Relatório 15 - Prática: Redes Neurais Convolucionais

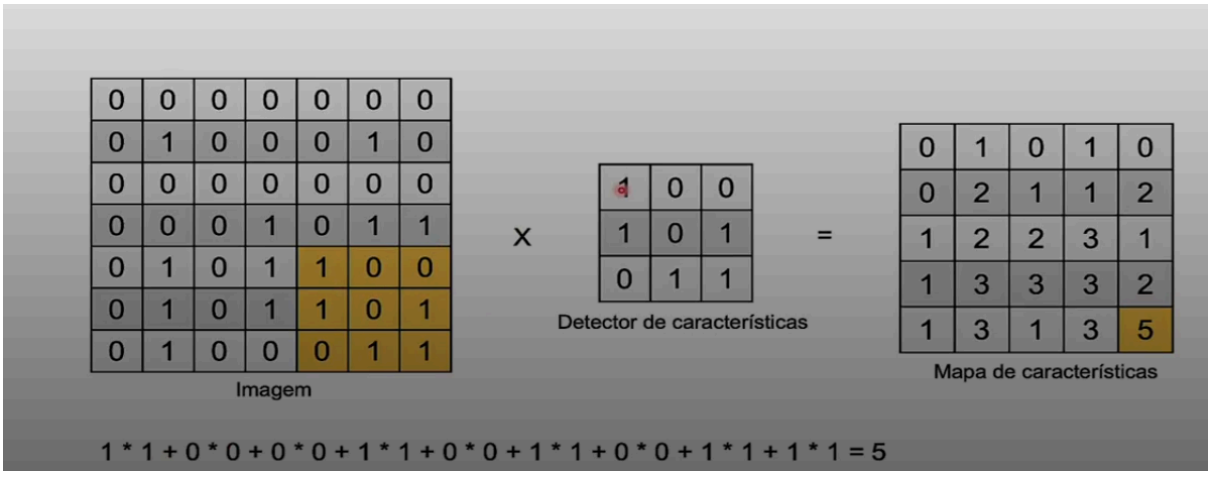
Jefferson korte junior

Seção 9 - Teoria sobre as redes Convolucionais.

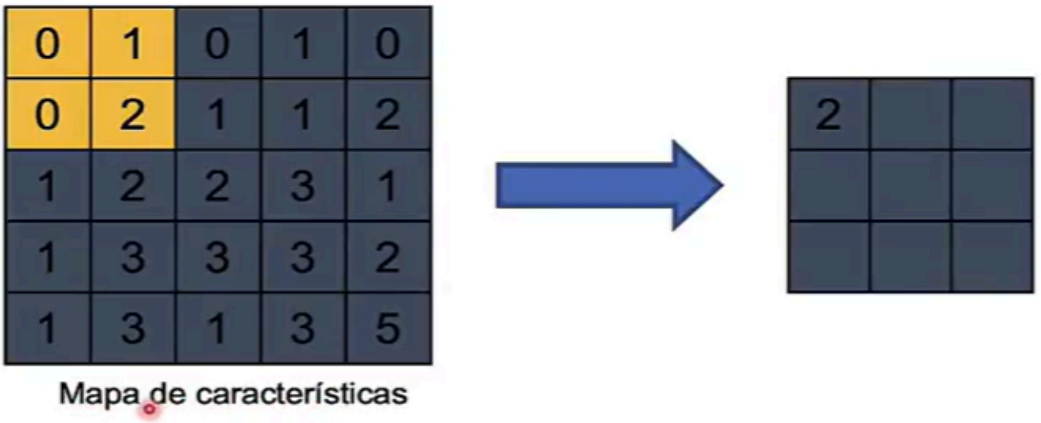
Aula 01 -> imagens e pixels: Nessa aula aprendemos oque é pixel e que pixel é a menor informação possível em uma imagem, e que cada pixel tem uma combinação de 3 cores, (RGB) com a combinação dessas 3 cores podemos chegar em qualquer outra.

Aula 02 -> Introdução a redes neurais convolucionais: No deep learning, os pixels servem como a entrada para os modelos. Redes neurais convolucionais (CNNs), por exemplo, processam esses pixels para extrair características como bordas, texturas e até padrões mais complexos, que são essenciais para reconhecimento de objetos e tarefas de classificação e dentro da rede neural há 4 passos, 1 - Operador de convolução, 2 - Pooling, 3 - Flattening, 4 - Rede neural densas.

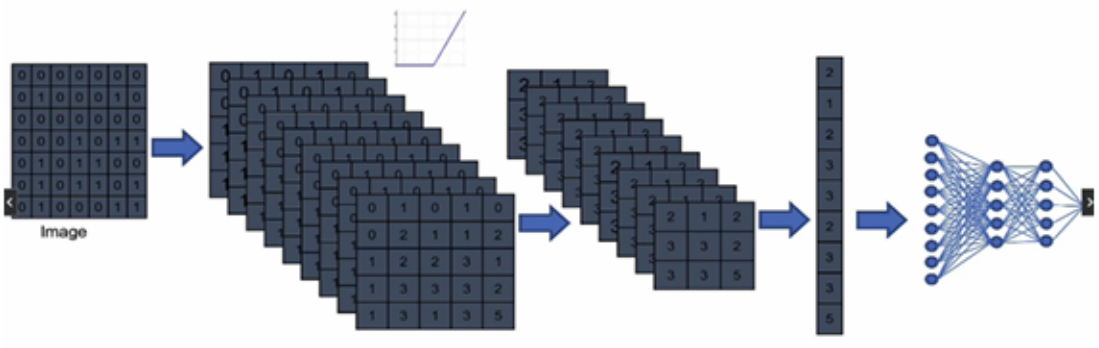
Aula 03 -> Etapa 1 - operador de convolução O operador de convolução é uma operação matemática fundamental em visão computacional, especialmente em redes neurais convolucionais, utilizada para extrair características importantes de uma imagem, como bordas, texturas e padrões. Esse processo funciona através da aplicação de um pequeno **filtro** (ou **kernel**) ou **(Detector de características)** geralmente no formato de uma matriz 3x3 ou 5x5, que percorre toda a imagem original. Em cada posição, o filtro realiza uma multiplicação ponto a ponto com os pixels cobertos e soma os resultados obtidos. O valor final dessa soma é atribuído à posição correspondente em uma nova imagem chamada **mapa de características**. Em geral, são usados **vários filtros diferentes**, e cada um gera seu **próprio mapa de características**, permitindo que a rede aprenda e represente diversos padrões da imagem de entrada. Além de destacar características visuais relevantes, a convolução também pode reduzir a dimensionalidade da imagem.



Aula 04 -> Etapa 2 - Pooling: A ideia dessa segunda etapa é que a rede consiga se adaptar aos diversos tipos de ambiente de uma imagem, ainda deve ser possível identificar que seja um cachorro em tal imagem mesmo que ele esteja na neve ou na água. Existem vários tipos de pooling, e o mais utilizado é o max pooling, onde é feita a extração do maior valor dentro do mapa de características:



Aula 5 - Etapa 3 - Flattening: Após o pooling é retornando uma matriz menor ainda, e transformamos essa matriz em um vetor, é necessário transformar em vetor, pois esses dados vão ser enviados para uma camada de entrada de uma rede neural densa.



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

```
[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 e uma matriz de 28 linhas e 28 colunas

E cada um desses valores da matriz e o dados especifico 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]]], dtype=uint8)
```

```
[7] Y_treinamento
```

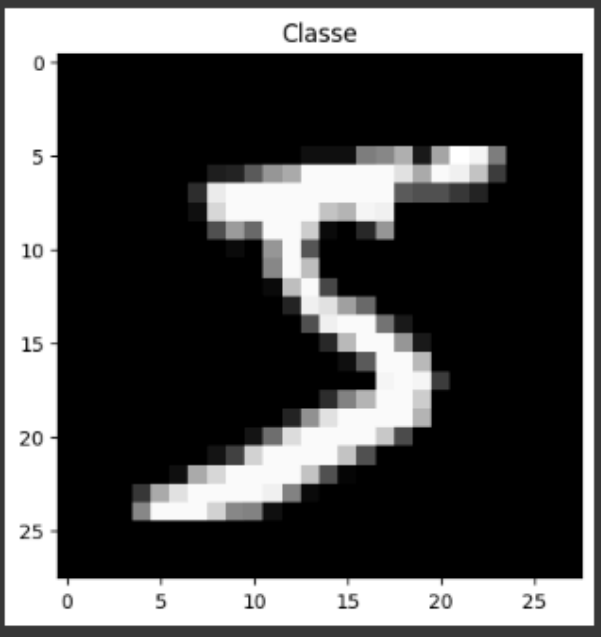
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
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'
```



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
```

```
((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()
```

```
(np.float32(1.0), np.float32(0.0))
```

```
[15] Y_treinamento
```

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

```
[16] # aplicando one hot enconder 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
```

```
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
```

```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.])
```

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]
```

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)
```

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

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

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 59s 120ms/step - accuracy: 0.8528 - loss: 0.4638 - val_accuracy: 0.9669 - val_loss: 0.1179
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 81s 118ms/step - accuracy: 0.9799 - loss: 0.0677 - val_accuracy: 0.9883 - val_loss: 0.0380
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 83s 119ms/step - accuracy: 0.9851 - loss: 0.0502 - val_accuracy: 0.9886 - val_loss: 0.0341
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 82s 119ms/step - accuracy: 0.9881 - loss: 0.0377 - val_accuracy: 0.9885 - val_loss: 0.0374
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 56s 120ms/step - accuracy: 0.9905 - loss: 0.0301 - val_accuracy: 0.9894 - val_loss: 0.0366
<keras.src.callbacks.history.History at 0x7a60f4d73a50>
```

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9869 - loss: 0.0438
```

```
[34] resultado
```

```
[0.03658820316195488, 0.9894000291824341]
```

validaçao cruzada

```
[35] from sklearn.model_selection import StratifiedKFold

     kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
[36] (X, y), (X_teste, y_teste) = mnist.load_data()
```

```
[37] X = X.reshape(X.shape[0], 28, 28, 1)
     X_teste = X_teste.reshape(X_teste.shape[0], 28, 28, 1)
```

```
[38] X = X.astype('float32')
     X_teste = X_teste.astype('float32')
```

```
[39] X /= 255
     X_teste /= 255
```

```
[40] y = np_utils.to_categorical(y, 10)
     y_teste = np_utils.to_categorical(y_teste, 10)
```

```
[41] resultados = []
     for indice_treinamento, indice_teste in kfold.split(X, np.zeros(shape=(y.shape[0], 1))):
         print(f'Indice treinamento: {indice_treinamento} \t Indice teste: {indice_teste}')
         rede_neural = Sequential()
         rede_neural.add(InputLayer(shape=(28, 28, 1)))
         rede_neural.add(Conv2D(filters = 32, kernel_size=(3, 3), activation='relu')) # Conv2D(quantidade_filtros, tamanhos_filtros, função_ativação)
         rede_neural.add(MaxPooling2D(pool_size=(2, 2)))
```

Argumentation

```
[43] from tensorflow.keras.preprocessing.image import ImageDataGenerator #usada para pré-processamento e aumento de dados de imagem
```

```
[44] 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'))
```

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

```
[47] gerador_treinamento = ImageDataGenerator(
         rotation_range=7,         # Rotação aleatória de até 7 graus
         horizontal_flip=True,     # Espelhamento horizontal aleatório
         shear_range=0.2,          # Distorção no eixo (cisalhamento)
         height_shift_range=0.07,  # Deslocamento vertical aleatório
         zoom_range=0.2            # Zoom aleatório nas imagens
     )
```

```
[48] gerador_teste = ImageDataGenerator()  # Gera dados de teste sem transformações; usado apenas para normalizar as imagens
```

```
base_treinamento = gerador_treinamento.flow(X_treinamento, Y_treinamento, batch_size=128)  # Aplica aumento de dados e gera lotes para treino
base_teste = gerador_teste.flow(X_teste, Y_teste, batch_size=128)  # Gera lotes de teste sem aumento de dados
```

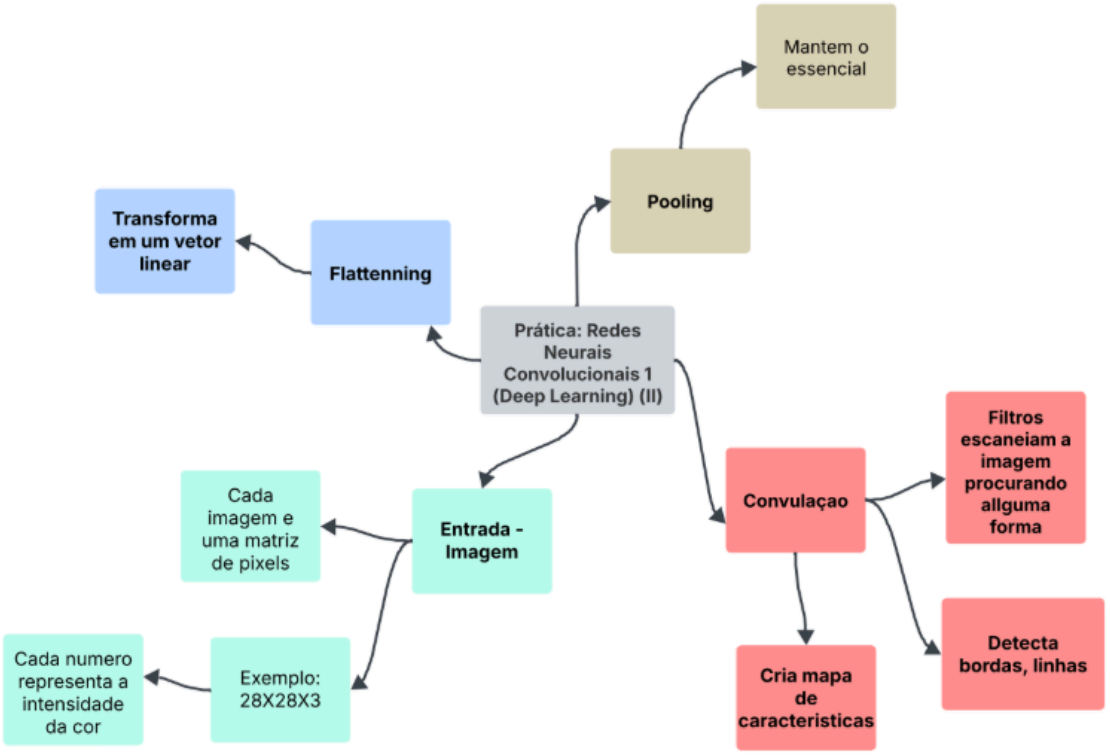
Nao conseguimos ver essas novas imagens

```
[50] Rede_neural.fit(base_treinamento, epochs=5, validation_data=(base_teste))
```

```
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call
  self._warn_if_super_not_called()
469/469 ━━━━━━━━━━━━━━━━━━━━ 45s 93ms/step - accuracy: 0.7594 - loss: 0.7659 - val_accuracy: 0.9508 - val_loss: 0.1572
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 81s 91ms/step - accuracy: 0.9291 - loss: 0.2311 - val_accuracy: 0.9636 - val_loss: 0.1123
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 44s 95ms/step - accuracy: 0.9448 - loss: 0.1796 - val_accuracy: 0.9696 - val_loss: 0.0939
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 42s 90ms/step - accuracy: 0.9569 - loss: 0.1412 - val_accuracy: 0.9698 - val_loss: 0.0955
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 42s 89ms/step - accuracy: 0.9606 - loss: 0.1233 - val_accuracy: 0.9779 - val_loss: 0.0710
<keras.src.callbacks.history.History at 0x7a60efdd4690>
```

```
[ ] Comece a programar ou gere código com IA.
```

Insight visual original sobre o conteúdo estudado no Card 15



Conclusão: Minha conclusão sobre o curso é que redes neurais convolucionais (CNNs) são extremamente eficazes para o processamento de imagens. Percebi que sua estrutura não é tão diferente das redes neurais tradicionais, com a principal diferença sendo o uso de camadas específicas para padrões visuais. E também a normalização dos dados para imagens é um pouco diferente.

Referências:

Bootcamp - Lamia - Card 15