Escola de Talento do Instituto Principia

Brenno Koiki Ishiyi
Luiz Cláudio Germano da Costa

Aprendizado profundo para a detecção de objetos em imagens: uma aplicação à detecção de zebrafishes

Orientador: Prof. Dr. João Batista Florindo

Julho de 2024

Agradecimentos

Ao Prof Dr João Batista Florindo, por se dispor a nos orientar e permitir que este trabalho ocorresse.

Ao diretor, Prof Dr Marcelo Guzzo, por oferecer e organizar esta oportunidade de iniciação acadêmica a tantos alunos, que é a Escola de Talentos.

Aos Prof Dr Ricardo Matheus, Leonardo Lima e Rickson Mesquita, por nos acompanhar durante o primeiro ano da Escola de Talentos.

E à Stark Bank, por patrocinar este projeto com o financiamento dos encontros e o oferecimento de bolsas aos estudantes.

Resumo

O presente estudo adentra no uso do Aprendizado Profundo para a detecção de objetos em imagens. Como ponto de partida, tem-se uma apresentação do funcionamento das redes neurais para posteriormente discutir-se o algoritmo YOLO e sua aplicação na detecção de *zebrafishes* (*Danio rerio*), um peixe que tem sido utilizado pela Ciência em diversas pesquisas, devido a sua semelhança genética com os seres humanos. Nesse sentido, a fim de detectar e rastreá-los, realizou-se um treinamento com o modelo YOLO os resultados foram analisados, atingindo bom nível de precisão nessa tarefa.

Sumário

1 Introdução

O termo *Machine Learning*, ou Aprendizado de Máquinas, foi criado em 1959 pelo engenheiro Arthur Samuel. Segundo ele, *Machine Learning* é "um campo de estudos que dá aos computadores a habilidade de aprender sem terem sido programado para tal". Um dos primeiros exemplos dessa tecnologia foi um programa desenvolvido por Arthur que jogava damas, em que, a partir da análise de diversos jogos, o computador identificava posições que estariam em vantagem ou desvantagem, e, com isso, o melhor lance era calculado. Desde então, essa área de estudo desenvolveu-se e pôde ser aplicada em diversas ferramentas atuais, desde algoritmos de recomendação de filmes até carros autônomos (por meio da detecção de objetos ao redor do veículo, por exemplo).

O *Machine Learning* tem como ponto fundamental o treinamento de um algoritmo que identifica padrões, seja em imagens, textos, áudios, entre outros. Esse processo se dá de duas formas diferentes, as quais constituem o *Supervised Learning* e o *Unsupervised Learning*.

No *Supervised Learning*, treina-se o computador com dados de entrada rotulados, ou seja, cujas saídas são pré-definidas. Já no *Unsupervised Learning*, o computador é treinado com dados não rotulados. Dessa forma, o computador não possui uma orientação da saída que ele deve produzir.

Neste trabalho, faz-se uma análise de como ocorre o treinamento no *Supervised Learning*, de forma a abordar as redes neurais, as convolucionais, e por último, o funcionamento do algoritmo YOLO na detecção de objetos. Mais especificamente, aplica-se o YOLO à detecção do peixe *zebrafish* (*Danio rerio*) em imagens de ambientes de controle. O *zebrafish* é um modelo animal amplamente utilizado em todo o mundo. Sua similaridade genética da ordem de 70% com o ser humano faz com que o mesmo seja muito usado, por exemplo, para o teste de novos fármacos ou na avaliação da toxicidade de determinados componentes químicos. Uma característica interessante desses animais é o seu padrão de movimentos como resposta a algum efeito no organismos deles causado por alguma dessas substâncias de teste. Poder identificar esse movimento de forma automática é um grande avanço nesse tipo de pesquisa e o algoritmo aqui estudado pode ser facilmente adaptado para essa tarefa.

2 O Básico do Treinamento de Computadores

Suponha uma situação em que seja necessário criar uma função linear $f(x)$ que melhor se ajuste a um conjunto de dados. Essa função é expressa por parâmetros w e b: $f(x) = wx + b$

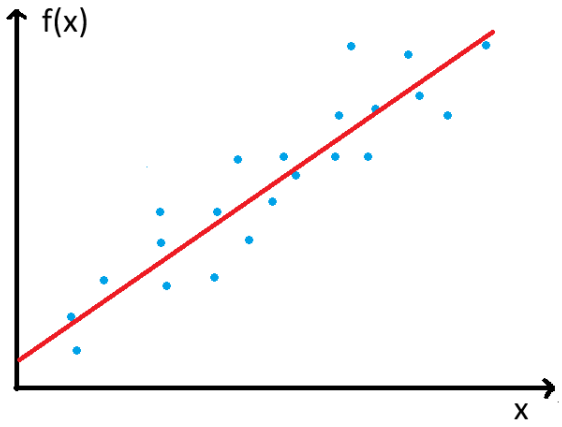
Na figura 1, x pode representar, por exemplo, o nível educacional e $f(x)$ o prestígio de uma pessoa. Treinar o computador nesse caso significa encontrar os parâmetros w e b da função que melhor se ajustam aos dados. O que está está sendo feito aqui é chamado de Regressão. Assim, dado um valor x de nível educacional, $f(x)$ é uma previsão do prestígio da pessoa.

2.1 A Função Custo

Ao analisar o gráfico da Figura 1, depreende-se que a reta representa os dados de maneira razoavelmente boa. Porém, para o computador, é preciso quantificar esse fator. Para isso, a função de custo associa um número a uma função, sendo ela zero se $f(x)$ adequar-se perfeitamente aos dados ou um número maior quanto menos adequado ela for. Existem vários tipos de função de custo, mas um dos mais simples é o seguinte:

$$J(w,b) = \frac{1}{2m} \sum_{i=0}^{m=1} (f(x^i) - y^i)^2 \tag{1}$$

Fig. 1: Exemplo de Regressão Linear



em que:

m = número de amostras

$f(x^i)$ = previsão para o dado i

J = custo de $f(x)$

y^i = valor que deve ser previsto pela função

Continuando o exemplo anterior, a expressão $(f(x^i) - y^i)^2$ representa a distância de f(x) a um dado. Com o somatório, calcula-se esse valor para todos os dados. Dessa forma, pode-se interpretar essa função de custo como a média das distâncias (ao quadrado) da reta aos pontos.

Com isso, o objetivo de um algoritmo de treinamento é achar a função com o menor custo possível. Isso faz sentido pois um valor baixo do custo indica uma boa adequação da função aos dados.

2.2 O Gradiente Descendente

Achado o custo da função atual, pode-se atualizar os parâmetros w e b com as seguintes expressões:

$$w = w - \alpha \frac{\partial J}{\partial w} \tag{2}$$

$$b = b - \alpha \frac{\partial J}{\partial b}, \tag{3}$$

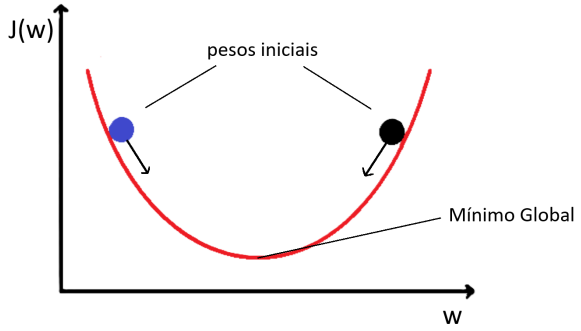
em que α é uma constante nomeada "taxa de aprendizagem". Quando maior seu valor, mais significativas são as atualizações.

Depois, calcula-se o custo com os novos parâmetros, o que possibilita um novo ciclo de atualizações. Tal processo se repete até que o custo seja o mínimo possível, algoritmo esse chamado de Gradiente Descendente.

2.3 Por Que O Gradiente Descendente Funciona?

Primeiramente, é mais fácil estudar o que ocorre com um parâmetro, apenas, durante as atualizações. A curva em vermelho da Figura 2 mostra como varia a função de custo de acordo com o valor do parâmetro w:

Fig. 2: Custo em função de w



A expressão (2) calcula a derivada parcial de J em relação a w e a subtrai do valor inicial de w. Nesse sentido, analisando o gráfico acima, quando a derivada é positiva (caso da bolinha da direita), o parâmetro w diminui, o que diminui também o custo J. Já quando a derivada é negativa (bolinha da esquerda), o parâmetro w aumenta, o que também diminui J.

Esse processo ocorre para todos os parâmetros, não apenas para um. Por isso, conforme w e b são atualizados, a função custo tende a diminuir até chegar no local mínimo. Ou seja, o próprio computador encontra, por meio de um algoritmo, os melhores parâmetros da função para um grupo de dados.

3 As Redes Neurais

No cérebro humano, os neurônios são interligados um com outro por meio de regiões denominadas sinapses. Por meio dessas conexões, uma célula nervosa consegue comunicar-se com a outra e transmitir mensagens. Essa é a inspiração da chamada rede neural, modelo de *Machine Learning* que revolucionou a área da tecnologia, sendo ela aplicada na Medicina, Engenharia, entre muitas outras áreas. Analogamente aos neurônios do cérebro, a unidade básica das redes neurais é chamada de "nó" e, em vez de serem conectadas por sinapses, essas unidades são interligadas por números (pesos).

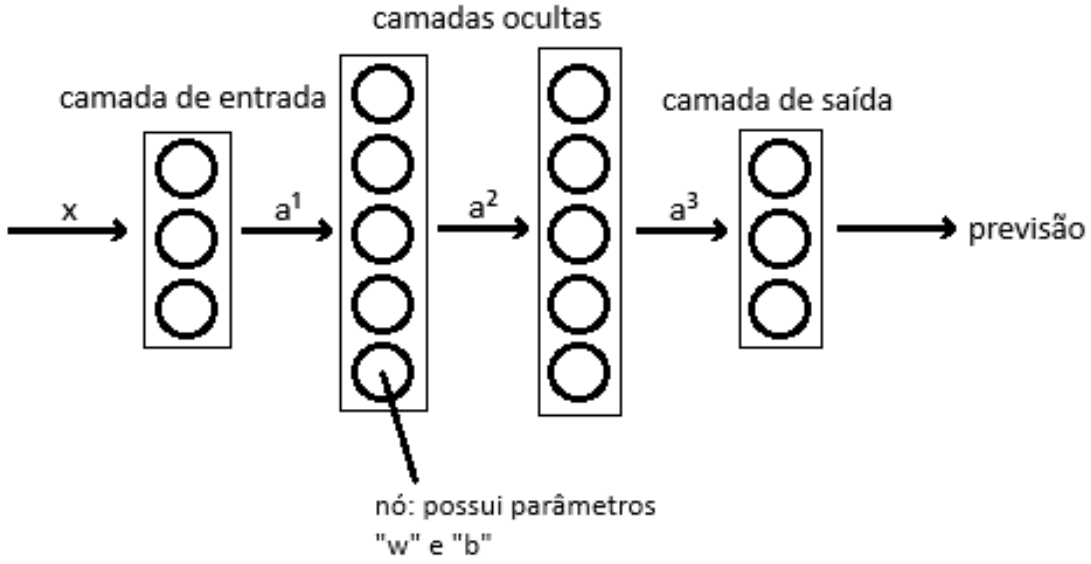
3.1 Estrutura Geral

Como mostrado na Figura 3, existem três tipos de camadas nas redes neurais: a camada de entrada, as camadas ocultas e a camada de saída. O primeiro tipo é responsável por receber os dados; o segundo é composto por várias camadas internas, as quais processam esses dados. Por último, a camada de saída realiza a previsão correspondente ao dado de entrada.

Cada camada é composta por vários "nós", que recebem os dados da camada anterior e os aplicam em uma função linear com parâmetros w (pesos) e b (*bias*), sendo depois usados como entrada em uma função de ativação. Depois, as saídas de todos os nós de uma camada são combinadas e repassadas para a camada seguinte, até a camada final, onde se realiza a previsão. Essa etapa descrita é o chamado *Forward Propagation*.

Após isso, tem-se a fase do treinamento (atualização) dos parâmetros, quando as derivadas do custo em relação a cada parâmetro são calculadas (para serem utilizadas no Gradiente Descendente). Essa fase é conhecida como *Backward Propagation*. Percebe-se que uma rede neural possui um funcionamento bastante semelhante ao descrito no Capítulo 2. A principal diferença

Fig. 3: Estrutura de uma rede neural



está que, em vez de treinarem-se os parâmetros de apenas uma função, treinam-se os de várias, pois cada nó representa uma.

3.2 Forward Propagation

Primeiramente, é necessário compreender o funcionamento de um nó. Cada nó possui os seguintes parâmetros:

$$\vec{w} = \begin{bmatrix} w_1 & w_2 & w_3 & \dots & w_n \end{bmatrix}$$
$$b$$

em que $\vec{w}$ é um vetor contendo os pesos w e b é chamado de "*bias*". Esses são os parâmetros que são treinados. Já os dados são representados por um outro vetor:

$$\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{bmatrix}$$

Quando os dados chegam no nó, calcula-se o seguinte:

$$z = \vec{w} \cdot \vec{x} + b \tag{4}$$

Em seguida, utiliza-se a chamada função de ativação e calcula-se a ativação:

$$a = g(z) \tag{5}$$

A função de ativação serve para dar à rede neural uma não-linearidade e, portanto, uma complexidade maior. Caso ela não fosse utilizada, a rede neural seria um conjunto de funções lineares – Equação 9 - ou seja, não haveria uma complexidade maior do que aquela discutida

na Seção 2. Existem várias funções de ativação, entre elas a ReLU (Equação 6), a Sigmoide (Equação 7) e a Linear (Equação 8).

$$g(z) = max(0, z) \tag{6}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{7}$$

$$g(z) = z \tag{8}$$

Assim, cada nó de uma camada tem como saída a ativação a, que por sua vez é a entrada dos nós da próxima camada. Na rede neural, como uma camada possui diversos nós, então a saída da camada é um vetor de ativação $\vec{a}$:

$$\vec{a} = \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_m \end{bmatrix}$$

sendo m o número de nós na camada. Esse vetor é utilizado como entrada para a próxima camada, de forma que se repete o processo em todas as camadas, até chegar na última, quando é feita a previsão.

3.3 Softmax

Nesta seção, será abordado um tipo de previsão: a Classificação Multi-Classes. Ao contrário da classificação binária, cujo objetivo é classificar um dado como "1" ou "0" (existem somente duas classes), na classificação multi-classes, existem mais que duas classes. Um exemplo desse tipo de previsão é quando, por meio de uma imagem de um animal, o computador descobre se é um cachorro, gato, cavalo, entre outros (existem várias classes). Para que isso seja possível, aplica-se a chamada ativação *Softmax* na última camada da rede neural.

A camada *Softmax*, como todas as outras camadas, possui diversos neurônios com parâmetros w e b a serem treinados. Cada neurônio representa uma classe e, portanto, sua quantidade nessa camada representa o número de classes. A seguir, está uma representação da *Softmax*:

$$\begin{bmatrix} \vec{w_1}^{\,k}, b_1^k \\ \vec{w_2}^{\,k}, b_2^k \\ \vec{w_3}^{\,k}, b_3^k \\ \dots \\ \vec{w_m}^{\,k}, b_m^k \end{bmatrix},$$

em que a linha i representa os parâmetros do i-ésimo neurônio da camada e o k significa que essa é a k-ésima camada.

Como dito anteriormente, cada camada recebe como entrada a saída da anterior. Considerando que existem k camadas, no caso da *Softmax*, ela recebe um vetor ativação $\vec{a}^{k-1}$ da penúltima. Com isso, cada neurônio calcula o seguinte:

$$z_1^k = \vec{w_1}^{\,k} \cdot \vec{a}^k + b_1^k$$

$$z_2^k = \vec{w_2}^{\,k} \cdot \vec{a}^k + b_2^k$$

$$z_3^k = \vec{w_3}^{\,k} \cdot \vec{a}^k + b_3^k$$

$$\dots$$

$$z_m^k = \vec{w_m}^{\,k} \cdot \vec{a}^k + b_m^k$$

A seguir, utiliza-se esta expressão para cada um dos neurônios:

$$a_i^k = \frac{e^{z_i^k}}{e^{z_1^k} + e^{z_2^k} + \ldots + e^{z_m^k}} \tag{9}$$

Ou seja, como cada neurônio tem como saída um valor a^k, a saída da camada inteira é o conjunto desses valores, sendo esse um vetor:

$$\vec{a}^k = \begin{bmatrix} a_1^k & a_2^k & a_3^k & \ldots & a_m^k \end{bmatrix}$$

Pela própria Equação (9), os valores de a_i^k estão compreendidos entre 0 e 1, ou seja, a saída da *Softmax* é:

$$\vec{a}^k = \begin{bmatrix} 0,2 & 0,05 & 0,7 & 0,1 & \ldots \end{bmatrix}$$

Nesse caso, o maior número é o neurônio 3, e, então, a previsão do computador para esse dado seria que ele pertence à classe 3. Ou seja, a previsão seria:

$$\hat{y} = 3$$

Por fim, para calcularem-se os gradientes, a fim de atualizar os parâmetros, usa-se a função de custo:

$$J(a_1^k, a_2^k, \ldots, a_m^k, y) = \begin{cases} -\log a_1^k & \text{se } y = 1 \\ -\log a_2^k & \text{se } y = 2 \\ \ldots & \\ -\log a_m^k & \text{se } y = m \end{cases}, \tag{10}$$

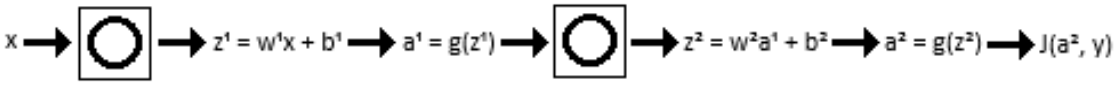
em que y é a previsão esperada que o computador faça.

3.4 Back Propagation

Agora, feita a etapa do *Forward Propagation*, inicia-´se a atualização dos parâmetros. É necessário, portanto, calcular as derivadas do erro em relação a cada parâmetro. O *Back Propagation* entra aqui como uma forma de, sem um custo computacional muito grande, realizar esses cálculos, os quais iniciam-se na última camada e terminam na primeira. Por isso o nome *Back Propagation*: ao contrário do *Forward Propagation*, as operações começam no fim e terminam no início da rede neural.

Na Figura 4, está uma rede neural mais simples, em que cada uma das duas camadas possuem apenas um nó.

Fig. 4: Rede Neural Simples



O dado x entra na primeira camada e na segunda (última) calcula-se o erro associado a ele. Imediatamente depois, calcula-se a derivada de J em relação a a^2:

$$\frac{\partial J}{\partial a^2}$$

Com isso, pode-se achar a derivada de J em relação a z^2:

$$\frac{\partial J}{\partial z^2} = \frac{\partial J}{\partial a^2}\frac{\partial a^2}{\partial z^2}$$

Veja que $\frac{\partial J}{\partial a^2}$ já foi calculado anteriormente, por isso, basta substituir seu valor na expressão acima.

É possível, agora, calcular as derivadas de J em relação aos parâmetros da segunda camada pois todas as derivadas que se precisava já foram achadas:

$$\frac{\partial J}{\partial w^2} = \frac{\partial J}{\partial a^2}\frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial w^2}$$

$$\frac{\partial J}{\partial b^2} = \frac{\partial J}{\partial a^2}\frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial b^2} = \frac{\partial J}{\partial a^2}\frac{\partial a^2}{\partial z^2}$$

Na segunda expressão, como $\frac{\partial z^2}{\partial b^2} = 1$, foi possível simplificar um pouco.

Agora, o mesmo processo ocorre para a primeira camada:

$$\frac{\partial J}{\partial a^1} = \frac{\partial J}{\partial z^2}\frac{\partial z^2}{\partial a^1}$$

Mais uma vez, o valor de $\frac{\partial J}{\partial z^2}$ já é conhecido, então basta substituir seu valor acima.

Para a derivada em relação a z^1:

$$\frac{\partial J}{\partial z^1} = \frac{\partial J}{\partial a^1}\frac{\partial a^1}{\partial z^1}$$

Finalmente, calculam-se as derivadas em relação aos parâmetros da primeira camada:

$$\frac{\partial J}{\partial w^1} = \frac{\partial J}{\partial z^1}\frac{\partial z^1}{\partial w^1}$$

$$\frac{\partial J}{\partial b^1} = \frac{\partial J}{\partial z^1}\frac{\partial z^1}{\partial b^1} = \frac{\partial J}{\partial z^1}$$

Portanto, nota-se que o *Back Propagation* baseia-se na reutilização de valores já computados para calcular novos valores. Dessa forma, ele é um rápido e eficiente algoritmo para achar os gradientes.

É necessário salientar, porém, que a forma como foi apresentado é uma versão simplificada, referente a um caso em que somente existe um nó por camada. Com mais nós, é preciso trabalhar não apenas com vetores (como foi apresentado ao longo deste capítulo), mas com matrizes, pois os dados ocupam mais uma dimensão.

4 Redes Neurais Convolucionais

Em 1980, O cientista da Computação Kunihiko Fukushima publicou um artigo que anos depois seria a base para a arquitetura das Redes Neurais Convolucionais (CNN). Nesse Contexto, as CNN são um subconjunto do aprendizado de máquina utilizadas com mais frequência para tarefas de classificação e visão computacional. As redes neurais convolucionais oferecem uma abordagem mais dimensionável para tarefas de classificação de imagens e reconhecimento de objetos.

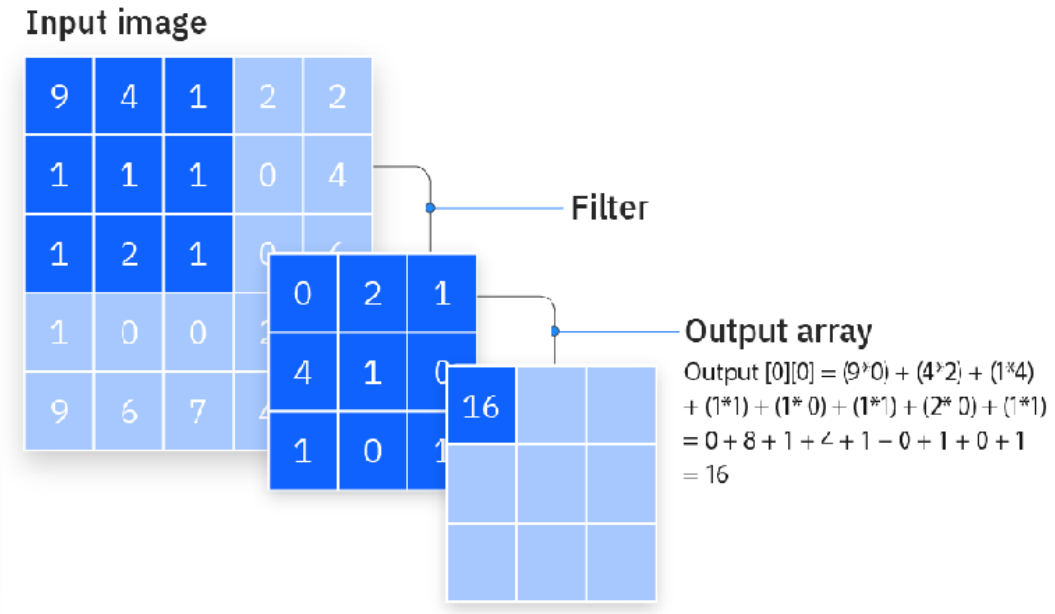
4.1 Funcionamento das Redes Neurais Convolucionais

As CNN são, essencialmente, um tipo de rede neural que utiliza a operação de convolução em vez da multiplicação por matrizes em ao menos uma de suas camadas. Essa operação pode ser descrita pela seguinte equação:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{11}$$

em que:
I, K = Matrizes
i, j = coordenadas da matriz resultante
m, n = coeficientes do somatório

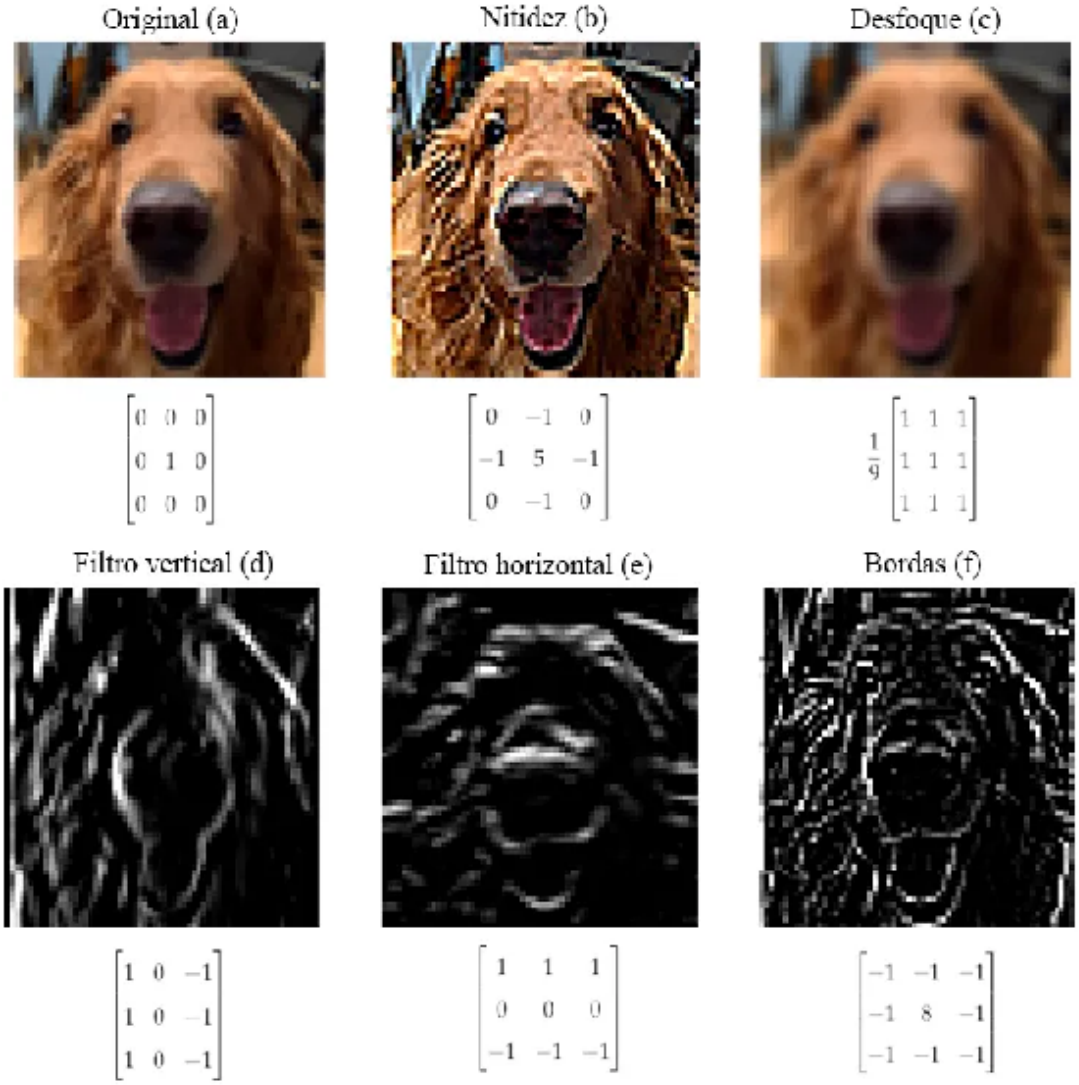
Fig. 5: Convolução



4.2 Convolução em imagens

No contexto de CNN, a matriz I é a imagem de entrada e a matriz K é o filtro aplicado à essa imagem. A depender do filtro utilizado, é possível manipular a nitidez e o desfoque, estilizar a imagem, ou detectar as arestas dos objetos presentes. A Figura 6 exemplifica a aplicação de alguns diferentes tipos de filtro em uma imagem base.

Fig. 6: Aplicação dos Filtros à imagem



5 You Only Look Once - YOLO

A detecção de objetos é a capacidade que um algoritmo possui em classificar um objeto presente em uma imagem e fornecer sua localização por meio de caixas delimitadoras (Figura 7). O

algoritmo *YOLO (You Only Look Once)*, desenvolvido em 2015, revolucionou a maneira de se fazer isso, sendo atualmente o principal modelo para essa tarefa.

Na época em que o YOLO foi desenvolvido, um dos maiores problemas da detecção de objetos era o seu alto custo computacional. Por exemplo, a abordagem do modelo *R-CNN (Region Based Convolutional Neural Networks)* consistia em primeiramente achar regiões que potencialmente possuíam objetos, para depois utilizar um algoritmo de classificação nessas regiões, a fim de identificar os objetos presentes. Esse processo, pelo fato de exigir várias redes neurais e uma inspeção das diversas regiões da imagem, era lento.

Entretanto, no YOLO, a imagem é repassada para uma única rede neural, e apenas uma vez. Por isso seu nome: *You Only Look Once* (Você Só Olha Uma Vez). Isso o torna muito rápido, de forma que permite até mesmo utilizá-lo na detecção em tempo real em vídeos, coisa que não era feita com boa precisão por outros algoritmos.

Outra vantagem desse algoritmo é o fato dele analisar toda a imagem de uma vez (e não apenas regiões dela, como era no R-CNN). Isso possibilita que o computador identifique também o cenário do objeto, e dessa forma, erre menos na detecção do objeto, quando ele encontra-se em ambientes diferentes.

Por outro lado, o YOLO tem certa dificuldade em detectar objetos pequenos que estejam em grupos, ou então, objetos com cores ou em posições não vistas durante o treinamento. Porém, em suas novas versões (atualmente já existe o YOLO v10), os quais mudaram certos aspectos da rede convolucional e adicionaram algumas técnicas que refinaram a detecção [4], não só esse problema, mas também a precisão do algoritmo tem melhorado.

6 Experimento Com Zebrafishes

Fig. 7: Detecção de *zebrafish*

Uma rede neural do tamanho usado no YOLO, com milhões de parâmetros, exige muitas imagens para o treinamento, que pode durar dias. Porém, pode-se realizar o chamado *Transfer Learning*, o qual consiste em reaproveitar os parâmetros pré-treinados de uma rede para realizar uma tarefa parecida.

Fig. 8: Detecção de *zebrafishes*

No caso, a rede do YOLO possui parâmetros já treinados para detectar certos objetos, como computadores, pessoas e carros. Assim, para detectar *zebrafishes*, não é necessário um treinamento do zero. Algumas poucas imagens do peixe já bastam para alcançar uma boa precisão.

Nesse experimento, usou-se a arquitetura do YOLOv8 Nano e 100 imagens foram utilizadas. Mas criaram-se outras novas por meio de algumas modificações, como a rotação e o embaçamento delas. Com esse último processo, denominado *Augmentation*, ao todo ficaram 300 imagens para o treinamento. Essa parte foi feita com a ajuda do *Roboflow*, uma plataforma que permite rotular as imagens e deixá-las prontas para o treinamento.

Percebe-se na Figura 7 que apenas um *zebrafish* não foi detectado. Isso pois a posição que ele se encontra foge do normal, e por isso, o algoritmo não o viu, ou viu poucas vezes algo parecido durante o treinamento.

Agora, na Figura 8, verifica-se uma dificuldade que provavelmente advém do fundo, que possui uma cor parecida com a dos *zebrafishes*. E também muitos deles aparecem borrados, o que dificulta a detecção.

Apesar disso, o algoritmo conseguiu detectar de forma boa aqueles peixes que estavam em uma posição lateral (como a maioria se encontra na Figura 7).

6.1 Funcionamento do YOLO

Agora, nesta Seção, será explicado como é possível que o algoritmo consiga detectar os objetos com base na primeira versão do YOLO [3]. Primeiramente, as imagens são divididas por uma grade $S \times S$. Cada célula (pedaço) da grade é responsável por detectar apenas um tipo de objeto, não podendo detectar duas ou mais classes diferentes.

A detecção se dá por meio das caixas delimitadoras, que possuem 5 valores associados a cada um deles: x, y, w, h e uma confiança c. x e y representam as coordenadas do centro da caixa, enquanto w e h sua largura e altura, respectivamente (Figura 9).

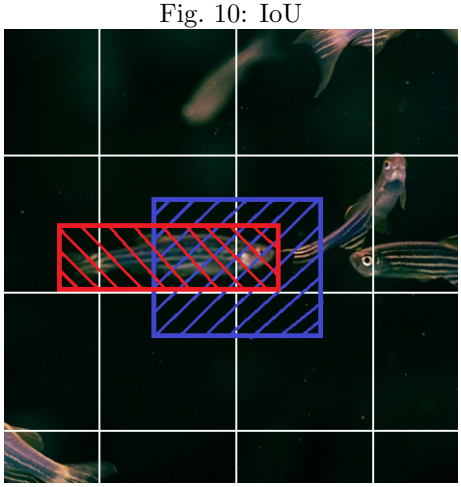
Já a confiança refere-se ao quão certo o algoritmo está em relação à presença do objeto dentro da caixa. Seu valor, entre 0 e 1, é a multiplicação entre a probabilidade de existir um objeto na célula e o *IoU (Intersection Over Union)* entre a caixa prevista e a caixa real (Figura 10).

O IoU quantifica a sobreposição entre duas caixas. Na Figura 10, em vermelho está a caixa real que delimita o peixe, e em azul, uma possível previsão. O cálculo do IoU é a razão entre a intersecção e a união entre as duas áreas.

Com isso, tem-se um valor mais próximo de 1 quanto mais as áreas se sobrepuserem e um valor próximo de 0, caso contrário.

Fig. 9: Valores x, y, w, h das Caixas Delimitadoras

Fig. 10: IoU



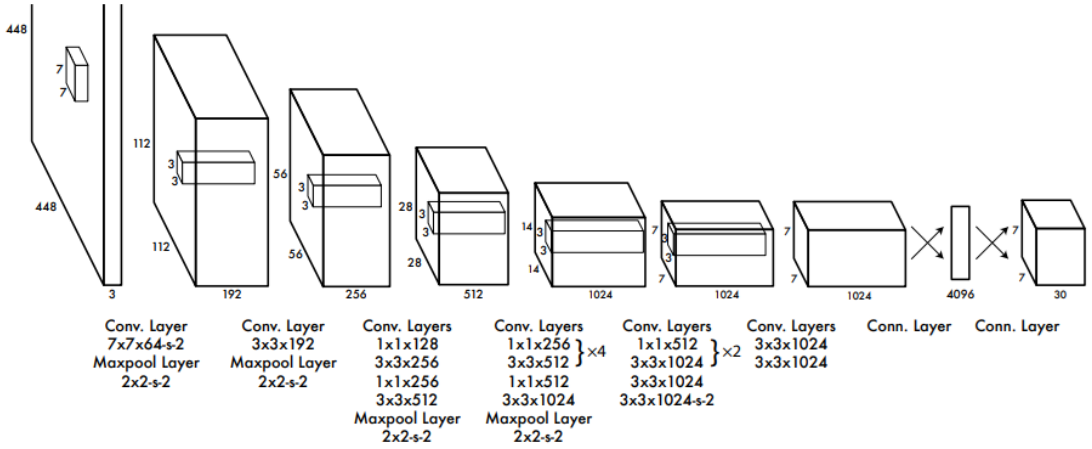
Continuando, cada pedaço da grade também prevê a probabilidade de encontrar dentro dela o centro de cada uma das classes. Logo, todos os dados possuem nas células, além dos valores x, y, w, h (referentes às caixas), um valor 0 ou 1 para cada classe, que indica a ausência ou presença do objeto.

No artigo original do YOLO [3], $S = 7$, o número de caixas previsto por cada pedaço da grade era 2 e existiam 20 classes diferentes. Isso totaliza $20 + 5 * 2 = 30$ valores por célula. Então, a previsão final de uma imagem, nesse caso, é um tensor $7 \times 7 \times 30$, que contém valores referentes às caixas delimitadoras e às classes presentes ou não.

Para transformar a imagem em um tensor com esse tamanho, ela é processada por uma rede convolucional mostrada na Figura 11. O objetivo do modelo, durante o treinamento, é prever um tensor de uma imagem, de forma que seus valores sejam os mais próximos possíveis dos dados. Esse fator é quantificado por uma função de custo, que permite, depois, atualizar os parâmetros da rede neural, para que a previsão se torne cada vez melhor.

Feito o treinamento, pode-se utilizar o algoritmo para detectar objetos em novas imagens. Nessa etapa, cada caixa delimitadora recebe um valor p_c de 0 a 1 que mostra o quão certo

Fig. 11: Rede Convolucional do YOLO

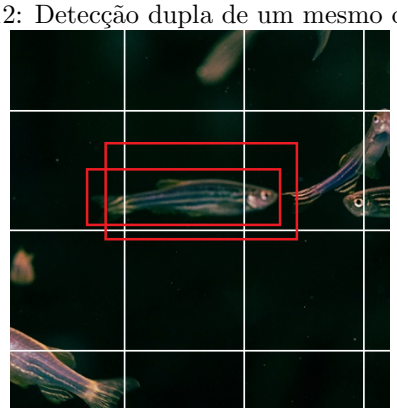


Fonte: [3]

o computador está em sua previsão. Ele é expresso pela multiplicação entre a confiança e as probabilidades das classes. Mas é possível que um único objeto possua várias caixas previstas para ele (Figura 12), e por isso, utiliza-se uma técnica denominada *Non Max Suppression (NMS)*.

Primeiro, eliminam-se todas as caixas que possuem um p_c menor que um valor determinado. Depois, escolhe-se aquela com o maior p_c e calcula-se o IoU entre ela e as caixas restantes. Por fim, todas as caixas cujo IoU deu acima de um valor são eliminadas. Com isso, em geral, sobra apenas uma caixa por objeto.

Fig. 12: Detecção dupla de um mesmo objeto



7 Conclusão

Portanto, com o algoritmo YOLO, é possível realizar uma eficiente detecção de objetos em imagens. Por ser relativamente acessível aos usuários interessados e poder ser treinado sem um banco de dados muito extenso, estudos envolvendo detecção de objetos se beneficiam imensamente com ele, o que pode ser bastante proveitoso à sociedade. No caso da detecção de *zebrafishes*, a automatização desse processo permite a ocorrência de estudos envolvendo a análise do comportamento

desses peixes. Isso é útil pois eles possuem uma grande similaridade genética com os seres humanos, tendo respostas imunológicas parecidas, e por isso, a análise comportamental pode servir como um parâmetro da eficácia ou não de fármacos bem como da reação a substâncias em geral. Sendo assim, o *Machine Learning* é uma área em constante desenvolvimento que tem aberto portas para novos tipos de pesquisas e este projeto permitiu um breve contato inicial com esta área da ciência.

Referências

[1] Andrew Ng, Kian Katanforoosh, and Younes Bensouda Mourri. Deep learning specialization, 2017. Coursera.

[2] Andrew Ng, Eddy Shyu, Geoff Ladwig, and Aarti Bagul. Machine learning specialization, 2012. Coursera.

[3] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[4] Abirami Vina. From YOLO to YOLOv8: Tracing the evolution of object detection algorithms, 2023. Disponível em: https://medium.com/nerd-for-tech/from-yolo-to-yolov8-tracing-the-evolution-of-object-detection-algorithms-eaed9a982ebd. (Acessado: 08 de Julho de 2024).