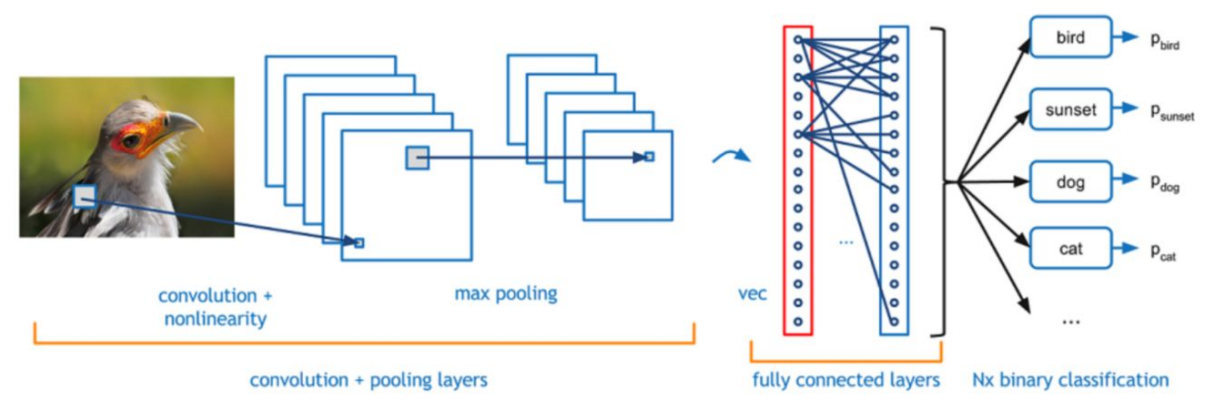
Rede Neural Convolucional (CNN - Convolutional Neural network)

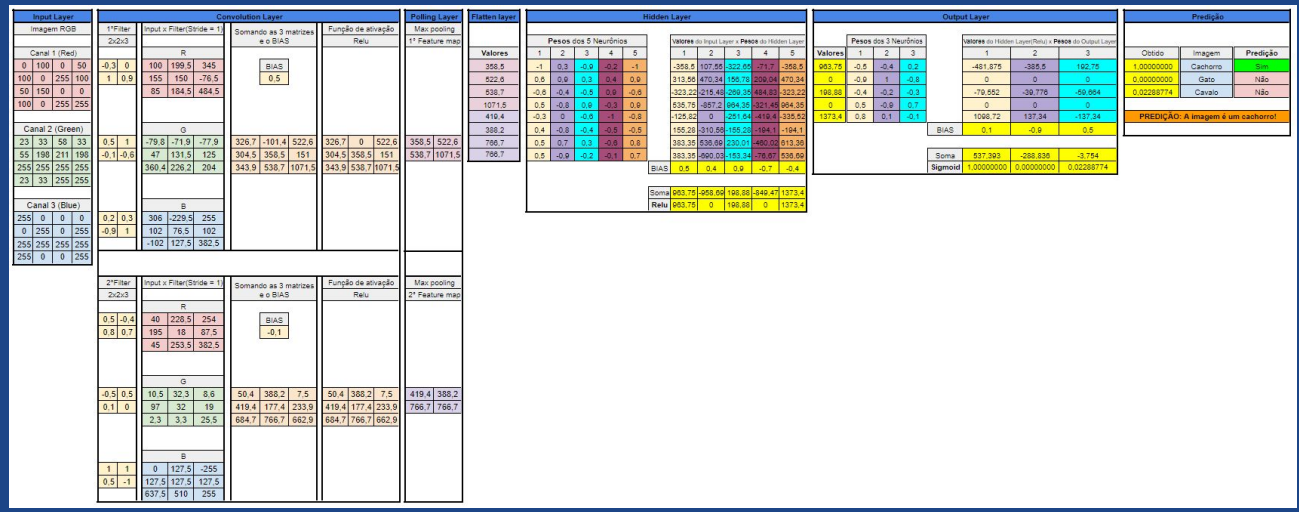
GEIA - Grupo de estudos de inteligência artificial



Fábio Lofredo Cesar

Links para acompanhar a apresentação

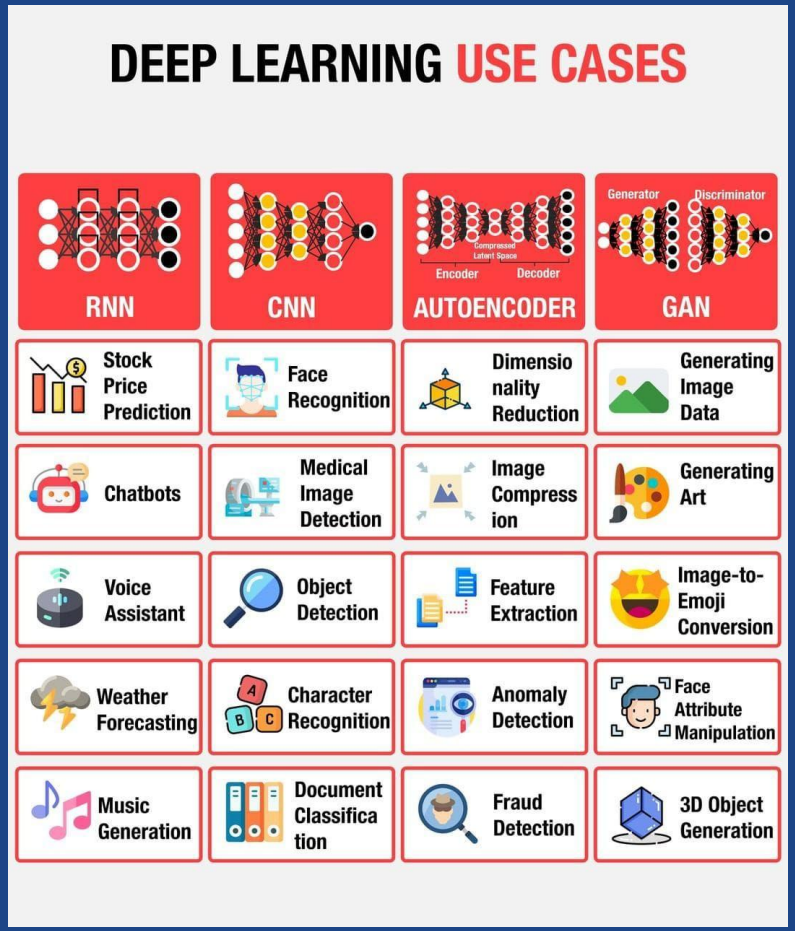
Foi criada uma planilha para acompanhar todo processo de predição de um algoritmo de CNN durante a apresentação, confira nos links para mais detalhes.

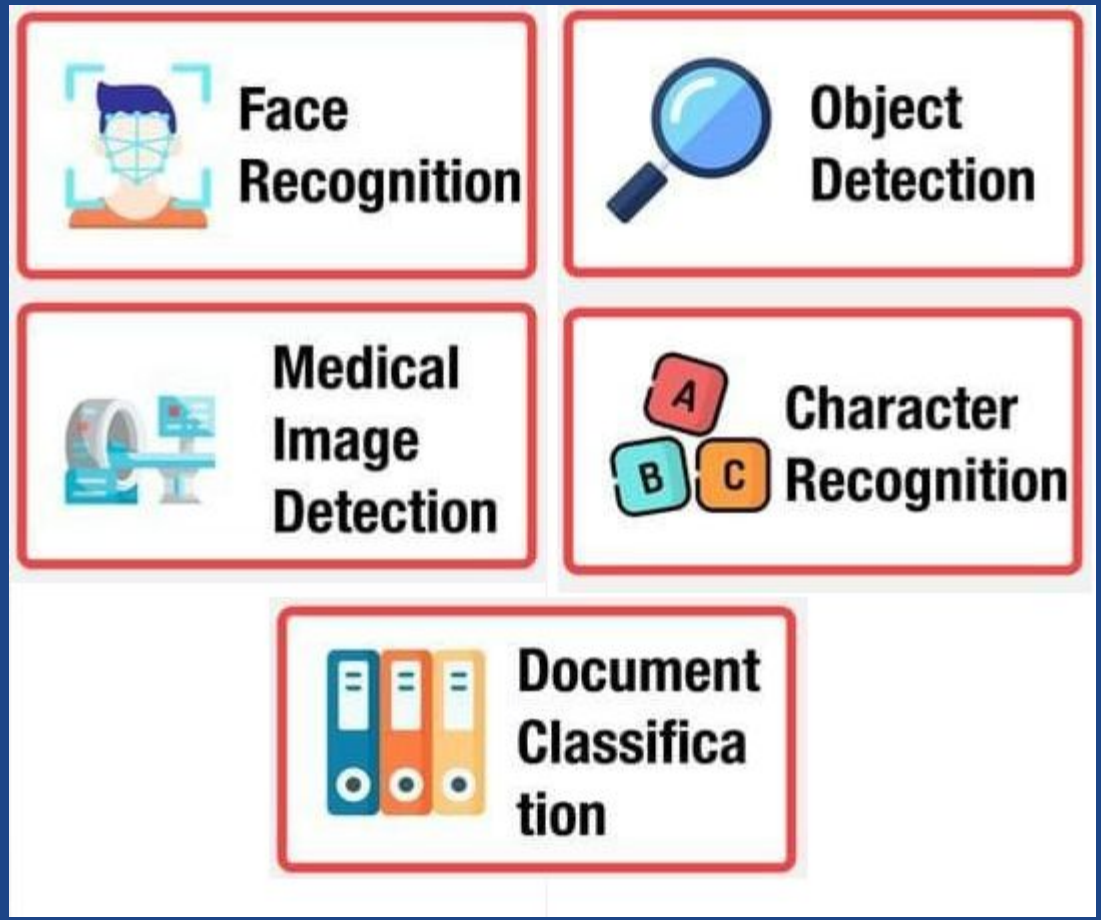


Planilha:
https://github.com/FabioLofredo/CNN-Convolutional-Neural-Network

Simulação:
https://adamharley.com/nn_vis/cnn/2d.html

Aplicações da CNN





Aplicações da CNN

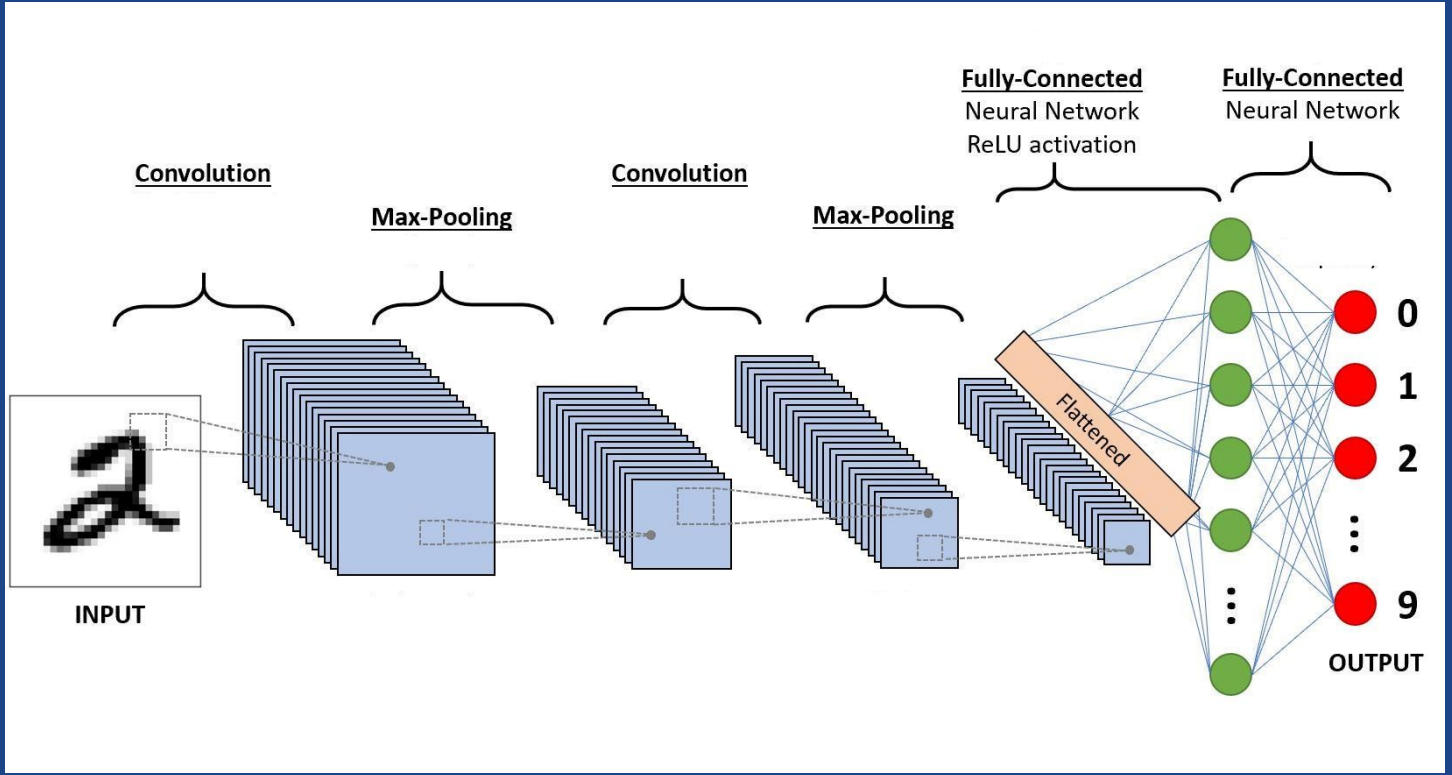
Redes totalmente convolucionais(FCN do inglês) é uma variante das CNN's clássicas.

Classificação da rede Alexnet da imagem de um gato do banco de imagens PASCAL VOC.

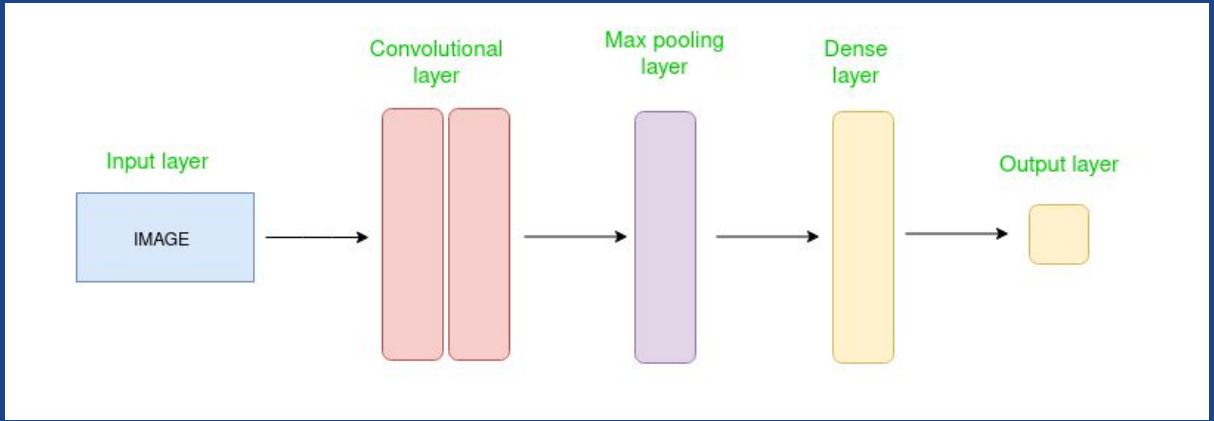
Aplicações da CNN

CNN de dígito numérico

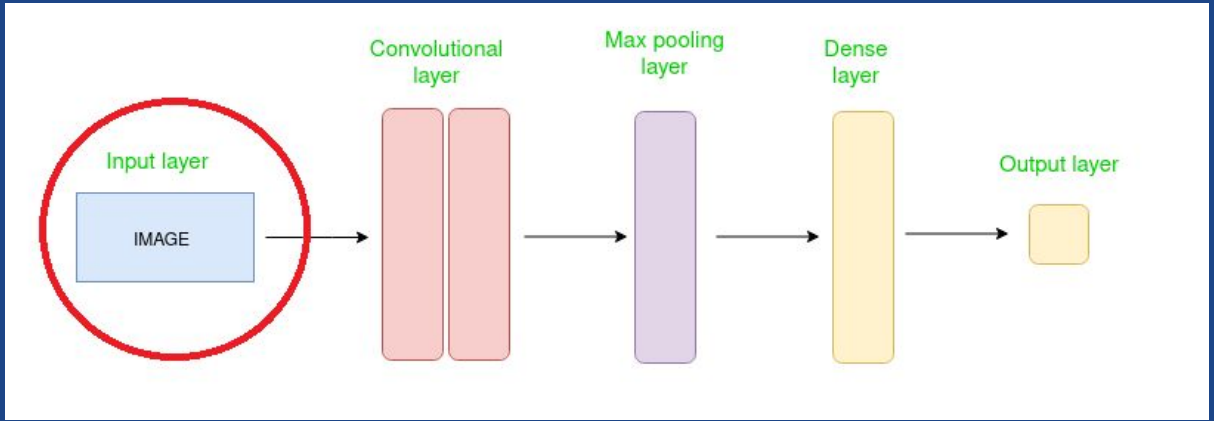
https://adamharley.com/nn_vis/cnn/2d.html



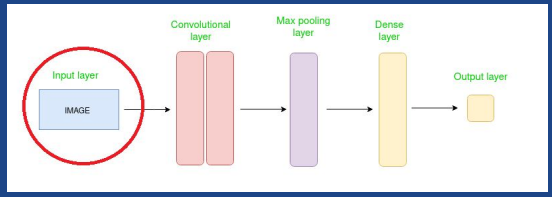
Mapa da apresentação



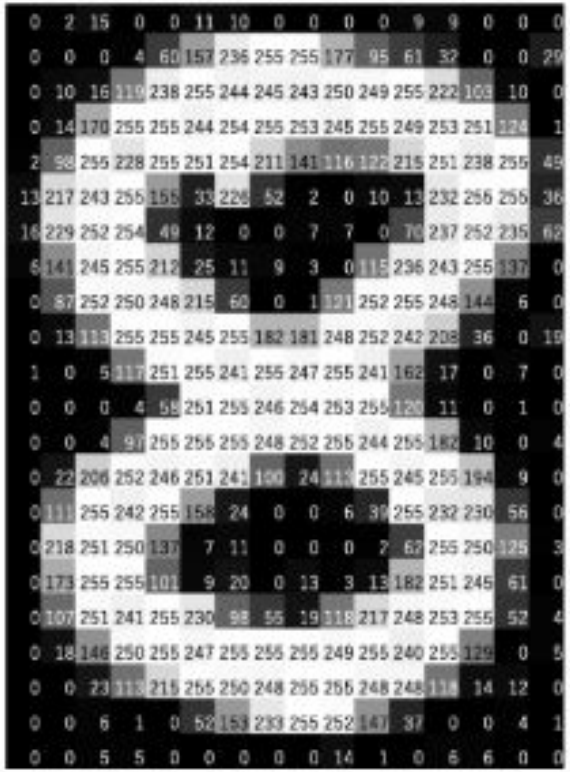
Entrada da Imagem



Entrada da Imagem



Interpretação de uma imagem para a máquina



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	239	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	239	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Entrada da Imagem

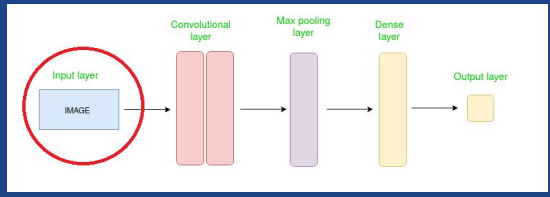
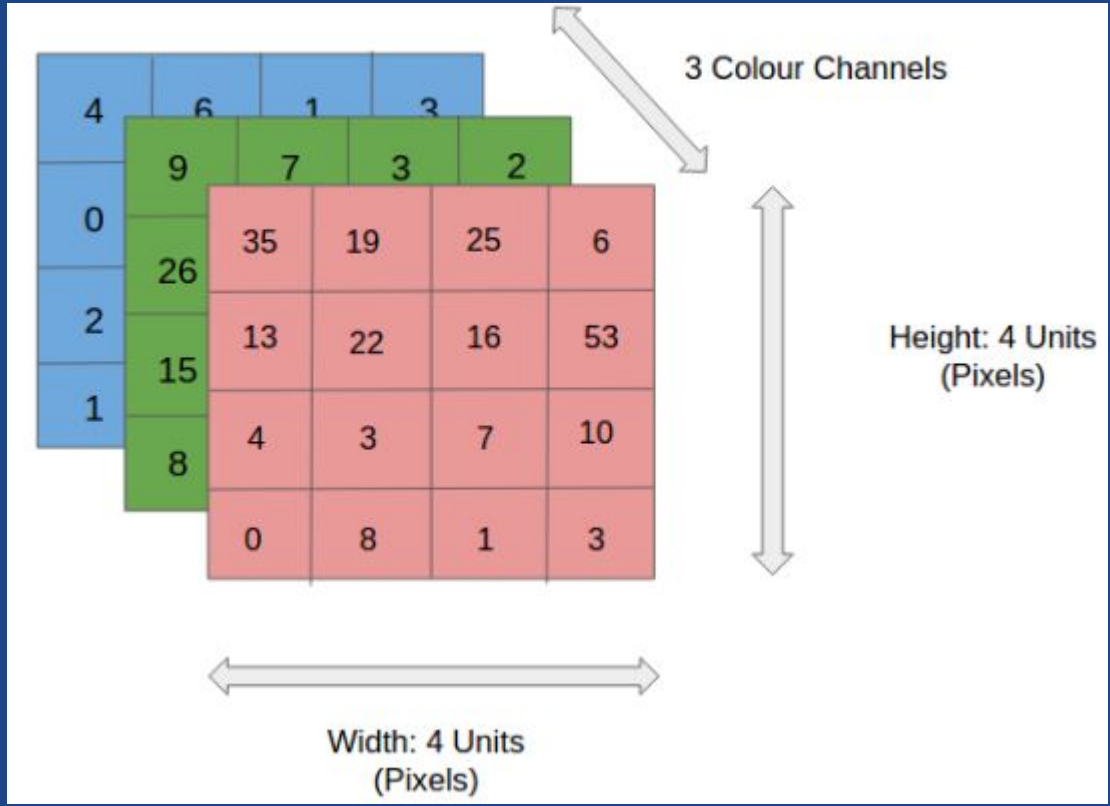


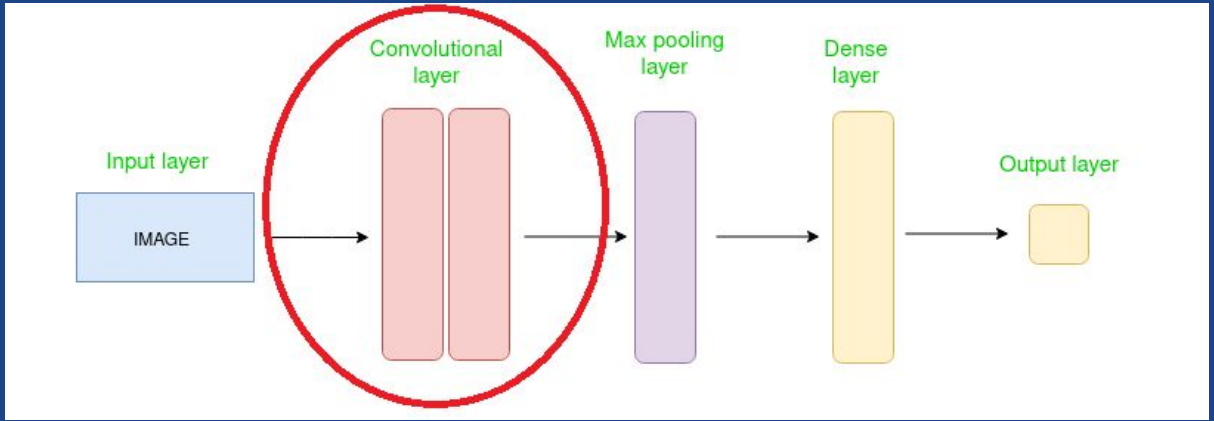
Imagem RGB (Red, Green e Blue).

A imagem RGB possui **3 canais**.

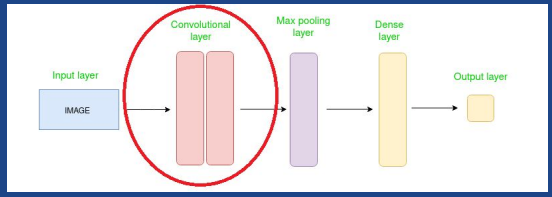
Já uma imagem monocromática possui somente **1 canal**.



Camada de Convolução



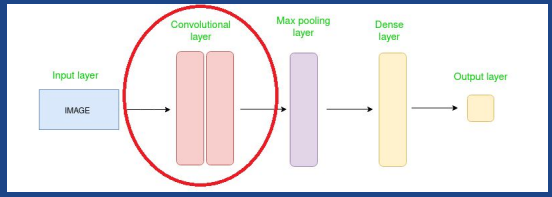
Camada de Convolução



Kernel (habitualmente chamado de **filtro** ou **filter**) é uma matrix, que percorrerá a imagem fazendo cálculos matemáticos, produzindo assim uma nova imagem.

1	0	1
0	1	0
1	0	1

Camada de Convolução



Convolução de uma imagem monocromática.

6 x 6 Image

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Convolution

*

3 x 3 filter

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

=

3^{1}	0^{0}	1^{-1}	2	7	4
1^{1}	5^{0}	8^{-1}	9	3	1
2^{1}	7^{0}	2^{-1}	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

3 x 1
1 x 1
2 x 1
0 x 0
5 x 0
7 x 0
1 x -1
8 x -1
\+ 2 x -1

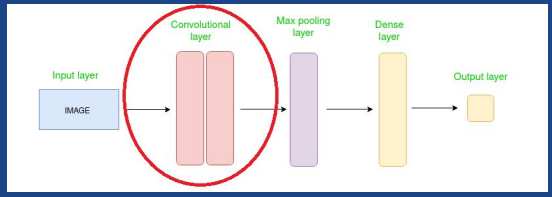
-5

=

4 x 4 output

-5			

Camada de Convolução



Convolução de uma imagem monocromática.

6 x 6 Image

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Convolution

*

3 x 3 filter

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

=

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1^{1}	7^{0}	8^{-1}
4	2	1	6^{1}	2^{0}	8^{-1}
2	4	5	2^{1}	3^{0}	9^{-1}

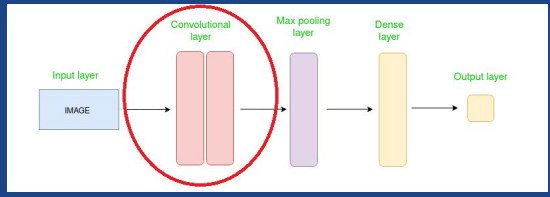
1 x 1
6 x 1
2 x 1
7 x 0
2 x 0
3 x 0
8 x -1
8 x -1
\+ 9 x -1
-16

=

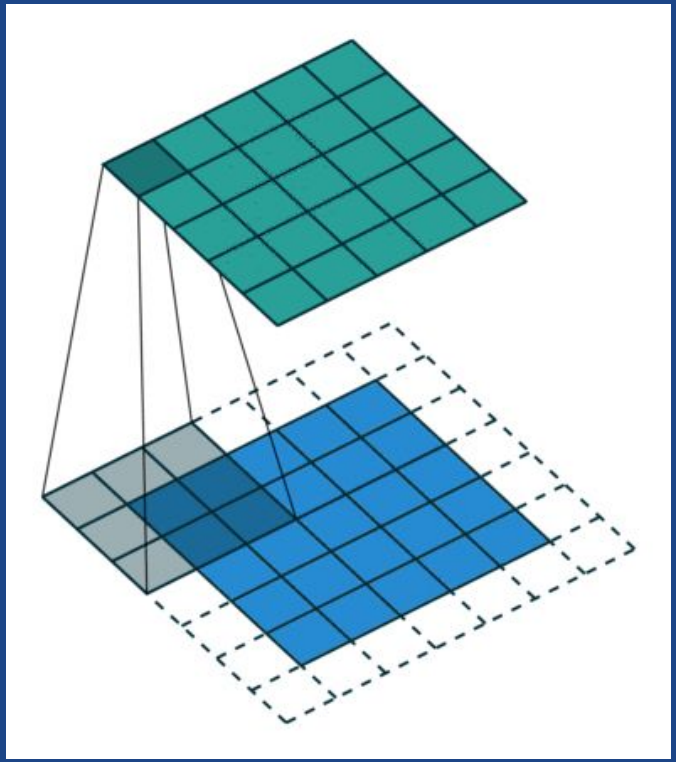
4 x 4 output

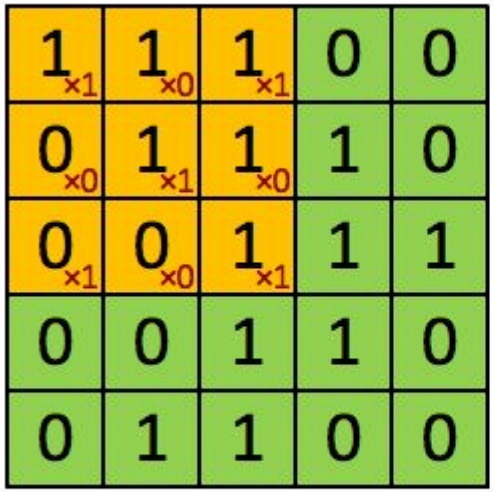
-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

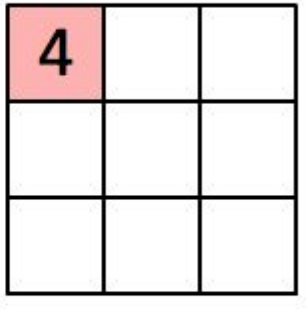
Camada de Convolução



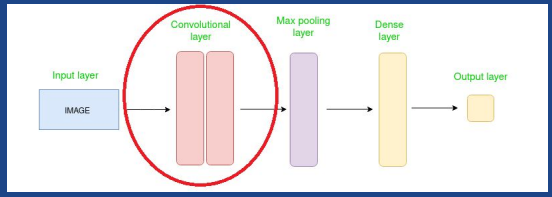
Animação de uma Convolução.



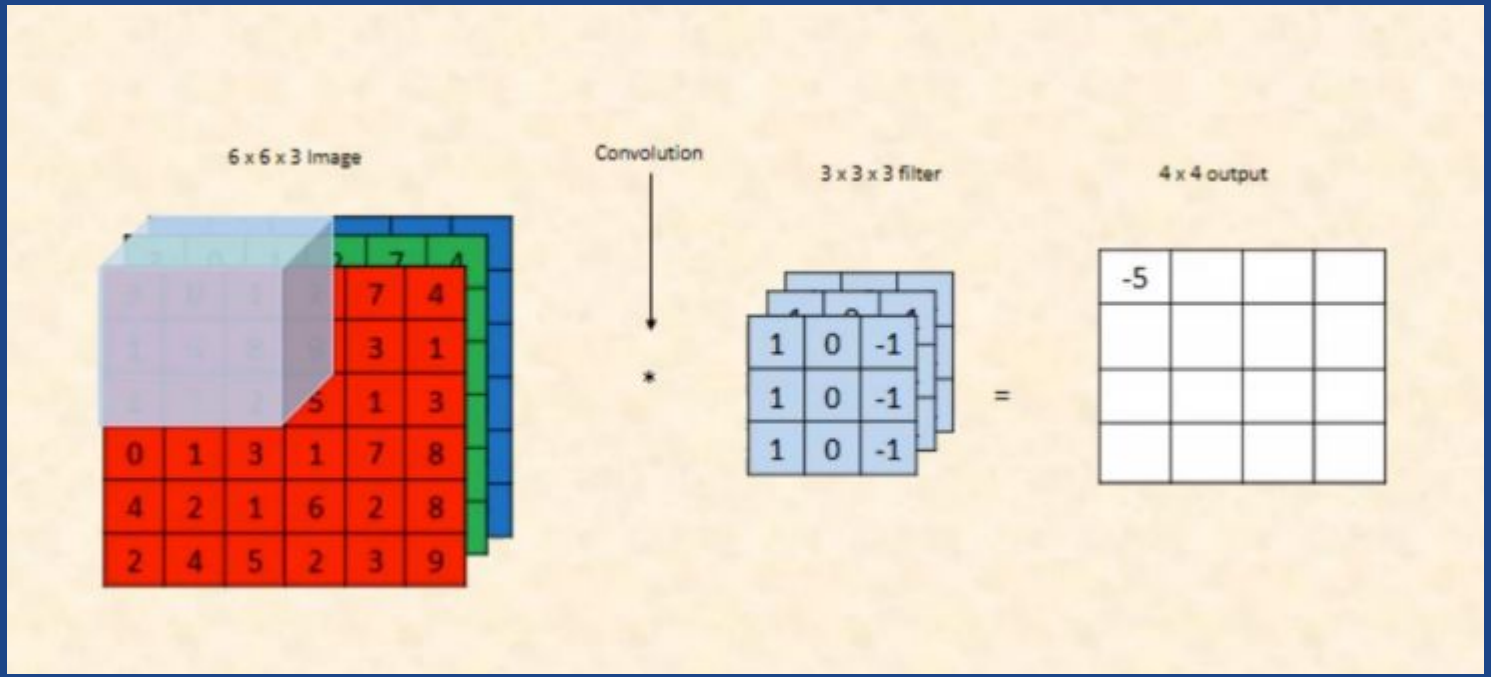




Camada de Convolução

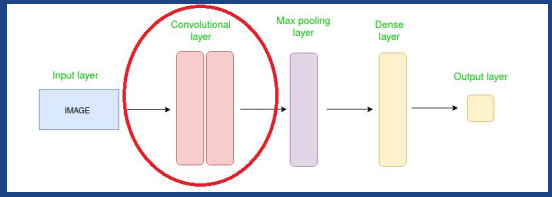


Convolução em 3 canais.



Camada de Convolução

Animação em 3 canais.



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)

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)

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)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

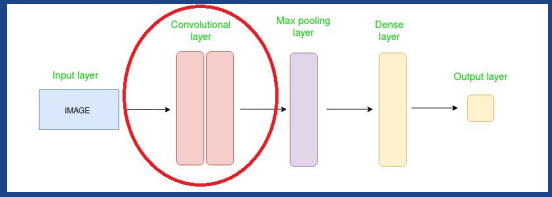
$308 + (-498) + 164 + 1 = -25$

Bias = 1

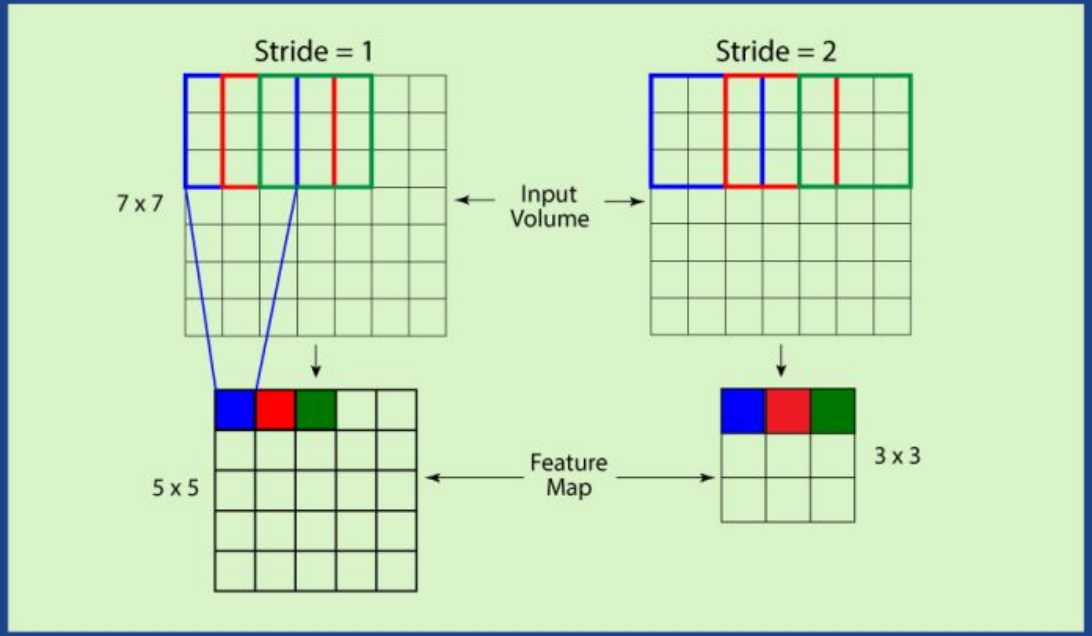
Output

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

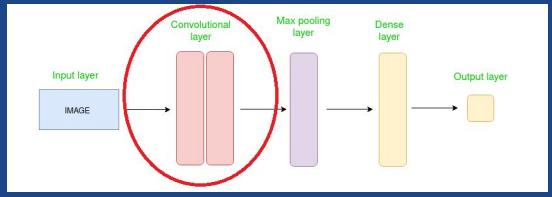
Camada de Convolução



Stride é o espaçamento que o kernel utilizará no seu movimento.

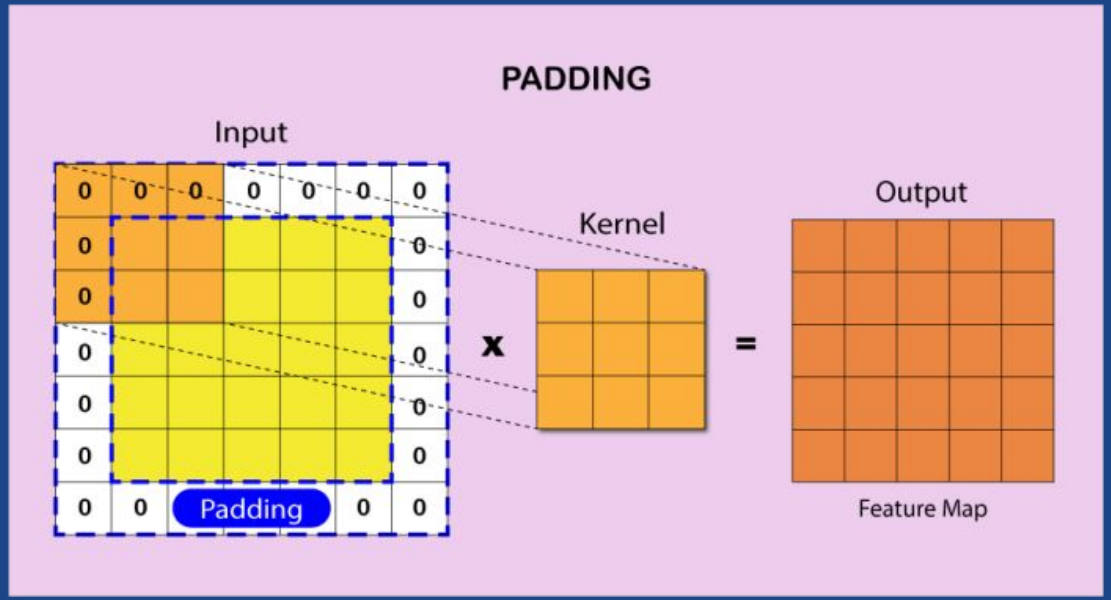


Camada de Convolução

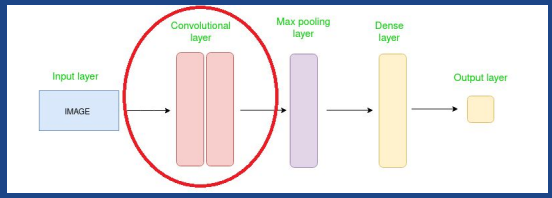


Padding é usado para ampliar a imagem em seus extremos, e usar o kernel fora dos limites da imagem inicial.

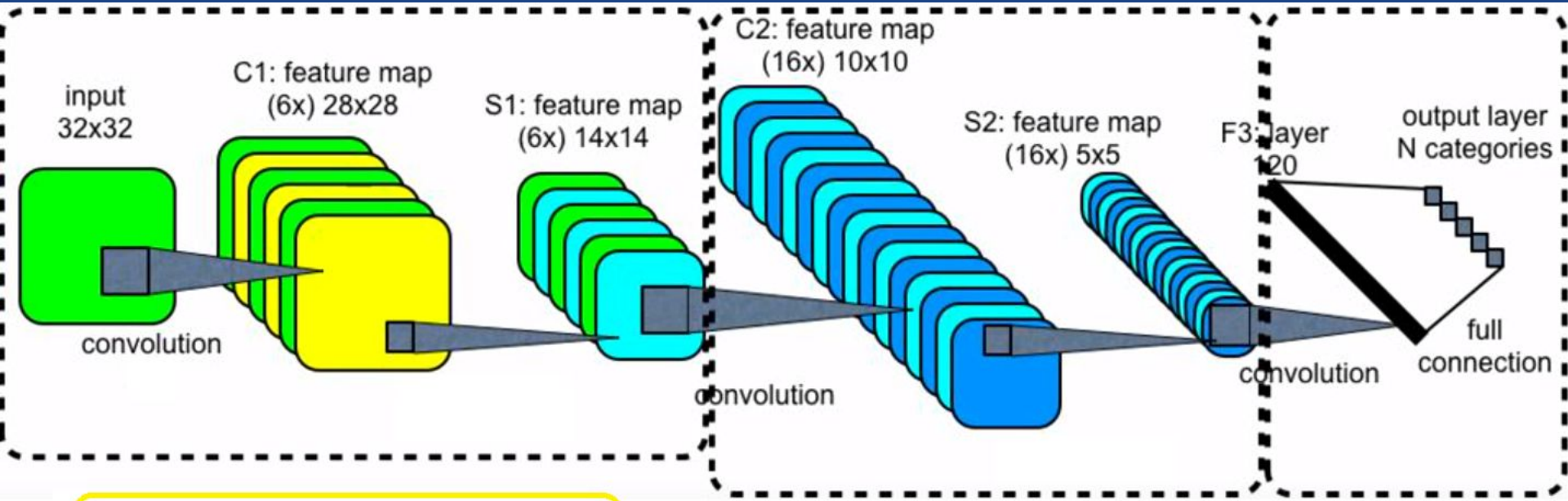
Uma **Feature map** é a saída desse processo de convolução



Camada de Convolução



É possível ter várias feature map e camadas. Veja o exemplo:
Na primeira camada temos 6 feature maps.
Na segunda camada temos 16 feature maps.

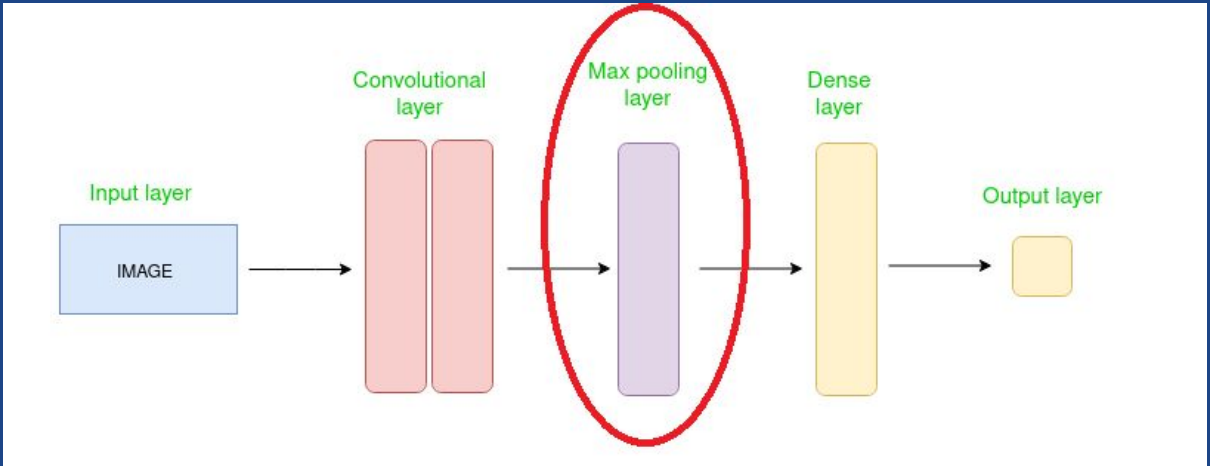


Camada 1(layer 1)

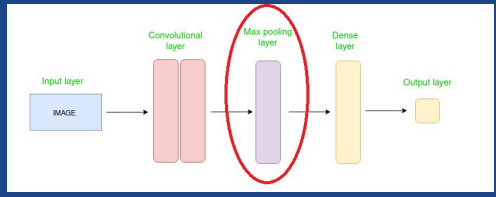
Camada 2(layer 2)

Classificação

Camada de Pooling

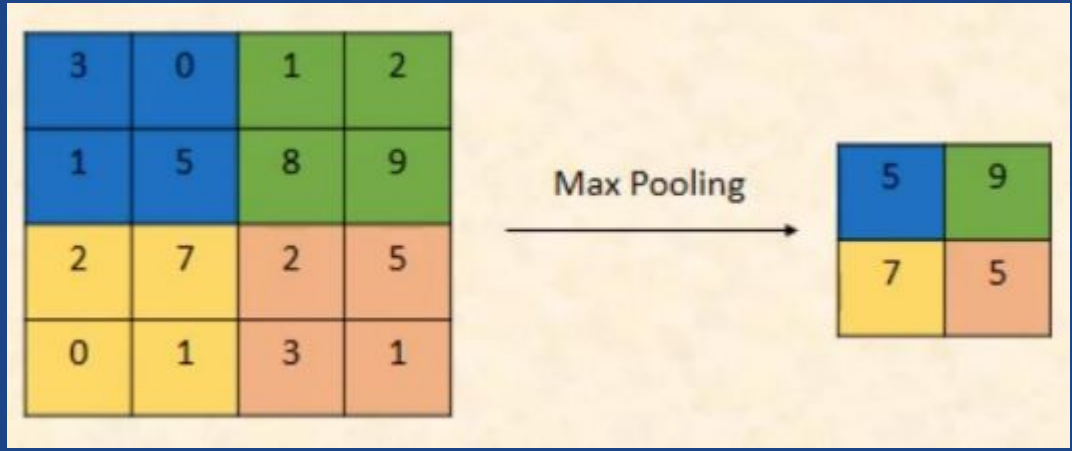


Camada de Pooling

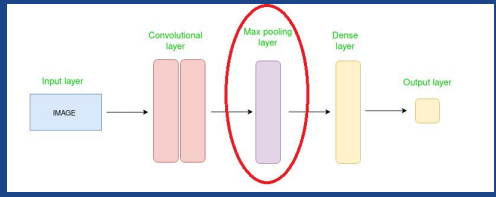


Pooling reduz o tamanho da imagem.

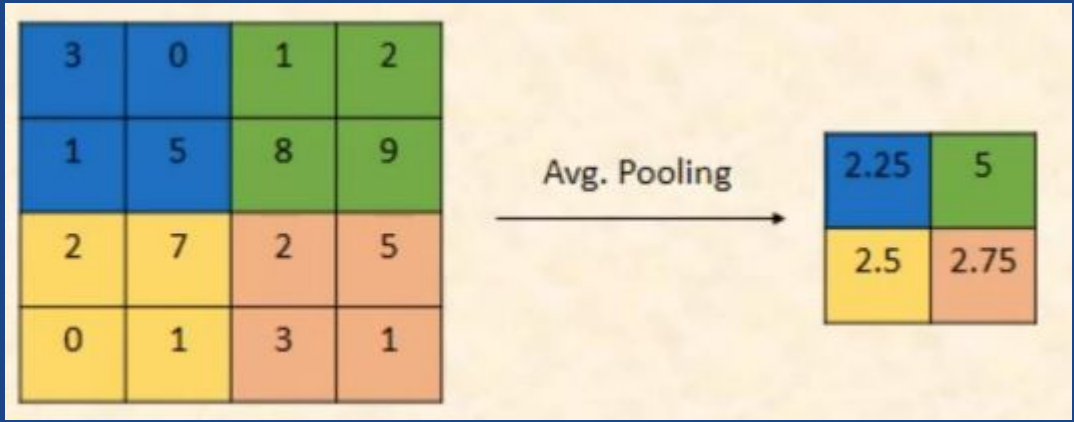
Max-pooling: Retorna o valor máximo na área sobreposta do kernel. Funciona como supressor de ruído.



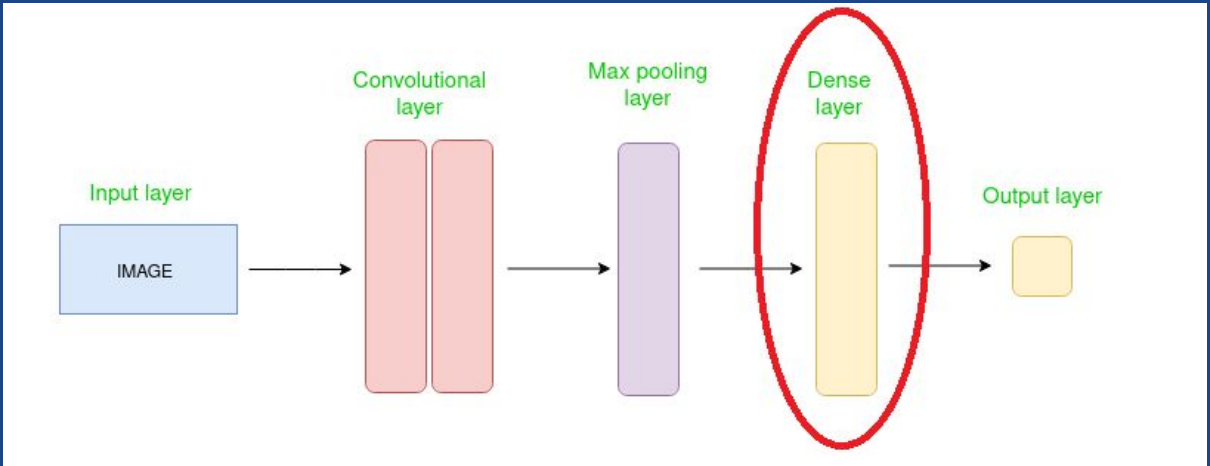
Camada de Pooling



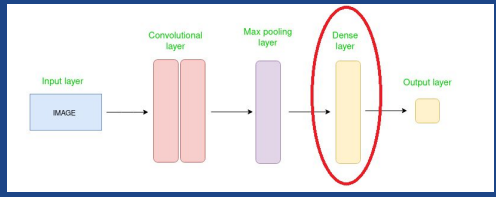
Average-pooling: Retorna a média na área sobreposta do kernel



Camada Densa

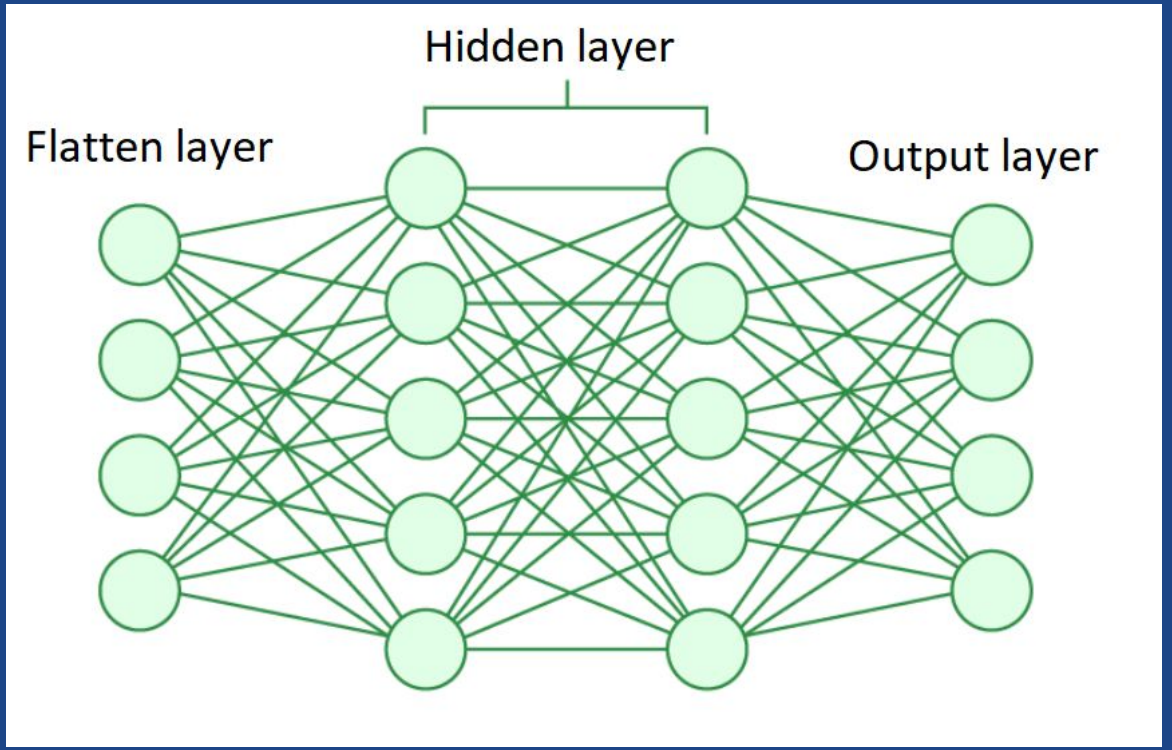


Camada Densa

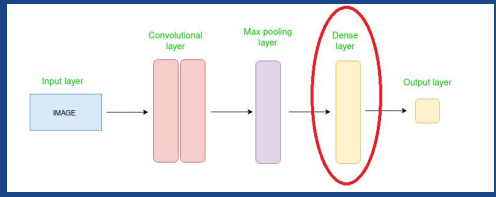


A **camada densa** é composta de uma **rede neural densa**.

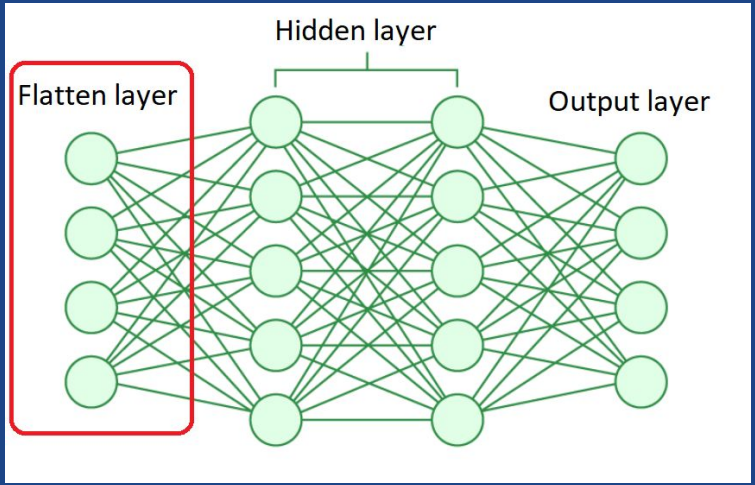
Uma **rede neural densa** é uma rede neural em que todos os neurônios estão conectados entre eles mesmos

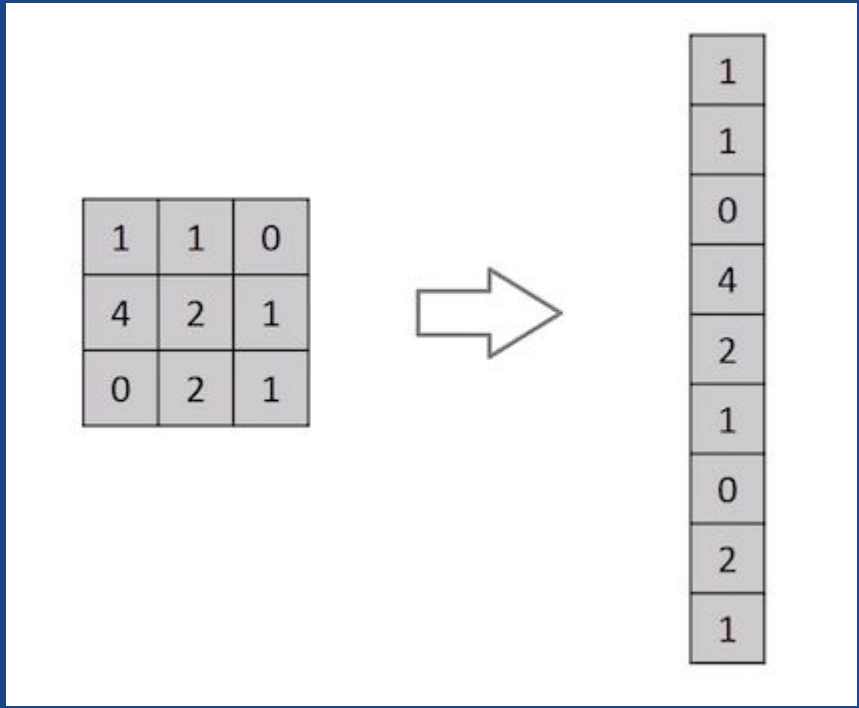


Camada Densa

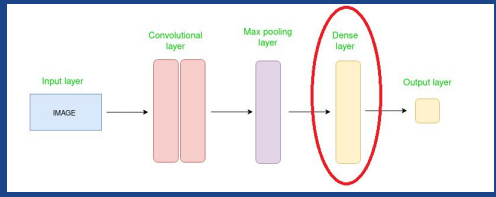


Flatten layer é um vetor linear obtido no processo de **Flattening**, consiste em transformar a imagem no formato de matrix, para um vetor linear, rearranjando os valores linearmente.

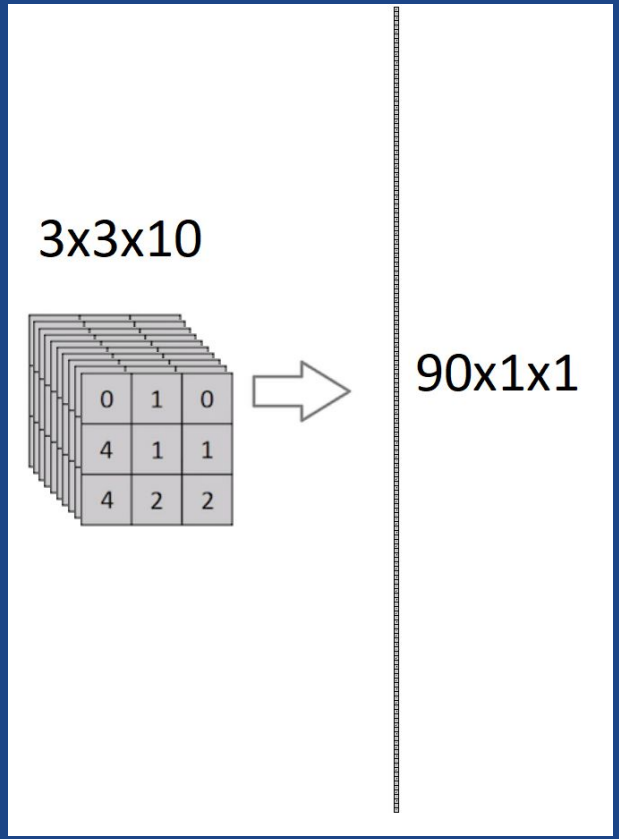




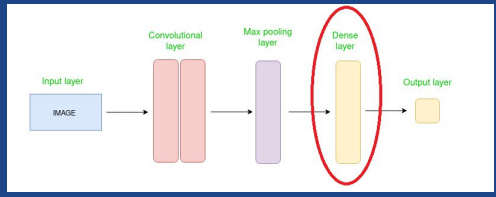
Camada Densa



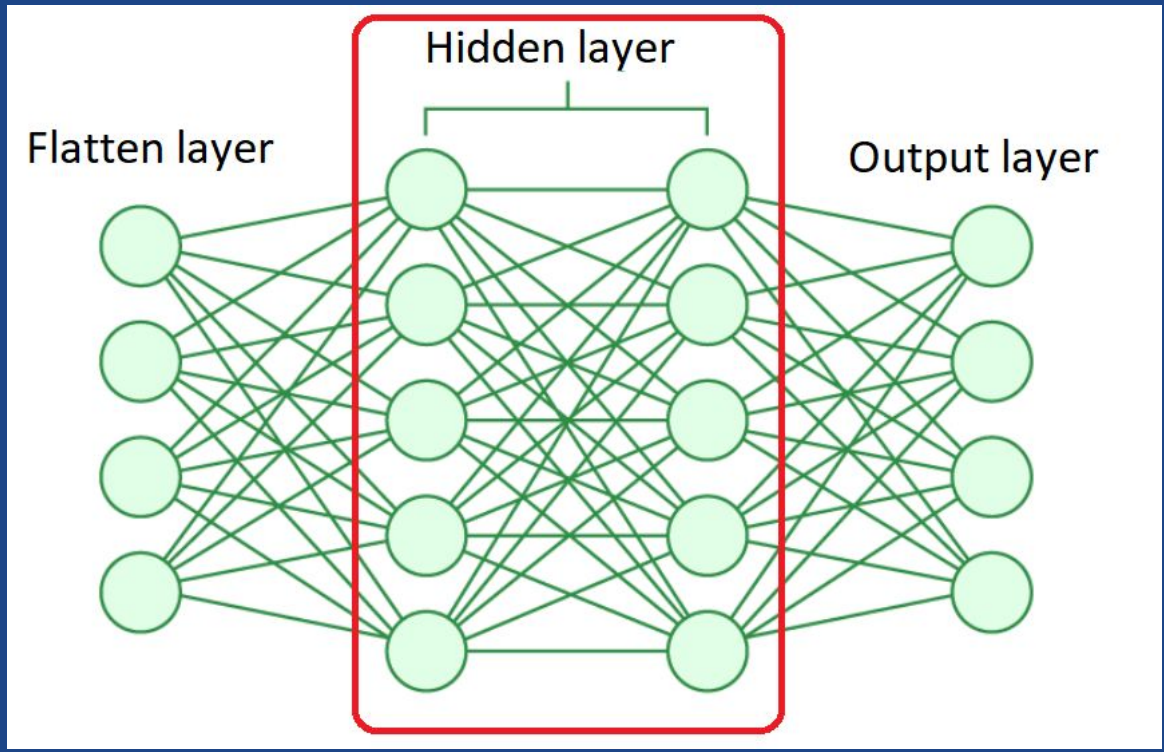
Se 10 feature map de 3x3 passassem pelo processo de flattening, seria produzido um flatten layer de 90x1.



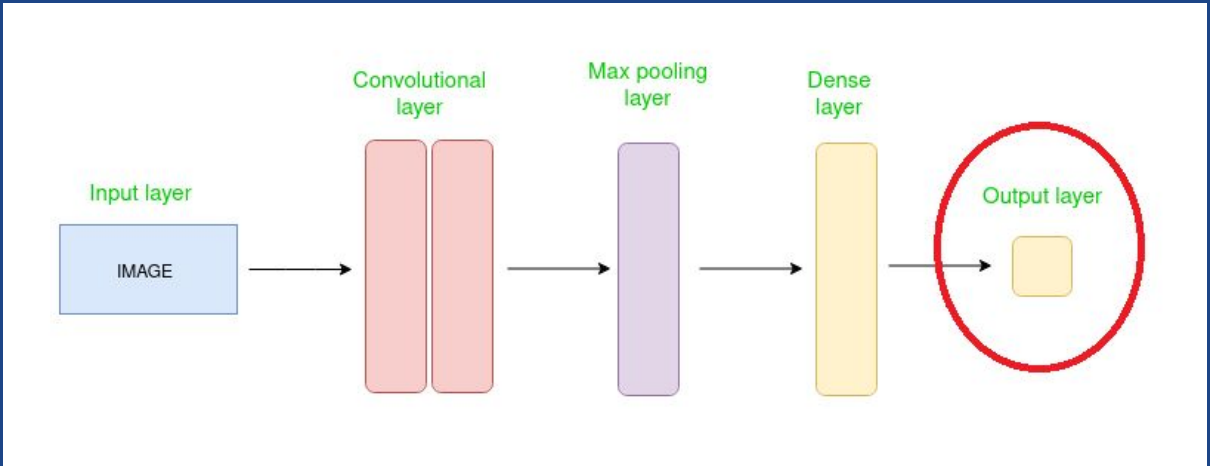
Camada Densa



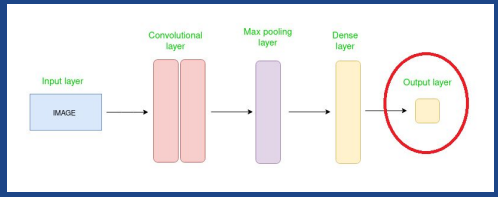
Hidden Layers: calcula (com os inputs do neurônio anterior, os pesos e bias definidos no treinamento) um valor de output para o próximo neurônio.



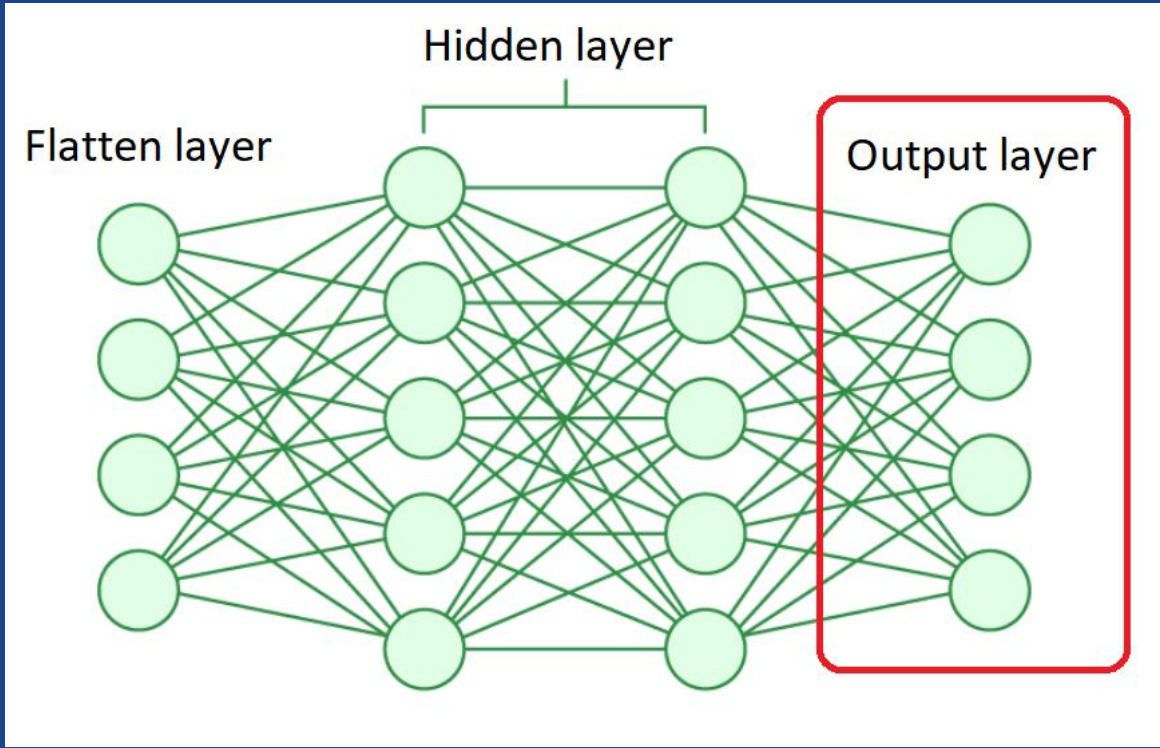
Camada de Saída



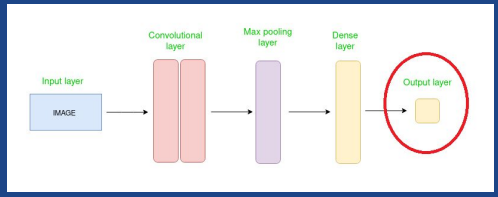
Camada de Saída



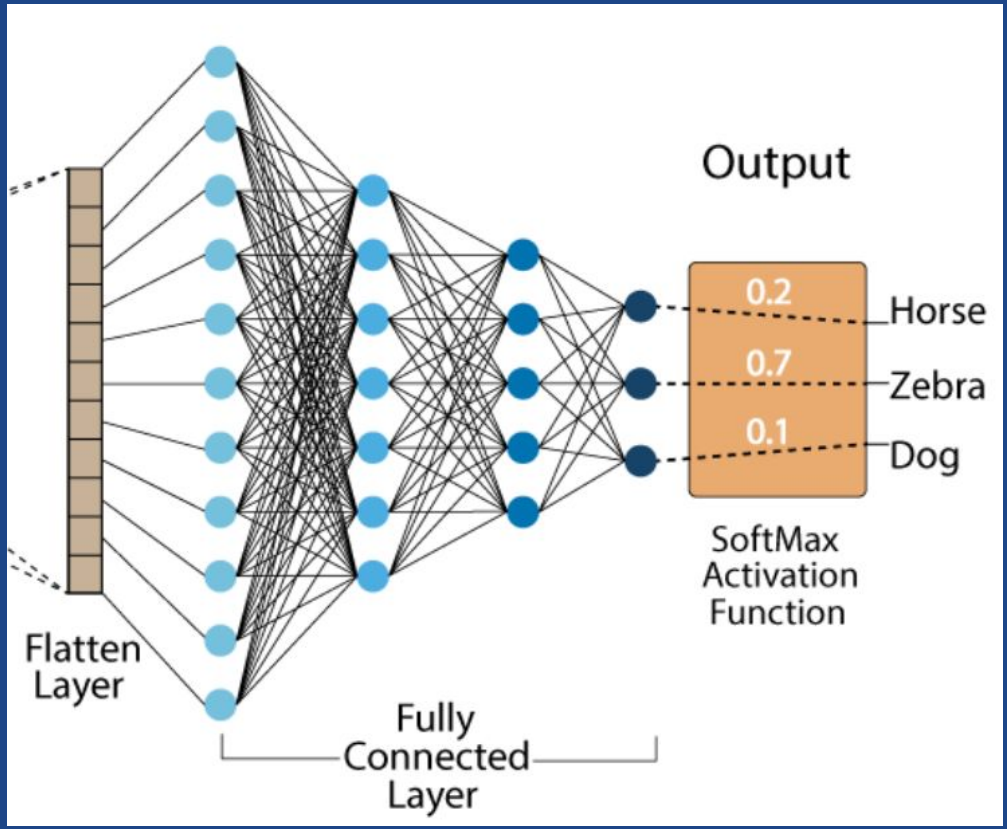
Output Layer: cada neurônio representa uma classificação definida, onde o neurônio com maior valor definirá o resultado.



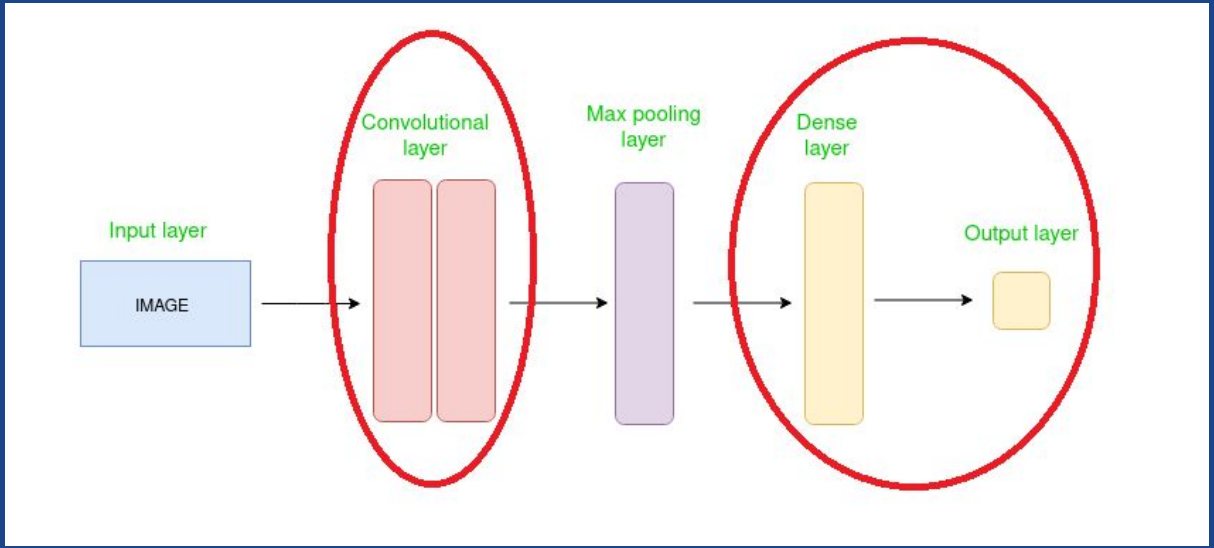
Camada de Saída



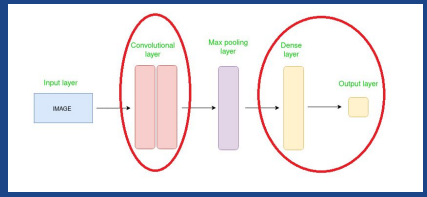
Predição após a rede neural densa. A imagem foi classificada como Zebra, pois obteve o maior valor de 0,7.



Função de Ativação

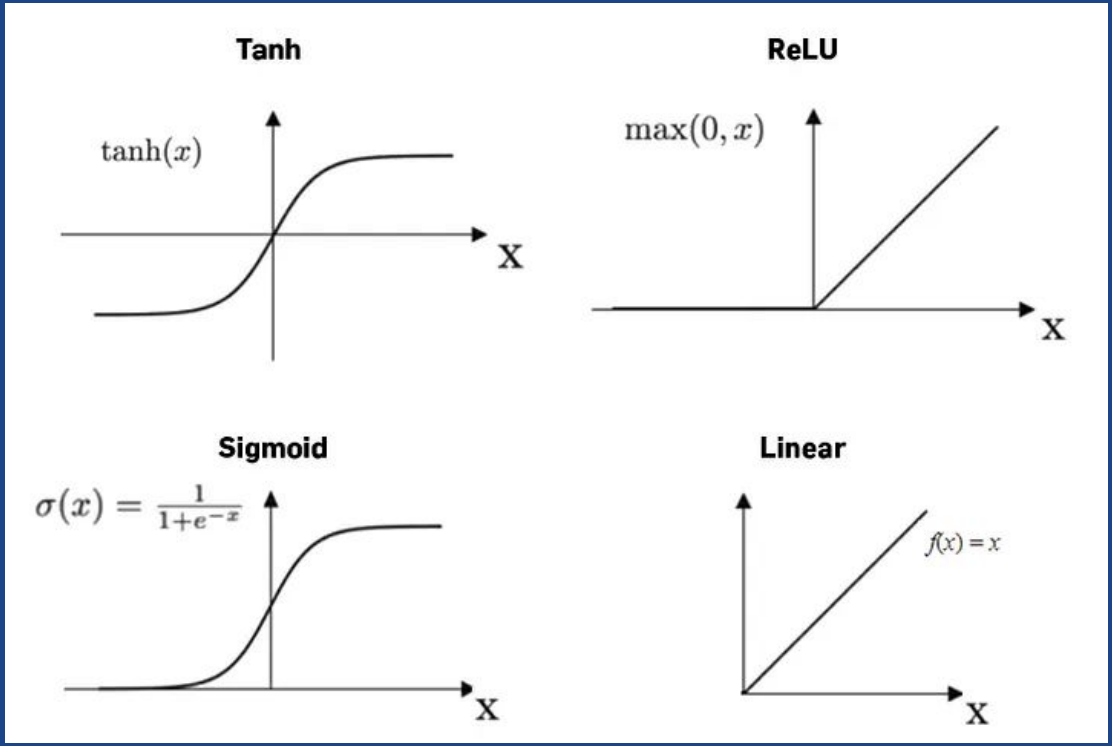


Função de Ativação

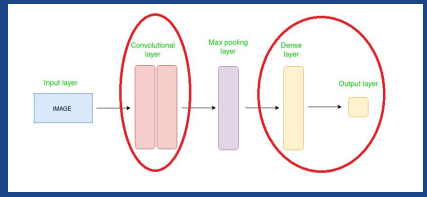


A **função de ativação** é usada na camada de convolução e na rede neural densa.

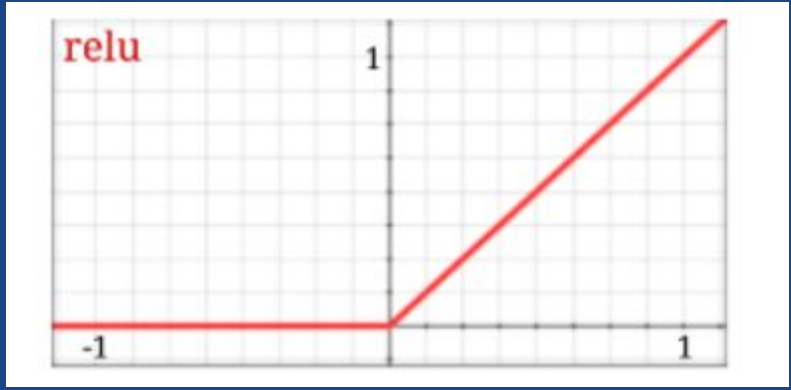
Ela aplica uma função no valor de saída de um neurônio.



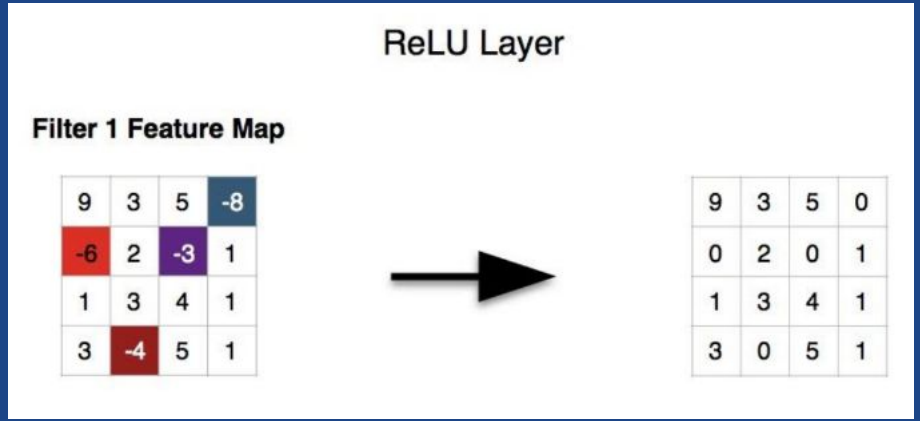
Função de Ativação



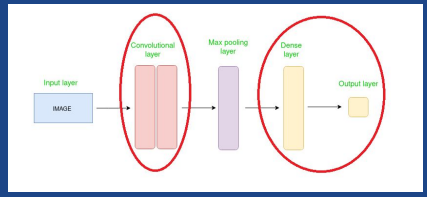
A função de ativação ReLU (*Rectified Linear Unit*) é bastante usada, ela será usada nas camadas de convolução e na rede neural densa.



O output da função de ativação ReLU troca todos os valores negativos por 0.

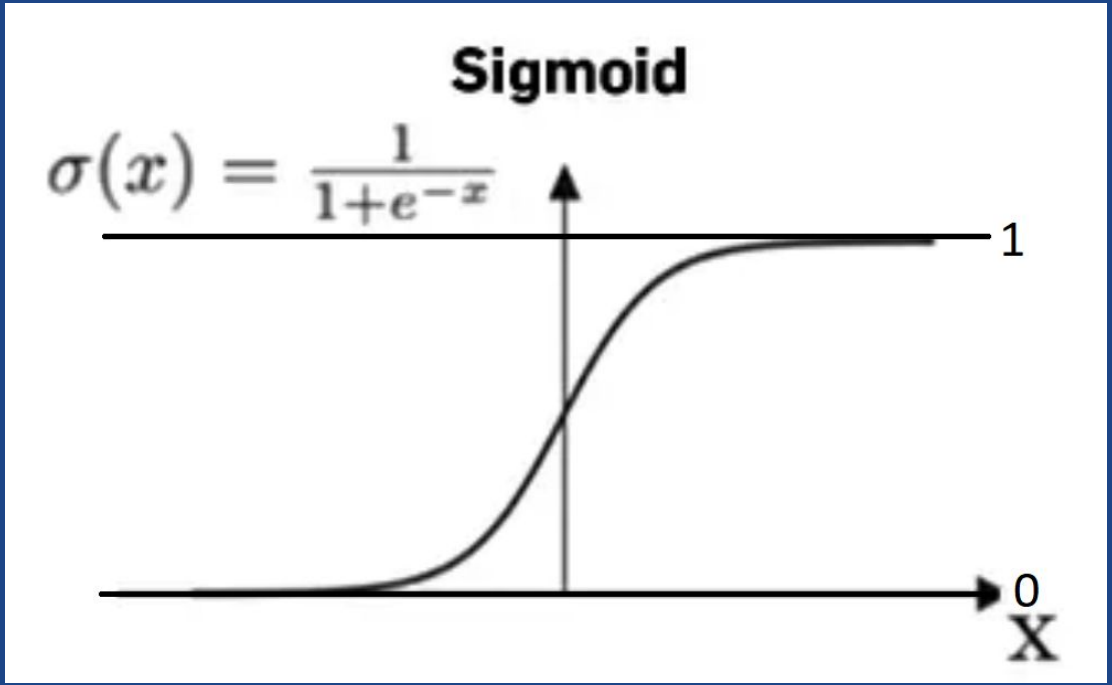


Função de Ativação

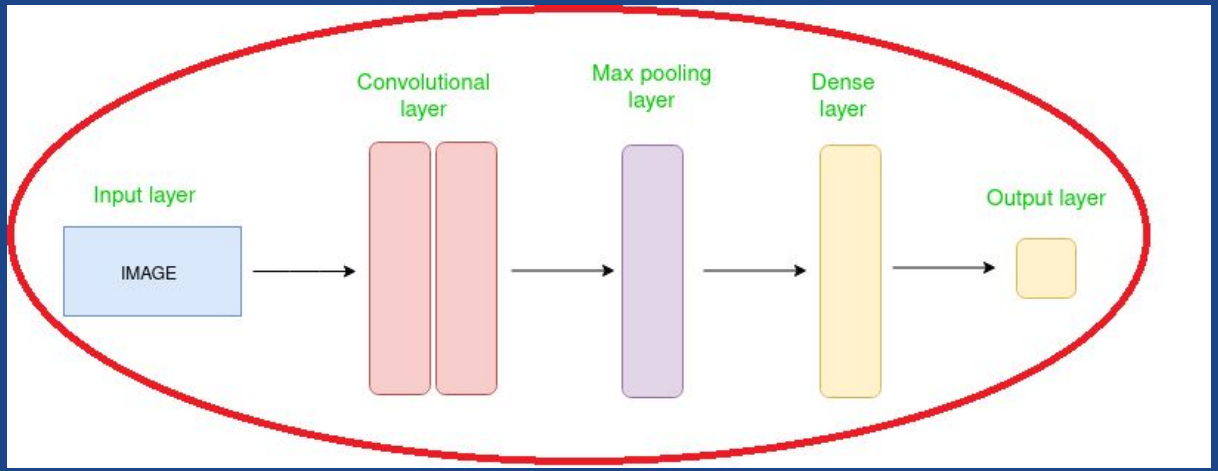


A função de ativação Sigmoid será usada na camada de saída.

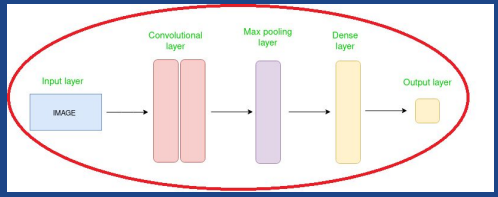
O output da função de ativação Sigmoid varia de 0 a 1.



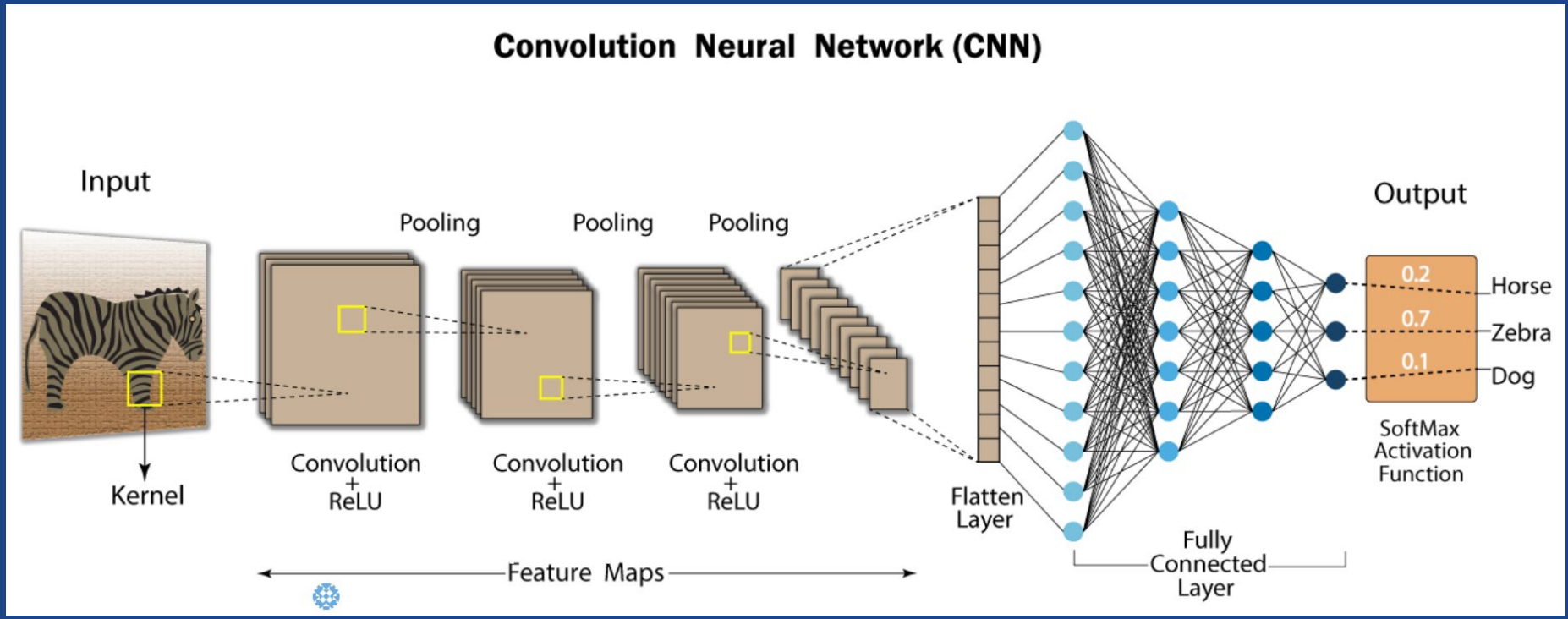
Visão Geral


Input layer
IMAGE
Convolutional layer
Max pooling layer
Dense layer
Output layer

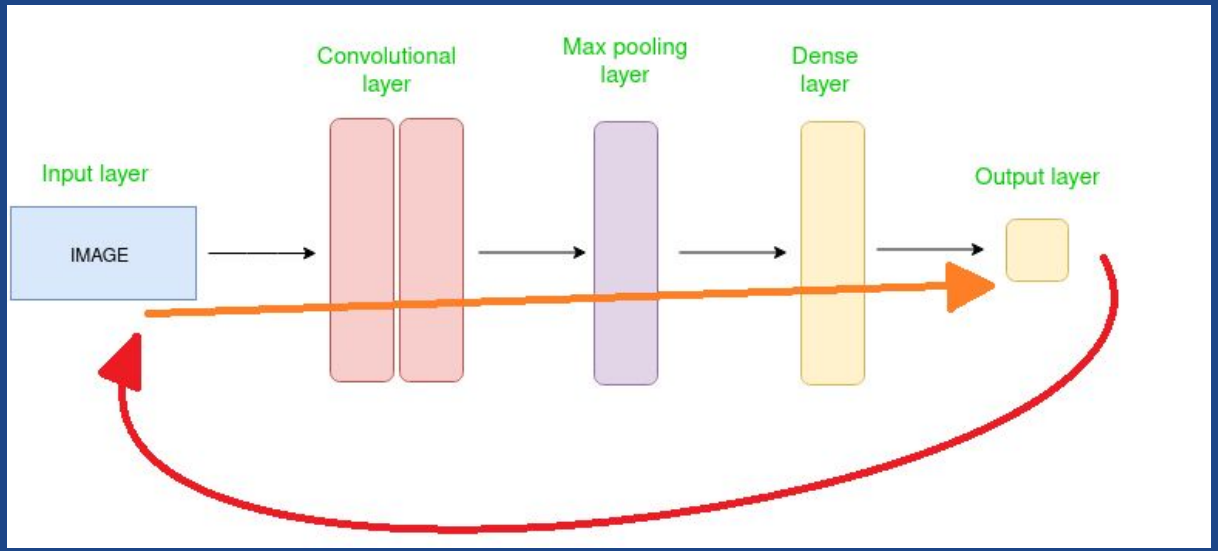
Visão Geral



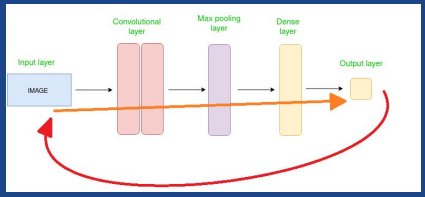
Classificação de imagem



Treinamento

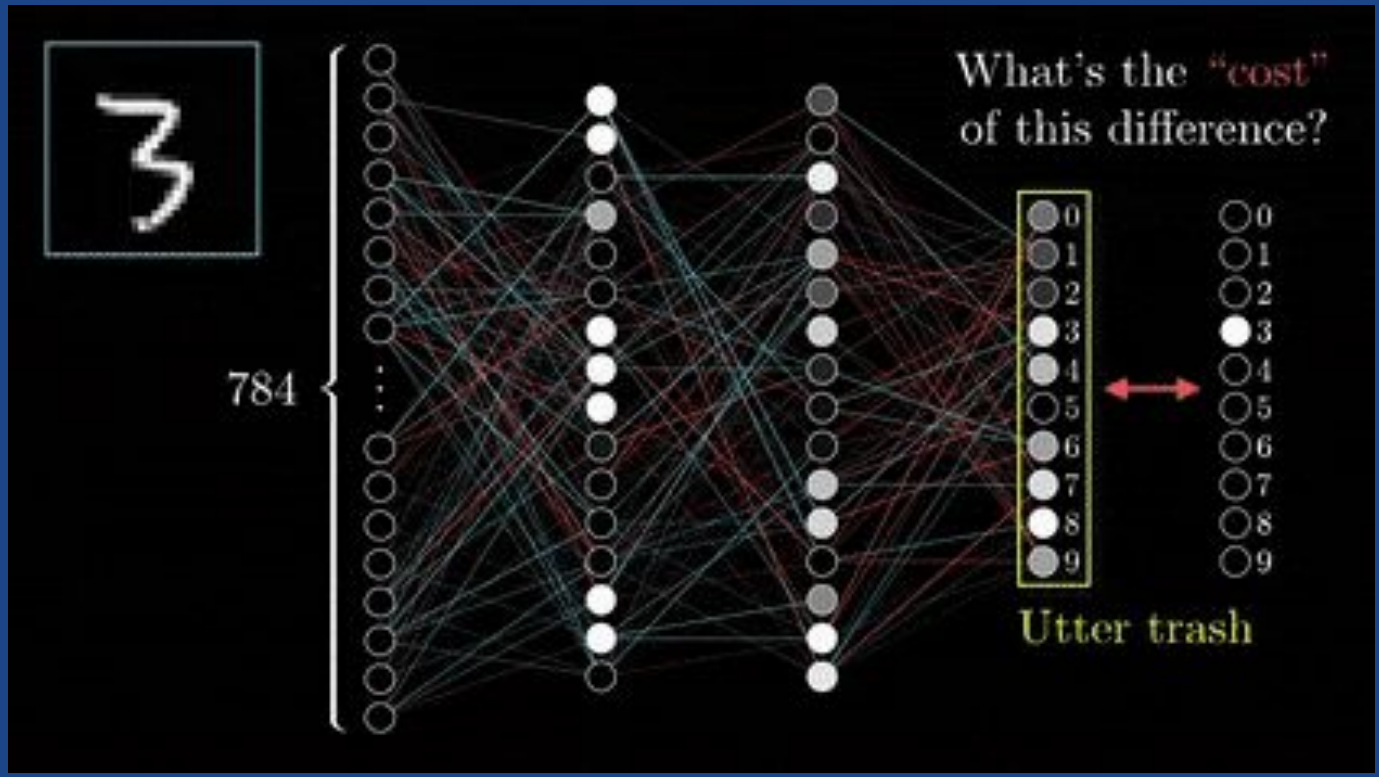


Treinamento

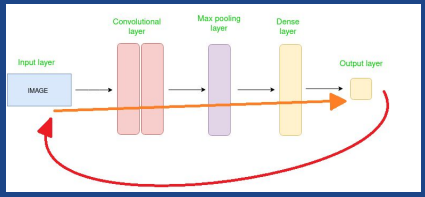


É preciso treinar uma CNN antes de usar.

A CNN é treinada utilizando várias imagens como referência.



Treinamento



Alguns termos no treinamento:

Backpropagation é o processo para ajustar os pesos do CNN com o objetivo de minimizar o erro. O processo de atualizar os pesos se repete até um valor desejado.

Época, ou **epoch**, é cada passagem pelo processo de atualizar os erros no backpropagation.

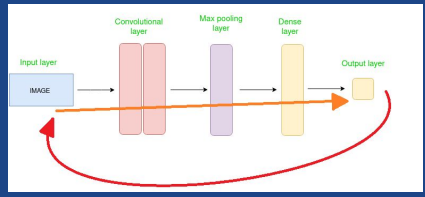
Learning Rate é um valor que define o passo ou velocidade de aprendizagem, se o valor for alto chegará mais rapidamente no resultado, mas possivelmente terá dificuldades em convergir. E valores muito baixos acontecerá o oposto.

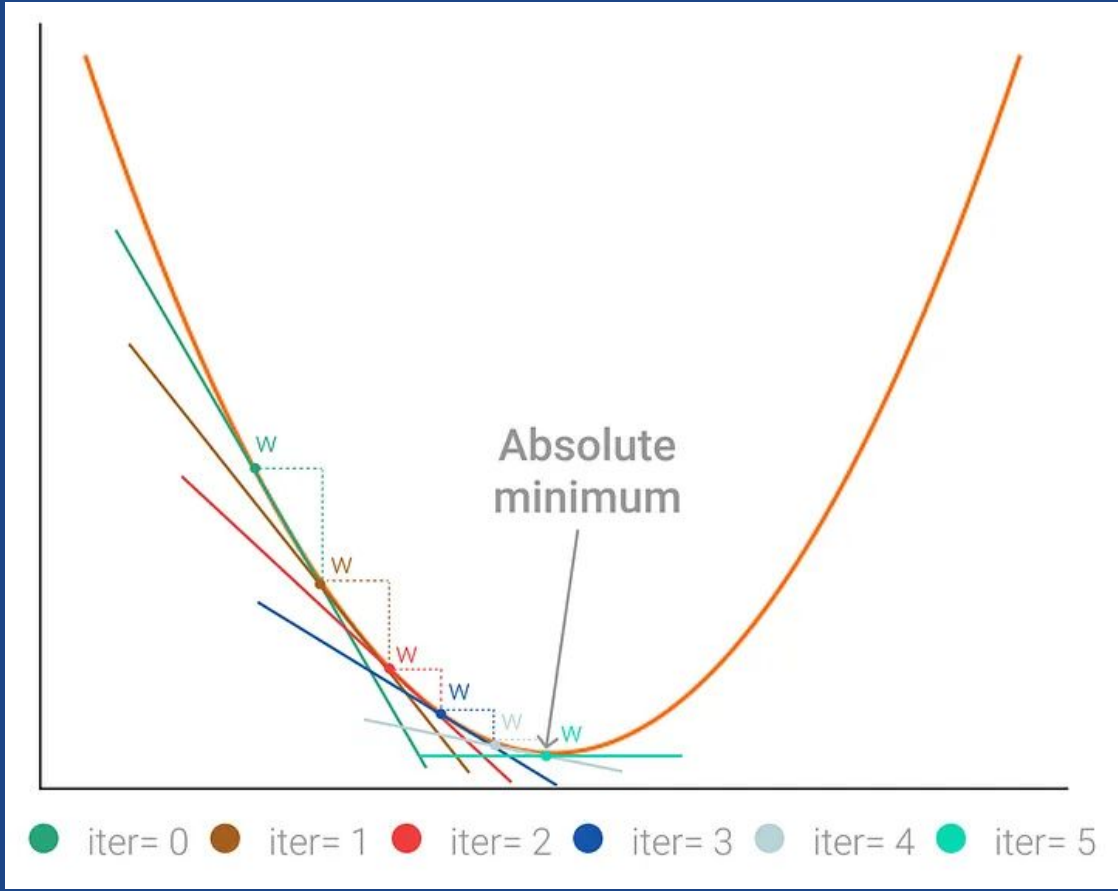
Função de custo, ou **Loss Function**, é uma função matemática que define o quão distante se está do resultado desejado, para regressão geralmente se usa a **Mean Square Error (MSE)** e para classificação se usa geralmente **Cross-Entropy**.

Gradiente descendente, ou **Gradient Descent**, é um algoritmo que optimiza a busca pelo erro mínimo usando a função de custo.

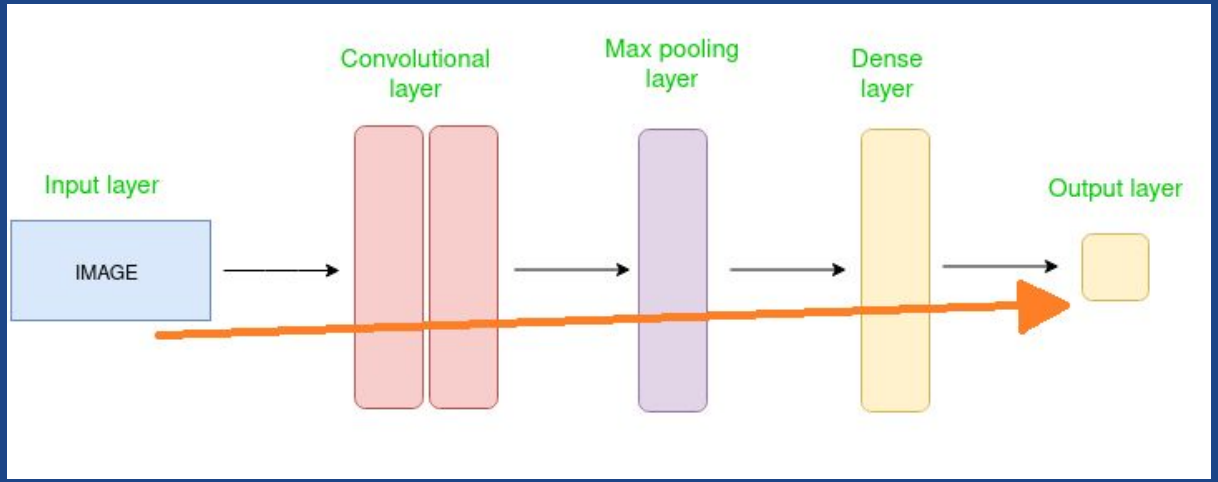
Batch size é a quantidade de amostras que é utilizada antes de atualizar os pesos.

Treinamento

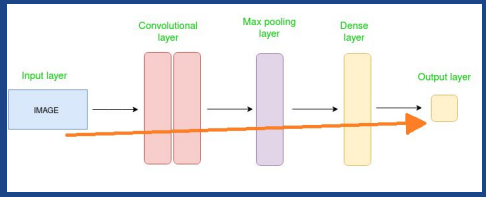




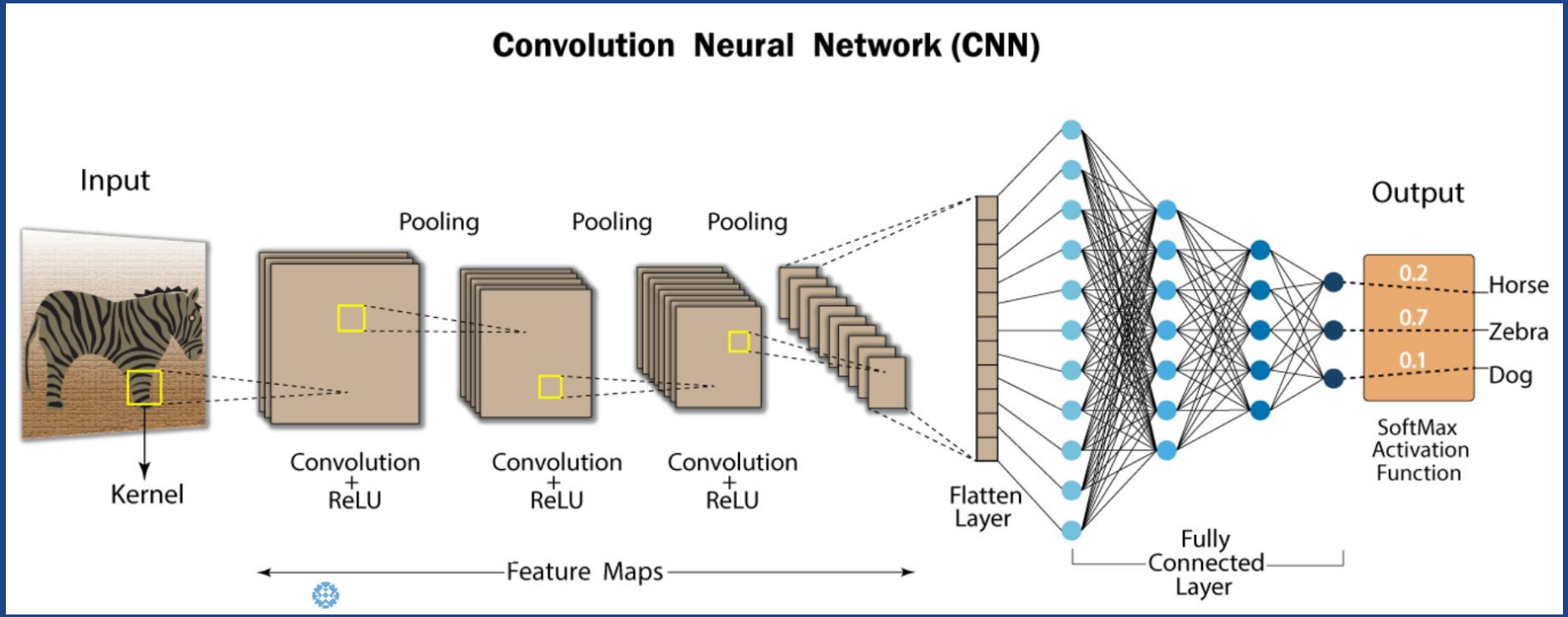
Predição


Convolutional layer
Max pooling layer
Dense layer
Input layer
Output layer
IMAGE

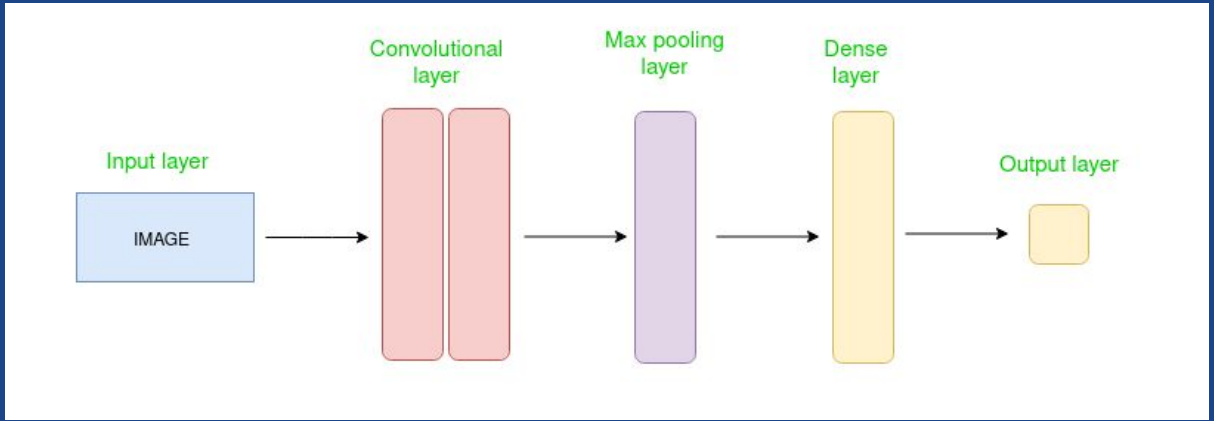
Predição



Após o treinamento com os ajustes dos pesos, é possível usar a CNN para prever casos.



Prática

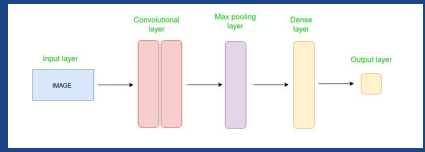

Convolutional layer
Max pooling layer
Dense layer
Input layer
IMAGE
Output layer

Prática

```
# Carregamento das bibliotecas
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import
Sequential
from tensorflow.keras.layers import Conv2D,
MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import
to_categorical

# Carrega o dataset MNIST que contém dígitos
escritos à mão
(train_images, train_labels), (test_images,
test_labels) = mnist.load_data()

# Pré-processamento dos dados:
# Redimensiona as imagens de treinamento e
teste para um tensor 4D (amostras, altura,
largura, canais)
# Normaliza os valores dos pixels para o
intervalo [0, 1]
train_images = train_images.reshape((60000,
28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000,
28, 28, 1)).astype('float32') / 255

# Transforma os rótulos de treinamento e
teste em vetores binários (one-hot encoding)
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

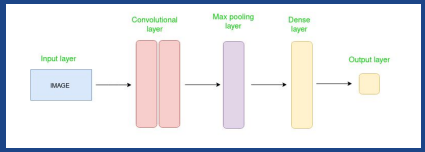
Código de Treinamento

```
# Constrói o modelo da Convolutional Neural
Network (CNN):
model = Sequential()
# Adiciona uma camada de convolução com 32 filtros
3x3 e função de ativação ReLU
# A entrada tem o formato (28, 28, 1)
correspondente a imagens em escala de cinza de
28x28 pixels
model.add(Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)))
# Adiciona uma camada de MaxPooling 2x2 para
redução de dimensões
model.add(MaxPooling2D((2, 2)))
# Adiciona outra camada de convolução com 64
filtros 3x3 e função de ativação ReLU
model.add(Conv2D(64, (3, 3), activation='relu'))
# Adiciona outra camada de MaxPooling 2x2 para
redução de dimensões
model.add(MaxPooling2D((2, 2)))
# Adiciona outra camada de convolução com 64
filtros 3x3 e função de ativação ReLU
model.add(Conv2D(64, (3, 3), activation='relu'))
# Transforma os dados 2D em um vetor 1D para a
etapa de classificação
model.add(Flatten())
# Adiciona uma camada densa com 64 neurônios e
função de ativação ReLU
model.add(Dense(64, activation='relu'))
# Adiciona a camada de saída com 10 neurônios
correspondentes aos 10 dígitos (0-9)
# A função de ativação softmax é usada para
transformar as saídas em probabilidades
model.add(Dense(10, activation='softmax'))
```




```
# Compila o modelo:
# Configura o otimizador "Adam", a
função de perda
"categorical_crossentropy" e a métrica
"accuracy"
model.compile(optimizer='adam',

loss='categorical_crossentropy' ,
              metrics=['accuracy'])

# Treina o modelo usando os dados de
treinamento:
# Realiza 5 épocas de treinamento,
usando um tamanho de lote (batch size)
de 64
model.fit(train_images, train_labels,
epochs=5, batch_size=64)

# Avalia o modelo no conjunto de teste:
# Calcula a perda e a acurácia usando os
dados de teste
test_loss, test_accuracy =
model.evaluate(test_images, test_labels)
print(f'Acurácia no conjunto de teste:
{test_accuracy}')
```

Prática

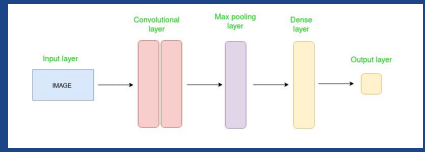


Resultados do treinamento

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
938/938 [==============================] - 61s 63ms/step - loss: 0.1846 - accuracy: 0.9444
Epoch 2/5
938/938 [==============================] - 59s 63ms/step - loss: 0.0512 - accuracy: 0.9838
Epoch 3/5
938/938 [==============================] - 58s 62ms/step - loss: 0.0362 - accuracy: 0.9887
Epoch 4/5
938/938 [==============================] - 59s 63ms/step - loss: 0.0274 - accuracy: 0.9916
Epoch 5/5
938/938 [==============================] - 60s 64ms/step - loss: 0.0217 - accuracy: 0.9935
313/313 [==============================] - 3s 10ms/step - loss: 0.0297 - accuracy: 0.9906
Acurácia no conjunto de teste: 0.9905999898910522
```

Prática

Código de Predição




```
# Carregamento das bibliotecas
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Carrega a imagem que você deseja prever:
new_image_path = '/content/4.png'
new_image = load_img(new_image_path, color_mode='grayscale', target_size=(28, 28))
new_image_array = img_to_array(new_image)
new_image_array = new_image_array.reshape((1, 28, 28, 1)).astype('float32') / 255

# Faz a previsão usando o modelo treinado:
predictions = model.predict(new_image_array)
predicted_label = np.argmax(predictions[0])

print(f'A imagem foi classificada como o dígito:{predicted_label}')
```

Prática

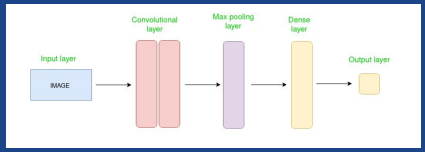
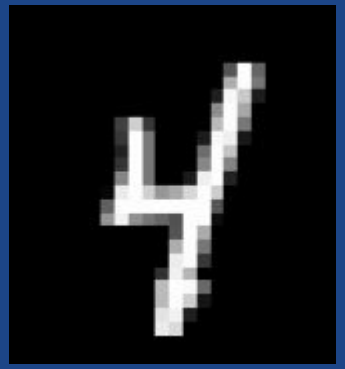


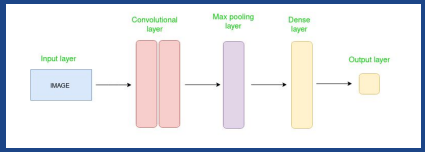
Imagem carregada



Resultado da predição

```
1/1 [==============================] - 0s 33ms/step
A imagem foi classificada como o dígito: 4
```

Prática



Tamanho das camadas

```
1 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 64)                36928

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 93322 (364.54 KB)
Trainable params: 93322 (364.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Referências

1) https://en.wikipedia.org/wiki/Convolutional_neural_network
2) https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/
3) https://www.geeksforgeeks.org/introduction-convolution-neural-network/
4) https://adamharley.com/nn_vis/cnn/2d.html
5) https://developersbreach.com/convolution-neural-network-deep-learning/
6) https://docs.gimp.org/2.8/pt_BR/plug-in-convmatrix.html
7) https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/
8) https://towardsaws.com/activation-function-f76bfc03e215
9) https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/
10) https://www.tensorflow.org/datasets/catalog/mnist?hl=pt-br
11) https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/
12) https://datahacker.rs/one-layer-covolutional-neural-network/
13) https://rsdjournal.org/index.php/rsd/article/view/26101/22964
14) https://sol.sbc.org.br/index.php/sbcas/article/view/6241
15) https://repositorio.ufpb.br/jspui/bitstream/123456789/15606/1/DAR20052019.pdf
16) https://www.monografias.ufop.br/bitstream/35400000/2872/6/MONOGRAFIA_RedesNeuraisConvolucionais.pdf
17) https://www.facebook.com/mltutblogs
18) https://www.youtube.com/watch?v=vXj4z_Vo8G0&ab_channel=EugenioCulurciello
19) https://studentsxstudents.com/image-classification-using-quanvolutional-neural-networks-qnn-cd13b287ceca

Contato

Fábio Lofredo Cesar
linkedin.com/in/fábio-lofredo-cesar-050613262