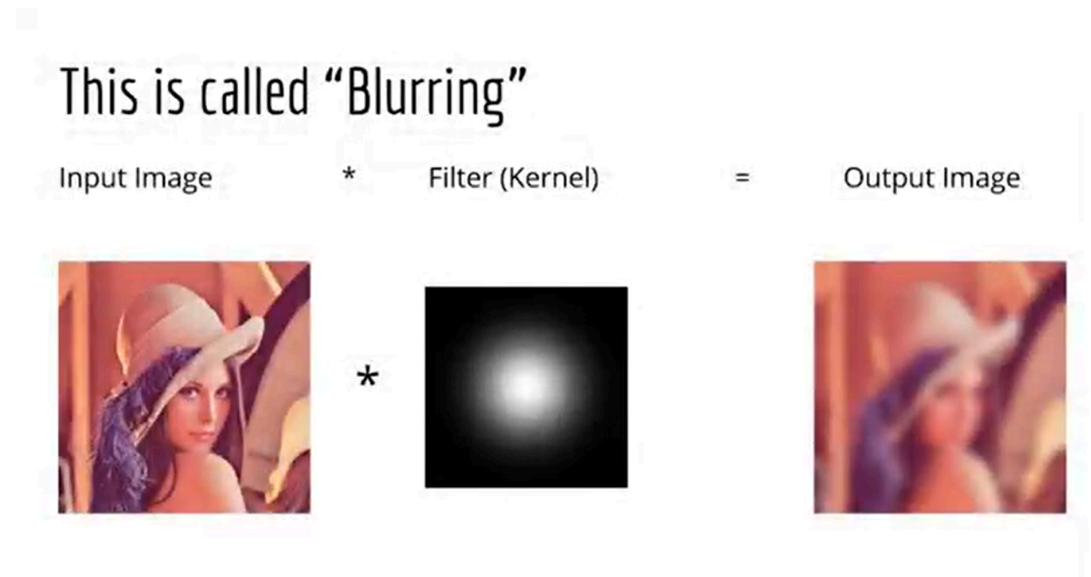


## Relatório - Card 16 - Prática: Redes Neurais Convolucionais 2 (Deep Learning) (II)

Jefferson korte junior

### Seção 5:

**Aula 29, 30, 31 - What is convolution:** Uma forma não matemática de pensar em convolução é apenas pegar uma imagem, passar por um filtro ou um 'kernel' e ter uma imagem de saída. **Exemplo:**



Também existe um calculo para saber a imagem de saída:

### Convolution Equation

- This explains how to calculate the (i,j)th entry of the output
- Why study this if Tensorflow already does it for us?

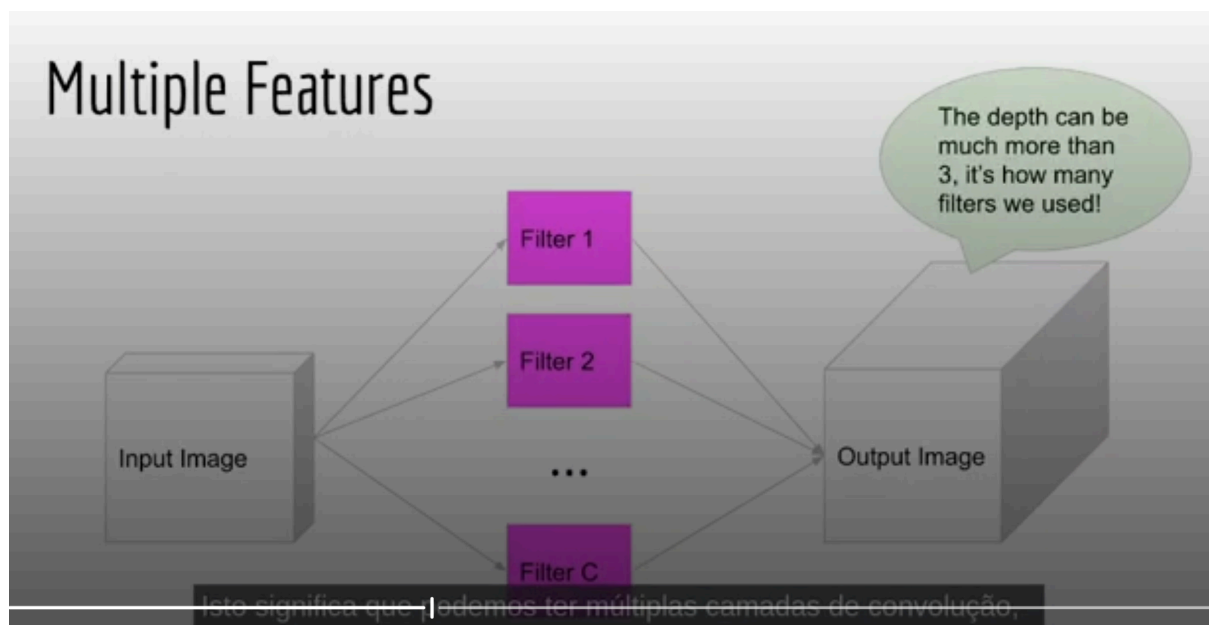
$$(A * w)_{ij} = \sum_{i'=0}^{K-1} \sum_{j'=0}^{K-1} A(i + i', j + j') w(i', j')$$

**Aula 32 - Why use 0-indexing:** Iniciar índices com zero na computação não é incomum e em redes neurais também é uma convenção amplamente adotada, por razões práticas e matemáticas. A indexação em zero facilita, por exemplo, o cálculo de endereços de memória.

**Aula 33 - Convolution on Color Images:** Quando tratamos de imagens coloridas estamos tratando de uma dimensão a mais dentro da matriz, o cálculo apresentado na imagem acima estava levando em conta a imagem esteja em uma escala de cinza. Caso ela seja colorida a soma fica diferente

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

Também é ensinado que podemos utilizar vários filtros e obter cada um uma imagem diferente.



Nessa aula ele demonstra como a convolução pode ajudar a diminuir o tamanho da rede, num caso de uma imagem de entrada de 32X32X3 e um filtro de 3X5X5X64 resultaria numa imagem de saída com 28X28X64(32-5+1=28) totalizando 4800 parâmetros necessários para a saída

- Let's calculate our savings from doing convolution instead of matrix multiplication
- Input image: 32 x 32 x 3
- Filter: 3 x 5 x 5 x 64
- Output image: 28 x 28 x 64 (32 - 5 + 1 = 28)
- # of parameters (ignoring bias term) = 3 x 5 x 5 x 64 = 4800

utilizando uma Rede Neural Densa:

Now consider a Dense layer (full matrix multiply)  
Flattened input image:  $32 \times 32 \times 3 = 3072$   
Flattened output vector:  $28 \times 28 \times 64 = 50176$   
Weight matrix:  $3072 \times 50176 = 154,140,672 \sim 154 \text{ MILLION}$

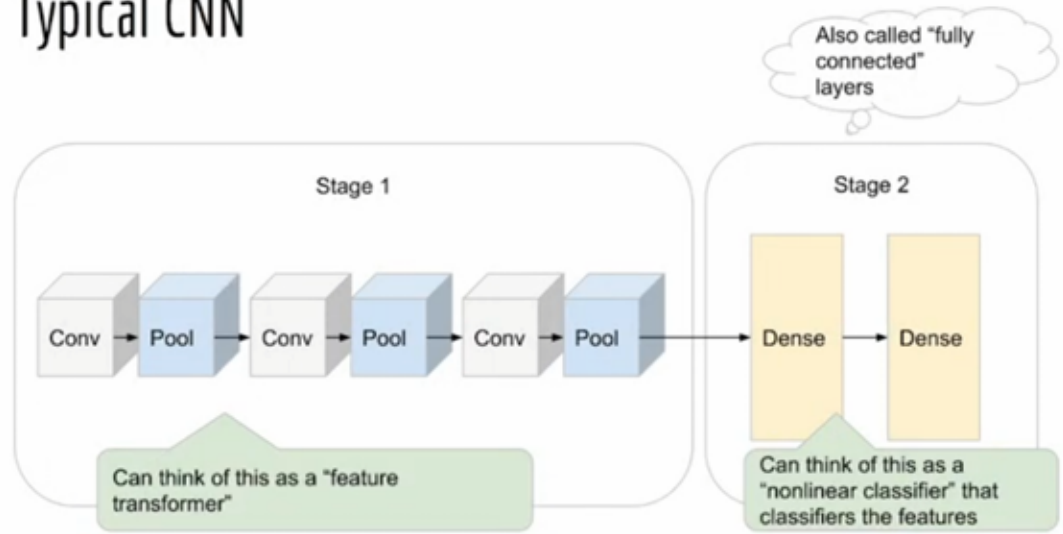
Comparando é uma diferença de 32000 vezes maior do que se usássemos a convolução, exigindo uma enorme quantidade de RAM e de tempo computacional.

**Aula 34 - CNN Architecture:** A estrutura básica de uma CNN (Convolutional Neural Network) é composta por camadas que recebem os dados de entrada, para extrair características e realizar tarefas como classificação ou detecção. As principais camadas são:

- Entrada: Recebe os dados pré-processados e no formato esperado pelo modelo.
- Convolução: Aplica filtros (kernel's) nos dados de entrada para detectar características. Os filtros compartilham pesos, o que reduz a quantidade de parâmetros e permite a detecção de padrões.
- Pooling: Reduz a dimensão da saída da camada de convolução, preservando as características mais importantes.
- Flatten: Faz a conversão em vetores de uma dimensão que serão recebidos como entrada pela camada densa.
- Densa: Conecta todos os neurônios de uma camada à próxima, geralmente usada no final da rede para combinar as características extraídas e realizar a classificação.
- Saída: Produz as previsões.

Imagem:

## Typical CNN



Essas camadas de pool após as camadas de convolução servem para reduzir a dimensão das imagens, como já tínhamos apreendido anteriormente. **Exemplo:**



O pooling é uma técnica de redução de dimensionalidade usada em redes neurais convolucionais para diminuir o tamanho espacial das imagens, mantendo as características mais importantes, isso ajuda a termos menos dados para processar e menos multiplicações a se fazer.

Após as camadas de seguida de convolução e Pooling, se tem a camada densa, porém essa camada aceita dados em uma dimensão só, aí é necessário usar recursos como 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**

```
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 #normalizado 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

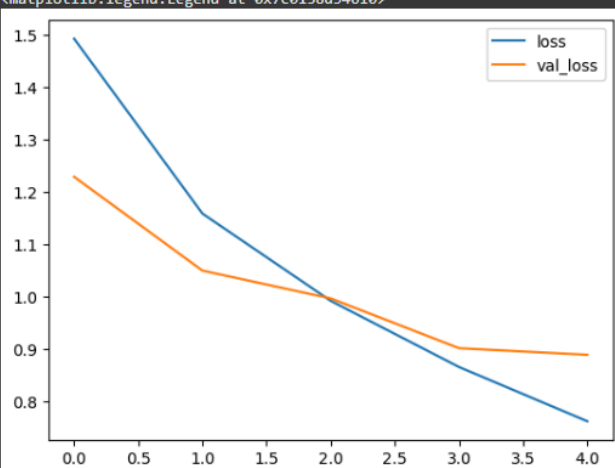
model = Model(i, x)
```

```
[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

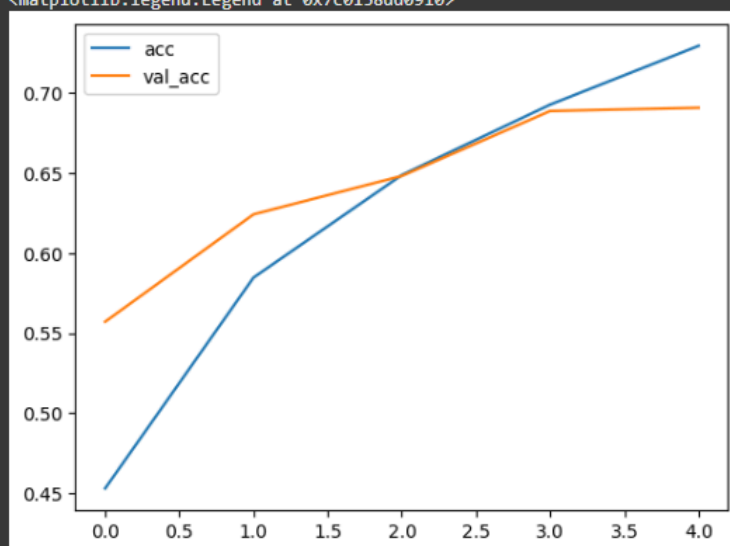
```
[6] plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend()
```

<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>



```

[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]

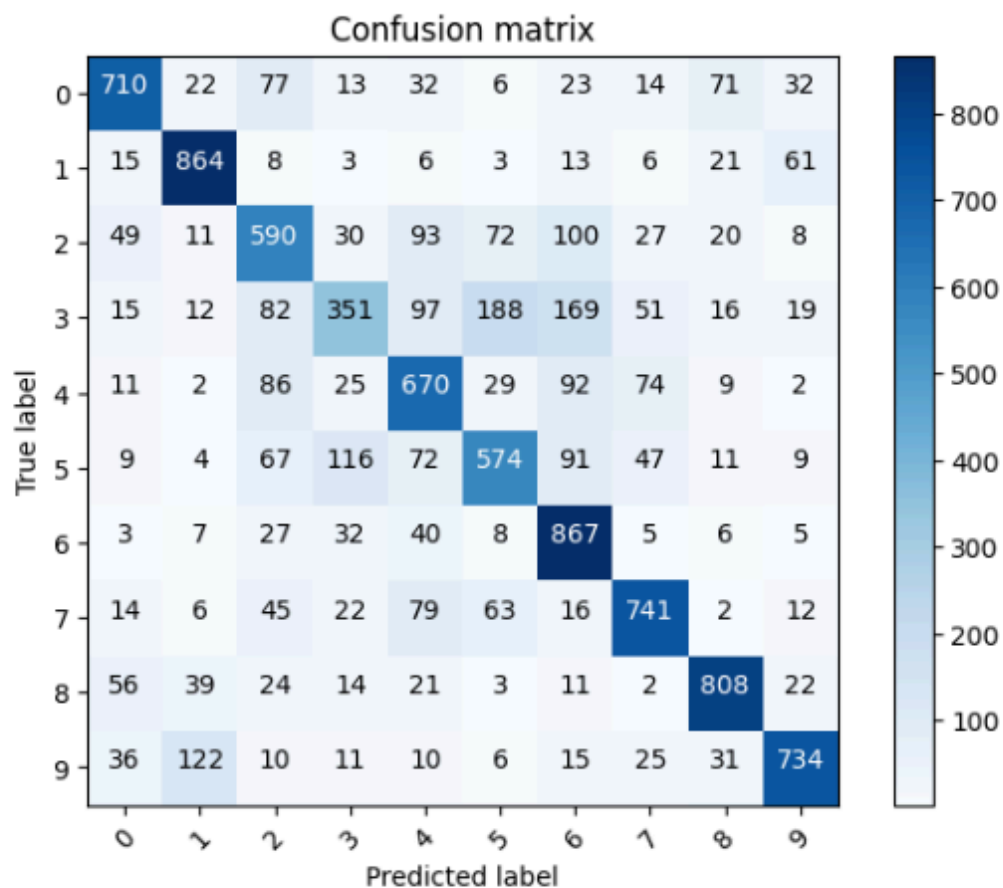
```

```
[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)))
```

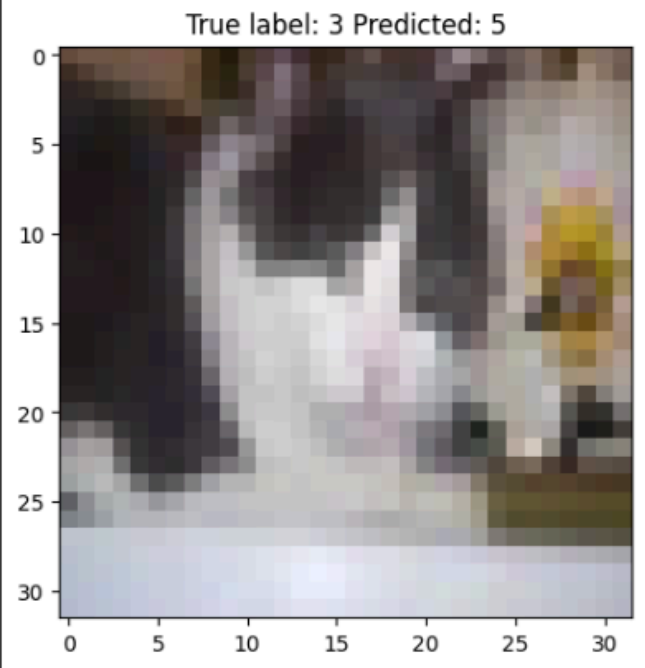
313/313 ————— 4s 12ms/step  
 confusion matrix, without normalization

```
[[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]]
```





```
[9] misclassified_idx = np.where(p_test != y_test)[0]
    i = np.random.choice(misclassified_idx)
    plt.imshow(x_test[i], cmap='gray')
    plt.title("True label: %s Predicted: %s" % (y_test[i], p_test[i]));
```



Comece a programar ou gere código com IA.

**Aula 38 - Data Argumentation:** Os modelos de Deep Learning, que dependem de grandes volumes de dados, geralmente superam algoritmos tradicionais em tarefas complexas, como reconhecimento de imagens ou processamento de linguagem natural. No entanto, isso só é possível quando há dados suficientes e de alta qualidade para o treinamento. O Data Augmentation serve para gerar mais dados em que se consiga trabalhar, mas ele não gera dados 'novos', ele apenas pega uma das imagens e pode dar um zoom, pode girar ela na vertical, deixar de ponta cabeça, para que a rede neural se adapte e consiga entender que aquilo é o mesmo objeto mas em posições diferentes ou de maneiras diferentes. Isso é especialmente útil quando o conjunto de dados original é pequeno ou desequilibrado.

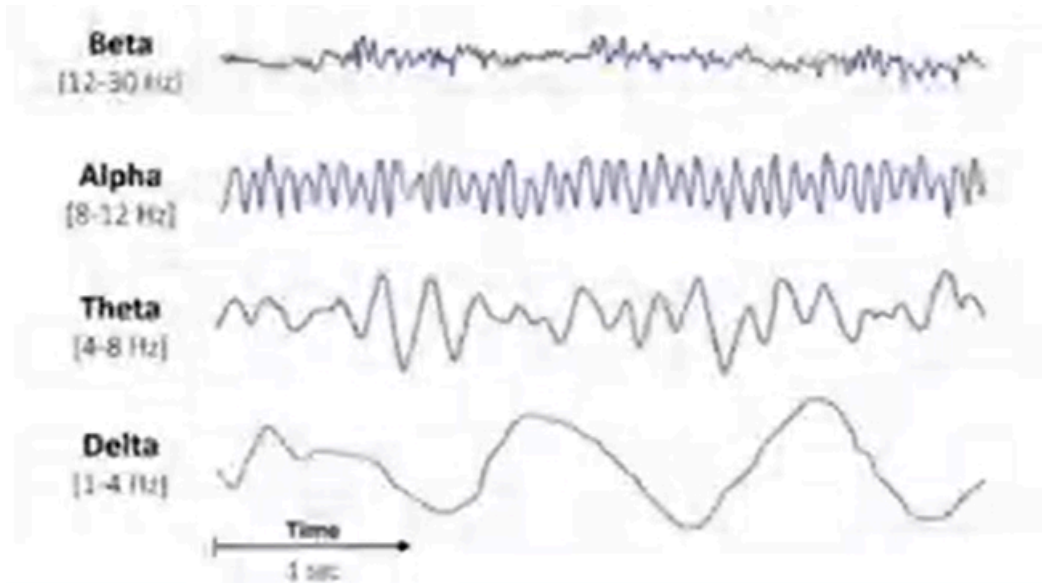


**Aula 39 - Batch Normalization:** A padronização dos dados antes de passá-los para algoritmos de Deep Learning é um passo fundamental para garantir o bom desempenho do modelo. Um problema comum é que apenas a primeira camada da rede recebe os dados normalizados, e à medida que esses dados passam por camadas densas, sua distribuição é alterada. E com isso pode dificultar o processo de treinamento.

Para resolver isso, aplica-se o Batch Normalization entre as camadas da rede neural. Essa técnica normaliza os dados de entrada de cada camada, mantendo a média próxima de zero e a variância próxima de um, mesmo após as transformações provocadas pelas camadas densas. Isso ajuda a estabilizar o treinamento, acelera a convergência, e permite que o modelo aprenda de forma mais eficiente e robusta ao longo de todas as camadas da rede.

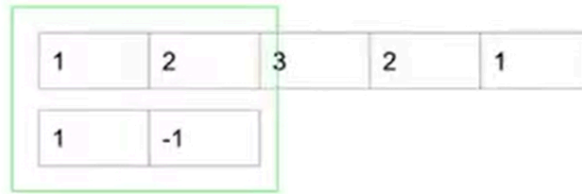
## Seção 6 - Natural Language Processing (NLP):

**Embeddings:** Nesta aula vimos como redes neurais convolucionais (CNNs) também podem ser usadas com sequências, e não apenas com imagens.



A ideia é parecida: assim como pixels próximos em uma imagem costumam ter valores parecidos, em uma sequência, os dados próximos também costumam ser semelhantes. Isso faz com que a convolução funcione bem em apenas uma dimensão. Em vez de uma matriz, temos um vetor, e o filtro que percorre esses dados também é um vetor menor. Essa técnica permite identificar padrões locais ao longo da sequência.

# 1-D Convolution: Example



- Result:  $1 * 1 + 2 * (-1) = -1$



## Seção 7 - Convolution In-Depth:

**Aula 46 - Real-Life Examples of Convolution:** Um dos primeiros exemplos que nos é passado é o de um áudio e a aplicação de efeitos nele, ele diz que o áudio no começo é aplicado um efeito ou um filtro que retorna um outro áudio de saída. Podemos concluir que a convolução é apenas entra um sinal e devolve um sinal que é semelhante a entrada porém modificado de alguma forma.

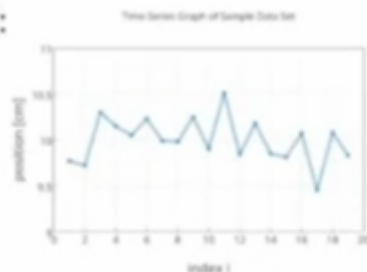
**Aula 47 - Beginner's Guide to Convolution:** A aula começa falando que para a convolução acontecer, só precisa somar e multiplicar, e é passado o mesmo exemplo da multiplicação da matriz de entrada pelo filtro e o resultado é a imagem de saída. Apenas nos é mostrado mais didaticamente o que já foi dito.

## Seção 8 - Convolutional Neural Network Description

**Aula 49 - Convolution on 3-D Images:** Agora vamos ver como trabalhar com imagens coloridas utilizando as 3 dimensões:

### Convolution over 3-D Images

1-D signal:



2-D signal:



3-D signal:



(C x W x H) Theano  
(H x W x C) Tensorflow

A detecção de bordas é uma técnica essencial na visão computacional, usada para encontrar mudanças bruscas na intensidade dos pixels, que normalmente indicam os contornos dos objetos em uma imagem. Em redes neurais profundas, isso é feito por meio de filtros que percorrem a imagem e identificam essas transições — ou seja, regiões onde os valores dos pixels mudam rapidamente. Esses filtros ajudam a destacar as formas e estruturas presentes na imagem, servindo como base para etapas mais complexas do processamento visual.

**Aula 50 - Tracking Shapes in a CNN:** Em uma rede neural convolucional (CNN) típica, os dados de entrada — como uma imagem — passam por várias camadas convolucionais e de pooling, seguidas por camadas totalmente conectadas (densas). As camadas convolucionais são responsáveis por extrair mapas de características, que identificam padrões como bordas, texturas ou formas. Já as camadas de pooling reduzem a dimensionalidade desses mapas, mantendo as informações mais relevantes e tornando o processamento mais eficiente.

Ao final do processo, as camadas densas recebem essas representações condensadas e as utilizam para realizar a classificação ou outra tarefa de saída. Um ponto importante é que alguns parâmetros, como o tamanho do filtro, não são aprendidos durante o treinamento. Em vez disso, são definidos manualmente e ajustados com base em experimentação, utilizando métodos como grid search ou busca aleatória.

Conforme a rede se aprofunda, o tamanho dos mapas de ativação diminui por causa do pooling, enquanto o número de filtros (e, portanto, de mapas de características) costuma

aumentar. Isso permite que a CNN aprenda uma variedade cada vez maior de padrões, desde os mais simples até os mais complexos.

## Seção 9:

Uma estrutura comum em redes neurais convolucionais consiste em duas partes principais: uma série de camadas convolucionais seguidas por uma série de camadas totalmente conectadas. As camadas convolucionais são responsáveis por extrair características espaciais da entrada enquanto as camadas totalmente conectadas integram essas características para realizar tarefas como classificação.



Essa é a VGG16, pois tem 16 camadas, 13 camadas de convolução e 3 conectadas. Em resumo ele diz que o necessário para aprender a fazer uma CNN é leitura do que já foi feito e escrito, pois a base quase sempre será a mesma, e treinos e testes para ver como a sua própria rede neural se adapta a tais mudanças.

## Secao 10 - In-Depth: Loss Functions

**Aula 52 - mean Squared Error:** O Erro Quadrático Médio (MSE - Mean Squared Error) é uma métrica comum quando se precisa medir a diferença entre os valores previstos e reais. A razão pela qual o erro é elevado ao quadrado no MSE é para garantir que os erros sejam sempre positivos. Se os erros não fossem positivos, os sinais opostos poderiam se cancelar, levando a uma interpretação enganosa de que a taxa de erros do modelo é baixa.

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

:

**Aula 53 - Binary Cross-Entropy:** A Entropia Cruzada Binária (Binary Cross-Entropy, BCE) é a função de perda padrão para problemas de classificação binária. Ela mede a diferença entre as probabilidades previstas pelo modelo e os rótulos reais. Sua fórmula é:

$$Loss = -\frac{1}{N} \sum_{i=1}^N \{y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\}$$

onde esses zeros e uns representam algo, como fraude ou não fraude, carro ou não carro, spam ou não spam...

**Aula 54 - Categorical Cross-Entropy:** A categorical cross-entropy é uma função de perda amplamente usada em problemas de classificação multiclasse. Ela mede a diferença entre a distribuição real das classes (geralmente representada como one-hot) e a distribuição prevista pela rede. Quanto mais distante a previsão estiver da classe correta, maior será a penalização. O objetivo do treinamento é minimizar essa perda, ajustando os pesos da rede para que as previsões se aproximem das classes corretas.

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

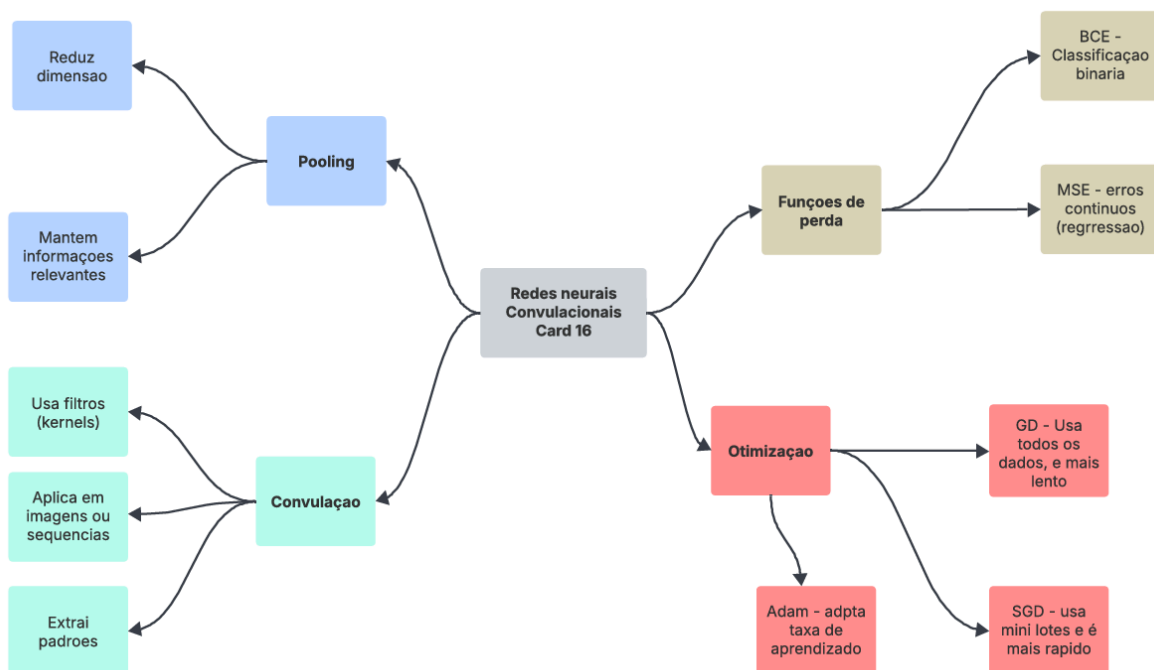
## Seção 11 - In-Depth: Gradient Descent

**Aula 55 - Gradient Descent:** A descida do gradiente é importante pois é usada em todos os tipos de redes neurais, é uma das bases do deep learning e também pode ser utilizada para treinar outros modelos de machine learning. Essa descida do gradiente serve para otimização de parâmetros, ajustando automaticamente os pesos para minimizar a função de perda, e também para encontrar o mínimo localizando os parâmetros que resultam nas previsões mais precisas.

**Aula 56 - Stochastic Gradient Descent:** O *Stochastic Gradient Descent* é um algoritmo de otimização usado para treinar redes neurais, ajustando os pesos da rede com base no gradiente da função de perda. A principal diferença em relação ao *Gradient Descent* tradicional é que, enquanto o GD calcula o gradiente usando todo o conjunto de dados (o que pode ser lento em grandes bases), o SGD atualiza os pesos a cada amostra ou mini-lote, tornando o treinamento mais rápido e eficiente. Essa abordagem introduz uma certa variação nas atualizações, mas ajuda a escapar de mínimos locais e acelera a convergência.

**Aula 59 - Adam :** O Adam (Adaptive Moment Estimation) é um algoritmo de otimização amplamente utilizado no treinamento de redes neurais. Ele combina as vantagens do *Stochastic Gradient Descent* com técnicas de adaptação de taxa de aprendizado. O Adam ajusta automaticamente a taxa de atualização dos pesos com base no histórico dos gradientes (momentum) e na média quadrática dos gradientes passados. Isso torna o treinamento mais estável e eficiente.

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