Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Processamento e Análise de Imagens

Notas de Aula

Vítor Wilher

PAI · 2025.2

Versão 1.0

Resumo. Redes neurais aplicadas a visão computacional: CNNs, transfer learning, Vision Transformers, modelos multimodais e detecção de objetos.

Palavras-chave: visão computacional; CNN; ViT; multimodal

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **Processamento e Análise de Imagens (PAI)** percorre, de forma cumulativa, o caminho que vai do neurônio artificial isolado até os sistemas modernos de detecção e segmentação de objetos em tempo real. A premissa didática adotada nos slides é clara: antes de aplicar arquiteturas profundas a imagens, é preciso entender o que é uma rede neural, como ela aprende e quais são as limitações do processo de otimização; e antes de treinar uma rede sobre pixels, é preciso entender o que é um pixel, como uma imagem é adquirida, quantizada e representada numericamente.

O percurso começa com **Redes Neurais Artificiais**, cobrindo a inspiração biológica, o histórico da área (de McCulloch-Pitts ao Backpropagation), a estrutura do neurônio artificial, funções de ativação, topologias feed-forward e o Perceptron de Múltiplas Camadas (MLP). Em seguida, aprofunda-se o **aprendizado**: o algoritmo Backpropagation, o Gradiente Descendente e suas variantes (estocástico e mini-batch), hiperparâmetros como taxa de aprendizado e termo de momento, e os problemas clássicos de paralisia da rede, mínimos locais, overfitting e underfitting.

A partir daí a disciplina entra no domínio visual propriamente dito. **Introdução ao Processamento de Imagens** trata de aquisição, representação matemática, espaços de cor (RGB, HSV/HSL, CMYK), quantização, resolução, histogramas, transformações pontuais, filtragem por vizinhança e equalização de histograma (global e CLAHE). Sobre essa base entram as **Redes Convolucionais**, com convolução, padding, stride, pooling, hierarquia de características, data augmentation e arquiteturas históricas (LeNet, AlexNet, VGG, ResNet). Depois, **Transfer Learning** formaliza a reutilização de modelos pré-treinados, apoiada em leituras dos artigos originais de ResNet, GoogLeNet/Inception e da Lottery Ticket Hypothesis.

O último terço da disciplina é aplicado e moderno: **Deploy de Modelo com FastAPI** transforma o modelo treinado em serviço acessível por API; **Vision Transformers** reconstrói a arquitetura Transformer (self-attention, Q/K/V, multi-head, add and norm, FFN, encoder-decoder) e sua adaptação para imagens via patches; e **Detecção e Segmentação de Objetos** fecha o ciclo com bounding boxes, IoU, mAP, NMS, tipos de segmentação e a evolução completa da família YOLO. Há ainda dois tópicos da ementa — Trabalho Final e Modelos Multimodais — cujo material de slides não consta no acervo disponível.

2 Trabalho Final

Material de slides não disponível para esta aula.

O Trabalho Final aparece como primeiro item da ementa oficial, funcionando como eixo integrador dos conteúdos técnicos das demais aulas. Sem slides no material disponível, não é possível descrever critérios de avaliação, formato de entrega ou escopo exigido. O que os slides das aulas subsequentes permitem inferir como repertório mobilizável são os estudos de caso trabalhados ao longo do curso: classificação de dígitos manuscritos (MNIST), classificação de imagens naturais (CIFAR-10), classificação binária Dogs vs. Cats, classificação de doenças foliares (dataset Beans) e detecção/segmentação com YOLO sobre datasets como tumores cerebrais e peças automotivas.

3 Introdução à Redes Neurais Artificiais

3.1 Inteligência computacional e inspiração biológica

O objetivo da **inteligência computacional** é desenvolver paradigmas ou algoritmos que permitam a máquinas realizar tarefas cognitivas. Um sistema desse tipo deve ser capaz de três coisas: **armazenar conhecimento**, **aplicar o conhecimento armazenado** para resolver problemas e **adquirir novo conhecimento** através da experiência.

Boa parte das técnicas da área nasce de inspirações na natureza. Sistemas especialistas imitam a inferência humana; a lógica fuzzy, o processamento linguístico; as redes neurais, os neurônios biológicos; os algoritmos genéticos, a evolução biológica; o particle swarm, o comportamento social de pássaros e peixes; e os sistemas híbridos combinam esses aspectos.

Na rede biológica, cada neurônio recebe estímulos de outros neurônios por meio de **sinapses**. O efeito de cada estímulo é controlado por um **peso sináptico**, positivo ou negativo, que se adapta para que a rede como um todo aprenda — a reconhecer objetos, entender outras línguas, controlar o próprio corpo. O processamento é altamente paralelo.

Um experimento citado nos slides ilustra bem o que se busca replicar: **Pigeons as art experts** (Watanabe et al., 1995). Pombos colocados em caixas de Skinner foram expostos a pinturas de Chagall e Van Gogh, recebendo recompensa ao bicar o botão diante de um Van Gogh. Os pombos atingiram 95% de acurácia nas pinturas usadas no treinamento e 85% em pinturas nunca vistas. A conclusão é decisiva: eles não memorizaram as pinturas, mas extraíram e reconheceram um padrão — o estilo — e generalizaram a partir dele. Essa capacidade de generalização é comum a redes neurais biológicas e artificiais, e ausente em um computador convencional.

3.2 Definição e características básicas

Redes Neurais Artificiais são sistemas inspirados nos neurônios biológicos e na estrutura massivamente paralela do cérebro, com capacidade de adquirir, armazenar e utilizar conhecimento experimental. A semelhança com o cérebro se dá em dois aspectos: o conhecimento é adquirido do ambiente por um processo de aprendizado, e as forças de conexão entre neurônios (pesos sinápticos) armazenam esse conhecimento.

A correspondência entre os dois mundos pode ser resumida assim:

Rede Biológica	Rede Artificial
Neurônio biológico	Neurônio artificial
Rede de neurônios	Estrutura em camadas
Bilhões de neurônios	Dezenas a milhões de neurônios
Conexões sinápticas	Pesos e bias (parâmetros)

As características básicas destacadas nos slides são:

- **Procura paralela e endereçamento pelo conteúdo**: o cérebro não possui endereço de memória nem busca informação sequencialmente.

- **Aprendizado**: a rede aprende por experiência, sem necessidade de explicitar o algoritmo da tarefa.
- **Associação**: a rede associa padrões distintos — foto e pessoa, sintomas e doença, leitura de sensores e falha, características do cliente e fraude.
- **Generalização**: responde corretamente a padrões novos, lidando com ruídos e distorções.
- **Abstração**: extrai a essência de um conjunto de entradas, recuperando o padrão sem ruído a partir de padrões ruidosos.
- **Robustez e degradação gradual**: a perda de alguns elementos processadores ou conexões não causa o mau funcionamento da rede.

3.3 Marcos históricos

- **1943 — McCulloch-Pitts**: modelo computacional do neurônio artificial, ainda sem capacidade de aprendizado.
- **1949 — Hebb, The Organization of Behavior**: primeira formulação explícita de uma regra de aprendizado (Hebbian Learning Rule).
- **1958 — Perceptron de Rosenblatt**: método de aprendizado supervisionado.
- **1969 — Minsky e Papert**: provam as limitações do perceptron de uma camada, que só resolve funções linearmente separáveis. O caso emblemático é o **OU-EXCLUSIVO (XOR)**, cuja tabela verdade `(0,0)->0`, `(0,1)->1`, `(1,0)->1`, `(1,1)->0` não admite separação por uma única reta.
- **1982 — Hopfield**: retomada da pesquisa na área.
- **1982 — Kohonen**: mapas auto-organizáveis.
- **1983 — Barto**: aprendizado por reforço e controle.
- **1986 — Rumelhart**: **Backpropagation**, o método mais popular para treinar perceptrons de múltiplas camadas e modelos de aprendizado profundo.
- **Atualmente**: interpretações probabilísticas, redes spiking e deep learning.

3.4 O neurônio artificial

O elemento processador combina **entradas** (atributos independentes, normalizados, representando uma única observação), **pesos sinápticos**, um **bias** e uma **função de ativação**. As saídas podem ser contínuas, binárias ou categóricas, e pode haver mais de uma saída.

O comportamento é descrito por duas equações. Primeiro a soma ponderada das entradas, e depois a aplicação da função de ativação sobre essa soma acrescida do bias:

$u_k = \sum_j w_{kj} x_j$

$y_k = \varphi(u_k + b_k) = \varphi(net)$

Os slides enfatizam que os **pesos** são o ponto crucial: eles são o que de fato representa uma rede neural, e criar uma rede neural é essencialmente o processo de ajustar esses pesos. É possível que pesos assumam valor zero — a conexão continua existindo, mas não influencia a saída daquele neurônio, o que corresponde à ideia de que nem todo atributo é importante para todo neurônio.

O **bias** tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação, deslocando o ponto de operação do neurônio.

As funções de ativação apresentadas são: **degrau ou limiar**, **logística (sigmoid)**, **linear**, **tanh** e **ReLU**.

3.5 Topologias

- **Redes feed-forward**: uma ou mais camadas de processadores com fluxo de dados em uma única direção, sem realimentação.
- **Redes recorrentes**: possuem conexões entre processadores da mesma camada e/ou com camadas anteriores (realimentação).

A forma mais simples é a rede com apenas uma camada de saída — o **Perceptron de uma camada**, usado para classificação de padrões linearmente separáveis. O **teorema da convergência do Perceptron** garante que, se os padrões de treinamento vêm de classes linearmente separáveis, o perceptron converge e posiciona a superfície de decisão como um hiperplano entre as duas classes.

3.6 Multilayer Perceptron (MLP)

O problema do XOR se resolve adicionando uma camada intermediária de processadores. Nos slides, a solução é construída combinando as funções lógicas **OR**, **NAND** e **AND**, com retas de separação explicitadas: `20x1 + 20x2 - 10` corresponde a $x_2 = -x_1 + 0{,}5$, e `-20x1 - 20x2 + 30` corresponde a $x_2 = -x_1 + 1{,}5$. Duas fronteiras lineares combinadas resolvem um problema não linearmente separável.

A MLP é uma extensão do Perceptron de Rosenblatt, composta de várias camadas de neurônios, e é a arquitetura clássica mais utilizada. Contém três tipos de camadas: **camada de entrada**, **camada(s) escondida(s) ou intermediária(s)** e **camada de saída**. Qualquer neurônio de uma camada pode se interligar a qualquer neurônio da camada seguinte.

O treinamento da MLP é supervisionado, feito com **Backpropagation** (retropropagação do erro, baseado na regra de aprendizado por correção de erro). O desafio central que ele resolve é encontrar como atualizar os pesos das camadas intermediárias, onde não há saída desejada explícita: o sinal de erro é determinado recursivamente, começando nos neurônios da camada de saída e propagando-se até as camadas intermediárias.

3.7 Processamento neural: aprendizado e recall

O processamento de uma rede neural divide-se em duas fases:

- **Aprendizado**: processo de atualização dos pesos sinápticos para aquisição do conhecimento — a aquisição da informação.
- **Recall**: processo de cálculo da saída da rede dado um padrão de entrada — a recuperação da informação.

Como recurso de apoio, os slides indicam o **Neural Network Playground** (`http://playground.tensorflow.org`) para experimentação visual com arquiteturas e hiperparâmetros.

3.8 Estudos de caso

Dois conjuntos de dados estruturados foram usados nesta aula:

- **Predição da faixa de preço de celulares** (Kaggle, mobile-price-classification): 2000 observações, 20 atributos e 4 classes de faixa de preço. Os atributos incluem `battery_power` (energia da bateria em mAh), `blue` (bluetooth), `clock_speed`, `dual_sim`, `fc` (megapixels da câmera frontal), `four_g`, `int_memory`, `m_dep` (profundidade em cm), `mobile_wt`, `n_cores`, `pc` (megapixels da câmera principal), `px_height` e `px_width` (resolução), `ram`, `sc_h` e `sc_w` (altura e largura da tela), `talk_time`, `three_g`, `touch_screen` e `wifi`.
- **Câncer de mama** (University of Wisconsin, Clinical Sciences Center): 30 atributos mais classe e id, entre eles raio (distância média do centro aos pontos do perímetro do tumor), textura (desvio padrão dos valores em escala de cinza), perímetro e área. São 569 instâncias, sendo 357 benignas e 212 malignas.

4 Introdução à Redes Neurais Artificiais - Parte 2

4.1 Definições operacionais de treinamento

Três definições são retomadas com cuidado porque governam todo o restante do curso:

- **Época**: a apresentação de todos os registros da base de dados à rede, com a retropropagação do erro para ajuste dos pesos. Em uma época podem ocorrer inúmeras iterações; a cada iteração há um ajuste de pesos. Cinquenta épocas significam que todos os registros são apresentados à rede cinquenta vezes.
- **Batch size**: quantos registros compõem cada iteração. Numa base com 4 registros, `batch_size = 4` implica 1 iteração por época, ou seja, 1 ajuste de pesos; `batch_size = 2` implica 2 iterações por época, ou seja, 2 ajustes de pesos.
- **Loss (função custo)**: a função que calcula o erro entre o valor real e o previsto. O exemplo dos slides compara uma saída prevista `[0.7, 0.2, 0.1]` com a saída real `[1.0, 0.0, 0.0]`.

4.2 Algoritmos de aprendizado

As propriedades de importância primordial de um algoritmo de aprendizado são aprender a partir do ambiente e melhorar o desempenho através da aprendizagem. A rede aprende sobre seu ambiente por um processo iterativo de ajustes aplicados a pesos sinápticos e níveis de bias; idealmente, torna-se mais instruída após cada época.

Um algoritmo de aprendizado é um conjunto preestabelecido de regras bem definidas para a solução de um problema de aprendizado. Os slides listam a variedade existente: por correção de erro, baseada em memória, hebbiana, competitiva, Boltzmann e Backpropagation (com Gradiente Descendente e Levenberg-Marquardt).

4.3 Backpropagation

O algoritmo opera em dois passos:

- **Feed Forward (propagação)**: um padrão é apresentado à camada de entrada e propagado em direção à camada de saída; a saída obtida é comparada com a desejada e o erro é calculado.
- **Feed Backward (retropropagação)**: o erro é propagado da camada de saída até a camada de entrada, atualizando os pesos das conexões dos neurônios internos conforme o erro retropropaga. Nas camadas intermediárias, o sinal do erro é determinado recursivamente em termos dos erros dos neurônios diretamente conectados a elas e dos pesos dessas conexões.

Suas **vantagens** são a simplicidade de implementação e a boa capacidade de generalização. Suas **desvantagens** são o custo computacional significativo e a baixa velocidade de aprendizado.

A grandeza minimizada é a soma dos erros quadráticos sobre todos os padrões e todos os processadores da camada de saída, onde `s_pk` é o estado de ativação do processador k ao apresentar o padrão p e `t_pk` é a saída desejada correspondente:

$E_{SSE} = \frac{1}{2} \sum_p \sum_k (t_{pk} - s_{pk})^2$

A minimização é feita pelo método do **Gradiente Descendente**: cada peso sináptico i do elemento processador j é atualizado proporcionalmente ao negativo da derivada parcial do erro em relação a esse peso.

$\Delta w_{ji} = \eta \frac{\partial E_{SSE}}{\partial w_{ji}}$

4.4 Otimizadores e hiperparâmetros

Três variantes do gradiente foram trabalhadas: **Gradiente Descendente**, **Gradiente Descendente Estocástico** e **Gradiente Descendente em Mini Batch** — as duas últimas diferindo do primeiro pela quantidade de exemplos usada para estimar o gradiente a cada atualização.

Sobre a **taxa de aprendizado** (η): quanto menor o parâmetro, menores as variações dos pesos e mais suave a trajetória, ao custo de um aprendizado mais lento; uma taxa muito grande pode tornar a rede instável.

O **termo de momento** (α) acrescenta à atualização atual uma fração da atualização anterior:

$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \Delta w_{ji}(n)$

O ganho prático é expressivo no exemplo mostrado: o gradiente descendente puro levou **690 iterações**, enquanto com termo de momento foram **84 iterações**.

Um exemplo de implementação ilustrativa em Python:

```python
# Atualizacao de pesos: gradiente descendente com e sem momento
def passo_gd(w, grad, eta):
    return w - eta * grad

def passo_gd_momento(w, grad, eta, alpha, delta_anterior):
    delta = alpha * delta_anterior + eta * grad
    return w - delta, delta
```

4.5 Capacidade e limitações da MLP

Foi demonstrado que a MLP é um **Aproximador Universal**, isto é, pode representar qualquer função. O Backpropagation e suas variações são os algoritmos de treinamento mais utilizados em aplicações práticas de previsão, classificação e reconhecimento de padrões.

O **overfitting** (supertreinamento) ocorre quando a rede fica especializada nos dados de treinamento e perde a capacidade de extrapolar para dados nunca vistos. Graficamente, é o momento em que o erro no conjunto de treinamento continua caindo enquanto o erro no conjunto de validação passa a subir.

Os **critérios de parada** discutidos incluem: número de épocas, taxa absoluta de variação do erro médio quadrático por época suficientemente pequena, e avaliação da capacidade de generalização por validação cruzada (**K-fold** e **leave-one-out**).

Apesar do sucesso do Backpropagation, três problemas persistem:

1. Definição do tamanho da rede. Trata-se de definir o número de camadas escondidas e o número de processadores em cada uma, buscando um compromisso entre **convergência** e **generalização**. Convergência é a capacidade da rede de aprender todos os padrões do conjunto de treinamento: uma rede pequena demais não consegue armazenar os padrões necessários (underfitting). Generalização é a capacidade de responder corretamente a padrões nunca vistos: uma rede grande demais modela também o ruído (overfitting). A rede não deve ser rígida a ponto de não modelar os dados, nem excessivamente flexível a ponto de modelar o ruído.

2. Paralisia da rede neural. Com o treinamento, os pesos podem alcançar valores muito grandes; a soma ponderada `net_j` de cada processador cresce junto; a derivada da função de ativação nesse ponto tende a zero; logo, a variação de peso tende a zero e a rede deixa de aprender. Para evitar: escolher valores de pesos e bias uniformemente distribuídos dentro de um intervalo pequeno, e dimensionar o número de processadores nas camadas escondidas de acordo com a necessidade.

3. Mínimos locais. A atualização dos pesos é função da taxa de aprendizado, que não deve ser nem muito pequena (treinamento lento) nem muito grande (oscilações). Com η pequeno e dependendo da inicialização aleatória dos pesos, pode não ser possível calcular um ajuste que tire a rede de um mínimo local — qualquer mudança no peso faz o erro aumentar. Com η grande, as mudanças não são suficientemente pequenas para atingir o mínimo global, e a rede fica oscilando em torno dele. As soluções apontadas são usar **taxa de aprendizado adaptativa** e o **termo de momento**.

4.6 Levenberg-Marquardt

A principal crítica ao Backpropagation é o longo tempo de treinamento. O algoritmo de **Levenberg-Marquardt (LM)** é um método mais rápido para treinar redes feed-forward de tamanho moderado (até algumas centenas de pesos sinápticos), e por ser baseado em matrizes tem implementações eficientes em Matlab, R e Python.

O Backpropagation tradicional baseia-se na derivada de primeira ordem da saída em relação aos pesos. Acelerar o treinamento exigiria derivadas de segunda ordem, mas o cálculo da **matriz Hessiana** é muito custoso computacionalmente. O LM é projetado para atingir velocidades de treinamento de segunda ordem sem calcular explicitamente a Hessiana, apoiando-se na **matriz Jacobiana**, que contém as primeiras derivadas parciais dos erros em relação aos pesos sinápticos.

4.7 Guia de funções de ativação

Um material complementar apresenta uma tabela de orientação sobre onde usar cada função de ativação. As recomendações usuais são:

- **Degrau**: apenas para saída binária.
- **Linear**: camada de saída em problemas de regressão.
- **Sigmoid** e **Tanh**: não lineares, aplicáveis em camada escondida e em camada de saída, incluindo casos multiclasse e multilabel.
- **Softmax**: camada de saída em problemas multiclasse.
- **ReLU**: não linear, tipicamente em camadas escondidas.

A distinção entre os dois regimes de saída categórica é importante: **multiclasse** significa que a rede infere somente uma classe entre todas; **multilabel** significa que a rede pode inferir nenhuma, uma ou mais de uma classe por entrada. Os slides advertem que a tabela indica o mais usual e que configurações diferentes podem ocorrer em problemas específicos.

4.8 Estudos de caso adicionais

- **Expectativa de vida** (World Health Organization): 20 atributos mais classe, incluindo ano, status, mortalidade em adultos e consumo de álcool, com 2938 registros.
- **KDD Cup**: detecção de intrusão em ambiente militar (rede local da U.S. Air Force), base preparada e gerenciada pelo MIT Lincoln Labs, com atributos de tráfego da rede a ser protegida. A base original tem 5 milhões de registros, 24 classes e 41 atributos.
- **Câncer de mama**, retomado da aula anterior.

5 Introdução ao Processamento de Imagens

5.1 Visão computacional

A **Visão Computacional** é o campo da inteligência artificial que capacita computadores a "ver" e interpretar o mundo a partir de imagens e vídeos digitais. Não se trata apenas de capturar fotos, mas de entender conteúdo e contexto. Embora a visão seja intuitiva para humanos, o sistema visual humano é extremamente complexo e envolve grande parte do cérebro; replicá-lo em máquinas exige algoritmos sofisticados e muita pesquisa.

5.2 O que é uma imagem

Uma **imagem** é uma representação espacial de uma cena 2D ou 3D, capturando informação visual do mundo real ou gerada digitalmente. Imagens digitais são compostas por **pixels**, elementos fundamentais organizados em uma grade de linhas e colunas, cada um armazenando informação de cor ou intensidade. Elas são criadas por dispositivos como câmeras e sensores, que convertem ondas eletromagnéticas em sinal por meio de fotossensores, com propriedades de **amplitude** e **frequência**.

Matematicamente, uma imagem digital é representada como uma função `f(x, y)` com valores discretos de intensidade, onde x e y são coordenadas espaciais.

- **Escala de cinza**: cada pixel possui um único valor representando brilho, geralmente de 0 a 255, sendo 0 preto completo, 255 branco puro e os valores intermediários tons de cinza.
- **RGB colorida**: cada pixel armazena três valores (vermelho, verde e azul) em três canais independentes; a combinação gera milhões de cores, com cada canal variando de 0 a 255 em imagens de 8 bits.

5.3 Representação de cores

- **RGB**: tensor tridimensional de dimensões largura por altura por 3 canais. Cada "fatia" representa a intensidade de vermelho, verde ou azul. É o formato dominante em displays digitais e se baseia em combinações de luz primária (modelo aditivo).
- **HSV/HSL**: focam em como humanos percebem cor, com **Hue** (matiz, o tipo de cor), **Saturation** (saturação, a pureza da cor) e **Value/Lightness** (valor ou luminosidade, o brilho). São ideais para ajustes intuitivos de cor.
- **CMYK**: ciano, magenta, amarelo e key (preto), usado em impressoras. É um modelo **subtrativo**, no qual as tintas absorvem luz, ao contrário do RGB, que emite luz.

5.4 Qualidade das imagens

A **quantização** determina quantos níveis distintos de intensidade podem ser representados. Em um bit armazenam-se 2 valores (0 e 1); em dois bits, 4 valores (00, 01, 10, 11); em 8 bits (1 byte), 256 valores; em 16 bits, 65536 valores. Quanto maior a profundidade de bits, mais finas as diferenças de brilho representáveis.

A **resolução** define quantos pixels compõem a grade da imagem, determinando o nível de detalhe espacial preservado.

5.5 Objetivos do processamento de imagens

Os slides organizam os objetivos em três frentes:

- **Interpretação humana**: melhorar a qualidade visual das imagens para facilitar análise e compreensão por observadores humanos.
- **Otimização técnica**: preparar imagens para armazenamento eficiente e transmissão por redes e sistemas.
- **Análise automática**: processar dados visuais para reconhecimento de padrões e decisões por sistemas de IA.

5.6 Operações básicas

Filtragem e operações em vizinhança. O filtro processa cada pixel considerando seus vizinhos por meio de uma máscara. O exemplo dado é o **filtro de mediana** aplicado para suavizar ruído sal (branco) e pimenta (preto), calculando a mediana dos pixels vizinhos para cada posição. O fluxo é: imagem original, com possível ruído ou imperfeições, aplicação da máscara, e resultado filtrado, suavizado, realçado ou com ruído reduzido.

Transformações pontuais. Modificam cada pixel individualmente, sem considerar vizinhos, sendo as transformações mais simples e rápidas. Exemplos: **inversão** (o claro torna-se escuro) e **clareamento** (intensidade aumentada uniformemente).

Histogramas. Um histograma é uma representação gráfica da distribuição dos níveis de intensidade (brilho) de uma imagem. O eixo horizontal representa a intensidade, tipicamente de 0 (preto absoluto) a 255 (branco puro) para 8 bits; o eixo vertical indica a quantidade de pixels com cada nível de intensidade. O histograma é crucial para entender:

- **Clareamento e escurecimento** (sub ou superexposição, que deslocam a distribuição de intensidades);
- **Contraste** (faixa dinâmica dos tons, isto é, a amplitude da faixa de intensidades utilizada);
- **Quantização** (número de níveis possíveis, por exemplo 8 ou 16 bits);
- **Qualidade radiométrica** (capacidade de distinguir diferenças sutis de brilho).

Equalização de histograma. A equalização global força a imagem a ocupar melhor toda a escala de intensidades, aumentando o contraste global: um histograma originalmente concentrado, indicando baixo aproveitamento da faixa dinâmica, torna-se bem mais espalhado. O **CLAHE** (equalização adaptativa com limitação de contraste) melhora o contraste **local**: equaliza pequenos blocos localmente e limita o contraste, não ocupando a escala de forma tão agressiva quanto a equalização global.

Um exemplo ilustrativo com OpenCV:

```
import cv2

img = cv2.imread("castelooriginal.png", cv2.IMREAD_GRAYSCALE)

# Equalizacao global de histograma
img_equalizada = cv2.equalizeHist(img)

# Equalizacao local com limitacao de contraste (CLAHE)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
img_clahe = clahe.apply(img)
```

5.7 Aplicações práticas e ponte para deep learning

As técnicas básicas são fundamentais para preparar dados visuais para visão computacional e deep learning. Três aplicações são destacadas:

- **Detecção de bordas**: identifica limites e contornos de objetos usando gradientes de intensidade;
- **Segmentação**: divide a imagem em regiões significativas baseadas em características visuais;

- **Change detection**: identifica mudanças em imagens pareadas.

A progressão proposta para o restante do curso é: (1) fundamentos de processamento básico de imagens; (2) características, com extração e análise de features; (3) redes neurais, com CNNs e Vision Transformers; (4) aplicações, com reconhecimento e decisões inteligentes.

5.8 Estudo de caso

O **MNIST**, com dígitos de 0 a 9 escritos à mão. Curiosidades registradas nos slides: Yann LeCun foi um dos criadores do dataset, e a motivação era servir de benchmark para reconhecimento de dígitos e viabilizar o processamento automático de cheques.

6 Introdução ao Processamento de Imagens - Continuação

Esta aula deu continuidade ao mesmo conjunto de slides da aula anterior (`LLMMaster_Intro_Processamento_Image`) retomando integralmente os tópicos de aquisição, representação matemática da imagem, espaços de cor RGB, HSV/HSL e CMYK, quantização em 8 e 16 bits, resolução, filtragem em vizinhança, histogramas, transformações pontuais e equalização de histograma global e CLAHE, além do estudo de caso do MNIST.

O foco da continuação, conforme indicado pelos arquivos de apoio (`Mnist.ipynb` e `Mnist_solução.ipynb`, além das imagens `imgoriginal.png`, `imgequalizada.png`, `imgclahe.png` e `castelooriginal.png`), esteve na parte prática: aplicar em código as operações de realce estudadas e construir um primeiro classificador de dígitos.

6.1 Consolidação prática

O ponto de convergência dos dois encontros é a compreensão de que **pré-processamento não é acessório**. A escolha da representação de cor, a profundidade de bits, o realce de contraste e a filtragem de ruído condicionam diretamente o que uma rede neural conseguirá aprender. Uma imagem com baixo aproveitamento da faixa dinâmica contém informação visual, mas apresenta ao modelo diferenças de intensidade muito comprimidas; a equalização redistribui essas intensidades e torna as diferenças mais discrimináveis.

Um esqueleto ilustrativo do carregamento do MNIST:

```
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalizacao das intensidades para o intervalo [0, 1]
x_train = x_train / 255.0
x_test = x_test / 255.0

print(x_train.shape)  # (60000, 28, 28)
```

7 Redes Convolucionais

7.1 O esquema básico de um projeto de visão computacional

Antes das CNNs propriamente ditas, a aula estabelece o pipeline de um projeto de visão computacional, começando pela **análise exploratória e pré-processamento**.

Desbalanceamento de classes. As estratégias de tratamento dividem-se em duas famílias. Baseadas em **dados**: oversampling aleatório, undersampling aleatório, transformações geométricas, transformações de cor, ruídos e filtros, sobreposição de imagens com transparência. Baseadas em **algoritmos**: função custo ponderada, classificação de uma classe somente (por exemplo one-class SVM), autoencoders treinados para uma única classe e modelos generativos.

Os slides alertam para o **Paradoxo da Acurácia**: em uma matriz de confusão fortemente desbalanceada, um modelo pode exibir 96,08% de acurácia enquanto outro exibe 94,11%, sem que o primeiro seja de fato melhor no que importa — a acurácia global mascara o desempenho na classe minoritária.

Inspeção visual e metadados. Recomenda-se visualizar uma pequena amostra da base observando características de shape, cor, iluminação, posição, qualidade e classe, além dos pixels em largura e comprimento. Metadados de captura e técnicos também podem ser relevantes: data, hora, localização, configuração da câmera e profundidade de bits (8, 16, 32 bits).

Adaptação de domínio e análise multivariada. Além de equalização e CLAHE, o **Histogram Matching** padroniza a distribuição de intensidade (brilho) segundo uma imagem de referência — um recurso de adaptação de domínio. Técnicas de redução de dimensionalidade como **PCA**, **t-SNE** e **uMAP** servem para análise de outliers e anomalias. Também se recomenda **análise de variância** com estatísticas de mínimo, máximo, mediana e média, eventualmente com limiar.

Preparação espacial. Três operações são apresentadas sobre a imagem original: **crop**, **redimensionamento** e extração de **patches**.

7.2 Data Augmentation

Data augmentation é criar novas variações dos dados a partir dos dados existentes, sem coletar novos exemplos. Os objetivos são reduzir overfitting, aumentar robustez e ensinar a rede a ignorar variações irrelevantes. A importância central é a **generalização do modelo**.

Os tipos são:

- **Geométricos**: os mais populares (rotação, zoom, espelhamento). O princípio subjacente é que o que importa não é onde está o objeto, mas o que ele é.
- **Fotométricos**: alteram a aparência (brilho, contraste, saturação). Ensinam a CNN a não depender da qualidade perfeita da imagem. Exigem critério: alterações de cor, brilho ou contraste podem introduzir variações irrelevantes ou distorcer informação semântica crítica. O exemplo dado é a alteração de **Hue** (matiz), que gira o tom da cor — de verde para amarelo, por exemplo. Quando a cor é semântica, como em diagnóstico médico, bandeiras ou sinais de trânsito, essa transformação pode confundir o modelo e levar a classificações incorretas.
- **Estruturais**: técnicas mais avançadas, como **Mixup**, **Cutout** e **Cutmix**.
- **No espaço de features**: variações geradas no espaço de representação.

Sobre esta última categoria, os slides observam que às vezes a variação acontece dentro da própria rede:

- **Dropout**: desativa aleatoriamente neurônios durante o treinamento, forçando o modelo a aprender representações mais robustas e generalizáveis, atuando diretamente nas features internas.
- **Noise interno**: adiciona ruído a camadas intermediárias ou ativações, introduzindo variabilidade nas representações extraídas em vez de manipular a imagem original.
- **Augmentação implícita**: ocorre naturalmente devido à arquitetura ou ao processo de treinamento (por exemplo, normalização de batch e funções de ativação), introduzindo regularização e diversidade no espaço de features sem modificar os dados de entrada.

7.3 Compromisso viés-variância

Válido para modelos de machine learning em geral: **viés (bias)** é a incapacidade do modelo de capturar o verdadeiro relacionamento entre os dados; **variância** é a diferença no resultado do modelo para diferentes datasets. O equilíbrio entre ambos governa a escolha da capacidade do modelo e a intensidade da regularização.

7.4 O problema das redes tradicionais

Redes neurais tradicionais tratam cada pixel como um valor independente, "achatando" a imagem em uma lista enorme. A sequência é: imagem 2D com representação espacial, achatamento que transforma a matriz em lista linear, vetor 1D com sequência longa de números, e **perda espacial** — as relações entre pixels vizinhos são quebradas. Isso destrói informação crucial: exatamente o que torna uma imagem significativa. Usar os pixels achatados como input limita a performance do modelo.

7.5 A ideia central das CNNs

Em vez de processar todos os pixels de uma vez, as CNNs analisam **pequenas regiões** da imagem por vez, preservando a estrutura espacial. A analogia dos slides é examinar uma pintura com uma lupa, notando detalhes locais antes de compreender o quadro inteiro.

O que é uma convolução? Um pequeno quadrado (filtro) percorre toda a imagem como um scanner e, em cada posição, detecta padrões específicos. Diferentes filtros procuram diferentes características: bordas, linhas, texturas, cores específicas.

Padding. Em alguns cenários é desejável que a saída tenha o mesmo tamanho da entrada, o que se obtém adicionando padding. Além disso, o padding facilita que o modelo capture características importantes localizadas nas bordas da imagem. Em geral faz-se **zero-padding** (que não cria informação falsa), mas também é possível usar valor médio dos pixels, valor mínimo, valor replicado da borda ou reflexão da imagem.

Stride. Com imagens grandes pode ser desejável usar strides maiores, deslocando o filtro por vários pixels de uma vez. Isso reduz o tamanho da saída, mas strides maiores podem fazer com que características importantes não sejam capturadas.

Pooling. É comum adicionar uma camada de pooling após uma camada convolucional. O objetivo é reduzir o tamanho das features geradas pela convolução, melhorando a eficiência computacional. Dependendo do tipo, a redução preserva as ativações mais fortes (**Max Pooling**) ou a informação média da região (**Average Pooling**), ajudando a tornar o modelo mais robusto a ruídos.

Flatten e camadas densas. Convertem os mapas de características em um vetor único, permitindo que a rede tome a decisão final por meio de camadas densas.

Um exemplo mínimo de arquitetura:

```
from tensorflow.keras import layers, models

modelo = models.Sequential([
    layers.Conv2D(32, (3, 3), activation="relu", padding="same",
                  input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),
])

modelo.compile(optimizer="adam",
               loss="sparse_categorical_crossentropy",
               metrics=["accuracy"])
```

7.6 Hierarquia de características

O ponto mais importante conceitualmente é que cada camada aprende padrões mais sofisticados sobre o conhecimento das camadas anteriores, formando uma **hierarquia visual natural**:

- **Camadas iniciais**: padrões simples — bordas (onde objetos começam e terminam), linhas e ângulos (orientações específicas e formas geométricas básicas), contraste (mudanças abruptas de cor e luminosidade).
- **Camadas intermediárias**: texturas, isto é, combinações de bordas; e partes de objetos — orelhas, olhos, rodas.
- **Camadas finais**: objetos completos — cachorro, carro, casa.

O exemplo do reconhecimento de um cachorro percorre esse caminho: (1) a foto é inserida na rede; (2) as primeiras camadas detectam bordas, contornos e contrastes locais; (3) as camadas intermediárias reconhecem textura, focinho, orelhas e regiões do corpo; (4) as camadas finais combinam partes e reconhecem a estrutura completa do animal; (5) a decisão final combina tudo e conclui a classe.

O insight destacado é que **ninguém programa a rede para procurar focinhos ou orelhas** — ela descobre isso sozinha. Durante o treinamento, a CNN vê milhares de imagens rotuladas e ajusta automaticamente seus filtros para identificar as características que melhor distinguem cada categoria. É um processo de tentativa e erro refinado.

7.7 Forças e limitações

Forças: precisão superior em tarefas visuais, aprendizado automático de padrões complexos, boa generalização para novas imagens e redução da necessidade de engenharia manual de features.

Desafios: precisam de muitos exemplos rotulados, requerem grande poder computacional, podem errar de forma não intuitiva e há dificuldade em explicar decisões específicas.

Sobre os **erros não intuitivos**: uma CNN pode ter 99% de precisão, mas seus erros ocasionais podem ser surpreendentes — pode confundir um cachorro com um muffin se os padrões visuais forem semelhantes. Isso ocorre porque CNNs reconhecem correlações estatísticas em padrões visuais, não significados conceituais como humanos.

O papel do dado de treinamento é decisivo. O ciclo é coleta de imagens, rotulagem em lote, treinamento da CNN e melhoria contínua. A performance depende diretamente da quantidade e diversidade dos dados. Dados enviesados levam a modelos enviesados: treinar apenas com fotos de cachorros brancos produzirá dificuldade com cachorros pretos.

7.8 Arquiteturas populares

- **LeNet (1998)**: primeira CNN bem-sucedida, usada para reconhecer dígitos escritos à mão.
- **AlexNet (2012)**: revolucionou o campo ao vencer a competição ImageNet com margem impressionante.
- **VGG (2014)**: demonstrou que redes mais profundas melhoram a precisão.
- **ResNet (2015)**: introduziu conexões residuais, permitindo redes com centenas de camadas.

7.9 O futuro

Quatro tendências são apontadas: modelos **mais eficientes** (menores com mesma precisão, rodando em dispositivos móveis); **mais interpretáveis** (técnicas para entender como as decisões são tomadas); **menos dependentes** de dados rotulados, via técnicas auto-supervisionadas; e **mais generalistas**, com modelos únicos capazes de resolver múltiplas tarefas visuais simultaneamente.

7.10 Estudo de caso

O **CIFAR-10**: 50 mil imagens de treino, 10 mil de teste, 10 classes, imagens de 32x32x3, 8 bits, RGB. Curiosidade registrada: Geoffrey Hinton foi um dos criadores do dataset, tanto do CIFAR-10 quanto do CIFAR-100, combinando um total de 80 milhões de imagens rotuladas.

8 Transfer Learning

8.1 Por que Transfer Learning

Treinar CNNs do zero exige muitas imagens anotadas e dias de processamento computacional intensivo. O **transfer learning** permite aproveitar modelos pré-treinados para acelerar drasticamente

o desenvolvimento e melhorar resultados. É fundamental em cenários com dados limitados, como imagens médicas, imagens de satélite ou nichos específicos da indústria.

A técnica aproveita o conhecimento de um modelo treinado em uma tarefa para aplicá-lo em outra tarefa relacionada. Em CNNs, reutilizam-se camadas convolucionais que já aprenderam a extrair características visuais genéricas como bordas, texturas e formas complexas. Essas redes podem ser ajustadas por **fine-tuning** ou simplesmente usadas como **extratores fixos de características**.

8.2 Cenários comuns

- **Extrator fixo de características**: congelar todas as camadas convolucionais e treinar apenas o classificador final. Ideal para datasets muito pequenos.
- **Fine-tuning parcial**: descongelar apenas as camadas superiores da rede para adaptar melhor ao novo domínio, mantendo as características genéricas.
- **Fine-tuning total**: treinar toda a rede com taxa de aprendizado baixa. Indicado para grandes datasets.

O processo prático é direto: parte-se de um modelo pré-treinado (por exemplo, em ImageNet com 1000 classes), removem-se as últimas camadas classificadoras e adiciona-se uma nova camada final adaptada às classes específicas da nova tarefa.

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models

base = VGG16(weights="imagenet", include_top=False,
             input_shape=(224, 224, 3))
base.trainable = False  # extrator fixo de caracteristicas

modelo = models.Sequential([
    base,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation="relu"),
    layers.Dense(2, activation="softmax"),  # nova tarefa
])
```

8.3 Por que CNNs são ideais para transfer learning

- **Hierarquia de características**: camadas iniciais capturam padrões universais como bordas, texturas e formas geométricas simples, úteis em praticamente qualquer tarefa visual.
- **Especialização progressiva**: camadas profundas aprendem padrões cada vez mais específicos da tarefa original, que podem ser adaptados ou substituídos.
- **Eficiência com poucos dados**: reutilizar camadas iniciais reduz drasticamente a necessidade de grandes volumes de dados e ajuda a evitar overfitting em datasets pequenos.

8.4 Arquiteturas de base

VGG16 e VGG19. Arquiteturas simples e profundas, muito utilizadas para extração de características visuais básicas e intermediárias. A VGG16 é composta por 16 camadas profundas que

utilizam exclusivamente convoluções pequenas de 3x3, criando uma arquitetura uniforme e fácil de entender. É pré-treinada no ImageNet, com 1,2 milhão de imagens distribuídas em 1000 classes.

ResNet (50, 101, 152). Redes residuais com conexões de atalho que facilitam o treinamento de redes extremamente profundas. A motivação é clara nos slides: a ideia era melhorar performance aumentando profundidade, mas o problema encontrado foi a **degradação de desempenho** — uma dificuldade de otimização, não de overfitting. A proposta são as **skip connections**, que criam dois caminhos possíveis: o mapeamento identidade, caso as camadas extras não sejam necessárias, e a função residual, que aprende apenas a correção sobre a identidade.

O artigo original (He, Zhang, Ren e Sun, Microsoft Research) formaliza isso: em vez de esperar que camadas empilhadas aproximem um mapeamento desejado `H(x)`, deixa-se que elas aproximem a função residual `F(x) = H(x) - x`, de modo que o mapeamento original se torna `F(x) + x`. A hipótese é que otimizar o mapeamento residual é mais fácil do que otimizar o mapeamento original não referenciado; no extremo, se o mapeamento identidade for ótimo, é mais fácil empurrar o resíduo para zero do que ajustar uma identidade com camadas não lineares. As shortcut connections identidade não adicionam nem parâmetros extras nem complexidade computacional, e a rede continua treinável end-to-end por SGD com backpropagation. Resultados reportados: redes residuais com até 152 camadas no ImageNet, 8 vezes mais profundas que as VGG mas com menor complexidade; um ensemble atingiu 3,57% de erro top-5 no conjunto de teste do ImageNet, vencendo o primeiro lugar na tarefa de classificação do ILSVRC 2015, além de primeiros lugares em detecção e localização no ImageNet e detecção e segmentação no COCO. O artigo também documenta o fenômeno de degradação com redes "plain": no CIFAR-10, uma rede de 56 camadas apresentou erro de treinamento maior do que uma de 20 camadas.

Inception (v3, v4). Arquitetura sofisticada com módulos que capturam informações em múltiplas escalas simultaneamente, combinando convoluções de tamanhos diferentes (1x1, 3x3, 5x5) em paralelo dentro de cada módulo. Isso aumenta a capacidade representacional da rede. Além disso, usa convoluções 1x1 para redução de dimensionalidade, diminuindo drasticamente o custo computacional sem perda de performance. Vantagens-chave: processamento multi-escala, eficiência computacional, alto desempenho e versatilidade.

A motivação da Inception também parte de melhorar performance aumentando profundidade, mas encontrando como problemas o número maior de parâmetros, overfitting e custo computacional. A hipótese de partida é que **a rede neural ótima é esparsa**. A analogia utilizada — retomada em profundidade no artigo da **Lottery Ticket Hypothesis** (Frankle e Carbin, ICLR 2019) — é que a inicialização aleatória equivale a comprar bilhetes de loteria, poucos bilhetes são premiados, e o pruning revela quais são os vencedores. O problema prático é que GPUs modernas são altamente otimizadas para operações densas, o que limita o ganho de modelos esparsos em treinamento e inferência.

A solução da Inception é uma **esparsidade simulada**: convoluções 1x1 para controlar profundidade (canais) e stacking de convoluções 1x1, 3x3 e 5x5 em paralelo, organizados em blocos de Inception. A arquitetura inclui ainda **classificadores auxiliares** conectados a camadas intermediárias, que evitam o vanishing gradient em uma rede muito profunda.

O artigo da Lottery Ticket Hypothesis, incluído como leitura, sustenta que redes densas inicializadas aleatoriamente contêm sub-redes ("bilhetes premiados") que, quando treinadas isoladamente, atingem acurácia de teste comparável à rede original em número similar de iterações. Os autores encontram consistentemente bilhetes premiados com menos de 10 a 20% do tamanho de várias arquiteturas totalmente conectadas e convolucionais para MNIST e CIFAR-10.

8.5 Benefícios, desafios e boas práticas

Benefícios: velocidade acelerada, com redução do tempo de treinamento de semanas ou meses para horas ou dias; menos dados necessários, alcançando excelente desempenho com centenas ou poucos milhares de imagens anotadas em vez de milhões; e robustez aprimorada, já que modelos pré-treinados viram milhões de exemplos diversos.

Desafios e cuidados: risco de overfitting em fine-tuning excessivo sobre datasets pequenos, exigindo monitoramento constante das métricas de validação; diferença de domínios, pois grande distância entre dados originais e novos pode exigir mais camadas treináveis e ajustes mais profundos; e balanceamento de camadas, encontrando o equilíbrio certo entre congelar e descongelar.

Dicas: data augmentation inteligente (rotação, flip, zoom, ajustes de cor) para aumentar artificialmente a diversidade do dataset; monitoramento contínuo de métricas de treino e validação para detectar overfitting precocemente; e experimentação sistemática, testando congelar diferentes quantidades de camadas e comparando resultados.

9 Deploy de Modelo com FastAPI

9.1 Recapitulação do pipeline de aprendizado

A aula abre situando o modelo dentro do quadro geral de machine learning. No **aprendizado supervisionado**, cada registro tem atributos e um rótulo, e as tarefas são:

- **Classificação**: rótulo categórico (por exemplo, imagens de um equipamento classificadas como normal ou anomalia);
- **Regressão**: rótulo contínuo (por exemplo, temperatura ou estado);
- **Previsão de séries temporais**: rótulo contínuo e dependente do tempo, a partir de dados históricos.

No **aprendizado não supervisionado** há apenas atributos, sem rótulo, e as tarefas incluem **agrupamento** (descoberta de semelhanças e grupos entre registros) e **associação**.

Em seguida a aula recapitula neurônio artificial, funções de ativação, bias, topologia MLP, os dois passos do Backpropagation e os otimizadores (Gradiente Descendente, Estocástico e Mini Batch), além dos fundamentos de imagem, da ideia central das CNNs e das arquiteturas de transfer learning (VGG16/19, ResNet 50/101/152, Inception v3/v4).

9.2 Formas de disponibilizar um modelo

Um modelo treinado só gera valor quando fica acessível. As opções de deploy apresentadas são: aplicativo desktop ou web, executável por linha de comando, script bat, uso via código, e **API**. A vantagem decisiva da API é ser **agnóstica a linguagem**: qualquer cliente capaz de fazer uma requisição HTTP pode consumir o modelo, independentemente da linguagem em que foi escrito.

9.3 O objetivo prático da aula

Criar uma API rodando localmente usando o framework **FastAPI**, com documentação e utilização através do **OpenAPI (Swagger)**. Os slides indicam também a intenção de, mais adiante, fazer deploy no **Heroku**.

Os arquivos de apoio da aula (`main.py`, `app.py`, `requirements.txt`, `model_transfer.keras`, `inference.ipynb` e imagens de teste de cães e gatos) indicam a estrutura típica de um serviço de inferência: o modelo serializado em formato Keras, um módulo de aplicação expondo os endpoints e um arquivo de dependências.

Um esqueleto ilustrativo do serviço:

```
from fastapi import FastAPI, UploadFile, File
from tensorflow import keras
import numpy as np
from PIL import Image
import io

app = FastAPI(title="Classificador de Imagens")
modelo = keras.models.load_model("model_transfer.keras")

@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    conteudo = await file.read()
    img = Image.open(io.BytesIO(conteudo)).resize((224, 224))
    x = np.expand_dims(np.array(img) / 255.0, axis=0)
    probs = modelo.predict(x)[0]
    return {"classe": int(np.argmax(probs)),
            "confianca": float(np.max(probs))}
```

Executado localmente, o FastAPI gera automaticamente a documentação interativa via OpenAPI/Swagger, permitindo testar o endpoint pelo navegador:

```
uvicorn main:app --reload
# documentacao interativa em /docs
```

9.4 Estudo de caso

Dogs vs. Cats: 25 mil imagens rotuladas em 2 classes. O modelo de classificação obtido por transfer learning nessa base é exatamente o artefato colocado em produção pela API.

10 Vision Transformers

10.1 O Transformer original

Proposto por **Vaswani et al. (2017)** no artigo "Attention Is All You Need", o Transformer revolucionou o processamento de linguagem natural ao substituir redes recorrentes por mecanismos de **self-attention**. Três propriedades explicam o impacto:

- **Paralelização**: processa todos os tokens simultaneamente, diferentemente de RNNs sequenciais.
- **Self-attention**: cada elemento pode "prestar atenção" a qualquer outro elemento da sequência.
- **Escalabilidade**: permite treinar modelos com bilhões de parâmetros em grandes datasets.

A arquitetura original tem três componentes principais: o **Encoder**, que processa a sequência de entrada transformando-a em representações contextuais ricas; o **Decoder**, que gera a sequência de saída token a token usando a saída do encoder e sua própria atenção; e o **Positional Encoding**, que injeta informação de ordem dos tokens, compensando a natureza não sequencial do self-attention.

10.2 Do token à atenção: o fluxo completo

1. **Tokenização**: o texto é dividido em unidades menores (tokens) mapeadas para índices inteiros de um vocabulário. Exemplo dos slides: "Transformer é poderoso" torna-se `['transformer', 'é', 'poderoso']` e depois `[1224, 57, 9813]`.
2. **Embedding aprendido**: cada token é transformado em vetor denso que captura seu significado semântico. É uma matriz aprendida e treinada end-to-end. Exemplo: vocabulário de 50000 e `d_model` de 512.
3. **Informação posicional**: adiciona-se um positional encoding aos embeddings. Em 2017 eram senoidais; modelos modernos geralmente aprendem esses encodings.
4. **Projeções lineares**: a partir da representação de entrada X, três projeções lineares separadas geram Query (Q), Key (K) e Value (V), por meio de matrizes de pesos aprendidas `W_Q`, `W_K` e `W_V`, camadas lineares sem ativação ajustadas por backpropagation.
5. **Cálculo da atenção**: Q, K e V são usados para calcular o resultado da atenção, ponderando cada token pela sua relevância em relação aos outros.

A mensagem-chave é que **tudo no Transformer é aprendido de forma conjunta**, dos embeddings iniciais até os pesos de atenção.

10.3 Query, Key e Value

O coração do Transformer é o **Scaled Dot-Product Attention**, que mede a similaridade entre projeções aprendidas Q e K dos elementos da sequência, determinando a relevância entre eles para compor a saída ponderada via V. O fator de escala pela raiz de `d_k` estabiliza os gradientes durante o treinamento.

A intuição apresentada é a de um sistema de recuperação de informação:

- **Query**: representa o que estou buscando, a informação para a qual desejo encontrar relevância.
- **Key**: representa o que eu ofereço, a característica de um item que pode corresponder a uma Query.
- **Value**: representa qual informação eu carrego associada a uma Key, que será agregada se sua Key for relevante para a Query.

A analogia da biblioteca digital detalha o processo: a Query é a pergunta feita ao sistema; as Keys são os títulos e resumos de todos os livros disponíveis; os Values são os conteúdos completos. A pergunta é comparada com cada título e resumo para medir relevância; os livros mais alinhados

recebem peso maior; e o resultado é uma combinação ponderada dos conteúdos mais relevantes. O ponto crucial é que **a atenção não seleciona um único item, mas constrói nova informação combinando de forma ponderada os valores mais relevantes**. Cada token decide dinamicamente de quem ele precisa "ouvir" na sequência.

Note-se ainda que o mecanismo não compara os embeddings originais diretamente, mas suas representações projetadas — o que permite capturar tipos diferentes de relação de forma mais rica e flexível.

10.4 Multi-Head Attention

Em vez de uma única função de atenção, o Transformer usa múltiplas cabeças em paralelo, permitindo capturar diferentes tipos de relações simultaneamente: uma cabeça pode focar relações locais, outra relações globais, outra padrões semânticos, e assim por diante. Cada cabeça aprende projeções diferentes, e a concatenação final é projetada de volta ao espaço original.

10.5 Add and Norm, e a FFN

Após calcular a atenção, obtém-se um novo vetor contextualizado para cada token, mas o processamento continua.

Passo 1 — Residual Connection (Add). A saída da atenção é somada à entrada original: `Z = X + Attention(X)`. Isso preserva a informação original, facilita o fluxo de gradientes e permite treinar modelos profundos. Os slides destacam que se trata do mesmo princípio estrutural da ResNet: cada bloco aprende um refinamento da representação anterior.

Passo 2 — Layer Normalization (Norm). O resultado da soma residual é normalizado: `Output = LayerNorm(Z)`. Normaliza cada vetor, controla escala e variância e mantém o treinamento estável, funcionando como um z-score aprendido que estabiliza a normalização sem limitar a capacidade do modelo.

A síntese dos slides é precisa: **a atenção adiciona contexto, o residual preserva identidade, a normalização estabiliza o aprendizado**.

FeedForward Network (FFN). Também chamada de MLP, é composta por uma primeira matriz `W_1` que expande a dimensão da representação, uma função de ativação não linear (geralmente GELU ou ReLU) e uma segunda matriz `W_2` que projeta de volta à dimensão original `d_model`. A intuição é complementar: o multi-head attention funciona como especialistas olhando relações diferentes, cuja concatenação junta as opiniões; a FFN refina a representação final dessas informações concatenadas, e sua não linearidade permite aprender funções mais complexas.

10.6 O Decoder

O decoder gera a sequência de saída passo a passo. No exemplo de tradução dos slides, a entrada do encoder é "Eu gosto de café" e a saída esperada é "I like coffee".

- **Shifted Right (treinamento)**: o target original `I | like | coffee` entra no decoder deslocado como `<s> | I | like`.

- **Masked Multi-Head Attention**: cada posição na sequência de saída só pode olhar para os tokens já gerados; uma máscara impede que o decoder veja tokens futuros, garantindo que a geração seja autoregressiva.
- **Cross-Attention (Encoder-Decoder Attention)**: o decoder também olha para a saída do encoder, com a Query vindo do próprio decoder e as Keys e Values vindo do encoder.

10.7 Por que self-attention é poderoso

Em LLMs, cada palavra considera e pesa a relevância de todas as outras palavras no contexto, formando uma representação final rica em relações semânticas. Em ViTs, cada patch de uma imagem presta atenção a todos os outros patches, capturando relações de longo alcance e contexto global. Os dois ganhos centrais são a **captura de dependências longas**, correlacionando informações distantes na sequência ou no espaço, e a **modelagem de contexto global**, na qual cada elemento tem visão abrangente de todo o input e não apenas de uma vizinhança local.

10.8 De texto para imagens: o ViT

O **Vision Transformer**, proposto por **Dosovitskiy et al. (2020)** no artigo "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" (ICLR 2021), adaptou a arquitetura Transformer para visão computacional com uma ideia: **tratar patches de imagem como tokens**. Assim como um texto é uma sequência de palavras, uma imagem pode ser tratada como uma sequência de patches.

Arquitetura. A imagem de entrada é dividida em patches fixos de 16x16, que são linearizados e projetados em embeddings, formando uma sequência processada por um **Transformer Encoder-Only** — não há geração de sequência ou tradução, apenas classificação. Um token especial [**CLS**] aprendível é adicionado à sequência como patch 0, e seu estado final é usado para a classificação.

Position embeddings. Como o Transformer não possui noção inerente de ordem, somam-se embeddings posicionais aprendidos a cada patch embedding para preservar a informação espacial. Duas variantes são citadas: posicionais **1D**, aprendidos para cada posição na sequência linearizada de patches, usados no ViT original; e posicionais **2D**, que codificam separadamente linha e coluna, com resultados similares aos 1D na prática.

10.9 ViT versus CNNs

Aspecto	CNNs (ResNet)	ViT
Viés indutivo	Forte (localidade, invariância a translação)	Fraco (aprende do dado)
Poucos dados	Desempenho superior	Tende a sobreajustar
Muitos dados	Saturação mais rápida	Escala melhor
Pré-treino	ImageNet é suficiente	Requer JFT-300M ou similar

Os resultados do ViT original confirmam essa leitura: pré-treinado no JFT-300M, o **ViT-H/14** atingiu **88,6%** de top-1 no ImageNet, com computação cerca de 4 vezes menor em relação a CNNs de ponta; já o **ViT-B/16** treinado apenas no ImageNet ficou em **77,9%**.

10.10 Família de modelos e evolução

O ViT foi proposto em três escalas, variando dimensão dos embeddings, número de camadas e cabeças de atenção:

Modelo	Camadas	Dim. hidden / Heads	Parâmetros
ViT-Base	12	768 / 12	86M
ViT-Large	24	1024 / 16	307M
ViT-Huge	32	1280 / 16	632M

A notação **ViT-B/16** indica modelo Base com patches de 16x16 pixels.

Entre as variantes posteriores:

- **DeiT (2021)**: Data-Efficient Image Transformers, com treinamento eficiente apenas no ImageNet usando distillation token.
- **Swin Transformer (2021)**: atenção local em janelas hierárquicas, eficiente e adequado para detecção e segmentação.
- **BEiT (2021)**: pré-treino auto-supervisionado inspirado no BERT, com masked image modeling.
- **DINOv2 (2023)**: ViT auto-supervisionado da Meta, com representações visuais de propósito geral, sem labels.

As aplicações se expandiram muito além da classificação: **detecção de objetos** (DETR e variantes, end-to-end), **segmentação** (SegFormer e Mask2Former, estado da arte em segmentação semântica e de instâncias), **geração de imagens** (DiT, Diffusion Transformer, usando ViTs como backbone) e **imagens médicas** (TransUNet e MedViT).

10.11 Estudo de caso

Aplicação de transfer learning com o modelo `google/vit-base-patch16-224-in21k` sobre o dataset **Beans** do Hugging Face, para diagnóstico automatizado de doenças foliares. O dataset contém imagens reais de folhas de feijão capturadas em Uganda, classificadas em 3 categorias: **Angular Leaf Spot** (manchas angulares marrons causadas por fungo, comprometendo a fotossíntese), **Bean Rust** (pústulas alaranjadas na superfície foliar, reduzindo a produtividade) e **Healthy** (folhas saudáveis sem sinais de infecção, usadas como referência). São aproximadamente 1000 imagens de treino, 130 de validação e 130 de teste.

O modelo é pré-treinado no ImageNet-21k, com mais de 14 milhões de imagens, e depois passa por fine-tuning no Beans. A acurácia alcançada foi de aproximadamente **99,2%**. As aplicações apontadas são diagnóstico automático para agricultura de precisão, apoio ao pequeno produtor na detecção precoce e manejo, e um framework replicável extensível a outras culturas e datasets agrícolas.

11 Modelos Multimodais

Material de slides não disponível para esta aula.

O tópico consta da ementa oficial, mas não há slides correspondentes no acervo. O material disponível permite apenas registrar as conexões que o preparam. A aula de Vision Transformers estabelece a base arquitetural necessária: a mesma arquitetura Transformer, com self-attention, projeções Q/K/V, multi-head attention, conexões residuais e normalização de camada, opera indistintamente sobre tokens de texto e sobre patches de imagem. Essa unificação de representação — texto e imagem convertidos em sequências de embeddings processadas pelo mesmo mecanismo — é precisamente a condição técnica que torna modelos multimodais viáveis. Os slides também mencionam a **cross-attention** entre encoder e decoder, mecanismo em que Query vem de uma fonte e Keys e Values de outra, que é o padrão usado para relacionar modalidades distintas.

12 Detecção e Segmentação de Objetos

12.1 Fundamentos: tarefas em visão computacional

Os slides organizam as tarefas em ordem crescente de complexidade:

- **Classificação**: atribui uma classe à imagem inteira. Exemplo: "essa imagem contém um gato". Mapeamento de imagem para rótulo.
- **Detecção de objetos**: localiza e classifica múltiplos objetos com caixas delimitadoras. Exemplo: "gato em (x1, y1, x2, y2), cão em (x3, y3, x4, y4)". Mapeamento de imagem para bounding boxes mais classes.
- **Segmentação**: atribui uma classe a cada pixel da imagem, produzindo máscara pixel a pixel por objeto.

A progressão completa de complexidade é: classificação, localização, detecção, segmentação e estimativa de pose.

12.2 Bounding box e IoU

Uma **bounding box** admite seis tipos de representação de coordenadas, das quais três formatos principais são apresentados:

- `bbox = (x_min, y_min, x_max, y_max)`, equivalente a (left, top, right, bottom);
- `bbox = (x_center, y_center, width, height)`;
- `bbox = (x_min, y_min, width, height)`, equivalente a (left, top, width, height).

Os valores podem ser normalizados pela largura e altura da imagem, ficando no intervalo de 0 a 1, ou absolutos.

A métrica **Intersection over Union (IoU)** mede a sobreposição entre a bbox prevista e a real. Uma detecção é considerada correta se o IoU for maior ou igual a um limite, tipicamente 0,5.

12.3 Métricas de desempenho

Cada detecção é classificada como:

- **TP** (verdadeiro positivo): IoU acima do limite e classe correta;
- **FP** (falso positivo): bbox errada, classe errada ou duplicata;
- **FN** (falso negativo): objeto real não detectado.

A partir daí definem-se:

- **Precisão**: dos que o modelo disse existir, quantos existem de fato. `P = TP / (TP + FP)`.
- **Recall**: dos que realmente existem, quantos o modelo encontrou. `R = TP / (TP + FN)`.
- **Average Precision (AP)**: área sob a curva precisão-recall de uma única classe, resumindo o desempenho em todos os níveis de confiança.
- **mean AP (mAP)**: média dos APs de cada classe. No COCO, com 80 classes, é a média de 80 APs.

12.4 Non-Maximum Suppression

Detectores frequentemente geram múltiplas caixas para o mesmo objeto. O algoritmo **Non-Maximum Suppression (NMS)** seleciona a melhor em quatro passos: (1) ordena as caixas por score de confiança; (2) mantém a caixa com maior confiança; (3) remove as caixas que se sobrepõem a ela com IoU acima do limite; (4) repete o processo até não haver mais sobreposição significativa.

12.5 Tipos de segmentação

- **Segmentação semântica**: classifica cada pixel com uma categoria, mas não distingue instâncias individuais. Os pixels são coloridos por classe (carro = azul, pessoa = vermelho). Exemplos: FCN, U-Net, DeepLab.
- **Segmentação de instância**: distingue cada instância individualmente, combinando detecção e segmentação. Cada objeto recebe cor única (pessoa1 = azul, pessoa2 = verde). Exemplos: Mask R-CNN, YOLO.
- **Segmentação panóptica**: unifica semântica e instância — objetos contáveis recebem segmentação de instância, e a cena de fundo recebe segmentação semântica. Exemplos: Panoptic FPN, Mask2Former.

12.6 Evolução dos detectores

Abordagens pré-deep learning.

- **Sliding Window**: desliza janelas de múltiplos tamanhos e classifica cada sub-imagem. O pipeline clássico combina **HOG** (Histogram of Oriented Gradients) para extração de features com **SVM** como classificador (Dalal e Triggs, 2005). O problema é ser computacionalmente inviável para grandes imagens.

- **Viola-Jones (2001)**: detecção de faces em tempo real, combinando features Haar (contraste), AdaBoost (que seleciona as melhores features no treinamento) e Cascade (que descarta a janela que falha no teste inicial). Foi muito usado em câmeras digitais.
- **Selective Search (2013)**: propõe cerca de 2000 regiões candidatas por imagem, com base em cor, textura, tamanho e forma. É muito mais eficiente que sliding window e serviu de base para as primeiras R-CNNs. O princípio é que, em vez de classificar todas as janelas possíveis, propor apenas regiões de interesse é muito mais eficiente.

Single Shot MultiBox Detector (SSD). Proposto por Liu et al. (ECCV 2016), usa múltiplos feature maps de diferentes escalas: feature maps menores para objetos grandes e feature maps maiores para objetos pequenos. O legado do SSD é que a ideia de detecção multi-escala em diferentes feature maps foi fundamental para o YOLOv3 e para praticamente todos os detectores modernos.

Os slides organizam ainda a distinção entre detectores de **2 estágios** (proposta de regiões seguida de classificação, como a família R-CNN) e de **1 estágio** (predição direta, como SSD e YOLO).

12.7 YOLO

You Only Look Once aborda a detecção como um **problema de regressão**: uma única rede prevê ao mesmo tempo bounding boxes e probabilidades de classe em uma única passagem pela imagem. A série se tornou a família de detectores mais usada no mundo, presente de drones a smartphones, de câmeras de segurança a carros autônomos.

O contexto de 2015 explica a motivação: a R-CNN levava cerca de 47 segundos por imagem, a Fast R-CNN operava a 0,5 FPS, e os pipelines eram complexos e difíceis de otimizar — inviáveis para tempo real.

YOLO v1. Divide a imagem em um grid S por S (por exemplo 7x7). Cada célula prediz B bounding boxes (2), cada uma com `(x, y, w, h, confiança)`, e C probabilidades de classe (20). A saída final tem dimensão `S x S x (B x 5 + C)`, resultando em 7x7x30, ou seja, 1470 predições. A arquitetura usa backbone **Darknet**, inspirado no GoogLeNet, com 24 camadas convolucionais e 2 camadas fully connected, entrada de 448x448x3 e saída 7x7x30. O backbone foi pré-treinado no ImageNet (acurácia top-5 de 88%) e depois passou por fine-tuning para detecção. Há também o **Fast YOLO**, com 9 camadas convolucionais.

YOLO v2 (2016). Seis melhorias principais:

1. **Batch Normalization** após cada camada convolucional, estabilizando o treinamento e permitindo remover o Dropout. Ganho de 2% de mAP.
2. **High Resolution Classifier**: fine-tune do backbone em 448x448 em vez de 224x224. Ganho de 4% de mAP.
3. **Anchor Boxes** (como na Faster R-CNN): uma anchor box é uma caixa de referência pré-definida, e a rede prediz offsets em relação a ela. Usa K-means clustering no dataset para encontrar as melhores formas de anchors, resultando em 5 anchors.
4. **Multi-Scale Training**: muda a resolução de entrada a cada 10 batches, de 320 a 608, tornando o modelo mais robusto a diferentes tamanhos.
5. **Darknet-19**: backbone melhorado com 19 camadas convolucionais e 5 max pool, sem camadas fully connected, o que reduz parâmetros.
6. **YOLO9000**: treina simultaneamente em COCO (detecção) e ImageNet (classificação), predizendo mais de 9000 classes em tempo real.

YOLO v3 (2017). Predições em 3 escalas diferentes para objetos pequenos e grandes, backbone **Darknet-53** com 53 camadas, **Feature Pyramid** fundindo features de diferentes níveis, e melhor recall, detectando mais objetos, especialmente pequenos.

YOLO v4 (2020). Backbone **CSPDarknet53**, no qual as Cross Stage Partial Networks dividem o feature map em duas partes — uma passa pela rede e outra vai direta ao final — reduzindo computação e melhorando gradientes, com ativação **Mish** no lugar de Leaky ReLU. O neck combina **SPP** (Spatial Pyramid Pooling, com múltiplos tamanhos de pooling em paralelo, aumentando o campo receptivo) e **PANet** (que acrescenta um caminho bottom-up à FPN, melhorando a propagação de informações de localização). O **Bag of Freebies** reúne técnicas que melhoram o mAP sem custo de inferência: CutMix e **Mosaic** data augmentation (que combina 4 imagens em uma única, redimensionada e misturada aleatoriamente, ensinando o modelo a detectar objetos fora do contexto habitual), DropBlock, Label Smoothing, **CIoU loss** (Complete IoU) e cosine annealing da taxa de aprendizado. Resultado: mAP de 43,5% no COCO a 65 FPS em uma V100, a melhor relação velocidade/acurácia da época.

YOLO v5, v6 e v7. A **v5 (2020)** é a implementação open source da Ultralytics, com família escalável (S, M, L, X) oferecendo trade-off entre velocidade e precisão, implementação em PyTorch mais acessível que as versões anteriores, AutoAnchor para otimização automática de anchors do dataset, e maturidade de produção com deploy fácil e boa documentação. A **v6 (2022)** foca em edge e aplicações industriais, com backbone RepVGG (reparametrização com treino multi-branch e inferência single-branch), detector **anchor-free**, TAL (Task Alignment Learning) para matching, e resultados competitivos em baixa latência. A **v7 (2022)** introduz o **E-ELAN** (Extended Efficient Layer Aggregation Network), escalamento de modelo e reparametrização, e cabeça auxiliar coarse-to-fine, atingindo mAP de 51,4% a 161 FPS em V100. A tendência emergente nas versões 6 e 7 é a consolidação de detectores anchor-free, que eliminam a necessidade de definir anchors manualmente e simplificam o pós-processamento.

YOLO v8, v9 e v10. A **v8 (2023)** adota **head desacoplada**, com um branch de classificação (conv mais sigmoid, gerando classes) e outro de regressão (conv mais bbox, gerando coordenadas), de modo que cada tarefa tem seu próprio head, favorecendo a convergência. A partir da v8 os modelos são **anchor-free**, predizendo diretamente as distâncias do ponto central às 4 bordas (top, bottom, left, right), o que elimina a heurística de anchors e simplifica o pós-processamento. A **v9 (2024)** ataca a perda de informação e a degradação de gradiente em redes profundas, nas quais features iniciais "somem" durante a propagação, criando caminhos auxiliares para preservar informação. A **v10 (2024)** ataca o NMS como gargalo — por ser um passo de pós-processamento que não é end-to-end e adiciona latência, especialmente em hardware especializado — usando **Dual Label Assignment**: one-to-many durante o treino, com supervisão rica e múltiplas bboxes por objeto para melhor gradiente, e one-to-one durante a inferência, com apenas a melhor predição por objeto, eliminando a necessidade de NMS.

YOLO v11 e v12. A **v11 (2024)** traz arquitetura de backbone e neck aprimorada para extração de features mais precisa, pipelines de treinamento otimizados para maior velocidade de inferência, compatibilidade com edge devices, cloud e GPUs NVIDIA, e suporte a múltiplas tarefas: detecção, segmentação, pose estimation, classificação e OBB. A **v12 (2025)** introduz o módulo **A2 (Area Attention)**, mecanismo de atenção por regiões que reduz custo computacional limitando o campo de atenção a áreas locais; o **R-ELAN** (Residual-Efficient Layer Aggregation Networks), que substitui o ELAN tradicional por conexões residuais escalonadas para estabilizar o treinamento em modelos maiores; e **FlashAttention**, que reduz drasticamente o overhead de acesso à memória durante os cálculos de atenção.

YOLO26 (2026). End-to-end **NMS-free**, com inferência sem pós-processamento; otimização para CPU, com melhor performance em dispositivos edge; **ProgLoss** e **STAL**, que dão boost em objetos pequenos; e o otimizador **MuSGD**, para treinamento mais estável e eficiente.

12.8 Prática

A parte prática da aula foi conduzida em notebooks Colab com a biblioteca Ultralytics, cobrindo contagem de objetos, treinamento sobre um dataset de detecção de tumores cerebrais, treinamento sobre um dataset de segmentação de peças automotivas e um exemplo com YOLO26. Um esqueleto ilustrativo do uso da API:

```
from ultralytics import YOLO

# Carrega um modelo pre-treinado
modelo = YOLO("yolo11n.pt")

# Fine-tuning em um dataset proprio
modelo.train(data="dataset.yaml", epochs=50, imgsz=640)

# Inferencia
resultados = modelo("imagem.jpg")
resultados[0].show()
```

13 Síntese da Disciplina

A disciplina constrói um argumento coerente do começo ao fim: **representações aprendidas superam representações desenhadas à mão, desde que se entenda o que se está representando e como o aprendizado ocorre**.

O primeiro movimento é estabelecer o mecanismo de aprendizado. O neurônio artificial é uma soma ponderada seguida de uma não linearidade; a MLP empilha esses neurônios em camadas e resolve, com o Backpropagation, o problema que Minsky e Papert identificaram no perceptron de uma camada. As aulas de otimização mostram que o algoritmo não basta: taxa de aprendizado, termo de momento, tamanho de batch, critérios de parada e regularização decidem se a rede converge, paralisa, fica presa em mínimo local ou sobreajusta. Essas noções — época, batch, loss, overfitting, viés-variância — reaparecem literalmente em todas as aulas seguintes.

O segundo movimento é entender o dado. Antes de qualquer arquitetura, a disciplina insiste que uma imagem é uma função `f(x, y)` discretizada, com profundidade de bits e resolução determinadas, representada em RGB, HSV ou CMYK conforme o objetivo. Histogramas, equalização global, CLAHE, histogram matching, filtragem por vizinhança e transformações pontuais não são curiosidades: são as ferramentas que decidem quanta informação discriminativa chega ao modelo. O tratamento de desbalanceamento, o paradoxo da acurácia e a análise exploratória com PCA, t-SNE e uMAP completam o repertório de preparação.

O terceiro movimento resolve o problema estrutural: achatar uma imagem destrói a relação espacial entre pixels. A convolução preserva essa relação examinando regiões locais, e o empilhamento de camadas produz uma hierarquia natural — bordas, texturas, partes, objetos — que a rede descobre sozinha a partir de exemplos rotulados. Padding, stride e pooling são os controles dessa operação;

data augmentation, dropout e ruído interno são os mecanismos de generalização. LeNet, AlexNet, VGG e ResNet marcam a trajetória de aprofundamento, e a ResNet fornece a peça conceitual mais reutilizada de toda a disciplina: a **conexão residual**, que reaparece intacta dentro de cada bloco Transformer.

O quarto movimento é econômico: transfer learning torna viável fazer visão computacional sem milhões de imagens e semanas de GPU. Congelar, descongelar parcialmente ou fazer fine-tuning total são decisões guiadas pelo tamanho do dataset e pela distância entre domínios. As leituras de ResNet, Inception e Lottery Ticket Hypothesis dão a esse movimento uma fundamentação sobre por que profundidade, esparsidade e inicialização importam.

O quinto movimento leva o modelo ao mundo. FastAPI com OpenAPI/Swagger transforma um arquivo `.keras` em serviço agnóstico a linguagem — um lembrete de que a fronteira entre um experimento e um produto é operacional, não algorítmica.

O sexto movimento generaliza o mecanismo. O Transformer substitui a vizinhança local da convolução pela atenção global entre todos os elementos. Query, Key e Value, multi-head attention, conexão residual, layer normalization e FFN formam um bloco que opera sobre qualquer sequência. O ViT explora essa neutralidade: patches de 16x16 são tokens, um token [CLS] carrega a decisão de classe, e embeddings posicionais restauram a noção espacial que a atenção não tem. O trade-off com CNNs é explicitado: viés indutivo forte e eficiência com poucos dados de um lado, viés fraco e melhor escalabilidade com muitos dados do outro.

O movimento final aplica tudo isso a tarefas mais difíceis que classificação. Detecção exige localizar e classificar simultaneamente, com bounding boxes, IoU, precisão, recall, AP, mAP e NMS. Segmentação exige decidir por pixel, em três regimes distintos (semântica, instância, panóptica). A trajetória da família YOLO, do grid 7x7 da v1 ao pipeline end-to-end NMS-free do YOLO26, resume em uma única linha evolutiva quase todos os conceitos do curso: batch normalization, anchors e seu abandono, treinamento multi-escala, pirâmides de features, conexões residuais, data augmentation estrutural, atenção e reparametrização.

O fio condutor, portanto, é que cada camada de sofisticação resolve uma limitação concreta da anterior — e que entender qual limitação está sendo resolvida é mais útil do que memorizar a arquitetura que a resolve.