Imprimir

INTRODUÇÃO À VISÃO COMPUTACIONAL

104 minutos

Aula 1

FUNDAMENTOS DE VISÃO COMPUTACIONAL

A conexão entre a visão computacional e a Indústria 4.0 está estabelecendo novos limites para a utilização e disseminação da Internet das Coisas (IoT) e o aprendizado da visão computacional é estratégico para qualquer profissional que deseje uma formação diferenciada.

25 minutos

INTRODUÇÃO

Nesta aula sobre visão computacional você estudará um campo interdisciplinar que permite ao computador ganhar compreensão sobre vídeos e imagens. O tópico é de grande importância, pois habilita a automatização de diversas tarefas que envolvam algum tipo de processamento visual e existe uma demanda da Indústrias 4.0 por produtos que embarcam soluções inteligentes, capazes de enxergar o mundo e interagir com o ambiente de modo responsivo.

Para a compreensão do assunto é necessário contextualizar o seu histórico e sua evolução ao longo dos anos. Isso facilita o entendimento das demandas e aplicações da visão computacional, permitindo ao estudante entender como ela se encaixa no cotidiano moderno.

A conexão entre a visão computacional e a Indústria 4.0 está estabelecendo novos limites para a utilização e disseminação da Internet das Coisas (IoT) e o aprendizado da visão computacional é estratégico para qualquer profissional que deseje uma formação diferenciada. Interessante, não é? Então, vamos começar?

HISTÓRIA DA VISÃO COMPUTACIONAL

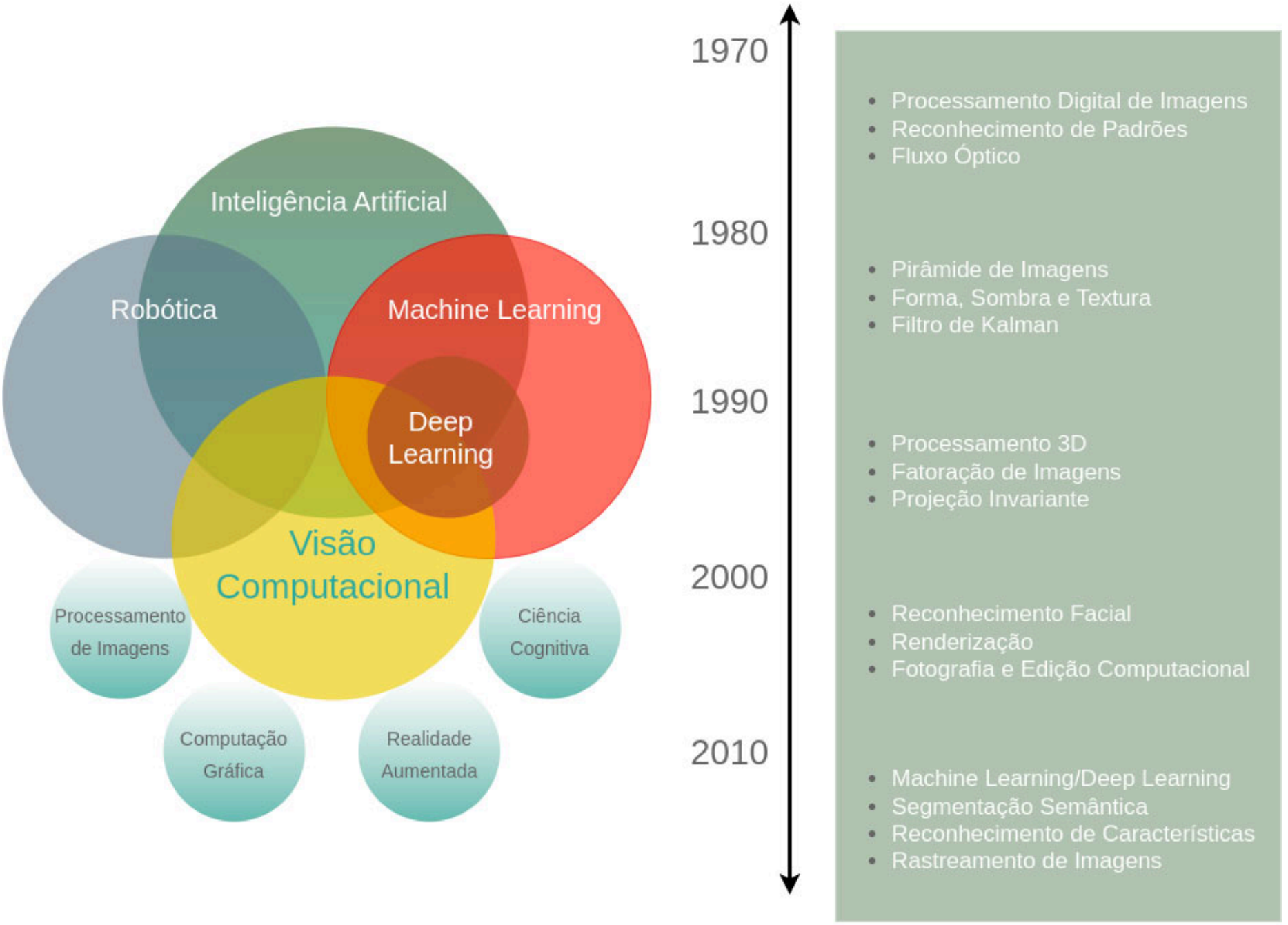
A visão computacional é um campo interdisciplinar que permite aos computadores compreender imagens e vídeos para automatizar tarefas e embutir algum tipo de conhecimento em determinado programa. A ideia é simular o sistema visual do ser humano, fornecendo à máquina um entendimento de alto nível sobre o mundo, dando um significado semântico para as imagens, de acordo com cada contexto de utilização e domínio de aplicação. As técnicas de visão computacional fornecem métodos para uma vasta gama de aplicações, em que, geralmente, a sequência de processamento consiste em adquirir, analisar e processar a imagem. Reconhecimento facial, carros autônomos, reconhecimento de caracteres, detecção de objetos e segmentação de imagens são apenas alguns dos exemplos de aplicações que empregam o conceito de visão computacional.

A história da visão computacional teve início em 1966, quando Marvin Minsky no MIT (*Massachusetts Institute of Technology*) solicitou ao seu aluno Gerald Jay Sussman para "passar o verão conectando uma câmera a um computador e fazendo com que o computador descrevesse o que viu" (SZELISKI, 2022). Na época, já existia o campo de processamento de imagens, mas o problema introduzido era mais complexo do que se imaginava e consistia em entender a estrutura tridimensional (3D) das coisas através das imagens. Nos anos 1970, o paradigma de compreensão de imagens evolui por meio de técnicas de extração de imagens e inferência 3D, partindo da estrutura topológica em duas dimensões das figuras. Os trabalhos de Winston (1975) e Hanson e Riseman (1978) foram marcos iniciais deste novo contexto e outros trabalhos seguiram essa vertente.

A partir de 1980, o foco das pesquisas direcionou-se para técnicas matemáticas quantitativas capazes de analisar determinada cena. Os pesquisadores da época, como Rosenfeld (1980), começaram a adotar métodos de pirâmides de imagens para combinar diferentes figuras ou fazer busca de correspondência entre elas. Este período é marcado por trabalhos que visam aprimorar a técnicas de detecção de contorno e bordas (SZELISKI, 2022). Além disso, alguns autores propuseram métodos qualitativos para entender a variação de sombras nas imagens (SZELISKI, 2022). Nos anos 1990 houve uma explosão de trabalhos que procuravam entender a estrutura do movimento e os sistemas de modelagem 3D automatizados foram desenvolvidos, incluindo soluções capazes de gerar uma superfície tridimensional completa (SEITZ; DYER, 1999).

Os anos 2000 introduziram uma mudança de paradigma considerável e muitos estudos direcionaram seus esforços para técnicas baseadas em dados e aprendizado. Além disso, essa década ficou marcada pela atenção dada à fotografia e diversos métodos de tratamento de imagens foram propostos e adotados, como mesclagem, síntese de textura, sobreposição e combinação. As técnicas de reconhecimento de padrões e extração de características receberam muita atenção neste período. A partir de 2010 houve um incremento substancial dos algoritmos e *hardwares* disponíveis para Inteligência Artificial e a visão computacional contribuiu e se beneficiou com este avanço. Hoje, ambos os tópicos são intimamente relacionados e a adoção de dados rotulados permitiu uma revolução no campo de reconhecimento de padrões. Esta evolução foi subsidiada pelo aumento do poder computacional entregue pelas Unidades de Processamento Gráfico (GPU), que permitiu o desenvolvimento de diversos algoritmos como a inovadora rede neural profunda AlexNet. O entendimento do histórico nos permite concluir que estamos em constante evolução e o tópico de visão computacional está em ampla ascensão. A Figura 1 ilustra a relação entre os conceitos apresentados e a evolução da visão computacional ao longo das décadas.

Figura 1 | Conceitos relacionados à visão computacional e técnicas desenvolvidas ao longo das décadas



Fonte: elaborada pelo autor.

IMPORTÂNCIA DE USO PARA O MERCADO E COMO SE GERA VALOR DA VISÃO COMPUTACIONAL

Para entender a importância da visão computacional dentro do mercado e da sociedade, podemos fazer um paralelo com a relevância da visão biológica em nosso dia a dia. O aparato visual é um dos componentes mais importantes do ser humano, ele facilita a interação com o meio em que vivemos e permite a execução desde tarefas simples até as mais sofisticadas. Diversas atividades cotidianas só são possíveis graças a esse sentido, que fornece uma capacidade ampla de orientação e posicionamento. Dirigir um carro, operar uma máquina em uma linha de produção ou realizar uma operação médica são exemplos de práticas que demandam uma visão acurada. Além disso, existem tarefas monótonas que precisam ser verificadas e conferidas por meio da visão. Partindo deste princípio, é possível ter uma noção das vantagens de embarcar a visão computacional nos dispositivos.

Computadores são precisos, não se cansam, automatizam tarefas e geram valor para diversos negócios. Permitir que as máquinas “enxerguem” é algo que traz uma grande vantagem competitiva para uma empresa ou pode gerar conforto para as pessoas. Já imaginou quantas horas o ser humano desperdiçou no trânsito, ou até mesmo quantas tarefas podem ser automatizadas por meio de um robô dotado de capacidades similares à visão humana? É neste contexto que a visão computacional se insere como um agente facilitador capaz de otimizar e baratear diversos produtos ou serviços, tais como:

- Carros autônomos.

- Reconhecimento de identidade por meio de imagens.
- Análise de texto.
- Identificação de *Fake News*.
- Tratamento de imagens.
- Automatização de linhas de produção.
- Criação de imagens.
- Análise e diagnóstico de doenças.
- Manutenção de peças.
- Sensoriamento por meio de satélites.

Outros dois aspectos importantes são a substituição da mão de obra humana em tarefas de risco e atividades que exigem alto grau de precisão. O primeiro caso, quando mal executado, pode levar à perda de vidas humanas e tem um forte impacto social. O segundo caso pode impactar negativamente nas receitas de uma empresa. Considere o diagnóstico de doenças por meio de imagens, situação em que há um fator de risco relacionado a decisões equivocadas, que invariavelmente pode incorrer em dois tipos de problemas. Primeiro: ao tratar um paciente que não tem a doença, pode-se gerar pânico e dor familiar sem necessidade, além de incorrer em custos desnecessários e riscos para o indivíduo. Segundo: quando a doença não é diagnosticada, o paciente pode morrer ou ter um diagnóstico tardio.

Tais aspectos tornam a visão computacional uma técnica-chave em diversos setores do mercado e da sociedade. Produtos e serviços que embarcam o conceito têm um valor e um diferencial muito grande quando comparados aos métodos tradicionais, que adotam apenas recursos humanos para realizar determinada tarefa. Além disso, diversos segmentos estão buscando soluções algorítmicas que usam a visão computacional e, ao mesmo tempo, as técnicas estão mais sofisticadas, o que exigirá um conhecimento amplo do assunto para atender demandas variadas.

Finalmente, pode-se dizer que o propósito é simular de forma realista, ampla e eficiente a visão do homem em dado contexto, a fim de automatizar algum processo tedioso e repetitivo, além de melhorar a eficiência e acurácia do sistema como um todo, mitigando erros potenciais que podem colocar vidas em perigo ou drenar recursos financeiros desnecessariamente. A Figura 2 ilustra uma das aplicações modernas da Visão Computacional.

Figura 2 | Detecção de objetos no trânsito usando visão computacional por meio da Inteligência Artificial

Fonte: Wikimedia Commons.

VISÃO COMPUTACIONAL PARA A INDÚSTRIA 4.0

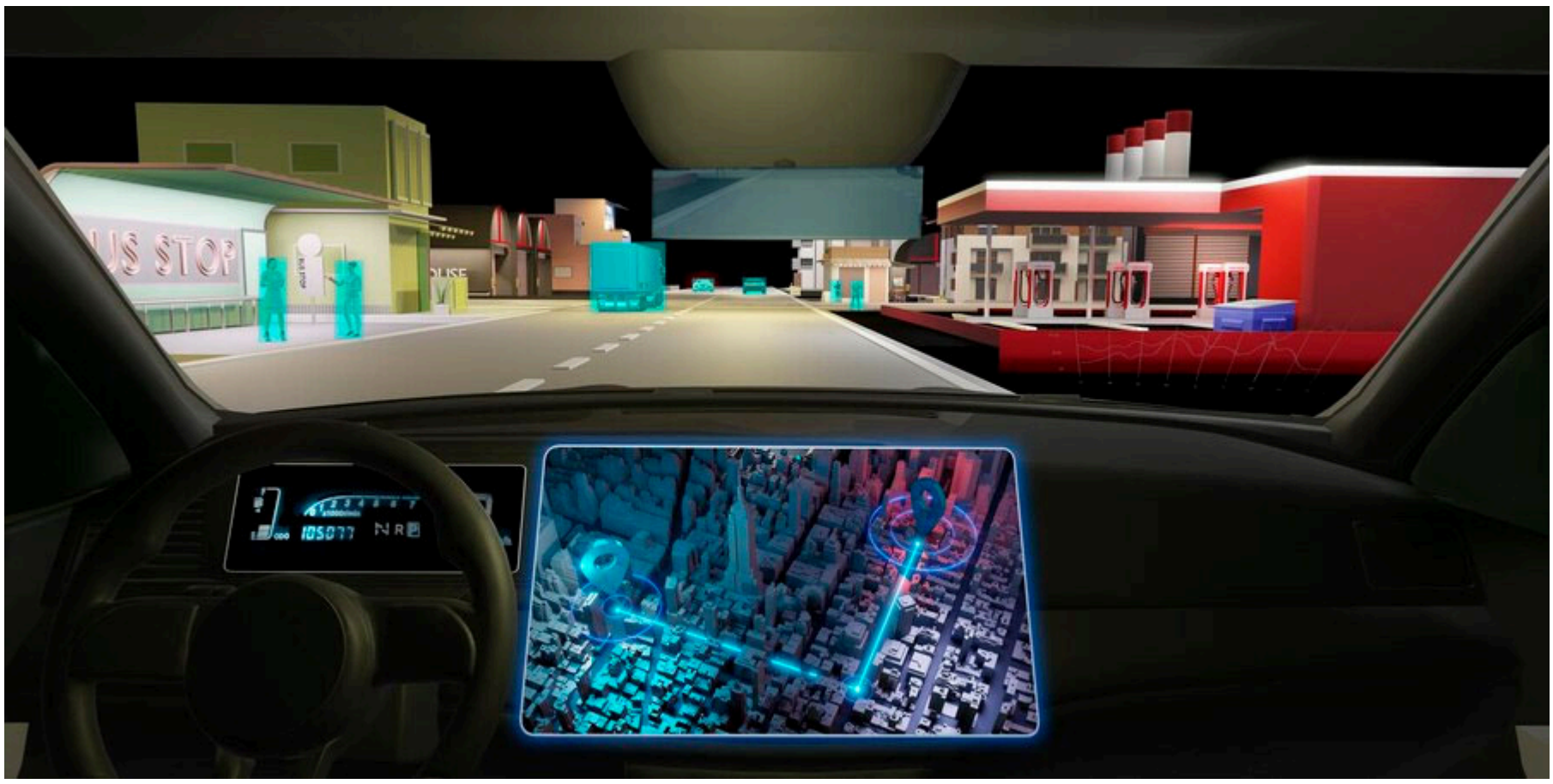
Na história passamos por algumas revoluções que foram de extrema importância para o crescimento da sociedade. A primeira Revolução Industrial aconteceu na Grã-Bretanha no final do século XVIII e foi responsável pela criação da máquina a vapor e esse acontecimento histórico trouxe diversos desdobramentos para a humanidade e impulsionou uma vasta gama de setores da indústria e do comércio, trazendo consigo uma série de componentes facilitadores para a vida cotidiana.

A Segunda Revolução Industrial aconteceu entre 1870 e 1945 e países como o Reino Unido, Estados Unidos, França e Alemanha foram os principais protagonistas. Neste período houve um grande incremento no número de ferrovias, a criação dos meios de comunicação a rádio, desenvolvimento do motor a combustão e outros adventos que tornaram a vida mais confortável e deram subsídios à medicina, química e física. A Terceira Revolução Industrial compreende o período entre 1950 e 2000 e esta fase trouxe mudanças profundas e tecnologias disruptivas para a humanidade. O desenvolvimento dos transistores e circuitos integrados é exemplo de criações que fomentaram a era digital e proporcionaram a criação de computadores poderosos. Além disso, a internet foi outro advento concebido neste período que alavancou os sistemas de comunicação e permitiu a troca de informações em uma escala nunca experimentada antes.

A partir dos anos 2000, a quarta Revolução Industrial ou Indústria 4.0 começou a se desenvolver. Os conceitos que caracterizam esse período são a automação, troca de dados entre os dispositivos e a capacidade de tomada de decisões. A conectividade foi permitida graças à tecnologia *Wireless*, da computação na nuvem e das redes 5G. O poder de decisões é fruto da Inteligência Artificial, que está sendo embarcada na borda dos dispositivos, que são capazes de se comunicar e cooperar entre si. A visão computacional, inserida na Indústria 4.0, torna a automatização extremamente eficiente, pois é possível utilizar o recurso de reconhecimento de padrões, segmentação semântica e reconhecimento facial para agilizar, baratear e otimizar variados processos. Dotar robôs e dispositivos embarcados da capacidade de visualizar e interpretar

o mundo é um diferencial muito grande, visto que a máquina pode exercer tarefas específicas que antes só podiam ser elaboradas por meio dos olhos humanos. A Figura 3 exemplifica o conceito de detecção de objetos usado em um sistema de rastreamento inteligente para condução autônoma.

Figura 3 | Detecção de objetos no trânsito



Fonte: Freepik.

A aplicabilidade da visão computacional na Indústria 4.0 é muito ampla e tem forte apelo comercial. Hoje já existem linhas de produção independentes capazes de identificar e separar componentes danificados e há projetos que visam integrar e controlar o tráfego de forma contínua, sem a intervenção humana. Também é indispensável citar a Internet das Coisas (IoT) que pretende incorporar múltiplos dispositivos de uso cotidiano de forma inteligente e coordenada. Por exemplo: imagine um sistema de automação para uma casa, que só permite acesso de pessoas autorizadas de acordo com um banco de imagem. Com uma abordagem parecida, o setor de segurança e vigilância pode aumentar a qualidade de seus serviços e melhorar a percepção dos consumidores.

Em conclusão, pode-se afirmar que o tema é amplo, relevante e apresenta diversos campos para serem explorados, o que, em última instância, irá fornecer as ferramentas e oportunidades para um leque de negócios e problemas, cujas soluções irão agregar valor para indústria e sociedade como um todo.

VIDEOAULA

Nesta videoaula, você aprenderá sobre a história e a evolução da visão computacional ao longo dos anos. Compreenderá como o tema impactou e continua impactando o nosso cotidiano e qual a sua importância na indústria e sociedade. Vamos entender como a visão computacional se relaciona com outras áreas, como a Indústria 4.0 e a automação de processos. O vídeo abordará os principais cenários de aplicação sobre o assunto e discutirá potenciais cenários de utilização. Espero você!

Videoaula

Para visualizar o objeto, acesse seu material digital.

Saiba mais

A evolução da visão computacional está intimamente ligada à evolução dos computadores, a qual serve de base para a execução de métodos mais complexos. O artigo *A Histórica Evolução da Computação Moderna!* apresenta e contextualiza os principais fatos históricos sobre essa máquina tão fantástica.

Antes de trabalhar com imagens e vídeos, devemos ficar atentos a questões éticas e legais. Para não infringir nenhum direito ou violar a lei, é bom estar por dentro da Lei Geral de Proteção de Dados. O artigo *Inteligência artificial e a Lei Geral de Proteção de Dados – LGPD* apresenta alguns aspectos importantes sobre o tema.

O artigo *Machine Learning, Visão Computacional e IoT* trata da interação entre Machine Learning, visão computacional e IoT, apresentando alguns exemplos de aplicação e discutindo o impacto futuro da junção destas três tecnologias.

Vale muito a pena ler!

Aula 2

UTILIZAÇÕES DA VISÃO COMPUTACIONAL

Esta aula terá um enfoque prático sobre temas relevantes de visão computacional, que são empregados por métodos modernos de processamento de imagens.

24 minutos

INTRODUÇÃO

Olá, estudante, tudo bem?

Esta aula terá um enfoque prático sobre temas relevantes de visão computacional, que são empregados por métodos modernos de processamento de imagens.

O primeiro tópico introduz o conceito de reconhecimento e localização de objetos, que é de suma importância para extrair informações úteis das imagens. Em seguida, trataremos do processo de detecção de objetos, uma tarefa desafiadora, em que instâncias particulares devem ser classificadas de acordo com rótulos/classes previamente definidos (“gato” ou “cachorro”, por exemplo). Finalmente, vamos explorar o conceito de segmentação de imagens, que permite dividir uma figura em diversas regiões de interesse, por meio de uma técnica que visa simplificar a representação semântica da figura e facilitar sua análise.

Para a compreensão dos assuntos abordados, a aula empregará exemplos práticos utilizando a plataforma Google Colab, que é gratuita e permite ao estudante fixar as habilidades aprendidas por meio da linguagem Python e sem a necessidade de instalar as bibliotecas no seu computador.

E aí, gostou do que leu até agora? Então, bora começar?

CONCEITOS DE RECONHECIMENTO E LOCALIZAÇÃO DE OBJETOS

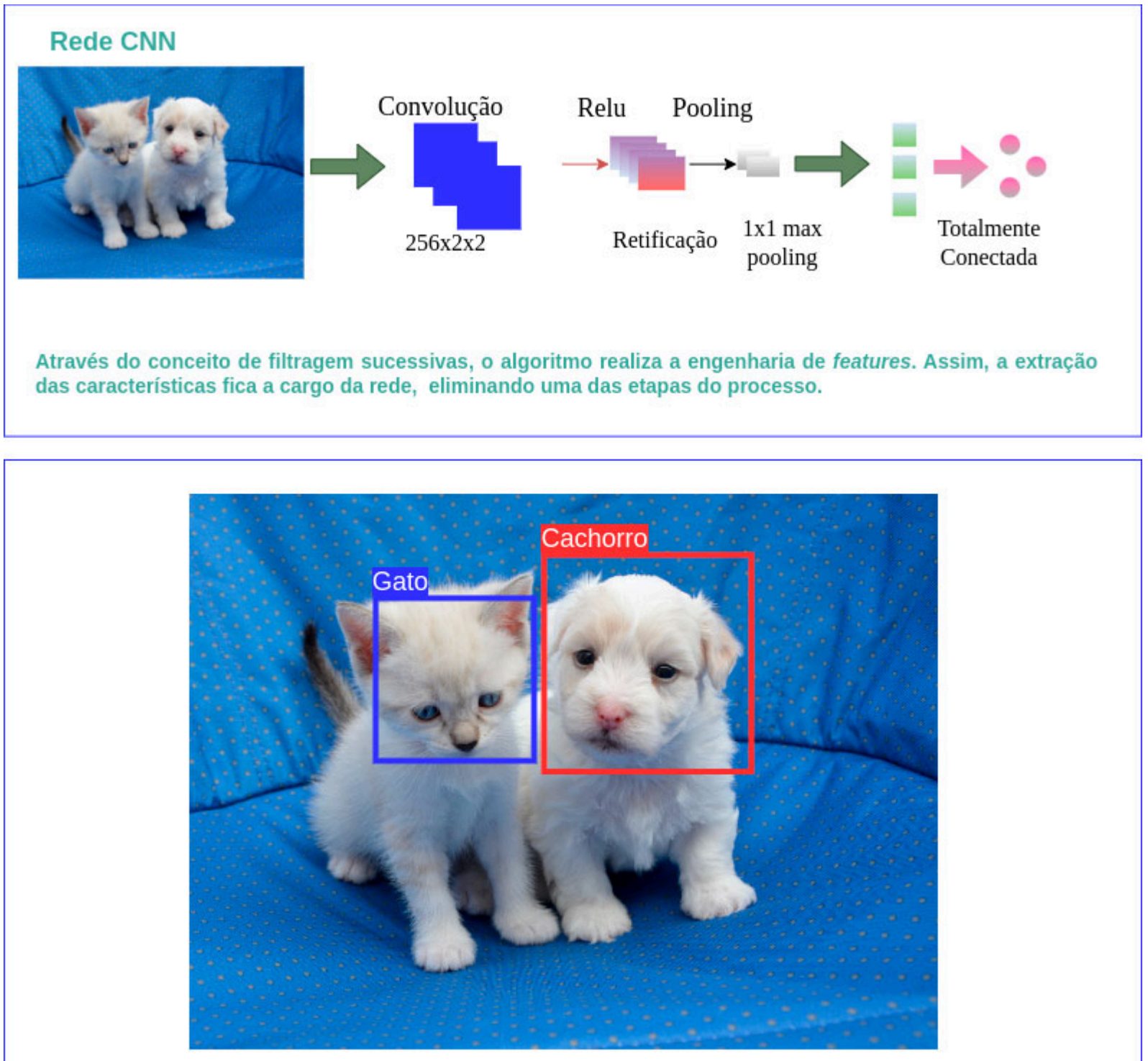
O reconhecimento de objetos é o ramo da visão computacional cujo objetivo é identificar determinadas classes de objetos (coisas do mundo real, como uma bola ou animal) em uma imagem. De forma geral, podemos dividir o método em duas categorias, **reconhecimento de instância** e **reconhecimento de classe**. No primeiro caso, estamos interessados em identificar um objeto com base em uma coleção prévia de objetos do mesmo tipo, assim, o problema se resume em reconhecer uma instância do objeto de interesse na imagem. Por exemplo: podemos definir uma classe de objetos chamada pintura e identificar o quadro *Girassóis* de Van Gogh como uma instância desta classe. O segundo método é mais desafiador, pois o propósito é determinar qualquer instância de uma classe particular entre diversas classes possíveis, como “carros” ou “pedestres” (LAZEBNIK, 2016). Além disso, em ambos os casos, a detecção dos elementos deve vir acompanhada de sua respectiva localização, isso é um aspecto importante do processo de reconhecimento, já que certas aplicações, como carros autônomos, precisam da orientação exata dos elementos na figura, caso contrário, o resultado da aplicação poderia ser caótico.

Reconhecer uma classe genérica entre diversas classes possíveis é campo de pesquisa aberto, em constante evolução e desafiador, visto que não há um método consolidado entre tantas técnicas introduzidas pela literatura. Os métodos de reconhecimento de objetos são categorizados em duas áreas com abordagens distintas e, dentro de cada uma delas, existem diversas variantes. A primeira é baseada em **características**, em que um algoritmo de extração de características é aplicado na imagem. Baseando-se na distribuição (histograma) obtida é possível dizer qual classe do conjunto de treino se assemelha mais às características encontradas. Outra variante desta abordagem é procurar por relacionamentos geométricos entre as imagens e as classes definidas na base de dados (SZELISKI, 2022). A segunda estratégia se baseia em **Redes Neurais Profundas** (*Deep Learning*), atualmente essa categoria corresponde ao estado da arte e já existem muitos *frameworks* gratuitos do tipo, como o Classy Vision e YOLO.

A introdução das Redes Neurais Convolucionais (*Convolutional Neural Network* - CNN) foi um marco facilitador para a visão computacional e para o reconhecimento de objetos. O algoritmo é inspirado no córtex cerebral dos gatos (que faz filtragens sucessivas nas imagens para reconhecer padrões complexos) e é capaz de automatizar o processo de extração de características de modo invariante ao posicionamento do objeto na figura. Com isso, as arquiteturas profundas ganharam uma ferramenta valiosa que tornou o processo de reconhecimento de padrões robusto. Atualmente, existem métodos tão poderosos que além de reconhecer uma classe simples com alta acurácia também são capazes de fazer um reconhecimento detalhado, ou seja, além de informar o rótulo (“cachorro” ou “gato”), o algoritmo é capaz de fornecer detalhes sobre a respectiva classe (“cachorro”, “preto”, “pastor-alemão”, “macho”). O grande contraponto é a quantidade de dados e a computação necessária para treinar essas redes que, ao contrário do ser humano que aprende por meio de poucos exemplos, necessitam de milhões de amostras rotuladas e *hardware* de ponta para entregar

resultados satisfatórios. A boa notícia é que uma vez treinada, a rede pode ser retreinada em outras tarefas semelhantes ou em base dados privados para atender uma tarefa mais específica e sem incorrer em alto custo computacional. A Figura 1 contextualiza o conceito de reconhecimento de classe usando uma rede do tipo CNN.

Figura 1 | Conceito de extração de características utilizando uma arquitetura em camadas com a CNN e o resultado do reconhecimento das classes de interesse na imagem



Fonte: elaborada pelo autor.

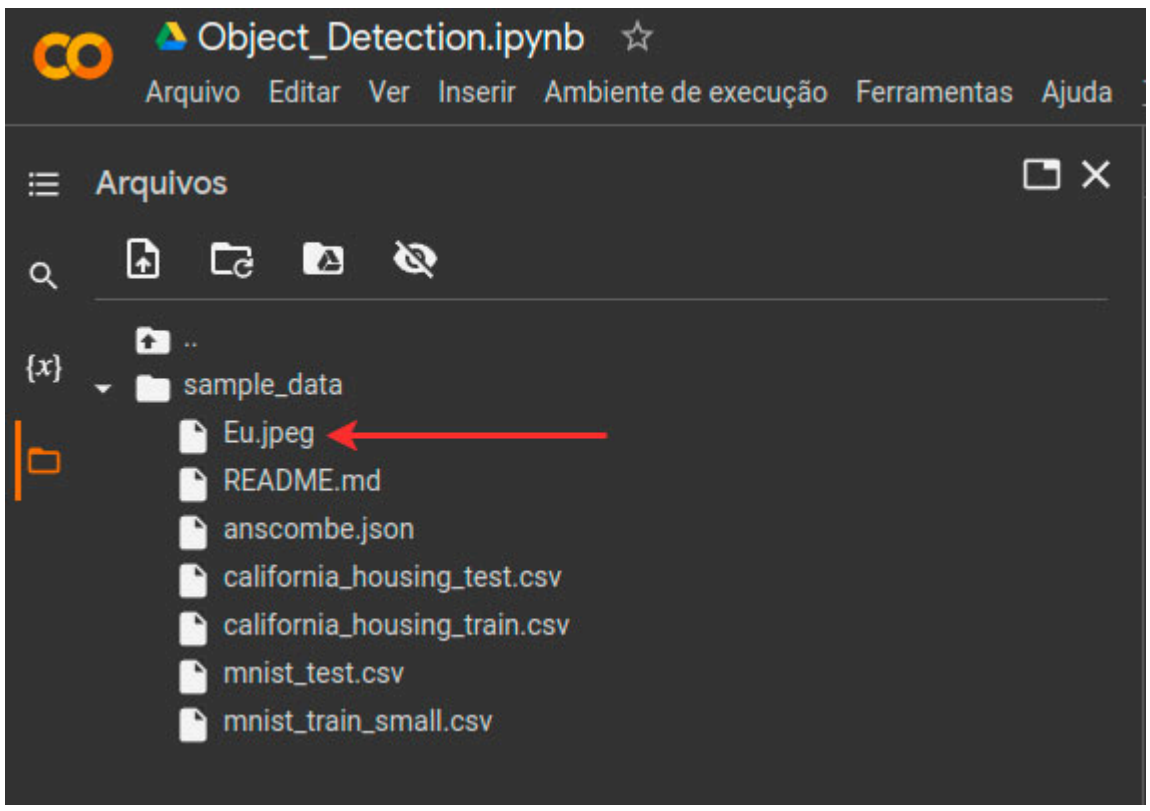
CONCEITOS DE DETECÇÃO DE OBJETOS

A detecção de objetos visa identificar onde as coisas de uma determinada imagem se encontram e dizer a qual classe o artefato pertence. Podemos, por exemplo, usar uma imagem do trânsito para classificar pedestres, carros, ciclistas e placas, também podemos obter sua respectiva posição na imagem. Esta seção terá uma abordagem prática e utilizaremos programação para criar uma aplicação capaz de reconhecer determinados padrões em uma imagem. Vamos usar a linguagem de programação Python, que é amplamente difundida no ramo de visão computacional e Inteligência Artificial. Para evitar a instalação e configuração de todo o ambiente, usaremos uma plataforma em nuvem chamada Google Colab, que fornece um ambiente de desenvolvimento completo para Inteligência Artificial e os recursos computacionais necessários. Para acessar

a plataforma é necessário ter uma conta do Google, que é gratuita e pode ser obtida por meio da Internet. Também empregaremos a biblioteca OpenCV, que é de uso livre e muito difundida em Visão Computacional, a biblioteca tem os métodos e funções que nos ajudarão na construção da solução.

Vamos criar uma aplicação para reconhecer rostos e olhos em uma determinada imagem. O conceito pode ser estendido para outros objetos como carros ou pedestres. Primeiro, devemos escolher uma imagem que contenha ao menos um rosto e fazer o upload para o Google Colab, você pode escolher uma pasta de interesse ou até mesmo associar um caminho do Google Drive. A Figura 2 ilustra a estrutura de pasta do Google Colab e a imagem escolhida.

Figura 2 | Estrutura do Google Colab e imagem escolhida



Fonte: elaborada pelo autor.

O Google Colab trabalha com o conceito de células de execução, o que facilita a criação do programa e o teste, assim não precisamos executar todo o código de uma única vez. Em seguida temos que importar as bibliotecas, vamos usar o OpenCV e o cv2_imageshow (responsável por exibir as imagens na tela). Cada célula deve ser executada com o botão "*play*" para que nossas ações tenham efeito. A Figura 3 contextualiza essa etapa.

Figura 3 | Importação das bibliotecas e leitura da imagem por meio do caminho específico

```
[18] #Importar o openCV
     import cv2
     from google.colab.patches import cv2_imshow

[36] # Carregar uma imagem.
     #É necessário fazer o upload do arquivo para a pasta no Google Colab
     image = cv2.imread('/content/Faces.jpg')
```

Fonte: elaborada pelo autor.

Para a detecção de objetos, vamos utilizar o algoritmo Haar Cascade do OpenCV, inicialmente, o algoritmo foi concebido para identificar rostos, mas hoje ele se estende para uma gama muito grande de objetos. Como estamos interessados em rostos, devemos usar o arquivo “haarcascade_frontalface_default.xml”, pois é ele que contém os parâmetros necessários para essa opção. Caso o interesse fosse “olhos”, teríamos que mudar para “haarcascade_eye_tree_eyeglasses.xml”, além disso, existe uma lista de objetos possíveis que podem ser consultados na documentação do OpenCV. Em seguida, temos que converter a imagem para cinza, pois o algoritmo funciona melhor com essa escala de cores. A Figura 4 ilustra esses passos.

Figura 4 | Criação do objeto de classificação, conversão da imagem para cinza e exibição

Fonte: elaborada pelo autor.

Finalmente, devemos passar para o nosso classificador a imagem de interesse e ele irá retornar uma tupla com as coordenadas de cada face encontrada na imagem, que permite criar um retângulo na imagem encontrada. A Figura 5 detalha esses passos.

Figura 5 | Reconhecimento de face em imagem usando o classificador Haar Cascade

```
[39] # Nosso classificador retorna a face detectada como uma tupla
     # Armazena a coordenada superior esquerda e as coordenadas inferior direita
     faces = face_classifier.detectMultiScale(imgGray, scaleFactor = 1.3, minNeighbors = 5)

     # Se faces for vazio, significa que o algoritmo não encontrou nenhum rosto
     if faces is ():
         print("Rostos não encontrados")

     # Nós iteramos através de nosso array faces e desenhamos um retângulo
     # sobre cada face em faces
     for (x,y,w,h) in faces:
         cv2.rectangle(image, (x,y), (x+w,y+h), (127,0,255), 2)

     cv2_imshow(image)
```

Fonte: elaborada pelo autor.

Podemos estender o conceito para detecção de olhos, conforme a Figura 6. É importante que o estudante refaça o programa em casa para fixar o conhecimento, além disso, você pode testar outros tipos de objetos ou usar até mesmo um vídeo.

Figura 6 | Detecção de face e olhos

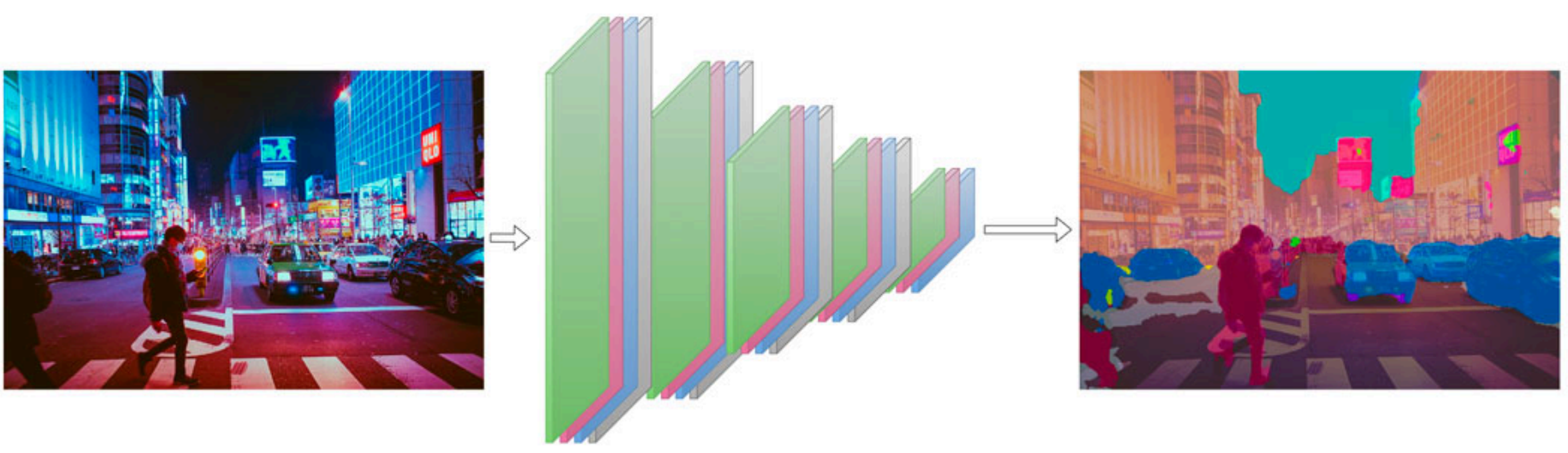
```
[40] olhosCascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_eye_tree_eyeglasses.xml")

# Nosso classificador retorna o ROI da face detectada como uma tupla
# Armazena a coordenada superior esquerda e as coordenadas inferior direita
olhos = olhosCascade.detectMultiScale(imgGray, scaleFactor = 1.2, minNeighbors = 1)

# When no faces detected, face_classifier returns and empty tuple
if olhos is ():
    print("Olhos não encontrados")

# Nós iteramos através de nosso array faces e desenhamos um retângulo
# sobre cada face em faces
for (x,y,w,h) in olhos:
    cv2.rectangle(image, (x,y), (x+w,y+h), (127,0,255), 2)

cv2_imshow(image)
```

Fonte: elaborada pelo autor.

DEFINIÇÃO DE SEGMENTAÇÃO DE INSTÂNCIAS

A **segmentação de objetos** é uma versão mais complexa do reconhecimento geral de objetos, pois além de distinguir múltiplos objetos em uma cena, devemos definir de modo criterioso e acurado a fronteira de separação entre as instâncias. A abordagem básica consiste em definir as classes de interesse e rotular cada pixel na imagem de acordo com a classe correspondente. A **segmentação de instância** é ligeiramente diferente da segmentação de objetos, ela produzirá uma imagem com máscaras separando as regiões de pixels, mas, neste caso, diferentes instâncias do mesmo objeto são separadas com cores diferentes. Assim como na detecção de objetos geral, as redes do tipo CNN desempenham um papel de destaque no paradigma de segmentação e algumas abordagens criam arquiteturas com camadas dispostas em forma de pirâmide para aprimorar a extração de características, como no trabalho de Lin *et al.* (2016). A Figura 7 apresenta um exemplo de segmentação de objetos usando uma pirâmide de CNN.

Figura 7 | Exemplo de segmentação de objetos usando pirâmide de CNN



Fonte: elaborada pelo autor.

Agora vamos voltar ao Google Colab e utilizar a biblioteca Pixellib para criar um exemplo prático. A biblioteca foi desenvolvida para fazer a segmentação de imagens e vídeos de maneira fácil e intuitiva. Com ela é possível integrar um projeto de *software* inteiro, em poucas linhas e sem gastar nada, já que a sua licença permite o uso comercial. Além disso, ela já fornece os modelos com as redes neurais treinadas, o que alivia substancialmente a computação e a necessidade de uma base de dados extensa. A primeira etapa do nosso projeto é a instalação das bibliotecas e a Figura 8 ilustra esse passo, que dependendo da conexão com a internet, pode demorar um pouco. É importante frisar que estamos instalando versões anteriores de algumas bibliotecas para ficar em consonância com os requisitos da Pixellib.

Figura 8 | Instalação das bibliotecas por meio do comando pip3

```
!pip3 install tensorflow==2.6.0
!pip3 install keras==2.6.0
!pip3 install imgaug
!pip3 install pillow==8.2.0
!pip install pixellib==0.5.2
!pip install labelme2coco==0.1.2
```

Fonte: elaborada pelo autor.

O passo seguinte é baixar para a nossa pasta de trabalho um modelo pré-treinado disponibilizado pelo site do fabricante. Podemos escolher dentre algumas opções e testar a diferença entre eles, neste caso, vamos empregar o “mask_rcnn_coco.h5” obtido por meio deste link; esse arquivo é usado para segmentação de instâncias. Outra dica importante é ler a documentação e os tutoriais disponibilizados pelo fabricante, isso facilita o entendimento do programa. Após subir o arquivo, você o enviará para sua pasta do Google Colab e importará as bibliotecas necessárias conforme a Figura 9.

Figura 9 | Importando o Pixellib e o módulo de segmentação de instâncias

```
import pixellib
from pixellib.instance import instance_segmentation
```

Fonte: elaborada pelo autor.

O último passo consiste em criar o objeto que irá realizar a segmentação, carregar o modelo treinado que acabamos de enviar para a pasta do Google Colab e, finalmente, escolher uma figura de origem e outra de destino para salvar o resultado da segmentação. O resultado da segmentação vai depender do tipo e da complexidade da imagem. Esses passos estão exemplificados na Figura 10.

Figura 10 | Sequência final de comandos para realizar a segmentação com a Pixellib

```
#Cria o objeto capaz de realizar a segmentação semântica
segment_image = instance_segmentation()
#Carrega o modelo treinado
segment_image.load_model("/content/mask_rcnn_coco.h5")
#indica uma imagem de origem e atribui um nome para imagem segmentada
segment_image.segmentImage("/content/sample_data/People.jpg",
                           output_image_name = "/content/Imagem_Segmentada.jpeg")
```

Fonte: elaborada pelo autor.

A Figura 11 exemplifica o resultado obtido em um dos experimentos.

Figura 11 | Exemplo de segmentação de instâncias usando Pixellib

Fonte: elaborada pelo autor.

VIDEOAULA

O vídeo desta aula contém um enfoque prático dos assuntos aprendidos. Você compreenderá a importância do reconhecimento de padrões em problemas de visão computacional e terá a oportunidade de trabalhar com a plataforma Google Colab em um desafio de detecção de objetos e em um problema de segmentação de instâncias. Ambos os conceitos são de suma importância em muitos desafios do mundo real. Não perca, espero por você!

Videoaula

Para visualizar o objeto, acesse seu material digital.

Saiba mais

Uma técnica interessante e amplamente empregada em reconhecimento de padrões é o casamento de *template*. O artigo *Reconhecimento de padrões em visão computacional via Template Matching* aborda esse assunto de forma prática. Vale a pena conferir!

A visão computacional trata também de vídeos, o que permite o reconhecimento de objetos em tempo real e nos fornece uma ferramenta poderosa. No *Construindo uma App para Reconhecimento de Objetos em Tempo Real com Tensorflow e OpenCV* você encontrará uma boa introdução sobre o assunto, além do código necessário para criar sua primeira aplicação com vídeo.

O OpenCV pode ser usado junto a modelos neurais profundos para aumentar a acurácia de detecção de faces, além disso, é possível capturar a imagem diretamente da câmera. É exatamente esse assunto que o texto *Detecção facial com Python e OpenCV* tratará.

Vale a pena conferir!

Aula 3

VISÃO COMPUTACIONAL CLÁSSICA

Esta aula abordará aspectos clássicos da visão computacional, que são essenciais para a compreensão das etapas de desenvolvimento de um software capaz de "enxergar" os objetos do mundo real.

26 minutos

INTRODUÇÃO

Olá, seja bem-vindo querido estudante!

Esta aula abordará aspectos clássicos da visão computacional, que são essenciais para a compreensão das etapas de desenvolvimento de um software capaz de "enxergar" os objetos do mundo real.

O conceito de pré-processamento de imagens nos permite formatar os dados de uma maneira conveniente para cada aplicação. As técnicas de extração de características são fundamentais para identificar padrões únicos nas imagens, que permitem diferenciar um contexto de outro. O treinamento das redes neurais é feito com base nos padrões extraídos previamente e a avaliação do modelo é uma etapa crítica antes de colocar o sistema em operação.

Essas habilidades permitem ao estudante compreender o ciclo de desenvolvimento da Inteligência Artificial e são aplicadas a diversos problemas do gênero. O profissional da Indústria 4.0 deve ser capaz de dominar os conceitos desta aula de forma ampla. Tudo muito legal e tecnológico, não é? Bora começar nossa aula?

DEFINIÇÃO DE PRÉ-PROCESSAMENTO DE IMAGENS

A visão computacional clássica compreende a família de algoritmos que não usam aprendizado de máquina, porém ela é a base de diversas abordagens (SZELISKI, 2022), desde soluções mais simples até artefatos sofisticados que usam *Deep Learning* em sua construção.

O pré-processamento ajuda a remover informações irrelevantes ou aprimor alguns recursos importantes para a análise adicional no contexto de *Machine Learning* (SANKA *et al*., 2018). Independentemente do objetivo do *software* de visão computacional, a maioria dos projetos irão recorrer a alguma técnica de pré-processamento de imagens, sendo as estratégias as mais variadas possíveis, como:

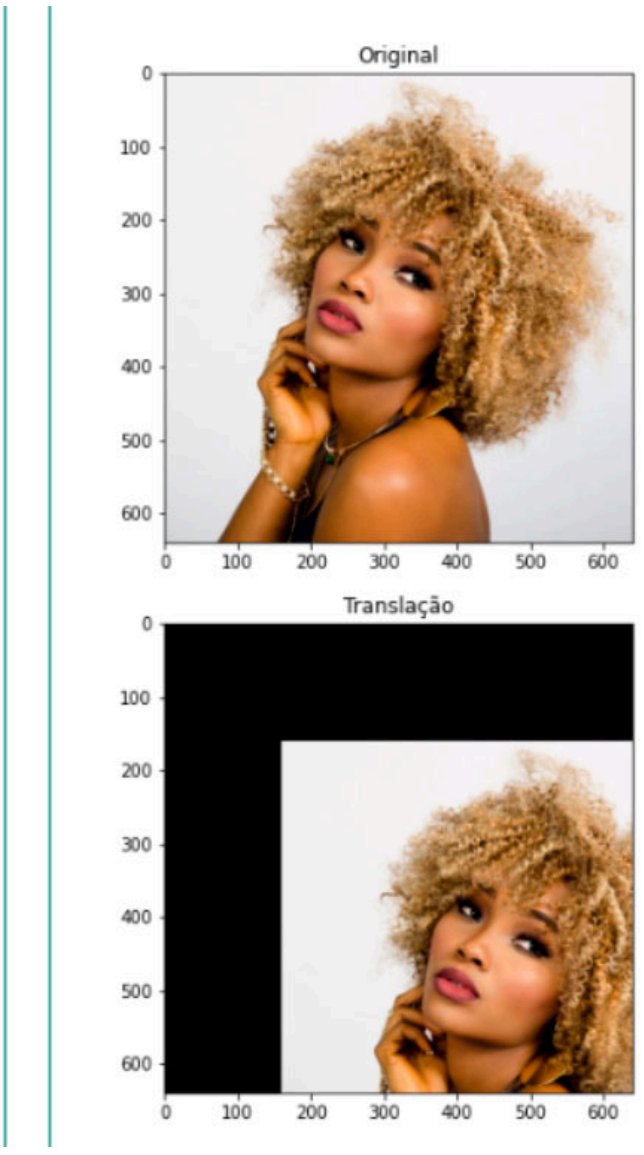
- Redimensionamento de imagens;
- Transformação de imagens: rotação, translação, oclusão;
- Remoção de fundo;
- Colorização de fotos em escala de cinza;
- Remoção de ruído;
- Extração de características;
- Detecção de texto.

Existem outros métodos além dos descritos acima e cada um deles é empregado em determinado contexto. Por exemplo: rotacionar uma imagem em diversos ângulos pode ser útil para enriquecer a base de dados, isso facilita a implementação dos algoritmos de classificação, que são treinados com imagens similares, mas que representam diferentes perspectivas da mesma pessoa ou objeto, e chamamos esse processo de aumento de dados. Outro exemplo é a extração de características, essa técnica é pré-requisito básico para qualquer mecanismo de classificação e, geralmente, é empregada em conjunto com outras técnicas de visão computacional clássica. O redimensionamento de imagem é muito útil quando possuímos uma base de dados com figuras que têm dimensões não uniformes, ou seja, cada item em nossa coleção está com um determinado tamanho. Redimensionar torna-se um passo indispensável antes de alimentar um algoritmo de classificação. Por exemplo: quando definimos a arquitetura do tipo CNN, devemos especificar a dimensão de entrada (28x28), o que força todas as imagens da base a corresponderem a esse padrão. Já a remoção de ruídos é interessante quando queremos aumentar a qualidade da imagem, além disso, o ruído é indesejável em problemas de aprendizado de máquina, já que pode dificultar o processo de treinamento e reduzir a acurácia do modelo. A Figura 1 exemplifica a técnica de translação de imagens usando OpenCV. Primeiro, temos de importar as bibliotecas necessárias e, em seguida, criamos uma função customizada de exibição, que recebe um título, a imagem e o tamanho desejado como parâmetros. Em seguida, temos de carregar a figura, obter a altura e largura da imagem. Esses valores são usados como base para definir o quanto queremos deslocar a imagem (no caso, usamos um quarto, mas você pode testar outras proporções). As proporções de deslocamento são empregadas para criar a matriz de translação ao longo dos eixos X e Y. Finalmente, usamos a função *warpAffine* do OpenCV para aplicar a transformação.

Figura 1 | Exemplo de código para translação de imagens usando OpenCV

```python
# Importar bibliotecas, criar nossa função Imshow
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Defina nossa função imshow
def imshow(titlo = "Image", image = None, tamanho = 5):
    w, h = image.shape[0], image.shape[1]
    proporcao = w/h
    plt.figure(figsize=(tamanho * proporcao,tamanho))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(titlo)
    plt.show()
```

```python
# Carrega a Imagem
image = cv2.imread('/content/Mulher.jpg')
imshow("Original", image)

# Armazenar altura e largura da imagem
altura, largura = image.shape[:2]

# Deslocar um quarto da imagem
quarter_altura, quarter_largura = altura/4, largura/4


# T é a matriz de translação
T = np.float32([[1, 0, quarter_largura], [0, 1,quarter_altura]])

# Usamos warpAffine para transformar a imagem usando a matriz T
img_translation = cv2.warpAffine(image, T, (largura, altura))
imshow("Translação", img_translation)
```

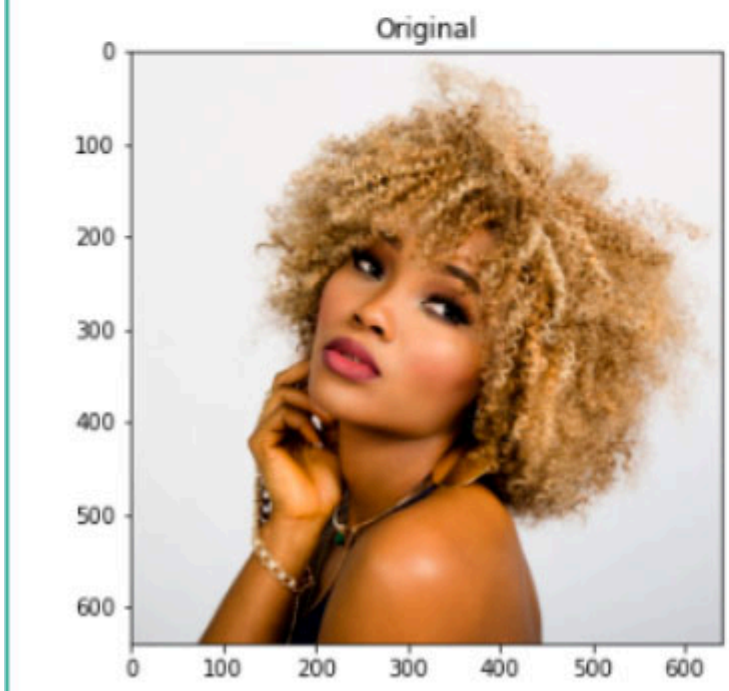


Fonte: elaborada pelo autor.

Podemos também aplicar o algoritmo de rotação utilizando a função apropriada e atribuindo os parâmetros necessários (centro da imagem, anglo e a escala) e, em seguida, usamos o método *warpAffine* novamente. A Figura 2 ilustra um exemplo de código e o resultado da transformação. Além das técnicas de pré-processamento introduzidas, existem outras que podem ser feitas de maneira similar usando o OpenCV, como remoção de fundo e redimensionamento de imagens. É muito importante testar outras técnicas de pré-processamento.

Figura 2 | Exemplo de código para rotação de imagens usando OpenCV

```python
# Carrega a Imagem
image = cv2.imread('/content/Mulher.jpg')
imshow("Original", image)

altura, largura = image.shape[:2]

# Divida por dois para girar a imagem em torno de seu centro
matrix_rotacao = cv2.getRotationMatrix2D((largura/2, altura/2), 45, 1)

# Insira a imagem, a matriz de rotação, largura e altura desejadas
image_rotacionada = cv2.warpAffine(image, matrix_rotacao, (largura, altura))
imshow("Rotacionada 45 graus com escala 1", image_rotacionada)
```



Fonte: elaborada pelo autor.

DEFINIÇÃO DE EXTRAÇÃO DE CARACTERÍSTICAS

Antes da definição teórica sobre extração de características, vamos imaginar o seguinte cenário: você atribui a uma criança a tarefa de agrupar diferentes frutas de acordo com o tipo. Você pode argumentar que o resultado desta tarefa é óbvio e que a criança não terá muita dificuldade em realizá-la. Mas, como o nosso cérebro executa esse trabalho? A resposta de alto nível para esse problema se baseia na semelhança entre os objetos. Porém, em um nível mais detalhado, o que fazemos é analisar determinadas características das frutas como cor, formato e peso e, com isso, julgar a qual classe cada fruta pertence. Estamos de maneira natural e incipiente extraindo características e padrões das imagens o tempo todo e são esses padrões que permitem distinguir um objeto de outro. De maneira análoga, a visão computacional necessita de técnicas para extrair características das imagens, caso contrário não haveria nenhum significado a ser apresentado aos computadores. A extração de características é a técnica que consiste em definir padrões de interesse e sintetizá-los a partir das imagens. As características devem apresentar significados importantes para classificação e análise, ou seja, procuramos encontrar um vetor de características que seja confiável o suficiente para descrever a sua localidade imediata (SANKA *et al.*, 2018), representando a semântica da imagem.

Na prática, cada problema envolve um grupo de características de interesse. Considerando a tarefa de reconhecimento facial, a primeira etapa do projeto consiste em identificar regiões na imagem que compreendem rostos e, partindo de sua localização, extrair padrões que sejam capazes de facilitar a

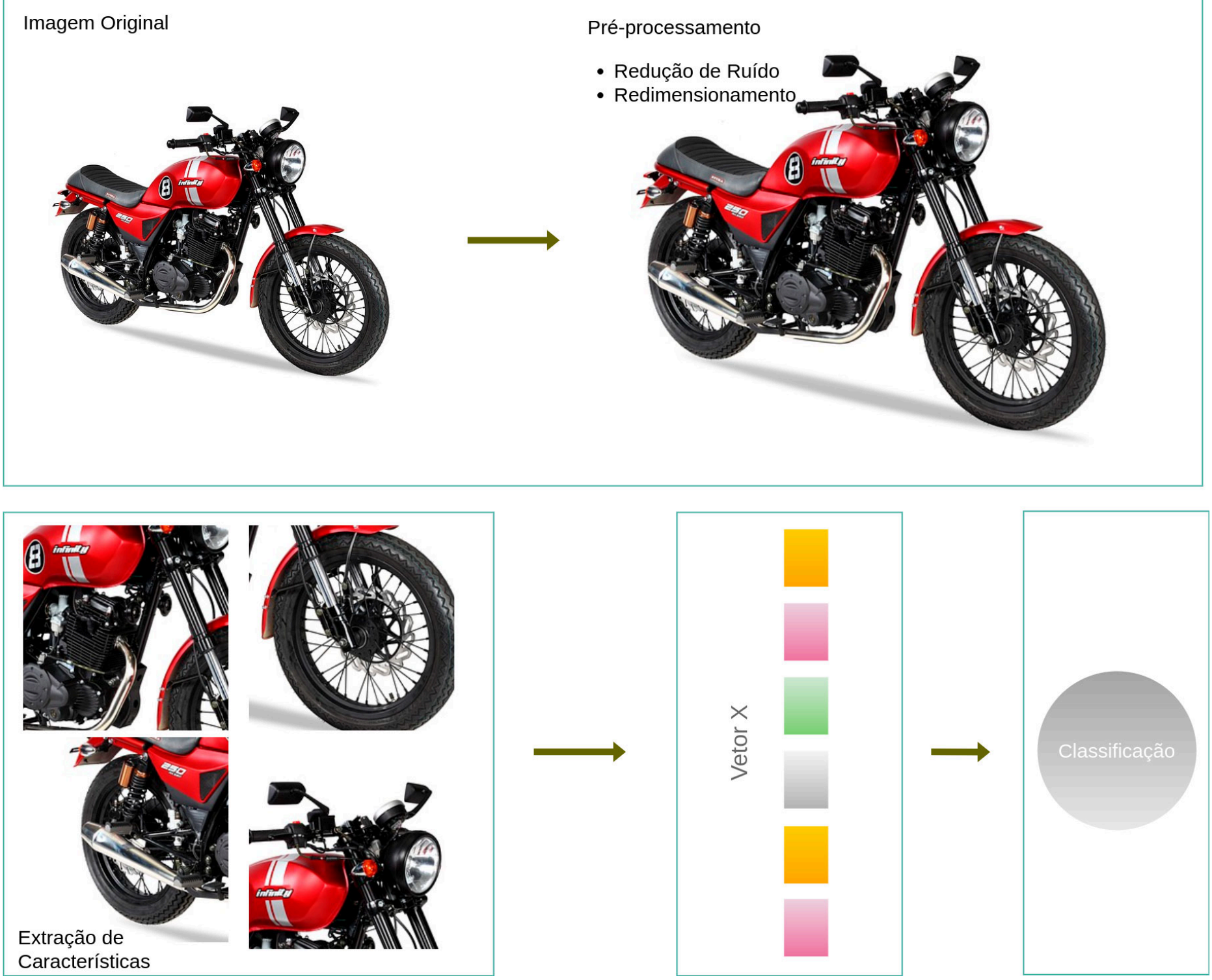
identificação do indivíduo. Neste caso, podemos empregar as seguintes características:

- Formato do rosto;
- Tamanho da boca e dos olhos;
- Cor dos olhos, cor da pele e cor do cabelo;
- Distância entre os olhos;
- Tipo de sobrancelha;
- Tamanho e formato do nariz.

Apesar de existirem pessoas muito semelhantes no mundo, cada face possui uma assinatura única, que é representada pelo somatório desses vetores de características. Essas informações são importantes para separação de classe e servem como entrada de um algoritmo de classificação, que é alimentado por meio de pares características/rótulo (X/y). O classificador é então treinado com base nessas informações e aprende a distinguir um rosto de outro por meio das peculiaridades de cada vetor de característica.

As Redes CNN pertencem a uma classe de algoritmos utilizados no reconhecimento de características em imagem e têm atributos que permitem a extração prévia de características. A rede consegue identificar padrões complexos, mesmo em condições adversas, como no caso de embaralhamento e iluminação fraca. Além disso, esses padrões são invariantes ao posicionamento, o que facilita identificar diversos tipos de características, mesmo sob situações opostas, como em uma imagem rotacionada. A maioria dos trabalhos recentes de visão computacional emprega algum tipo de abordagem que utiliza a rede CNN na etapa de atração de características (LIU, 2018). A Figura 3 ilustra os estágios envolvendo o processamento de imagens, a extração de características e a classificação do objeto. Esses passos são comumente chamados de *pipeline* de aprendizado de máquina.

Figura 3 | Exemplo de um *pipeline* de aprendizado de máquina, incluindo a etapa de extração de características, organização do vetor com os padrões aprendidos e a classificação



Fonte: elaborada pelo autor.

DEFINIÇÃO DE TREINAMENTO E AVALIAÇÃO DE CLASSIFICADORES

Quando desejamos atingir a excelência em uma dada atividade, ouvimos que devemos praticar, estudar ou treinar determinado conceito, que nos ajudará atingir certo nível de proficiência. De forma similar, as redes neurais precisam aprender uma certa classe de problema antes de serem colocadas em produção, em Inteligência Artificial chamamos esse processo de treinamento, em que os algoritmos aprendem por meio de exemplos (pares de característica/resposta). O aprendizado de máquina é a ciência de criar rotinas para que os computadores aprendam a partir dos dados, assim, a máquina consegue aprender por meio da experiência como resolver uma dada tarefa. O processo de treinamento é feito de modo iterativo e a cada passo do programa são apresentadas novas instâncias (X/y) à rede, que permitem ao algoritmo diferenciar os padrões de interesse entre as amostras. A cada iteração, a rede ajusta os pesos de suas unidades de acordo com a informação apresentada e leva em consideração aquilo que já está assimilado. O treinamento iterativo é feito por meio de incrementos conhecidos como épocas. Geralmente, o treinamento converge mais rapidamente nos estágios iniciais e iterações adicionais são usadas para refinar o modelo.

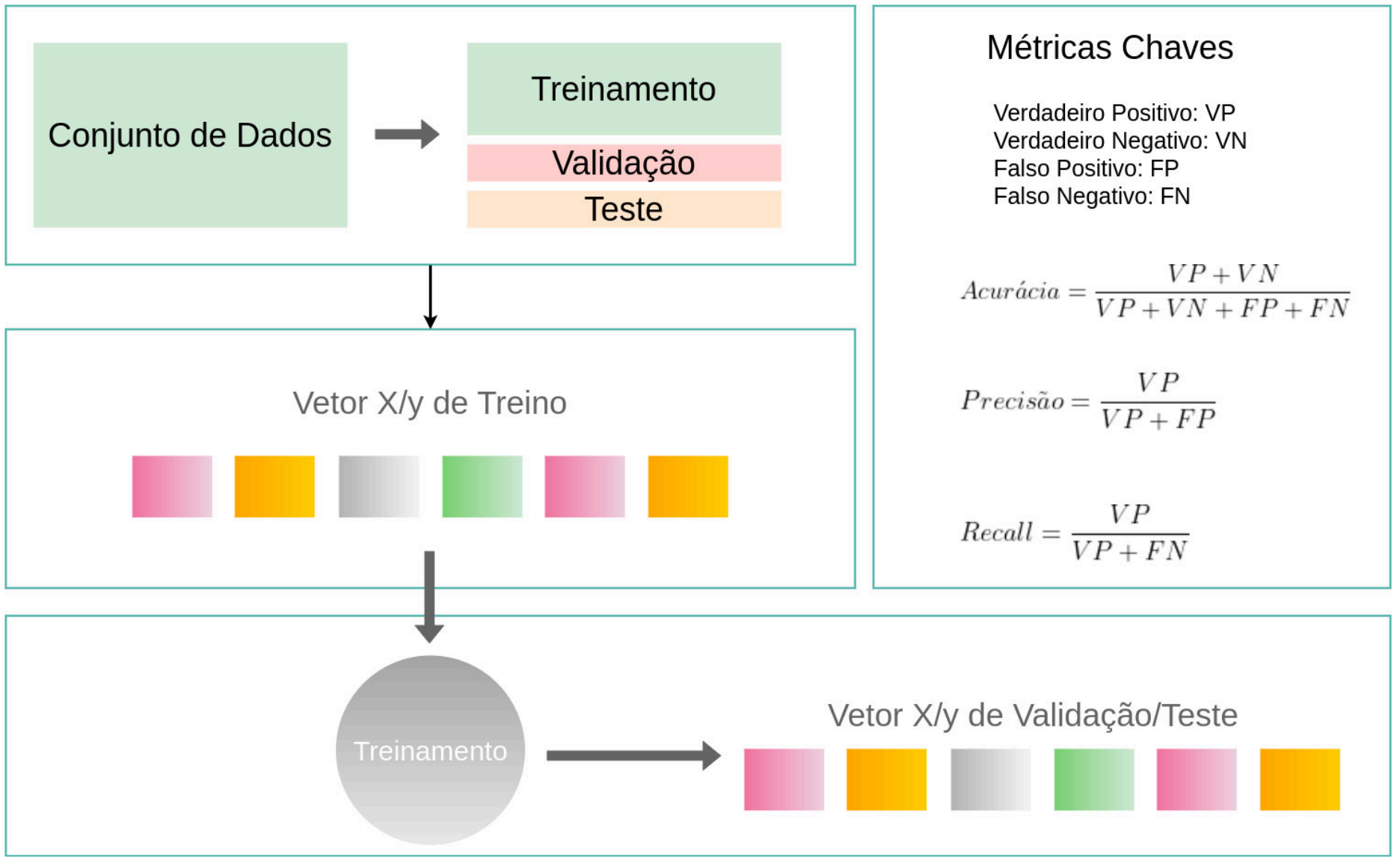
Imagine o problema de classificar as placas de trânsito partindo de uma coleção de imagens. A primeira etapa consiste em definir as classes do problema que, neste caso, são diversas (multiclasse). Em seguida, devemos ser capazes de extrair características que façam sentido no contexto de classificação, como:

- Símbolo da placa;
- Padrões geométricos;
- Texto na placa.

Com as características organizadas em um vetor X, devemos associá-las às respectivas classes correspondentes (nome da placa, vetor y). Assim, o algoritmo é capaz de compreender as relações entre os atributos que caracterizam cada placa. Antes de treinar qualquer abordagem, é necessário dividir o conjunto de dados em três grupos distintos: treino, validação e teste. Esta etapa é crítica e garante que a rede não decore os padrões, mas seja capaz de generalizar em amostras que estão fora do conjunto de treino. Deste modo, o conjunto de treino é usado na etapa de treinamento, o conjunto de validação ajuda no processo de ajuste de parâmetros e o teste é uma etapa adicional que assegura que o modelo treinado faça previsões corretas em novos dados. A metodologia de treinamento emprega a maioria dos dados para treinar a rede (geralmente 70%). As amostras que não foram disponibilizadas ao modelo devem ser utilizadas para validação e teste (15% para cada conjunto). Esses valores podem mudar dependendo do problema, mas são um bom palpite inicial.

É essencial verificar a capacidade de generalização do algoritmo, pois podem surgir dois problemas. Quando o modelo é muito simples, podemos obter uma resolução ruim, e dizemos que a rede está com pouco treinamento (*underfitting*). No segundo caso, dizemos que a rede apresentou uma solução excessivamente complexa (*overfitting*), sendo incapaz de fazer previsões fora do conjunto conhecido. A avaliação dos classificadores é feita por meio de métricas, que permitem mensurar o grau de acerto em dado contexto. A **acurácia** informa o número de acertos geral do modelo. Se o modelo acertar 75 em 100, dizemos que a acurácia foi de 75%. A **precisão** visa identificar a proporção de predições positivas corretas e, geralmente, é adotada em problemas em que um falso positivo tem um impacto maior (classificar um e-mail real como spam). O **recall** visa identificar a proporção de positivos verdadeiros identificados corretamente. Esta métrica é utilizada em problemas em que um falso negativo tem mais importância (classificar uma pessoa doente como saudável). Quanto mais próximo de 1, melhor o resultado da métrica. Os conceitos apresentados estão ilustrados na Figura 4.

Figura 4 | Conceito de divisão de conjuntos, treinamento e tipos de métricas de avaliação para classificação



Fonte: elaborada pelo autor.

VIDEOAULA

A videoaula irá detalhar conceitos teóricos essenciais para qualquer projeto de visão computacional. Você compreenderá que, apesar dos avanços da *Deep Learning*, as técnicas clássicas ainda têm papel de destaque nas etapas iniciais do projeto. Você entenderá a importância da extração de características no processo de reconhecimento de padrões e aprenderá o fluxo de etapas necessárias para treinar um modelo e a forma de avaliá-lo corretamente. Bora assistir? Aguardo você em nossa jornada!

Videoaula

Para visualizar o objeto, acesse seu material digital.

Saiba mais

Existem variadas técnicas de pré-processamento de imagens e a melhor forma de dominá-las é colocando a mão na massa, testando cada um dos métodos. O artigo *Introdução ao processamento digital de imagens com OpenCV* é um tutorial prático e profundo, que apresenta e discute diversas abordagens de pré-processamento.

O texto *O pipeline de Visão Computacional* descreve com detalhes o *pipeline* de visão computacional por meio de exemplos práticos e discute a extração de características necessária aos problemas de classificação.

O texto *Métricas de avaliação em machine learning* explica de maneira aprofundada os conceitos de avaliação de classificadores e explica a importância de cada métrica de acordo com o contexto de uso.

Bora lá conferir, combinado?

Aula 4

VISÃO COMPUTACIONAL CONTEMPORÂNEA

Nesta aula vamos botar a mão na massa e praticar, ao mesmo tempo que abordaremos conceitos teóricos-chave, que nos ajudam na construção e avaliação de um modelo de Deep Learning*. Criaremos uma aplicação para reconhecimento de dígitos em imagens.*

27 minutos

INTRODUÇÃO

Olá, seja bem-vindo querido estudante!

Nesta aula vamos botar a mão na massa e praticar, ao mesmo tempo que abordaremos conceitos teóricos-chave, que nos ajudam na construção e avaliação de um modelo de *Deep Learning*. Criaremos uma aplicação para reconhecimento de dígitos em imagens. Legal, não é?

Existem bibliotecas e *frameworks* que permitem a concepção de arquiteturas profundas e a inferência de um modelo neural. Elas fornecem ao desenvolvedor meios elegantes para a aplicação dos conceitos de IA, por meio de uma interface de alto nível que abstrai ideias complexas.

Vamos aprender como monitorar o treino da rede neural e avaliar a qualidade dos nossos resultados. Para isso, temos que saber identificar os principais desafios e lidar com eles de forma assertiva, garantindo a convergência do modelo e sua aderência ao problema. As habilidades introduzidas aqui evitam o desperdício de tempo e recursos, além de colaborar com o desenvolvimento de um perfil analítico.

Tudo muito legal e tecnológico, não acha? Bora começar nossa aula?

PRINCIPAIS *FRAMEWORKS* UTILIZADOS

Uma das grandes vantagens da computação é a sua capacidade de abstrair conceitos concretos do mundo real. Essa tarefa é feita de modo incremental, por meio de programas de computadores e bibliotecas, sem a necessidade de criar uma solução “do zero”. Esta noção se estende para a visão computacional, já que existem muitas bibliotecas e *frameworks* que nos auxiliam em diversas tarefas. Um *framework* é uma coleção de códigos genéricos que fornecem uma estrutura de desenvolvimento voltada a um tipo de aplicação, ou seja, é um modelo repleto de ferramentas de auxílio à programação. Existem muitos *frameworks* de IA que,

invariavelmente, se aplicam em problemas de visão computacional, o seu domínio e entendimento são peças fundamentais para a compreensão e prática do tema. Os seguintes *frameworks* de IA têm um papel de destaque na indústria e academia científica:

1. **TensorFlow** (ABADI *et al.*, 2016): desenvolvido pela equipe do Google Brain, de código aberto e organizado de modo hierárquico, é a plataforma de aprendizado de máquina mais utilizada. Permite o desenvolvimento de aplicações supervisionadas e não supervisionadas, por meio da criação de modelos profundos e utiliza o conceito de camadas (Convolucional, Recorrente, Normalização, etc.). Disponibiliza modelos pré-treinados, conjuntos de dados, métricas de avaliação e muito mais, além de poder operar em processadores e placas gráficas (o que acelera a computação);

2. **Keras** (GULLÌ; SUJIT, 2018): roda em cima do TensorFlow e outras plataformas como o Theano, ou seja, é um *framework* de alto nível. Foi construído para acelerar o desenvolvimento de *Machine Learning*, possui uma sintaxe amigável, foi escrito em Python e é totalmente modularizado, permitindo combinar diferentes modelos (convolucionais e recorrentes, por exemplo);

3. **Scikit-Learn** (SCIKIT-LEARN, 2011): é considerada por muitos a porta de entrada de ML, gratuita e escrita em linguagens como Python, C e C++. Permite o pré-processamento dos dados, a organização dos vetores de treino e teste, contém diversos classificadores, regressores e algoritmos de agrupamento. Abrange uma coleção de métricas e é fácil de integrar com outras bibliotecas, como Matplolib (visualização), SciPy (computação científica) e Numpy (computação numérica). O *framework* é intuitivo e pode ser combinado com outros para otimizar os resultados de ML;

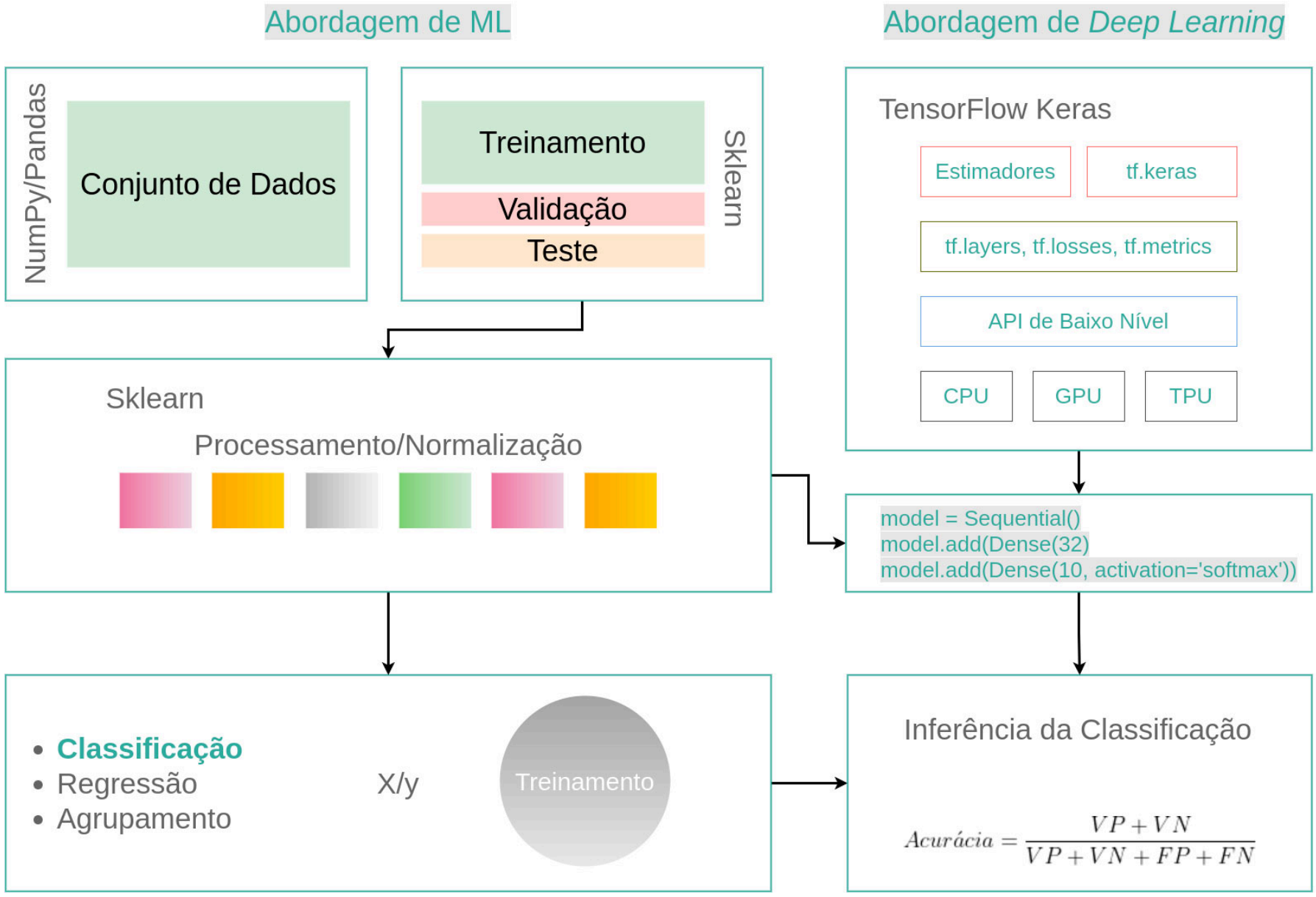
4. **PyTorch** (PASZKE *et al.*, 2019): construído sobre o Torch (uma biblioteca de aprendizado de máquina de código aberto) e desenvolvido pela META IA. Muito comum em aplicações de visão computacional e de processamento de linguagem natural. Assim como o Keras, usa o conceito de Tensor (entidades que generalizam escalares, vetores e matrizes) para representar os dados. Foi criado usando Python, C++ e CUDA. Alguns desenvolvedores consideram a sintaxe mais complexa, embora permita um maior nível de granularidade sobre a criação dos modelos;

5. **NumPy:** biblioteca composta por arrays multidimensionais e uma coleção de rotinas para processamento dos dados. Possui funções para trabalhar com álgebra linear, transformada de Fourier e matrizes;

6. **Pandas:** empregado para trabalhar com conjuntos de dados por meio de *dataframes*. Possui funções para analisar, limpar, explorar e manipular a informação. Foi desenvolvido em cima do pacote NumPy, dessa forma, o Pandas aproveita muita coisa do NumPy. Os *dataframes* são usados para alimentar análises estatísticas no SciPy, plotagem do Matplotlib e algoritmos de aprendizado de máquina no Scikit-learn, TensorFlow e outros.

Essas são as principais bibliotecas e *frameworks* de IA e, embora existam outras, trabalhos de visão computacional refinados podem ser endereçados por meio do uso e consórcio entre elas. A Figura 1 faz uma ilustração de alto nível sobre os conceitos introduzidos.

Figura 1 | Etapas do aprendizado de máquina



Fonte: elaborada pelo autor.

Você viu que interessante são essas ferramentas e como elas podem nos ajudar na construção de um método de aprendizado de máquina? Agora que você já sabe sobre o assunto, bora ver mais um pouco!

PREMISSAS PARA O USO DE DEEP LEARNING EM APLICAÇÕES DE VISÃO COMPUTACIONAL

Esta seção terá um enfoque prático e utilizaremos os *frameworks* TensorFlow e Keras para criar uma aplicação de reconhecimento de imagens. A partir da base de dados MNIST (com imagens de números escritos à mão) criaremos uma solução capaz de reconhecer um determinado número em uma figura. Apesar de simples, o conceito se estende para diversos problemas de classificação em visão computacional e muitas das ideias introduzidas aqui são estendidas para desafios mais complexos. Antes de iniciar a codificação, é importante que você compreenda que o desenvolvimento deve ser feito em etapas, isso facilita a criação e manutenção do código, além de permitir a compreensão geral sobre os estágios de ML. As etapas seguintes resumem o tutorial:

1. **Processamento da imagem:** temos que carregar, inspecionar, visualizar os dados e pré-processar as imagens para um formato adequado a ML;
2. **Construção do modelo:** devemos definir uma arquitetura, usando os *frameworks* propostos, que seja adequada ao nosso desafio. Por se tratar de reconhecimento de padrões em imagens, um bom palpite é usar camadas convolucionais. De qualquer forma o projetista tem a liberdade de sugerir outras configurações;

3. **Treinamento do modelo:** consiste em apresentar os dados processados a rede, de forma iterativa através das épocas;

4. **Inferência do modelo:** esta etapa consiste no teste e verificação da abordagem, usando novos dados para avaliar a rede.

A Figura 2 ilustra como podemos fazer o carregamento e a inspeção dos dados. Primeiro, dividimos o conjunto em treino (60000 amostras) e teste (10000 amostras), por simplicidade deixamos de fora o conjunto de validação. Além disso, verificamos que cada imagem tem a dimensão de 28x28 e cada uma delas está associada a uma resposta em y (número correspondente).

Figura 2 | Processamento da imagem

```
#Importa o conjunto de Dados
from tensorflow.keras.datasets import mnist

# Divide os vetores de treino e teste
(x_train, y_train), (x_test, y_test)  = mnist.load_data()


# Imprima as dimensões da imagem e o número
#de etiquetas em nossos dados de treinamento e teste
print("\n")
print ("Dimensão do X de treino:" + str(x_train[0].shape))
print ("Rótulos y:" + str(y_train.shape))
print("\n")
print ("Dimensão do X de teste:" + str(x_test[0].shape))
print ("Rótulos:" + str(y_test.shape))
```

```
Dimensão do X de treino:(28, 28)
Rótulos y:(60000,)


Dimensão do X de teste:(28, 28)
Rótulos:(10000,)
```

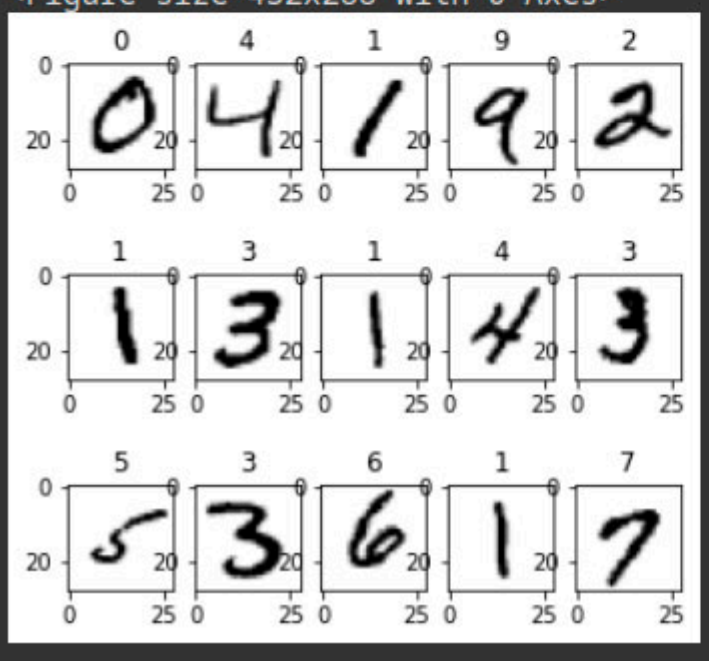
Fonte: elaborada pelo autor.

A Figura 3 mostra como podemos visualizar os dígitos e verificar quais são os rótulos associados, para isso, importamos a biblioteca de visualização Matplotlib, definimos o número de imagens a serem exibidas e organizamos em uma figura com 3 linhas e 5 colunas.

Figura 3 | Exibição das imagens e inspeção da classe de resposta associada

```python
# Biblioteca para visualização
import matplotlib.pyplot as plt

# Create figure and change size
figure = plt.figure()
plt.figure(figsize=(5,5))
# Definimos o número de imagens para visualizar
num_of_images = 15
# Iteramos até 15
for index in range(1, num_of_images + 1):
    #Organizamos a exibição em 3 linhas x 5 colunas
    plt.subplot(3, 5, index).set_title(f'{y_train[index]}')
    plt.axis('on')
    plt.imshow(x_train[index], cmap='gray_r')
```

```
<Figure size 432x288 with 0 Axes>
```



Fonte: elaborada pelo autor.

Em seguida, temos que preparar os dados para um formato conveniente para arquitetura da rede, adicionando mais uma dimensão e normalizando os valores em uma faixa que vai de 0 até 1 para facilitar a convergência do treino. Finalmente, codificamos a resposta para um formato categórico (*to_categorical*). A Figura 4 ilustra esses passos.

Figura 4 | Codificação da resposta

```python
# Salvar o número de linhas e colunas
linhas = x_train[0].shape[0]
colunas = x_train[0].shape[1]

# Devemos formatar a entrada corretamente para o Keras
# Para isso adicionamos mais uma dimensão (60000,28,28) to (60000,28,28,1)
x_train = x_train.reshape(x_train.shape[0], linhas, colunas, 1)
x_test = x_test.reshape(x_test.shape[0], linhas, colunas, 1)

#Salvamos o formato da entrada, vamos usar adiante
input_shape = (linhas, colunas, 1)

#Mudamos o valor das imagens para o tipo float
#Fazemos isso pois não podemos trabalhar com inteiros
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

#Normalizamos os dados entre 0 e 1. Facilita o treinamento
#Antes o valor era entre 0 e 255 (valores possíveis de um pixel)
x_train /= 255.0
x_test /= 255.0

from tensorflow.keras.utils import to_categorical

#Transformanos a resposta categorica em um vetor codificado
#Na prática representa a mesma coisa, apenas usamos um vetor sparso
#para representar as classes e formatar a resposta para ML
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Fonte: elaborada pelo autor.

A Figura 5 detalha as etapas de criação do modelo, por meio da importação das bibliotecas, definição do número de camadas e seus filtros. Escolhemos as funções de ativação, perda e a métrica de avaliação. A arquitetura pode ser modificada livremente, desde que atenda ao problema de classificação, podemos adicionar ou remover camadas ou modificar o número de filtros. Como estamos trabalhando em um problema de reconhecimento de padrões com múltiplas classes, usamos a função de ativação *categorical_crossentropy* na última camada, pois ela irá categorizar os padrões de acordo com rótulos possíveis. As demais funções de ativação podem ser testadas livremente, mas, geralmente, a ReLU entrega excelentes resultados. A função de otimização foi o Adam (*Adaptive Moment*), pois ela acelera o processo de treino, o passo escolhido ou a taxa de aprendizagem (*Learning Rate*) é o usado pelo otimizador para ajustar o modelo durante o treino.

Figura 5 | Criação de um modelo de classificação de imagens usando camadas convolucionais

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam
#Definimos o número de classes (0-9 dígitos)
num_classes = 10

# Criamos o modelo sequencial
model = Sequential()
#Adicionamos uma camada de convolução com 64 filtros
#Essa camada irá reconhecer padrões nas imagens
#A função de ativação define a saída da camada
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
#Adicionamos uma camada de convolução com 128 filtros
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
#Usamos MaxPooling para simplicar saída da rede
model.add(MaxPooling2D(pool_size=(2, 2)))
# Remodelamos a diemnsão para adequar a saída
model.add(Flatten())
#Usamos uma camada densemente conectada com 64 neurônios
model.add(Dense(64, activation='relu'))
#Criamos a camada de saída com 10 classes possíveis
model.add(Dense(num_classes, activation='softmax'))
#Função de perda para classificação categorical_crossentropy
#Algoritmo de otimização Adam e a métrica acurácia
model.compile(loss = 'categorical_crossentropy',
              optimizer = Adam(0.0005), metrics = ['accuracy'])
```

Fonte: elaborada pelo autor.

A última etapa desta fase consiste em treinar o modelo, etapa em que definimos um tamanho para o *batch* (número de amostras apresentadas a rede a cada iteração) e o número de épocas, além de fornecer os vetores de treino e validação ao modelo. A Figura 6 exemplifica o treinamento. Na seção seguinte, vamos avaliar o treino e verificar o modelo.

Figura 6 | Treinamento da arquitetura proposta

```python
#Definimos o número de amostras para apresentar em cada epoca
batch_size = 128
#Definimos o número de epocas
epochs = 10

#Treinamos o modelo através da função fit
#Passamos o cojunto de treino (x_train, y_train)
#Validamos no conjunto de teste validation_data = (x_test, y_test)
history = model.fit(x_train,
                    y_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (x_test, y_test))
```

Fonte: elaborada pelo autor.

Que tal praticar um pouco? É uma excelente oportunidade aprender mais e fixar o conhecimento! Tente fazer esse exemplo e conte para gente!

ABORDAGENS PARA A AVALIAÇÃO DE MODELOS DE VISÃO COMPUTACIONAL

Após treinamento, em que os pares rotulados de entrada-saída de treino foram processados por meio das épocas, o modelo está pronto para prever novas amostras (SZELISKI, 2022). Mas como sabemos se a rede de fato aprendeu e não decorou os padrões nas amostras conhecidas? Temos que testar, garantindo que o comportamento seja semelhante em imagens novas e desconhecidas pela rede; esta fase também é chamada de **inferência** do modelo. É por isso que a divisão dos conjuntos é fundamental, deixamos uma parcela ou duas (validação e teste) das amostras para averiguar a capacidade de generalização da nossa arquitetura. E como procedemos com essa avaliação de maneira sistemática?

Um bom projeto é verificado antes mesmo de o treino terminar. Durante o treinamento podemos acompanhar, por meio das épocas, o comportamento do aprendizado, verificando a acurácia de treino e acurácia de validação, além de diagnosticar a função de perda de ambos os conjuntos. Lembrando que uma acurácia perfeita é igual a 1 (100%). Já a perda (*loss*) representa o custo associado ao treino e, neste caso, queremos que ele seja o mais próximo de zero possível. Como a função de treinamento nos fornece tais informações ao longo das iterações, podemos diagnosticar o modelo da seguinte maneira:

- **Desejado:** a acurácia do treino e validação estão subindo ao longo das épocas e, consequentemente, a perda está diminuindo.
- ***Underfitting*:** a acurácia do treino e validação não atinge valores satisfatórios (maior que 0.8 é bom ponto de partida) e a perda não diminui em ambos os casos.
- ***Overfitting*:** a acurácia do treino continua **subindo** e a da validação fica **estagnada,** neste ponto as acurácias começam a divergir, indicando que épocas adicionais não melhoram o modelo.

A Figura 7 exemplifica os **resultados** da avaliação e a programação necessária para instanciar o método de treino, em que escolhemos o número de épocas e o tamanho do *batch*, parâmetros que irão influenciar diretamente na velocidade. O que você achou da acurácia obtida?

Figura 7 | Exemplo de treinamento

```python
#Definimos o número de amostras para apresentar em cada epoca
batch_size = 128
#Definimos o número de epocas
epochs = 65

#Treinamos o modelo através da função fit
#Passamos o cojunto de treino (x_train, y_train)
#Validamos no conjunto de teste validation_data = (x_test, y_test)
history = model.fit(x_train,
                    y_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (x_test, y_test))

#Avaliamos o modelo no conjunto de teste
score = model.evaluate(x_test, y_test, verbose=0)
print('Perda:', score[0])
print('Acurácia:', score[1])
```

```
Perda: 0.062000531703233 72
Acurácia: 0.9879000186920166
```

```
Epoch 23/65
469/469 [==============================] - 7s 15ms/step - loss: 0.0114 - accuracy: 0.9971 - val_loss: 0.0403 - val_accuracy: 0.9879
Epoch 24/65
469/469 [==============================] - 7s 14ms/step - loss: 0.0106 - accuracy: 0.9971 - val_loss: 0.0386 - val_accuracy: 0.9881
Epoch 25/65
469/469 [==============================] - 7s 14ms/step - loss: 0.0099 - accuracy: 0.9975 - val_loss: 0.0394 - val_accuracy: 0.9876
Epoch 26/65
469/469 [==============================] - 7s 15ms/step - loss: 0.0089 - accuracy: 0.9979 - val_loss: 0.0419 - val_accuracy: 0.9874
```

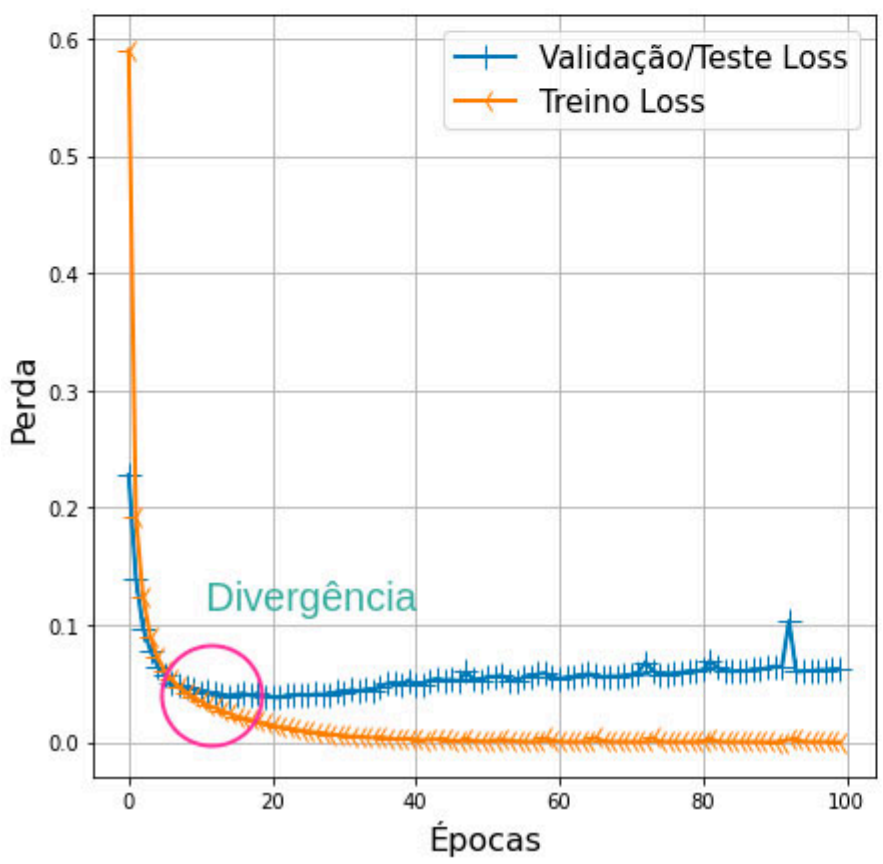
Fonte: elaborada pelo autor.

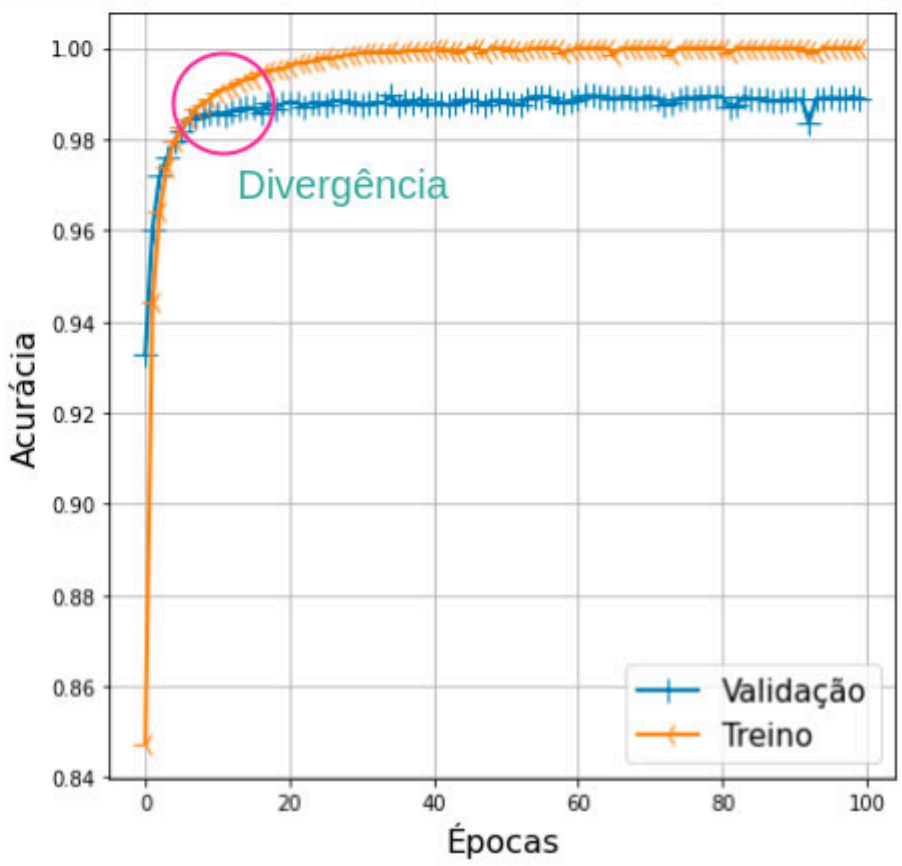
O *underfitting* pode indicar que a arquitetura proposta é muito simples ou que os dados são insuficientes para representar o problema. Com relação ao *overfitting*, podemos identificar em qual época o problema começa a acontecer e limitar o treino até ali. Outro aspecto essencial durante o monitoramento é a escolha do algoritmo de otimização e o valor da taxa de aprendizado, pois esses parâmetros vão ditar o ritmo de convergência de sua arquitetura. No exemplo anterior, utilizamos o Adam pois ele acelera bastante o treino, já a taxa de aprendizado pode ser testada por meio de diferentes valores. Devemos considerar que: taxas baixas demais tornarão o treino lento e taxas elevadas podem fazer com que a rede nunca alcance um valor ótimo (sendo incapaz de convergir). Podemos utilizar o modelo treinado para criar um gráfico de monitoramento, visualizando a perda e a acurácia ao longo das épocas.

Finalmente, é importante salientar que a escolha das métricas é fundamental para a avaliação correta do modelo e devemos sempre levar em consideração a aplicação e o contexto de uso. Os conceitos apresentados nesta seção estão detalhados na Figura 8, em que temos o código necessário para criar o gráfico da acurácia (conceito pode ser estendido para a perda) e o resultado da avaliação do processo de treino, em que verificamos que a partir da época 15 há uma divergência entre o conjunto de validação e treino.

Figura 8 | Criação do gráfico de treino para a acurácia e *plots* da avaliação para a acurácia e perda

```python
# Biblioteca para Plotar
import matplotlib.pyplot as plt

# Use o objeto com os resultados salvos
history_dict = history.history

# Crie um array com o número de épocas
epochs = range(0, 100)

figure = plt.figure()
plt.figure(figsize=(7,7))

line1 = plt.plot(epochs, history_dict['val_accuracy'], label='Validação')
line2 = plt.plot(epochs, history_dict['accuracy'], label='Treino')
plt.setp(line1, linewidth=2.0, marker = '+', markersize=12.0)
plt.setp(line2, linewidth=2.0, marker = '3', markersize=12.0)
plt.xlabel('Épocas', fontsize=16)
plt.ylabel('Acurácia', fontsize=16)
plt.grid(True)
plt.legend(prop={'size': 15})
plt.show()
```





Fonte: elaborada pelo autor.

Agora você já conhece todas as etapas na construção de um programa que usa Inteligência Artificial, não deixe de se desafiar mais um pouco e tente modificar a arquitetura da rede e veja outros resultados.

VIDEOAULA

A videoaula de hoje foca na resolução de um problema prático, clássico em visão computacional: como reconhecer dígitos em uma imagem? Vamos empregar técnicas de processamento de imagem para deixar os dados em um formato adequado para ML e, em seguida, construiremos um modelo profundo baseado em camadas convolucionais, usando os *frameworks* introduzidos, a rede será capaz de reconhecer imagens e classificá-las de acordo com dígito correspondente e você aprenderá a avaliar o modelo.

Bora assistir? Aguardo você em nossa jornada!

Videoaula

Para visualizar o objeto, acesse seu material digital.

Saiba mais

O Scikit-Learn é um dos *frameworks* de IA mais conhecidos e práticos, que permite trabalharmos em problemas de classificação, regressão e agrupamento. Que tal conhecer melhor essa ferramenta e fazer alguns exemplos práticos? O texto *Classificação com scikit-learn* é uma boa porta de entrada.

A matéria-prima de IA consiste nos dados e sua limpeza e organização são aspectos fundamentais antes de treinar um modelo. A ferramentas Pandas é um dos carros-chefes quando o assunto é manipulação e tratamento de dados. O artigo *Pandas Python: vantagens e como começar* faz uma bela introdução prática sobre essa biblioteca, essencial para o cientista de dados.

O TensorFlow é realmente fantástico e muito poderoso. Você pode ter acesso a diversos tutoriais sobre o *framework* e ninguém melhor que o fabricante para ensinar. Você encontrará diversos exemplos em português no site Tutoriais TensorFlow e também já pode aproveitar para verificar a documentação.

Bora lá conferir, combinado?

REFERÊNCIAS

2 minutos

Aula 1

HANSON, A. R.; RISEMAN, E. M. (eds). **Computer Vision Systems**. Nova York: Academic Press, 1978.

ROSENFELD, A. Quadtrees and pyramids for pattern recognition and image processing. *In*: **International Conference on Pattern Recognition** (ICPR), 1980, pp. 802-809.

SEITZ, S. M.; DYER, C. M. Photorealistic scene reconstruction by voxel coloring. **International Journal of Computer Vision**, ano 35, n. 2, 1999, pp. 151-173.

SZELISKI, R. **Computer Vision**: Algorithms and Applications. Springer International Publishing, 2022. Disponível em: https://doi.org/10.1007/978-3-030-34372-9. Acesso em: 11 dez. 2022.

WINSTON, P. H.; BERTOHOLD, H. The psychology of computer vision. McGraw-Hill, 1975.

Aula 2

LAZEBNIK, S. Generic Object Recognition. **Computer Vision**, Springer US, 2014, p. 325-26. Disponível em: https://doi.org/10.1007/978-0-387-31439-6_332. Acesso em: 12 dez. 2022.

LIN, T. *et al*. **Feature Pyramid Networks for Object Detection**, 2016. Disponível em: https://doi.org/10.48550/ARXIV.1612.03144. Acesso em: 12 dez. 2022.

SZELISKI, R. Computer Vision: Algorithms and Applications. **Springer International Publishing**, 2022. Disponível em: https://doi.org/10.1007/978-3-030-34372-9. Acesso em: 12 dez. 2022.

Aula 3

LIU, Y. H. Feature Extraction and Image Recognition with Convolutional Neural Networks. **Journal of Physics**: Conference Series, v. 1087, set. 2018. Disponível em: https://doi.org/10.1088/1742-6596/1087/6/062032. Acesso em: 12 dez. 2022.

SONKA, M. *et a*l. **Image processing, analysis, and machine vision**. 4. ed, Thompson Learning, 2015.

SZELISKI, R. Computer Vision: Algorithms and Applications. **Springer International Publishing**, 2022. Disponível em: https://doi.org/10.1007/978-3-030-34372-9. Acesso em: 12 dez. 2022.

Aula 4

ABADI, M. *et al*. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. **arXiv**, 16 mar. 2016. Disponível em: http://arxiv.org/abs/1603.04467. Acesso em: 13 dez. 2022.

GULLÌ, A.; SUJIT, P. Deep Learning with Keras: Implement Neural Networks with Keras on Theano and TensorFlow. **Packt Publishing**, 2017.

PASZKE, Ad. *et al*. PyTorch: An Imperative Style, High-Performance Deep Learning Library. **arXiv**, 3 dez. 2019. Disponível em: http://arxiv.org/abs/1912.01703. Acesso em: 13 dez. 2022.

PEDREGOSA *et al*. SCIKIT-LEARN: Machine Learning in Python, **JMLR 12**, 2011, pp. 2825-2830.

SZELISKI, R. Algorithms and Applications. Springer International Publishing **Computer Vision**, 2022. Disponível em: https://doi.org/10.1007/978-3-030-34372-9. Acesso em: 13 dez. 2022.

Imagem de capa: Storyset e ShutterStock.