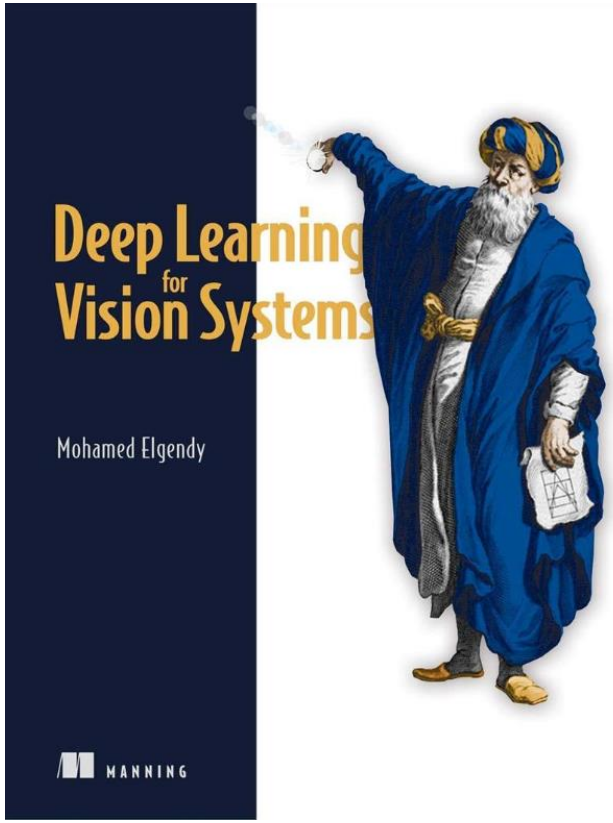
PUC Minas

Principais Referências



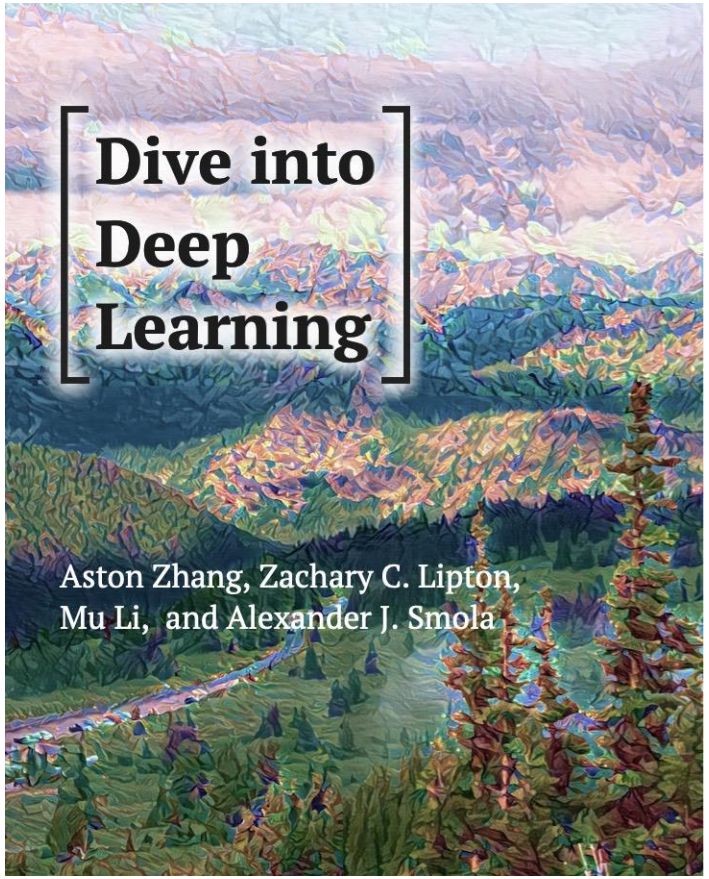
François Chollet. **Deep Learning with Python.** November 2017 ISBN 9781617294433 384 pages

Principais Referências



Mohamed Elgendy. **Deep Learning for Vision Systems**. 2020 (estimated)
ISBN 9781617296192 410 pages

Principais Referências



Aston Zhang; Zack C. Lipton; Mu Li; Alex J. Smola. **Dive into Deep Learning**. http://numpy.d2l.ai/