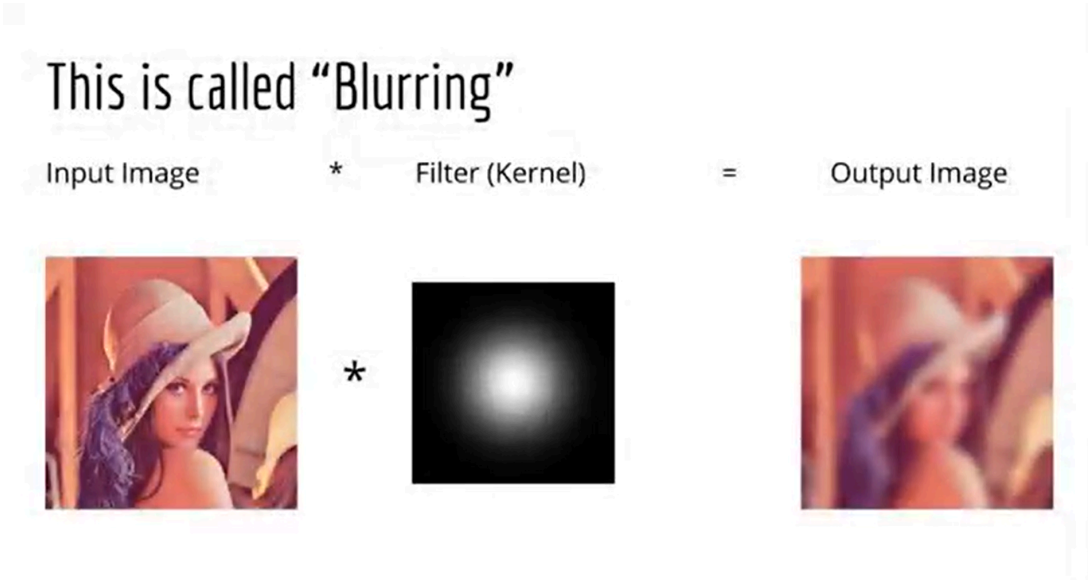
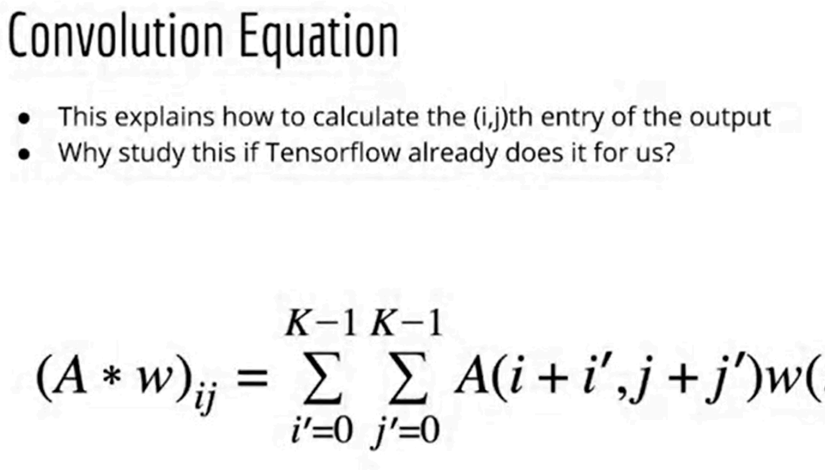
Relatório - Card 16 - Prática: Redes Neurais Convolucionais 2 (Deep Learning) (II)

Jefferson korte junior

Seção 5:

Aula 29, 30, 31 - What is convolution: Aprendi que convolução é como passar um filtro sobre a imagem para destacar partes importantes. É isso que ajuda a rede a entender o que tem na imagem. Também vimos que existe uma forma de calcular a saída com base nesse processo.



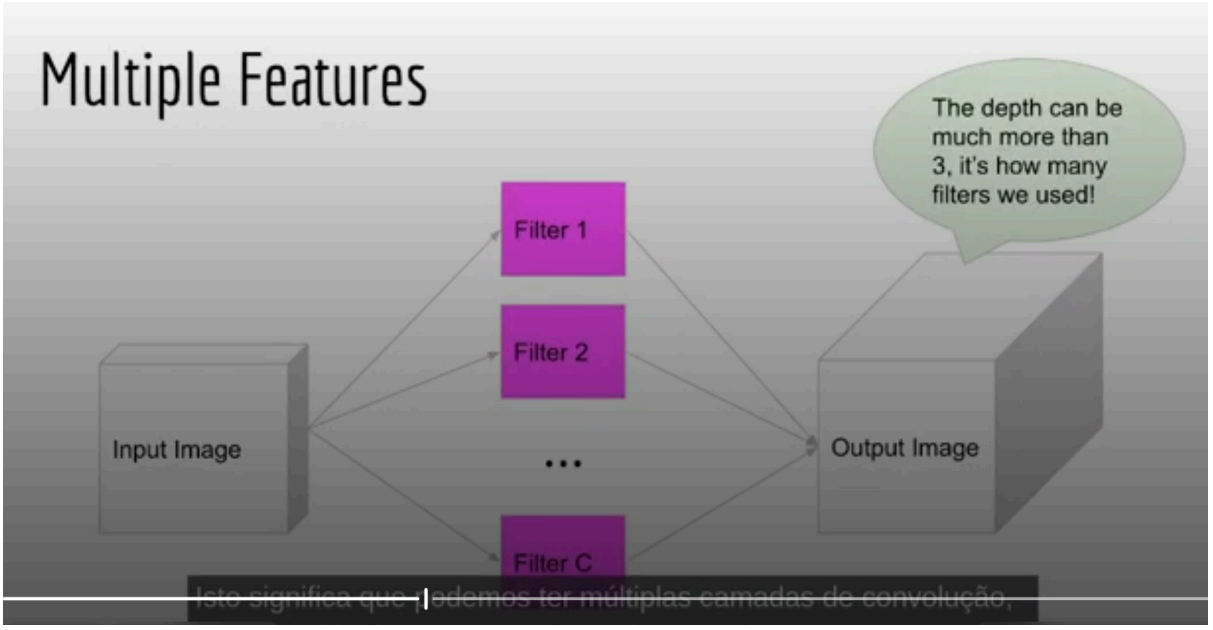


Aula 32 - Why use 0-indexing: Vi que começar a contar do zero é comum em programação, e nas redes neurais isso ajuda nos cálculos e na organização da memória.

Aula 33 - Convolution on Color Images: Aprendi que imagens coloridas têm uma dimensão a mais que as em preto e branco, e isso muda os cálculos da convolução.

$$(A * w)_{ij} = \sum_{c=1}^{3} \sum_{i'=0}^{K-1} \sum_{j'=0}^{K-1} A(i+i', j+j', c) w(i', j', c)$$

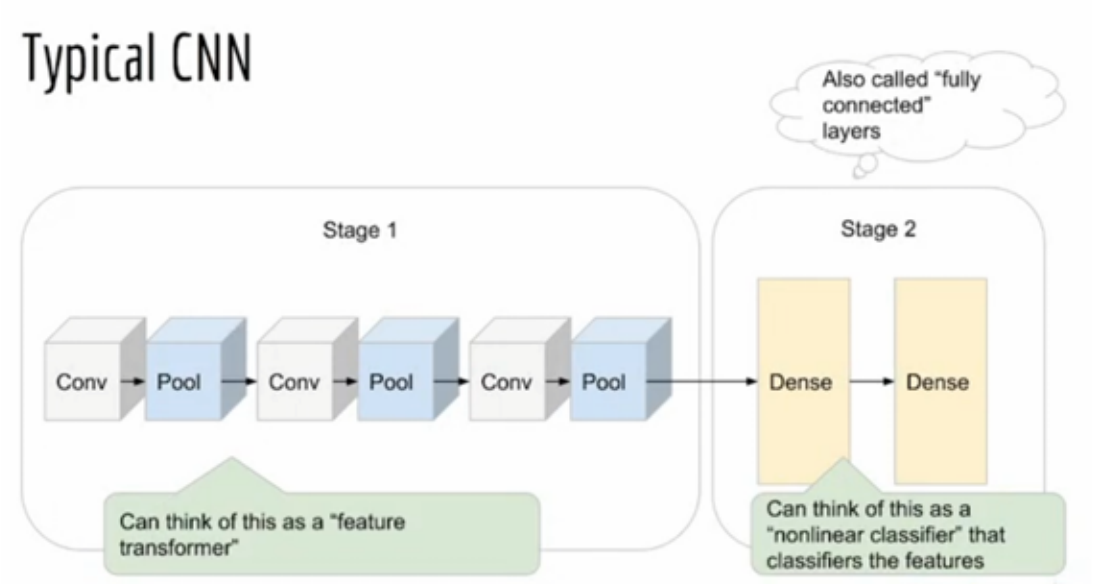
Também vi que dá pra usar vários filtros, e cada um gera uma imagem diferente.



Além disso, entendi que usar convolução ajuda a **reduzir o tamanho da rede** e o número de parâmetros. Por exemplo, com uma imagem de entrada 32x32x3 e filtro 5x5, a saída fica menor (28x28x64) e usa bem menos memória comparado com uma rede densa, que precisaria de muito mais RAM e tempo para treinar.

Aula 34 - CNN Architecture: Nessa aula entendi como é formada uma rede neural convolucional. Ela tem várias camadas que ajudam a entender a imagem e fazer a classificação.
Tem a entrada, onde os dados chegam; depois a convolução, que usa filtros pra pegar os padrões; o pooling, que diminui o tamanho; o flatten, que transforma tudo em vetor; a camada densa, que junta tudo; e por fim a saída, que dá o resultado da previsão.

Imagem:



Essas camadas de pooling, que vêm depois da convolução, servem pra diminuir o tamanho da imagem, mas mantendo o que é mais importante. Isso ajuda a rede a usar menos memória e fazer menos cálculos.
Antes de passar pra camada densa, que só aceita vetores, a gente precisa achatar os dados usando o Flatten ou o **GlobalMaxPooling2D**.

Aulas 36 e 37 é ensinado na prática o que foi ensinado durante as aulas anteriores, códigos utilizando a base de dados do tensor flow, como a **cifar 10** e **mnist fashion**

Aqui eu importo as bibliotecas, carrego o dataset, normalizo os dados e transformo tudo em vetor pra poder usar na rede neural. Depois conto quantas classe diferentes tem nos dados de treino e em seguida faço a estrutura da rede neural

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.models import Model
```

```
[ ] cifar10 = tf.keras.datasets.cifar10
    (x_train, y_train), (x_test, y_test) = cifar10.load_data() #carregamento base de dados

    x_train, x_test = x_train / 255.0, x_test / 255.0 #normaliznado em 0 até 1
    y_train, y_test = y_train.flatten(), y_test.flatten() #deixando em um vetor

    print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ——————————— 4s 0us/step
(50000, 32, 32, 3) (10000, 32, 32, 3) (50000,) (10000,)
```

```
[ ] K = len(set(y_train))
    K
```

```
10
```

```
[ ] i = Input(shape=x_train[0].shape) #camada de entrada com forma de 32x32x3

    x = Conv2D(32, (3, 3), strides=2, activation='relu')(i) #primeira convolução
    x = Conv2D(64, (3, 3), strides=2, activation='relu')(x) #segunda
    x = Conv2D(128, (3, 3), strides=2, activation='relu')(x) #terceira

    x = Flatten()(x) #deixando em apenas um vetor para a camada densa

    x = Dropout(0.2)(x) #dropout para evitar overffiting
    x = Dense(1024, activation='relu')(x) #camadad densa
    x = Dropout(0.2)(x) #denovo, evitando overfitting

    x = Dense(K, activation='softmax')(x) #camada de saída

    model = Model(i, x)
```

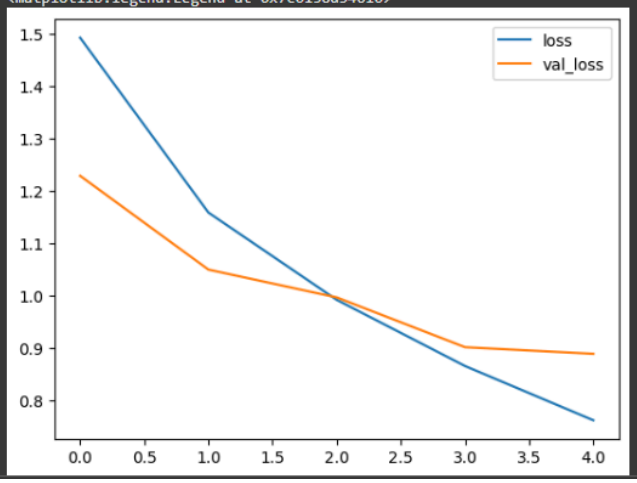
Aqui eu compilo e treino o modelo. Depois disso, faço um gráfico pra ver como ficou o erro (loss) e a acurácia (acc)

```
[5] model.compile(optimizer = 'adam',
                  loss = 'sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    r = model.fit(x_train,y_train, validation_data=(x_test, y_test), epochs = 5)
```

```
Epoch 1/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 35ms/step - accuracy: 0.3669 - loss: 1.7007 - val_accuracy: 0.5572 - val_loss: 1.2292
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 55s 35ms/step - accuracy: 0.5754 - loss: 1.1920 - val_accuracy: 0.6243 - val_loss: 1.0499
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 35ms/step - accuracy: 0.6467 - loss: 1.0011 - val_accuracy: 0.6483 - val_loss: 0.9967
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 86s 37ms/step - accuracy: 0.6933 - loss: 0.8650 - val_accuracy: 0.6889 - val_loss: 0.9016
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 78s 35ms/step - accuracy: 0.7343 - loss: 0.7451 - val_accuracy: 0.6909 - val_loss: 0.8890
```

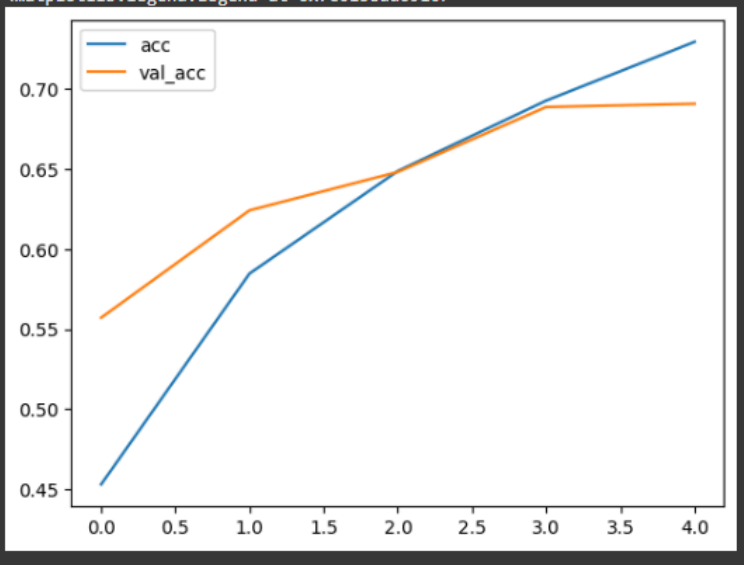
```
[6] plt.plot(r.history['loss'], label='loss')
    plt.plot(r.history['val_loss'], label='val_loss')
    plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c0158d34610>
```




```
plt.plot(r.history['accuracy'], label='acc')
plt.plot(r.history['val_accuracy'], label='val_acc')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c0158dd0910>
```



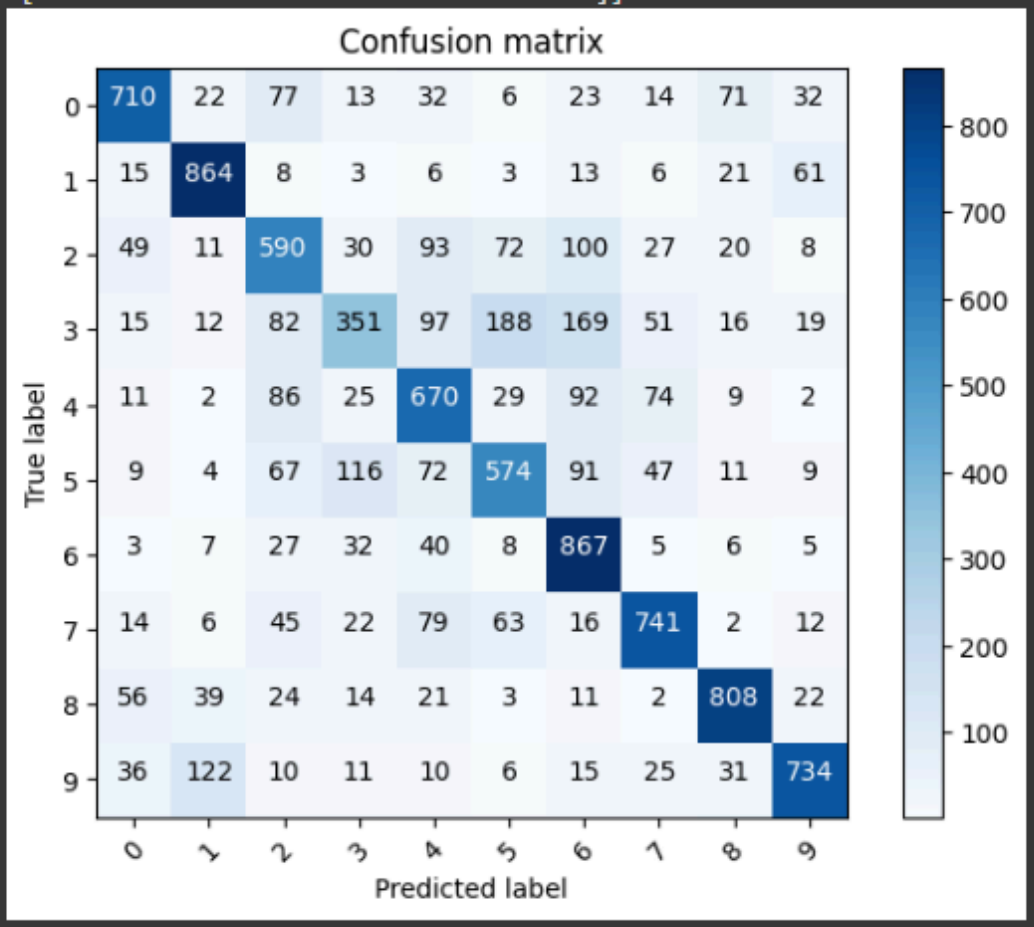
Aqui eu crio uma função pra mostrar a matriz de confusão e ver como o modelo tá

```
[8]
from sklearn.metrics import confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    if normalize:
      cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
      print("Normalized confusion matrix")
    else:
      print('confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
      plt.text(j, i, format(cm[i, j], fmt),
               horizontalalignment="center",
               color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

p_test = model.predict(x_test).argmax(axis=1)
cm = confusion_matrix(y_test, p_test)
plot_confusion_matrix(cm, list(range(10)))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step
confusion matrix, without normalization
[[710  22  77  13  32   6  23  14  71  32]
 [ 15 864   8   3   6   3  13   6  21  61]
 [ 49  11 590  30  93  72 100  27  20   8]
 [ 15  12  82 351  97 188 169  51  16  19]
```

```
[8]     plt.show()

p_test = model.predict(x_test).argmax(axis=1)
cm = confusion_matrix(y_test, p_test)
plot_confusion_matrix(cm, list(range(10)))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step
confusion matrix, without normalization
[[710  22  77  13  32   6  23  14  71  32]
 [ 15 864   8   3   6   3  13   6  21  61]
 [ 49  11 590  30  93  72 100  27  20   8]
 [ 15  12  82 351  97 188 169  51  16  19]
 [ 11   2  86  25 670  29  92  74   9   2]
 [  9   4  67 116  72 574  91  47  11   9]
 [  3   7  27  32  40   8 867   5   6   5]
 [ 14   6  45  22  79  63  16 741   2  12]
 [ 56  39  24  14  21   3  11   2 808  22]
 [ 36 122  10  11  10   6  15  25  31 734]]
```



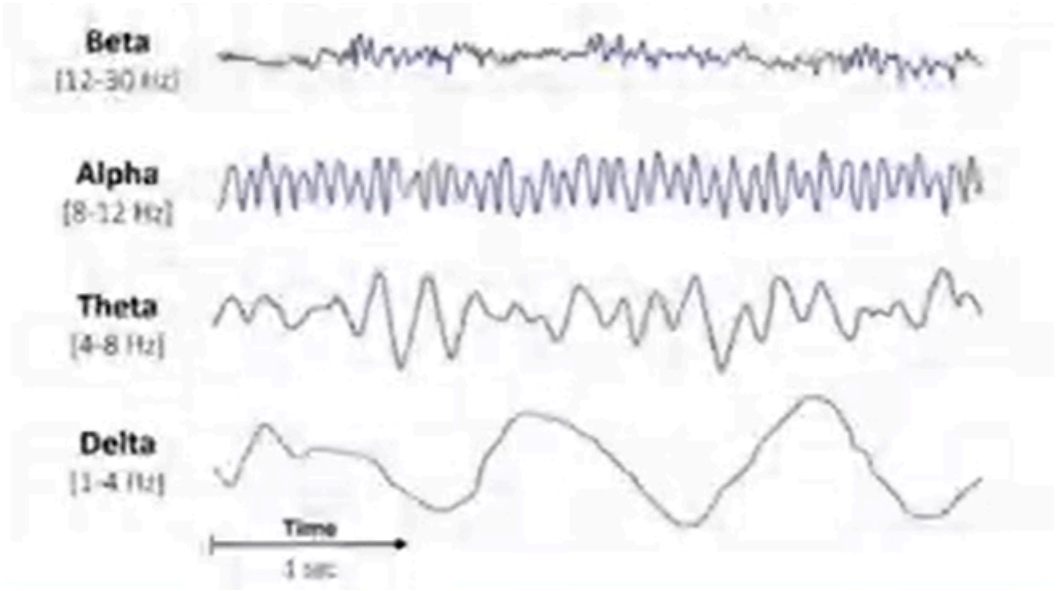
Aula 38 - Data Argumentation: Aprendi que, em redes neurais, quanto mais dados, melhor. Quando não temos muitos, usamos o Data Augmentation, que gira, amplia ou modifica imagens já existentes, pra ensinar a rede a reconhecer o mesmo objeto em

posições diferentes. Isso é muito útil quando o conjunto de dados é pequeno ou desequilibrado.

Aula 39 - Batch Normalization: Vi que o Batch Normalization serve pra padronizar os dados dentro da rede, mantendo eles com média próxima de zero e variância próxima de um. Isso evita que os dados “se baguncem” ao passar pelas camadas e ajuda a treinar mais rápido e com mais estabilidade.

Seção 6 - Natural Language Processing (NLP):

Embeddings: Aprendi que as CNNs também funcionam com sequências, não só com imagens. Em vez de trabalhar com uma matriz de pixels, usamos um vetor e um filtro que passa por ele. Isso ajuda a encontrar padrões locais nos dados, assim como nas imagens. A ideia é parecida: dados próximos costumam ter valores parecidos, e a rede aprende com isso.



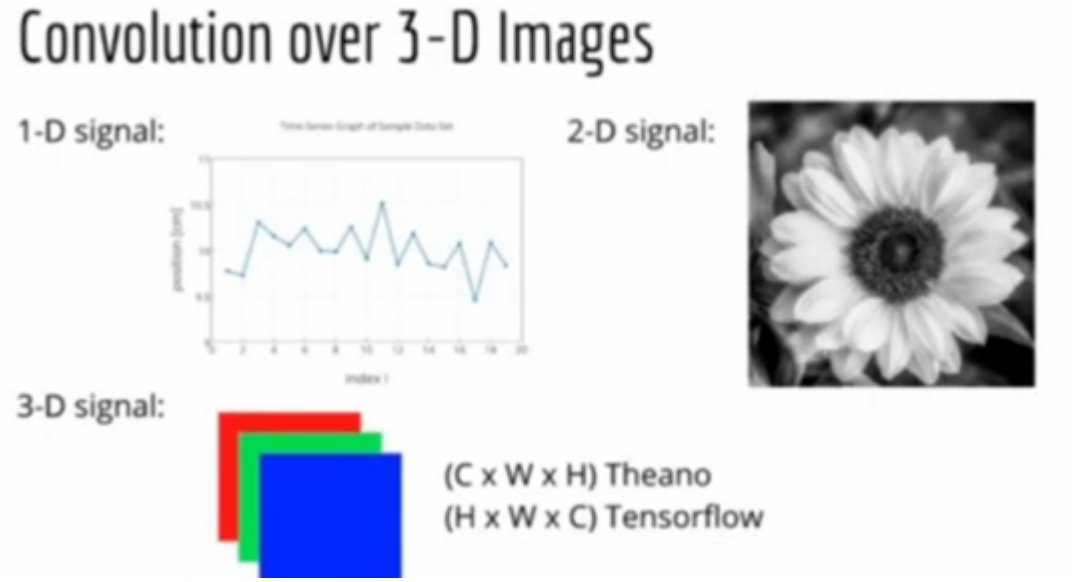
Seção 7 - Convolution In-Depth:

Aula 46 - Real-Life Examples of Convolution: Um dos primeiros exemplos é com áudio: aplicamos um efeito ou filtro e obtemos um novo som. Isso mostra que a convolução pega um sinal de entrada e devolve outro sinal parecido mas modificado.

Aula 47 - Beginner's Guide to Convolution: A aula explica que a convolução envolve apenas soma e multiplicação. Mostra como multiplicar uma matriz de entrada por um filtro para gerar a imagem de saída.

Seção 8 - Convolutional Neural Network Description

Aula 49 - Convolution on 3-D Images: Aprendemos como lidar com imagens coloridas (RGB), que têm 3 dimensões. Um exemplo é a detecção de bordas, onde filtros identificam mudanças bruscas nos pixels, destacando os contornos dos objetos. Esses filtros ajudam a rede a enxergar melhor as formas da imagem.



Aula 50 - Tracking Shapes in a CNN: Em uma CNN, a imagem passa por camadas que extraem padrões (como bordas e formas) e depois por camadas que reduzem o tamanho desses dados, deixando só o mais importante. No fim, camadas densas usam essas informações para classificar ou fazer outras tarefas. O tamanho dos filtros, por exemplo, é definido manualmente e testado.

Seção 9: Aprendi que uma uma CNN geralmente é dividida em duas partes: camadas de convolução, que extraem as características da imagem, e camadas totalmente conectadas, que usam essas informações para, por exemplo, classificar algo.

A VGG16 é um exemplo ela tem 16 camadas no total, sendo 13 de convolução e 3 conectadas. O principal aprendizado aqui é que, para criar uma CNN, é importante estudar modelos já existentes e testar bastante, porque a base costuma ser parecida, mas cada rede pode reagir diferente às mudanças.

Secao 10 - In-Depth: Loss Functions

Aula 52 - mean Squared Error: nessa aula vi que o MSE é usado para medir a diferença entre o valor previsto e o real. Os erros são elevados ao quadrado para evitar que números negativos se cancelem. Assim, conseguimos ver melhor o quanto o modelo está errando.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y - \hat{y})^2$$

:

Aula 53 - Binary Cross-Entropy: Já nessa aula aprendi que BCE é a função de perda mais usada em classificações com duas opções (sim ou não, spam ou não spam). Ela compara a probabilidade prevista com a resposta real (0 ou 1) e mostra o quanto o modelo errou.

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\{yi \,.\, log(\hat{y}i) + (1 - yi) \,.\, log(1 - \hat{y}i)\}$$

Aula 54 - Categorical Cross-Entropy: Usada quando tem várias classes. Compara o que era pra ser com o que o modelo previu. Quanto mais errado, maior a perda. O treino tenta diminuir isso.

$$Categorical\ Cross - Entropy = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log \hat{y}_{ik}$$

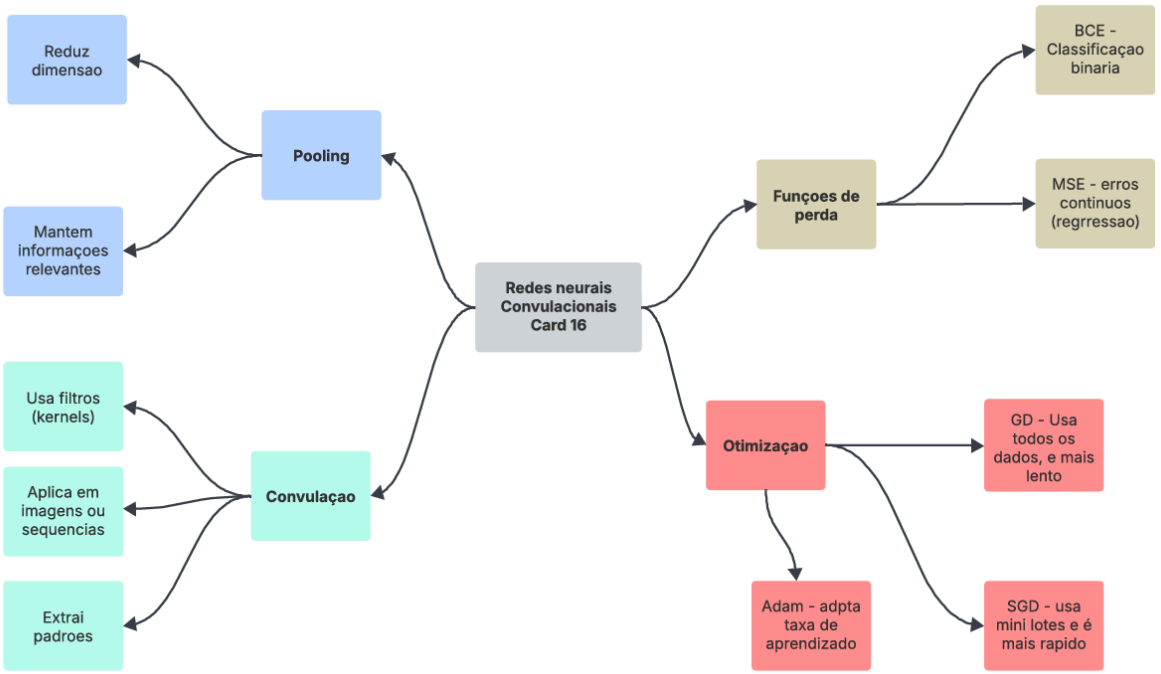
Seção 11 - In-Depth: Gradient Descent

Aula 55 - Gradient Descent: Aprendi que a decida do Gradiente é uma técnica usada para ajustar os pesos da rede e melhorar as previsões. ela tenta minimizar os erros do modelo, encontrando os melhores valores dos parâmetros

Aula 56 - Stochastic Gradient Descent: Parecido com o Gradient Descent, mas em vez de usar todos os dados de uma vez, o SGD atualiza os pesos com cada amostra ou mini-lote. Isso deixa o treinamento mais rápido

Aula 59 - Adam : O Adam é um otimizador que melhora o SGD. Ele muda a taxa de aprendizado sozinho, deixando o treino mais rápido e estável.

insight visual original sobre o conteúdo aprendido no card - 16



Conclusão: Concluo que as Redes Neurais Convolucionais (CNNs) se mostram extremamente poderosas e versáteis. Desde os fundamentos da convolução até aplicações mais avançadas, como o processamento de texto e o uso de técnicas de otimização, é certo como essa arquitetura transformou áreas como a visão computacional e o processamento de sinais.

Referências:

Bootcamp - Lamia - Card 16