

Relatório 15 - Prática: Redes Neurais Convolucionais

Jefferson korte junior

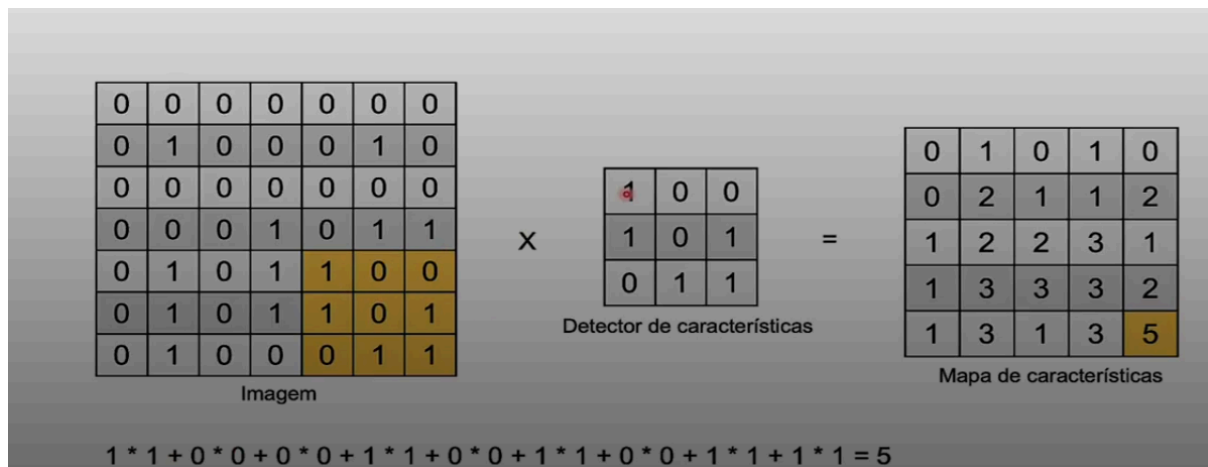
Seção 9 - Teoria sobre as redes Convolucionais.

Aula 01 -> imagens e pixels: Nessa aula aprendemos que o pixel é a menor informação possível em uma imagem, e que cada pixel tem uma combinação de 3 cores, (RGB) com a combinação dessas 3 cores podemos chegar em qualquer outra.

Aula 02 -> Introdução a redes neurais convolucionais: As redes neurais convolucionais (CNNs) usam os valores dos pixels como entrada para aprender padrões visuais. Dentro das redes convolucionais não é utilizado todos os pixels, por exemplo ele utilizou os emojis como exemplo, os emojis na parte lateral são todos iguais, algo que não vai mudar em nada a classificação deles, a própria rede neural escolhe os pixels necessários que representam a imagem. Elas são compostas por quatro etapas principais: convolução, pooling, flattening e uma rede neural densa para a classificação final. Essas etapas permitem que a rede extraia e interprete características visuais como bordas e texturas.

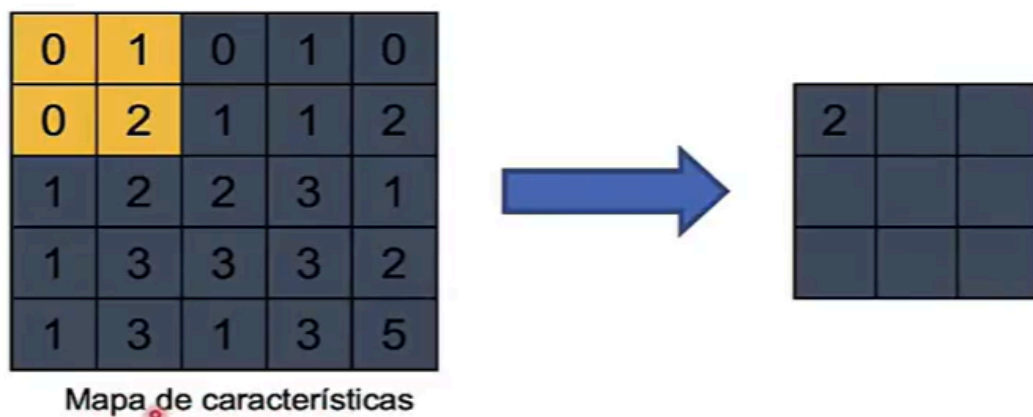
1 - Operador de convolução, 2 - Pooling, 3 - Flattening, 4 - Rede neural densas.

Aula 03 -> Etapa 1 - operador de convolução: Convolução é usada para destacar padrões importantes da imagem. Ela aplica pequenos filtros (como matrizes 3x3 ou 5x5) sobre a imagem, multiplicando os valores dos pixels por um conjunto de pesos. Isso gera um novo mapa chamado mapa de características, que representa visualmente aquilo que o filtro detectou.

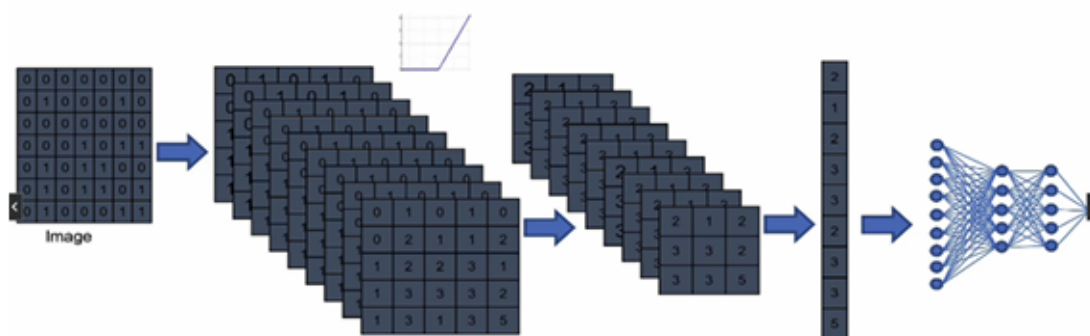


Aula 04 -> Etapa 2 - Pooling: Nesta etapa, o objetivo é tornar a rede neural capaz de reconhecer padrões em diferentes contextos visuais. Mesmo que um cachorro apareça em ambientes variados: como neve ou água. A rede deve conseguir identificar suas características principais. Para isso, utiliza-se o processo de **pooling**, que reduz a dimensão dos dados mantendo as informações mais relevantes.

O mais comum é o **max pooling**, que seleciona o maior valor dentro de uma região do mapa de características, destacando os traços mais fortes da imagem.



Aula 5 - Etapa 3 - Flattening: Após o pooling é retornando uma matriz menor ainda, para que essa informação ser utilizada em uma camada densa, é necessário transformar essa matriz em um vetor unidimensional. É necessário fazer essa transformação pelo fato que daí permite que cada valor no vetor representa algo da imagem



As outras aulas foram totalmente práticas, aplicando os conceitos aprendidos:

Aqui estamos importando as bibliotecas necessárias, separando os dados entre treino e teste, e exibindo os dados de **X_treinamento**

```
[2] import tensorflow as tf
import matplotlib
import keras
import numpy as np
```

```
[3] from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Dense, Flatten, Dropout, Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras import utils as np_utils
import matplotlib.pyplot as plt
```

```
[4] (X_treinamento, Y_treinamento), (X_teste, Y_teste) = mnist.load_data()
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 — 0s 0us/step

divisão do dataset em treino e teste

```
[5] X_treinamento.shape, X_teste.shape
```

```
((60000, 28, 28), (10000, 28, 28))
```

Temos 60 mil imagens diferentes e cada imagem é uma matriz de 28 linhas e 28 colunas

E cada um desses valores da matriz é o dados específico de um pixel dessa imagem

```
[6] X_treinamento
```

```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       ...,

        [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]]])
```

Aqui está sendo exibido os dados contidos em **Y_treinamento**, que representam as classes das imagens do conjunto de dados. E logo depois, uma imagem do conjunto **X_treinamento** é exibida utilizando a biblioteca matplotlib, com o parâmetro **cmap='gray'** para que ela seja mostrada em preto e branco.

```
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0]], dtype=uint8)

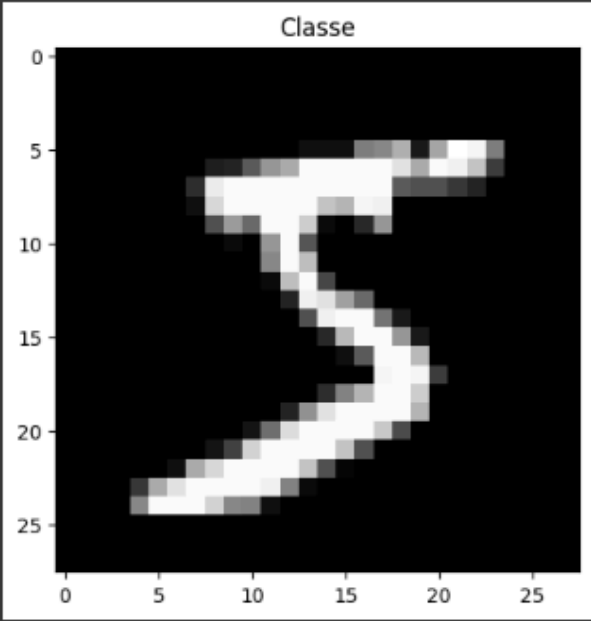
[7] Y_treinamento
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

plt.imshow(X_treinamento[0], cmap='gray') #gray serve para mostrar a imagem em preto e cinza
plt.title('Classe', str(Y_treinamento[0]))

-----
AttributeError                                Traceback (most recent call last)
/tmp/ipython-input-8-3039716751.py in <cell line: 0>()
      1 plt.imshow(X_treinamento[0], cmap='gray') #gray serve para mostrar a imagem em preto e cinza
----> 2 plt.title('Classe', str(Y_treinamento[0]))

-----
3 frames
/usr/local/lib/python3.11/dist-packages/matplotlib/cbook.py in normalize_kwargs(kw, alias_mapping)
    1785     ret = {} # output dictionary
    1786
-> 1787     for k, v in kw.items():
    1788         canonical = to_canonical.get(k, k)
    1789         if canonical in canonical_to_seen:

AttributeError: 'str' object has no attribute 'items'
```



Aqui a gente faz o pré-processamento das imagens e dos rótulos. Depois, os dados viram float e os pixels são normalizados entre 0 e 1. Os rótulos também são transformados em vetores com 10 posições, usando **one-hot encoding**, pra ajudar na hora de treinar o modelo."

```
Pre-processamento

[10] X_treinamento = X_treinamento.reshape(X_treinamento.shape[0], 28, 28, 1) #Adiciona o numero um pelo fato que nossas imagens nao sao coloridas
     X_teste = X_teste.reshape(X_teste.shape[0], 28, 28, 1)

[11] X_treinamento.shape, X_teste.shape
Out: ((60000, 28, 28, 1), (10000, 28, 28, 1))

[12] # Convertendo o tipo de dado do array de treinamento para float32.
     X_treinamento = X_treinamento.astype('float32')
     X_teste = X_teste.astype('float32')

[13] # normalizando para valores entre 0 e 1
     X_treinamento /= 255
     X_teste /= 255

[14] X_treinamento.max(), X_treinamento.min()
Out: (np.float32(1.0), np.float32(0.0))

[15] Y_treinamento
Out: array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

[16] # aplicando one hot encoder nas saídas
     # transformando o número inteiro em uma codificação binária de 10 elementos
     Y_treinamento = np_utils.to_categorical(Y_treinamento, 10)
     Y_teste = np_utils.to_categorical(Y_teste, 10)

[17] Y_treinamento
Out: array([[0., 0., 0., ..., 0., 0., 0.],
           [1., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.],
           ...,
           [0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 0., 0.],
           [0., 0., 0., ..., 0., 1., 0.]])

[18] Y_treinamento[0] #Numero Cinco
Out: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.])
```

Aqui está a nossa estrutura da rede neural convulacional

▼ Estrutura da Rede Neural

```
[27] Rede_neural = Sequential()
Rede_neural.add(InputLayer(shape = (28, 28, 1)))
Rede_neural.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
Rede_neural.add(MaxPooling2D(pool_size=(2, 2)))
Rede_neural.add(Flatten())
Rede_neural.add(Dense(units=128, activation='relu'))
Rede_neural.add(Dense(units=10, activation='softmax'))
```

```
[28] Rede_neural.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
flatten_2 (Flatten)	(None, 5408)	0
dense_4 (Dense)	(None, 128)	692,352
dense_5 (Dense)	(None, 10)	1,290

Total params: 693,962 (2.65 MB)
Trainable params: 693,962 (2.65 MB)
Non-trainable params: 0 (0.00 B)

```
[21] Rede_neural.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Treinando o modelo

```
[22] Rede_neural.fit(X_treinamento, Y_treinamento, batch_size=128, epochs=5, validation_data=(X_teste, Y_teste))
```

Epoch 1/5
469/469 ————— 33s 67ms/step - accuracy: 0.8847 - loss: 0.4245 - val_accuracy: 0.9723 - val_loss: 0.0853
Epoch 2/5
469/469 ————— 26s 55ms/step - accuracy: 0.9800 - loss: 0.0686 - val_accuracy: 0.9802 - val_loss: 0.0573
Epoch 3/5
469/469 ————— 27s 58ms/step - accuracy: 0.9868 - loss: 0.0457 - val_accuracy: 0.9829 - val_loss: 0.0506
Epoch 4/5
469/469 ————— 40s 55ms/step - accuracy: 0.9901 - loss: 0.0328 - val_accuracy: 0.9853 - val_loss: 0.0418
Epoch 5/5
469/469 ————— 26s 56ms/step - accuracy: 0.9929 - loss: 0.0231 - val_accuracy: 0.9859 - val_loss: 0.0477
<keras.src.callbacks.history.History at 0x7a610ef4e750>

```
[23] resultado = Rede_neural.evaluate(X_teste, Y_teste)
```

313/313 ————— 2s 5ms/step - accuracy: 0.9821 - loss: 0.0597

```
[24] resultado
```

[0.047701697796583176, 0.9858999848365784]

Aqui faço algumas otimizações na rede. Para ver se ela funciona melhor.

▼ Otimizando a rede neural

```
[29] Rede_neural = Sequential()
Rede_neural.add(InputLayer(shape = (28, 28, 1)))

Rede_neural.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
Rede_neural.add(BatchNormalization())
Rede_neural.add(MaxPooling2D(pool_size=(2, 2)))

Rede_neural.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
Rede_neural.add(BatchNormalization())
Rede_neural.add(MaxPooling2D(pool_size=(2, 2)))

Rede_neural.add(Flatten())

Rede_neural.add(Dense(units=128, activation='relu'))
Rede_neural.add(Dropout(0.2))

Rede_neural.add(Dense(units=128, activation='relu'))
Rede_neural.add(Dropout(0.2))

Rede_neural.add(Dense(units=10, activation='softmax'))
```

```
[30] Rede_neural.summary()
```

Model: "sequential_3"

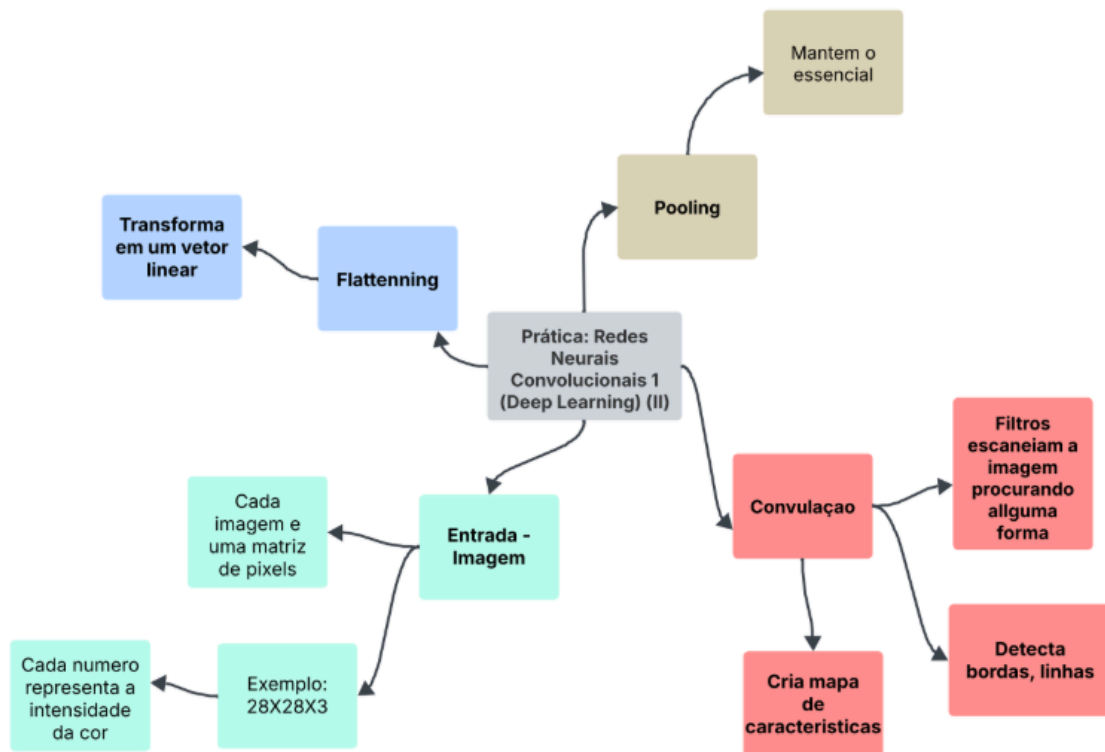
Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
batch_normalization (BatchNormalization)	(None, 26, 26, 32)	128
max_pooling2d_3 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_4 (Conv2D)	(None, 11, 11, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 11, 11, 32)	128
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 32)	0
flatten_3 (Flatten)	(None, 800)	0
dense_6 (Dense)	(None, 128)	102,528
dropout (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 128)	16,512
dropout_1 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 10)	1,290

Total params: 130,154 (508.41 KB)

Trainable params: 130,026 (507.91 KB)

Non-trainable params: 128 (512.00 B)

Insight visual original sobre o conteúdo estudado no Card 15



Conclusão: Ao final do card aprendi que as redes neurais convolucionais (CNNs) são ferramentas extremamente eficazes para o processamento e reconhecimento de imagens. Embora algumas coisas pareçam com as redes neurais tradicionais, as CNNs se destacam pelo uso de mais algumas camadas: **convolução** e **pooling**. Isso faz com que possam pegar extração de padrões visuais com maior precisão. Com certeza esse card foi muito importante para o meu conhecimento.

Referências:

Bootcamp - Lamia - Card 15