UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ROGERIO BARBOSA CHAVES

PIPELINE DE MONITORAMENTO DE VEÍCULOS PARA A UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA

VITÓRIA DA CONQUISTA - BA
2025

ROGERIO BARBOSA CHAVES

PIPELINE DE MONITORAMENTO DE VEÍCULOS PARA A UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA

Trabalho de conclusão de Curso apresentado ao Curso de Ciência de Computação da Universidade Estadual do Sudoeste da Bahia, como requisito parcial para o grau de Bacharel em Ciência da Computação.

Orientadora: Profa. Drª Alexsandra Oliveira Andrade

Vitória da Conquista - BA

2025

RESUMO

Este trabalho tem como objetivo propor uma pipeline de reconhecimento de veículos por meio da identificação de suas placas. Para isso, a pipeline deve atender a dois requisitos principais: (1) ser capaz de localizar a placa do veículo em uma imagem capturada e (2) identificar corretamente os caracteres da placa. Para atender a esses requisitos, utilizou-se o conjunto de dados ALPR para o treinamento de dois modelos. O primeiro é o modelo YOLO 11, treinado para detectar e localizar placas de veículos em imagens. O segundo é o modelo TPS-ResNet-BiLSTM-Attn, responsável pela identificação dos caracteres presentes nas placas. A metodologia adotada envolveu o tratamento do conjunto de dados para viabilizar o treinamento dos modelos, o treinamento propriamente dito e a avaliação dos resultados obtidos. Com isso, este trabalho busca desenvolver uma pipeline robusta o suficiente para ser integrada a um sistema de reconhecimento de veículos.

Palavras-chave: Reconhecimento de placas de veículos, monitoramento de veículos, visão computacional, inteligência artificial.

ABSTRACT

This work aims to propose a vehicle recognition pipeline through license plate identification. To achieve this, the pipeline must meet two main requirements: (1) it must be able to locate the vehicle's license plate in a captured image and (2) correctly identify the plate's characters. To fulfill these requirements, the ALPR dataset was used to train two models. The first is the YOLO 11 model, trained to detect and locate vehicle license plates in images. The second is the TPS-ResNet-BiLSTM-Attn model, responsible for identifying the characters on the plates. The adopted methodology involved preprocessing the dataset to enable model training, conducting the training itself, and evaluating the obtained results. With this, the study aims to develop a robust pipeline capable of being integrated into a vehicle recognition system.

Keywords: License plate recognition, vehicle monitoring, computer vision, artificial intelligence.

LISTA DE ILUSTRAÇÕES

Sumário

1. INTRODUÇÃO

1.1. Motivação e Contextualização do Problema

A aplicação da tecnologia na atualidade vem transformando diversas áreas da vida humana e o monitoramento de ambientes públicos não deixa de ser diferente. A necessidade de coibir atos criminosos e de colaborar com possíveis investigações torna-se indispensável nos dias atuais.

Entre algumas iniciativas, podemos citar o Centro de Segurança e Inteligência (CSI) da cidade de São José dos Campos, que é um sistema de monitoramento inteligente que é capaz de realizar reconhecimento facial, rastreamento de pessoas, objetos e veículos, detectar tempo de permanência e detecção de movimento. (JUNIOR, P., 2023)

Outra iniciativa, dentro deste contexto, temos o projeto Muralha Digital que iniciou a operação na cidade de Vitória da Conquista, que é um sistema de radares para controle de velocidade de veículos, que também é capaz de detectar e responder a incidentes de segurança em tempo real. (MURALHA, 2025)

Dito isso, temos a Universidade Estadual do Sudoeste da Bahia que é uma instituição pública de ensino superior com sede em Vitória da Conquista, porém esteja presente com outros dois campi, um em Jequié e o outro em Itapetinga.

O campus de Vitória da Conquista enfrenta uma dificuldade peculiar, pois se caractere por ser um campus aberto, em que possui uma escola técnica da rede estadual dentro de suas dependências, onde a circulação de ônibus do sistema de transporte público de Vitória da Conquista, de associações estudantis de cidades vizinhas, além dos diversos veículos particulares que acessam livremente campus.

Esses fatores trazem enormes dificuldades para a instituição controlar o acesso de veículos em suas dependências, de forma que não há o registro de quais veículos entraram em cada momento na instituição.

Com isso em mente, a proposta deste trabalho é desenvolver uma pipeline de monitoramento de placas veículos, utilizando de técnicas de visão computacional para identificar e registrar a entrada de veículos.

1.2. Trabalhos Relacionados

Dado a relevância do tema, há diversas aplicações e trabalhos que buscam contribuir com este problema, podemos citar diversos trabalhos como:

O trabalho publicado por Zherzdev e Gruzdev, *LPRNet: License Plate Recognition via Deep Neural Networks* (ZHERZDEV, GRUZDEV, 2018), em que apresenta uma abordagem de Reconhecimento automático de placas, aplicado em um conjunto de dados de placas de chinesas. Nessa abordagem os autores propuseram um modelo totalmente convolucional, sem utilizar um decodificador baseado em LSTM. Com essa abordagem o modelo atingiu resultados de 94.1%, 95% e 94%, respectivamente para as variações: *baseline*, *basic* e *reduced*.

Os diversos trabalhos publicados por Laroca, *A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector* (LAROCA, *et. al*, 2018), apresenta um sistema de reconhecimento automático de placas veiculares (ALPR) baseado no detector YOLO 2 e redes neurais convolucionais (CNNs) para garantir robustez em diferentes condições, comparando com os sistemas comerciais: *Sighthound* e OpenALPR. O sistema foi avaliado em dois conjuntos de dados: SSIG e UFPR-ALPR. No conjunto SSIG o sistema demonstrou uma taxa de reconhecimento de 93,53% a 47 FPS, enquanto no UFPR-ALPR, o sistema atingiu 78,33% de acurácia a 35 FPS, enquanto as soluções comerciais ficaram abaixo de 70%.

A First Look at Dataset Bias in License Plate Recognition, publicado por (LAROCA, *et. al*, 2022), neste trabalho os autores abordaram o problema de viés em conjuntos de dados para reconhecimento de placas veiculares. Assim, os autores realizaram experimentos em oito conjuntos de dados, quatro coletados no Brasil e quatro em China continental, e foram observados que cada conjunto de dados tem uma “assinatura” única e identificável, uma vez que um modelo de classificação leve prevê o conjunto de dados de origem de uma imagem de placa de carro com mais de 95% de precisão.

Do We Train on Test Data? The Impact of Near-Duplicates on License Plate Recognition (LAROCA, *et. al*, 2023a), que aborda a presença de quase-duplicadas nos principais conjuntos de dados que são amplamente utilizados em pesquisas de reconhecimento de placas veiculares. Esses quase-duplicadas são imagens diferentes que exibem a mesma placa, e este trabalho aborda a presença destas quase-duplicadas no conjunto de treinamento e de teste. Nos experimentos realizados com os dois principais conjuntos de dados, AOLP e CCPD, usando seis modelos renomados, mostrou-se que ao avaliar os modelos em conjuntos de dados mais justos, sem quase-duplicadas, houve redução substancial na taxa de reconhecimento do modelo.

No trabalho *Leveraging Model Fusion for Improved License Plate Recognition* (LAROCA, *et. al*, 2023b), os autores buscaram investigar as melhorias potenciais no reconhecimento de placas de veículos através do uso de múltiplos modelos de reconhecimento. Aplicando 12 modelos de reconhecimento para avaliar os caracteres utilizando como estratégias de seleção: o reconhecimento de maior confiança ou seleção baseada no voto majoritário.

1.3. Objetivos

1.3.1. Objetivo Geral

Desenvolver uma pipeline de identificação de veículos para a Universidade Estadual do Sudoeste da Bahia, usando algoritmos de visão computacional para permitir o monitoramento das placas dos veículos.

1.3.2. Objetivos Específicos

a) Revisar a literatura sobre aprendizado de máquina, focando em visão computacional e reconhecimento óptico de caracteres;
b) Implementar um modelo de detecção de objetos, para detectar e localizar as placas dos veículos.
c) Implementar um modelo de reconhecimento de caracteres.
d) Testar os resultados;
e) Analisar os resultados;

1.4. Metodologia

Para o desenvolvimento deste trabalho, foi adotado a metodologia, Cross Industry Standard Process for Data Mining (CRISP-DM), o modelo de referência desta metodologia define um ciclo de vida de seis fases para o desenvolvimento de um projeto: entendimento do negócio, entendimento dos dados, preparação dos dados, modelagem, avaliação e implementação. Cabe destacar que para este trabalho, foi aplicado até a Fase 5 do modelo CRISP-DM.

Conforme (JUNIOR, O., 2023), as seis etapas do CRISP-DM para um projeto de ciência de dados são:

Fase 1 - Entendimento do negócio: compreende a fase inicial, busca-se entender qual o problema que precisa ser resolvido, os objetivos e os requisitos do projeto para com isso definir um plano preliminar para atingir o objetivo.

Fase 2 - Entendimento dos dados: temos a coleta inicial de dados e a análise exploratória dos dados, com o objetivo de entender a sua estrutura, identificar possíveis problemas de qualidade de dados e descobrir os primeiros insights.

Fase 3 - Preparação dos dados: compreende todas as etapas necessárias para construir o conjunto de dados final.

Fase 4 - Modelagem: aplicação das técnicas de modelagem para treinar, avaliar e ajustar os parâmetros dos modelos.

Fase 5 - Avaliação: após a fase de modelagem, em que construímos o modelo, o avaliamos para definir a eficácia do modelo em resolver o problema definido na fase 1.

Fase 6 - Implantação: etapa final do projeto, em que se realiza a documentação, apresentação e colocamos em prática a solução.

1.5. Organização do Trabalho

Este trabalho apresenta um sistema de monitoramento de veículos. Além deste capítulo de introdução, para atingir este objetivo, serão desenvolvidas as seguintes etapas metodológicas. No segundo capítulo, revisão bibliográfica oriunda de diversas fontes, como livros, artigos científicos e vídeos relacionados a aprendizagem de máquina, visão computacional, reconhecimento óptico de caracteres. No terceiro capítulo, temos a modelagem e implementação do sistema monitoramento veículos, incluindo o modelo de navegação autônoma e o reconhecimento óptico de caracteres. No quarto capítulo, temos os resultados do sistema. Por fim, no quinto e último capítulo, temos a conclusão do trabalho.

2. FUNDAMENTOS TEÓRICOS

2.1. Visão Computacional

2.1.1. Visão Geral

A percepção das estruturas tridimensionais do mundo é obtida com certa facilidade para nós humanos. Pensar em quão vivido é a percepção tridimensional ao olhar os objetos, pessoas e ambientes à nossa volta, com as formas, padrões de luz e sombras sobre as superfícies. Os psicólogos perceptivos têm passado décadas na busca pelo entendimento de como funciona o sistema visual e, embora possa-se inventar ilusões de ótica para separar alguns de seus princípios, uma solução completa para este quebra-cabeça permanece indescritível. (SZELISKI, 2022)

De forma paralela, pesquisadores no campo da visão computacional têm desenvolvido técnicas matemáticas para recuperar a forma tridimensional e a aparência de objetos em imagens. O progresso neste campo tem sido rápido nas últimas duas décadas. Atualmente, têm-se técnicas confiáveis de forma precisa computar um modelo 3D de um ambiente com milhares de fotografias parcialmente sobrepostas. Dado um conjunto suficientemente grande de vistas de um determinado objeto ou fachada, pode-se criar modelos de superfície 3D densos e precisos com a utilização da correspondência estéreo. Consegue-se até mesmo, com sucesso moderado, delinear a maioria das pessoas e objetos em uma fotografia. Entretanto, apesar de todos esses avanços das últimas décadas, a capacidade de um computador explicar uma imagem com o mesmo nível de detalhe e causalidade de uma criança de dois anos permanece indefinida. (SZELISKI, 2022)

Então temos a pergunta, por que a visão é tão difícil? Em parte, é por se tratar de um problema inverso, no qual procura-se recuperar algumas incógnitas, dadas informações insuficientes para especificar completamente a solução. Assim, deve-se recorrer a modelos probabilísticos e baseados na física, ou à aprendizagem automática a partir de grandes conjuntos de exemplos, para eliminar a ambiguidade entre soluções potenciais. Contudo, a modelagem do mundo visual em toda a sua rica complexidade é muito mais difícil do que modelar o trato vocal que produz sons falados. (SZELISKI, 2022)

Os modelos avançados em que se usa na visão computacional são de forma geral desenvolvidos em física (radiometria, óptica e design de sensores) e em computação gráfica. Em ambos os campos, busca-se modelar como os objetos se movem e se animam, como a luz

que reflete em suas superfícies é espalhada pela atmosfera, refratada através de lentes de câmeras (ou olhos humanos) e, finalmente, projetada em um plano de imagem plano (ou curvo). (SZELISKI, 2022)

Embora a computação gráfica ainda não seja perfeita, a ilusão de realidade está essencialmente presente. Na visão computacional, tenta-se fazer o inverso, ou seja, descrever o mundo que se vê em uma ou mais imagens e reconstruir suas propriedades, como forma, iluminação e distribuição de cores. Embora isso seja feito tão facilmente por humanos e animais, constitui-se uma tarefa complexa para os algoritmos de visão computacional. (SZELISKI, 2022)

A partir dos avanços obtidos, a visão computacional está sendo usada hoje em uma ampla variedade de situações do mundo real: reconhecimento óptico de caracteres (OCR), inspeção de máquina, varejo, armazéns de logística, imagens médicas, veículos autônomos, fotogrametria, captura de movimento, vigilância, entre diversos outros. (SZELISKI, 2022)

2.1.2. Representação de imagens 2D

Uma imagem para ser processada por um computador é necessário que ela seja descrita por um número finito de pontos e representada por um número finito de tons ou cores. Ao tomarmos como exemplo uma imagem em tons de cinza, a representação adequada é como uma matriz cujas linhas e colunas identificam um ponto na imagem, chamado de pixel. Cada pixel é representado no computador como um número inteiro que corresponde a intensidade de luz no pixel. A cor é representada por 8 bits, variando entre 0 e 255, sendo 0 correspondendo a cor preta, e 255 a cor branca. (CONCI, AZEVEDO, LETA, 2008)

Ao considerarmos imagens coloridas, devemos ter três planos, se considerarmos o padrão de cor RBG, onde cada plano registra a quantidade de vermelho (R), azul (B) e verde (G), que ao serem combinadas geram uma visão de cor. (CONCI, AZEVEDO, LETA, 2008)

Nesta representação de imagem como uma matriz bidimensional, os índices da matriz são valores inteiros que especificam a linha e a coluna na matriz com a origem, isto é, o ponto (0,0) localizado no canto superior esquerdo da imagem. As posições dos pixels são representadas no plano da imagem pelas coordenadas x e y, que denotam as coordenadas espaciais. (CONCI, AZEVEDO, LETA, 2008)

2.1.3. Aplicações

Quer se trate de diagnóstico médico, veículos autônomos, monitoramento de câmeras ou filtros inteligentes, diversas são as aplicações no campo da visão computacional que estão intimamente relacionadas às nossas vidas atuais e futuras. Nos últimos anos, o aprendizado profundo, especialmente com as redes convolucionais (abordado na seção 1.2.4), tem sido o poder transformador para melhorar o desempenho dos sistemas de visão computacional. Desta forma, pode-se dizer que as aplicações mais avançadas de visão computacional são quase inseparáveis do aprendizado profundo. (ZHANG, 2023).

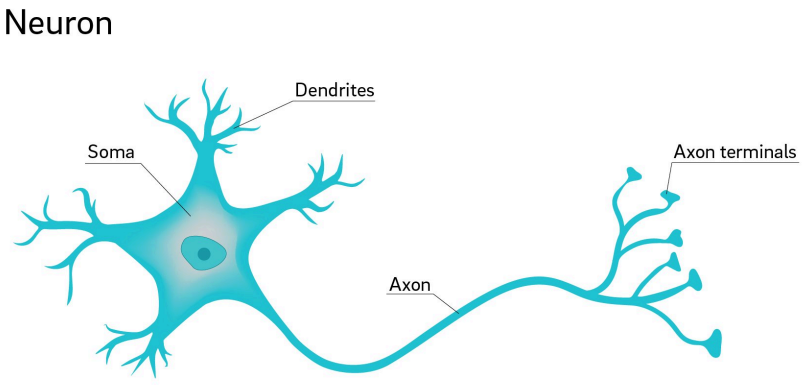
Como as redes convolucionais podem representar efetivamente imagens em vários níveis, essas representações em camadas têm sido usadas com sucesso em várias tarefas de visão computacional, como detecção de objetos, rastreamento de objetos, segmentação semântica, segmentação de instâncias, transferência de estilo e entre outros. (ZHANG, 2023).

2.2. Aprendizado Profundo

2.2.1. Redes Neurais Artificiais

A partir da inspiração biológica, temos que conceber alguns conceitos iniciais. Primeiro sobre o neurônio biológico, que de forma geral, é uma célula que recebe sinais de entrada (informações) através da árvore dendrítica, que passa pelo corpo celular e por fim a informação é transmitida a outros neurônios, músculos ou sensores por meio do axônio. Na figura 1 podemos ver uma representação de um neurônio biológico.

Figura 1 - Representação de um neurônio biológico.

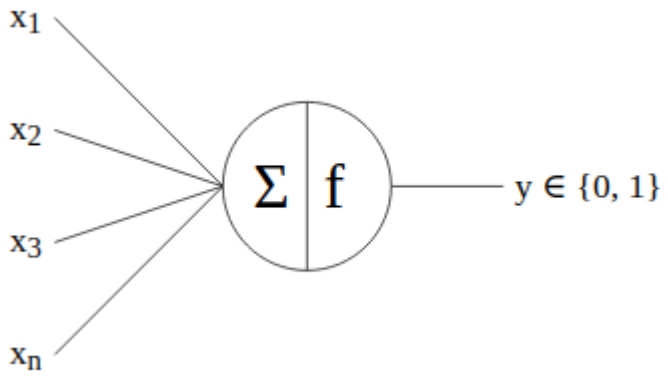


Fonte: (VADAPALLI, 2021)

Nas redes neurais biológicas, temos as conexões entre os neurônios, um neurônio pode se conectar com diversos outros neurônios, processo este chamado de sinapses. “A sinapse é a região responsável por realizar a comunicação entre dois ou mais neurônios, ou de um neurônio para um órgão efetor, ou seja, um músculo ou uma glândula”. (CASTILHO, 2025)

Partindo disso, temos as redes neurais artificiais, que possuem inspiração nas redes neurais biológicas, buscam modelar em computação as conexões entre neurônios. Na figura 2 podemos observar a representação do modelo de um neurônio de McCulloch-Pitts.

Figura 2 - Modelo de neurônio de McCulloch-Pitts.



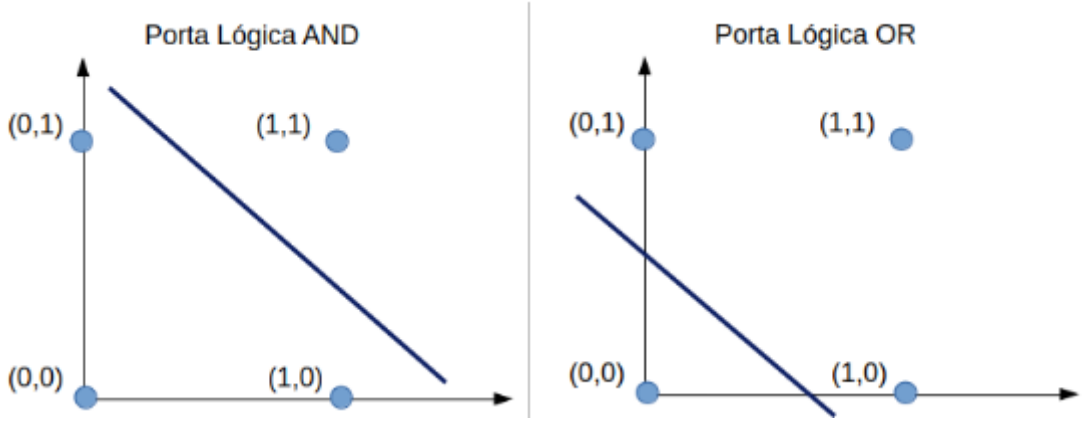
Fonte: De própria autoria.

Na construção do modelo de McCulloch-Pitts o interesse está em modelar os aspectos ligados ao processamento da informação em um neurônio. Por processamento de informação entende-se os caminhos e etapas pelas quais passam os potenciais de ação que trafegam de: um neurônio a outro neurônio; receptores sensoriais a um neurônio; ou de um neurônio a um músculo. Desta forma, os modelos matemáticos devem representar os dendritos, as sinapses, o corpo celular, e o axônio. (AUGUSTO, 2014)

Em 1958 temos a rede Perceptron, proposta por Rosenblatt, o primeiro modelo de aprendizagem supervisionada. Além disso, é a forma mais simples de uma rede neural usada para classificação (problemas linearmente separáveis). Entretanto, as redes Perceptrons possuem a limitação de classificação, para problemas que somente possam ser separáveis linearmente, ou seja, o algoritmo de perceptron converge e posiciona a superfície de decisão em um hiperplano entre duas classes. (HAYKIN, 2001)

Na figura 3, vemos dois exemplos de problemas linearmente separáveis, as portas lógicas AND e OR. Vemos que podemos separar as combinações que resultam em verdadeiro ou falso utilizando uma reta.

Figura 3 - Exemplos de problemas linearmente separáveis.



Fonte: De própria autoria.

Para realizar o treinamento da rede Perceptron seguimos as etapas que estão descritas a seguir:

Etapa 1: definição das entradas e das saídas desejadas.

Definição das entradas (neste exemplo serão apenas duas entradas e um único exemplo) e da saída desejada.

Entradas: X_1 e X_2.

Saída: Y

Etapa 2: definição da taxa de aprendizado.

Definição da taxa de aprendizado da rede (representado por n)

Etapa 3: inicialização dos pesos e do viés.

Inicialização dos pesos (há um peso associado a cada entrada) e do viés.

Pesos: W_1 e W_2.

Viés: bk.

Etapa 4: operação soma.

Operação da soma ponderada entre as entradas e os pesos, com a adição do viés.

$$Z = X_1 \cdot W_1 + X_2 \cdot W_2 + b_k$$

Etapa 5: aplicação da função degrau.

$$f(z) = \begin{matrix} 1, para\ z \geq 0; \\ 0, para\ z < 0. \end{matrix}$$

Etapa 6: comparação da saída da rede com a saída desejada.

Verificação da saída da rede com a saída desejada, que chamaremos de erro da rede (se o erro for igual a 0 significa que saída da rede é igual a saída da rede.).

$$err$$

$$erro = Y - f(z)$$

Etapa 7: ajuste dos parâmetros.

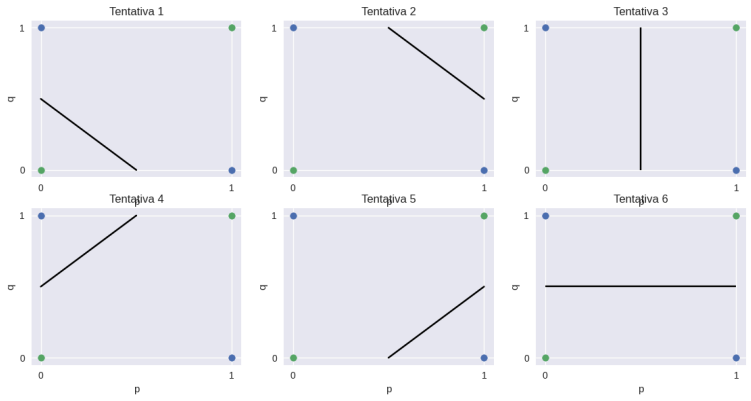
Atualização dos pesos e veis.

$$W_1 = W_1 + n \cdot erro \cdot X_1$$
$$W_2 = W_2 + n \cdot erro \cdot X_2$$
$$b_2 = b_2 + n \cdot erro \cdot 1$$

Por fim, esses passos são repetidos por um número definido de épocas ou pode-se utilizar uma outra condição parada.

Embora a perceptron tenha a capacidade de resolver alguns problemas (especificamente os problemas linearmente separáveis), ela mostrou-se limitada, não sendo possível resolver problemas não linearmente separáveis, como o problema do OU exclusivo (XOR). Pois, a solução da porta lógica XOR não pode ser realizada de forma linear, isto é, não há uma reta que possa ser traçada de forma que consiga encontrar uma solução, como podemos ver algumas tentativas na figura 4 a seguir.

Figura 4 - Algumas tentativas de realizar a separação linear da porta lógica XOR.



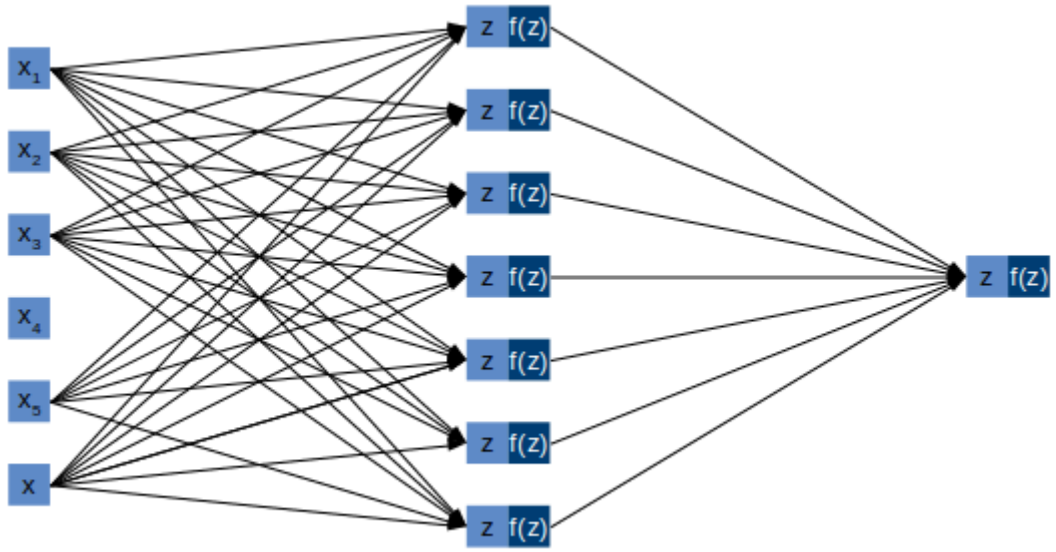
Fonte: De própria autoria.

2.2.2. Perceptron Multicamadas

A Perceptron de Multicamadas (MLP) é uma classe de redes neurais constituída de uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Neste tipo de rede, as entradas são propagadas para as camadas seguintes até chegar à camada de saída, chamado de feedforward. A aplicação de MLP é realizada através de um aprendizado supervisionado com o algoritmo de retropropagação de erro. (HAYKIN, 2001)

Na figura 5 vemos um exemplo de uma MLP, assim como no neurônio de uma perceptron, o neurônio de uma MLP possui um vetor de entradas em que é feito a soma ponderada com os seus respectivos pesos somado a um viés. Entretanto, cabe destacar uma diferença entre os neurônios de uma MLP e de uma Perceptron Simples no que diz respeito a função de ativação. A função degrau utilizada na perceptron não pode ser utilizado em uma MLP, pois em uma Perceptron Multicamadas se utiliza funções de ativação deriváveis e não lineares, como por exemplo: sigmoide, tangente hiperbólica, relu, softmax e outras.

Figura 5 - Exemplo da arquitetura de uma MLP.



Fonte: De própria autoria.

Aprendizado supervisionado é um método de aprendizagem de máquina, onde o problema possui uma entrada e uma saída e o objetivo é encontrar os parâmetros corretos para gerar a saída. Assim, em uma rede neural com aprendizagem supervisionada, os parâmetros

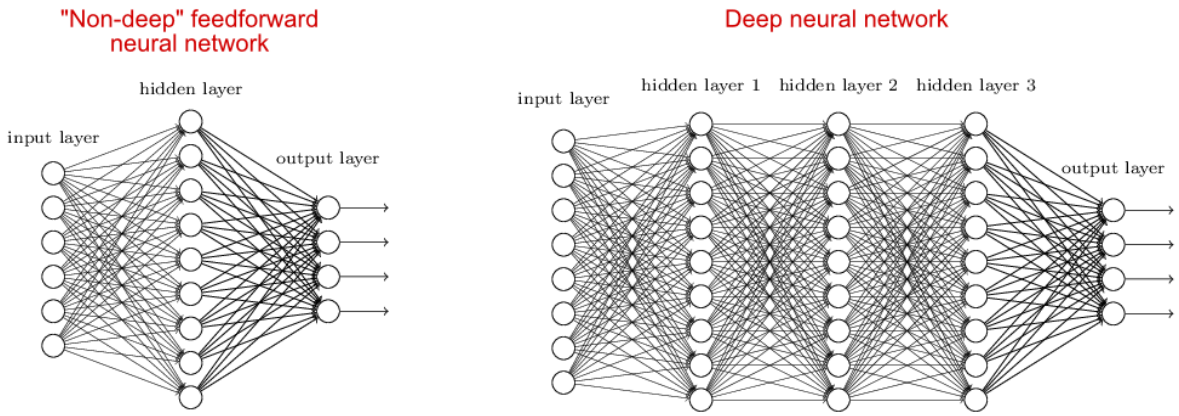
são inicializados com alguma técnica, como por exemplo, uma distribuição normal, e o algoritmo realiza o feedforward das entradas com esses parâmetros que gerará ao final a saída.

Comparando a saída gerada com saída desejada, realiza-se a retropropagação do erro para ajustar os valores dos parâmetros. Este processo é realizado iterativamente até o erro se aproximar de zero ou que número de épocas de treino determinado seja concluído. A retropropagação do erro é realizada através de um algoritmo de otimização, como por exemplo, SGD, Adam, Adamax, Adagrad, Adadelta, RMSprop e FTRL.

2.2.3. Aprendizado Profundo

O aprendizado profundo, também conhecido por redes neurais profundas, são redes neurais com muitas camadas ocultas, conforme a figura 6. A arquitetura mais simples são as chamadas perceptrons multicamadas, onde há mais de duas camadas ocultas, como dito anteriormente, consistem em multicamadas de neurônios totalmente conectadas, com a camada anterior (no qual recebe as entradas para a camada atual) e com a camada posterior (no qual ela exerce influência). A primeira camada é chamada de camada de entrada, a última camada é chamada de camada de saída e todas as camadas entre elas são chamadas de camadas ocultas. (ZHANG, et al., 2022).

Figura 6 - Representação da diferença entre rede neural e aprendizado profundo.



Fonte: (OZGUR, 2019).

Embora temos as MLP como os modelos mais simples, temos abordagens mais robustas e atuais como as máquinas de Boltzmann profundas, redes neurais recorrentes e redes neurais convolucionais.

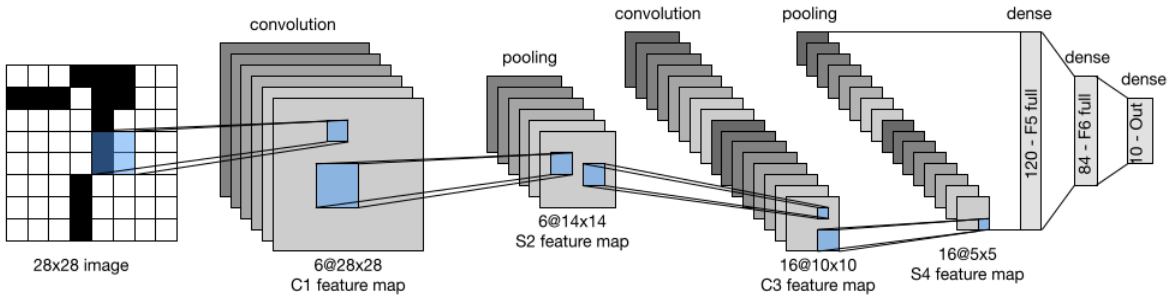
2.2.4. Redes Neurais Convolucionais

Nas técnicas de redes neurais com o uso de MLP, quando nos deparamos com dados de imagem, para os quais cada exemplo consiste em uma grade bidimensional de pixels. A forma de lidar com essas estruturas nessas abordagens se mostram insatisfatórias. Pois, simplesmente há o descarte da estrutura espacial de cada imagem, achatando-as em vetores unidimensionais para então inserir em uma MLP totalmente conectada. (ZHANG, 2023)

As Redes Neurais Convolucionais (Convolutional Neural Networks - CNN), é um conjunto de tipos de redes neurais que são projetadas especificamente para processar de forma eficiente informações espaciais. As arquiteturas baseadas nas CNN tornaram-se amplamente presentes nas pesquisas e aplicações no campo de visão computacional, como por exemplo, reconhecimento de imagens, detecção de objetos, segmentação semântica, segmentação de instância e entre outros. (ZHANG, 2023)

Nas arquiteturas baseadas em CNN, temos como operações básicas as camadas convolucionais e camadas de pooling, e os conceitos de preenchimento (*padding*) e passada (*stride*) (ZHANG, 2023). Esses tópicos serão mais discutidos nas de seção de visão computacional. Na figura 7 a representação de uma CNN para classificação de imagens.

Figura 7 - Representação de uma Rede Neural Convolucional - LeNet.



Fonte: (ZHANG, 2023).

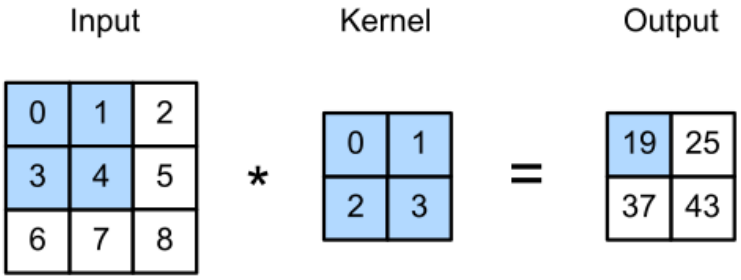
2.2.4.1. Camadas de Convolução

Embora tenhamos o nome convolução, na prática a operação aplicada e a correlação cruzada. Em cada camada temos um tensor de entrada e tensor de núcleo (*kernel*) que são combinados, através da correlação cruzada para produzir um tensor de saída. No início da

rede a imagem é o tensor de entrada, no restante da rede são as saídas da camada anterior (ZHANG, 2023)

Para exemplificarmos essa operação, desconsideraremos tensores de entrada com múltiplos canais, que serão abordados na seção 2.2.4.4. Além disso, também será desconsiderado o parâmetro viés, que é um valor adicionado na operação de correlação cruzada. Na figura 8 o temos um tensor de entrada de forma 3×3 e o tensor do núcleo com a forma de 2×2. A porção sombreada para o primeiro elemento da saída, como os tensores de entrada e de núcleo, utiliza para a computação da saída: $0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$. (ZHANG, 2023)

Figura 8 - Representação da operação de correlação cruzada.



Fonte: (ZHANG, 2023).

Conforme (ZHANG, 2023),

> "Na operação de correlação cruzada bidimensional, começamos com a janela de convolução posicionada no canto superior esquerdo do tensor de entrada e deslizamos através do tensor de entrada, da esquerda para a direita e de cima para baixo. Quando a janela de convolução desliza para uma determinada posição, o subtensor de entrada contido naquela janela e o tensor do kernel são multiplicados elemento a elemento e o tensor resultante é somado produzindo um único valor escalar. Este resultado fornece o valor do tensor de saída no local correspondente" (ZHANG, 2023)

No exemplo da figura 8, o tensor de entrada com a forma 3×3 e o núcleo 2×2, que produz um tensor de saída com a forma 2×2. De forma genérica, com o pressuposto da forma de entrada de $n_h x n_w$ e a forma do núcleo kernel de $k_h x k_w$, a forma de saída será $\left(h - k_h + 1\right)x\left(w - k_w + 1\right)$. Assim, o formato de saída da camada convolucional é determinado pela forma da entrada e pela forma do núcleo de convolução. (ZHANG, 2023)

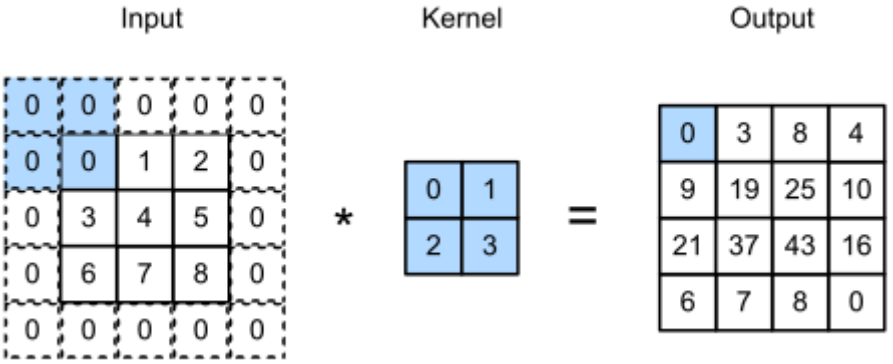
Em muitos casos, quando se faz necessário afetar o tamanho da saída, os hiperparâmetros de preenchimento e a passada são incluídos para afetar o formato de saída. Pois, como os núcleos geralmente têm largura e altura maiores que 1, após aplicar muitas convoluções sucessivas, tendemos a acabar com resultados consideravelmente menores que a entrada. Por exemplo, ao iniciarmos com uma imagem de 240×240, 10 camadas com o núcleo de 5×5, reduzem a imagem para 200×200, com o corte de 30% da imagem e a obliteração de informações importantes sobre os limites da imagem original. Nesta situação, o preenchimento é a ferramenta mais popular para lidar com esse problema. (ZHANG, 2023)

2.2.4.2. Preenchimento

Ao aplicar camadas convolucionais temos a tendência de perder as informações dos pixels no perímetro de nossa imagem, esse efeito tem uma tendência de aumento à medida em que se ocorre a aplicação sucessiva de camadas convolucionais. Para solucionar este problema em arquiteturas convolucionais, realizamos a adição de pixels extras ao redor do limite do tensor de entrada, isto é o parâmetro de preenchimento, estes pixels de forma geral possuem o valor igual a 0. (ZHANG, 2023).

Conforme podemos ver na figura 9, aplicamos o preenchimento no tensor de entrada 3×3, tendo como resultado um tensor de tamanho para 5×5. A saída correspondente então para uma matriz 4×4. (ZHANG, 2023).

Figura 9 - Operação validação cruzada com preenchimento.



Fonte: (ZHANG, 2023).

Assim, conforme a equação 2.1, em que o preenchimento é representado por p_h e p_w, podemos aumentar o tamanho do tensor de saída. Cabe destacar que esta representação é o número de linhas adicionado, no exemplo da figura 9, p_h e p_w foram iguais a 2.

$$\left(n_h - k_h + p_h + 1\right) \times \left(n_w - k_w + p_w + 1\right) \qquad (2.1)$$

2.2.4.3. *Stride*

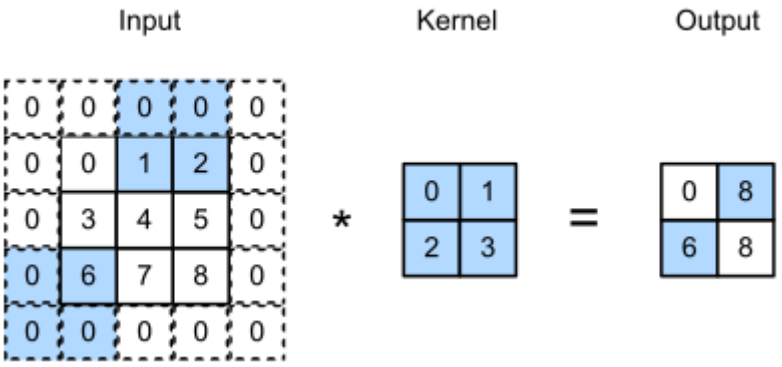
Conforme (ZHANG, 2023),

> "Ao calcular a correlação cruzada, começamos com a janela de convolução no canto superior esquerdo do tensor de entrada e, em seguida, deslizamos sobre todos os locais, da esquerda para a direita, e de cima para baixo. Até agora, o padrão utilizado é deslizar a janela um pixel de cada vez, porém, às vezes, seja por eficiência computacional ou para reduzir a resolução, há a movimentação da janela mais de um pixel por vez, pulando os locais intermediários." (ZHANG, 2023)

O parâmetro que define a quantidade de linhas ou colunas em que a janela é deslocada é chamado de *stride* (passo). Em relação ao *stride*, é definido um valor de passo, tanto para a altura, quanto para largura. Na figura 10, mostra uma operação de correlação cruzada bidimensional com *stride* de 3 na vertical e 2 na horizontal. No exemplo da figura 10, conforme (ZHANG, 2023),

> "Podemos ver que quando o segundo elemento da primeira coluna é gerado, a janela de convolução desliza três linhas para baixo. A janela de convolução desliza duas colunas para a direita quando o segundo elemento da primeira linha é gerado. Quando a janela de convolução continua a deslizar duas colunas para a direita na entrada, não há saída porque o elemento de entrada não pode preencher a janela, a menos que adicionemos outra coluna de preenchimento" (ZHANG, 2023).

Figura 10 - Operação validação cruzada com preenchimento e passo.



Fonte: (ZHANG, 2023).

Quando o *stride* para a altura é s_h e para a largura é s_w, a forma da saída é dada pela equação (2.2)

$$\left[\frac{n_h - k_h + s_h}{s_h}\right] \times \left[\frac{n_w - k_w + s_w}{s_w}\right] \qquad (2.2)$$

2.2.4.4. Entradas com Múltiplos Canais

Nas arquiteturas de CNN, é comum que os tensores de entrada das camadas tenham múltiplos canais, por exemplo em uma imagem colorida com o padrão RGB, onde há um canal para cada cor. Com esse tipo de entrada, os *kernels* de convolução precisam possuir o mesmo número de canais que temos nos dados de entrada, cada canal deste *kernel* chamados de filtro convolucional. Nos exemplos dados até aqui, *kernel* e filtro eram equivalentes. (ZHANG, 2023).

Conforme (ZHANG, 2023),

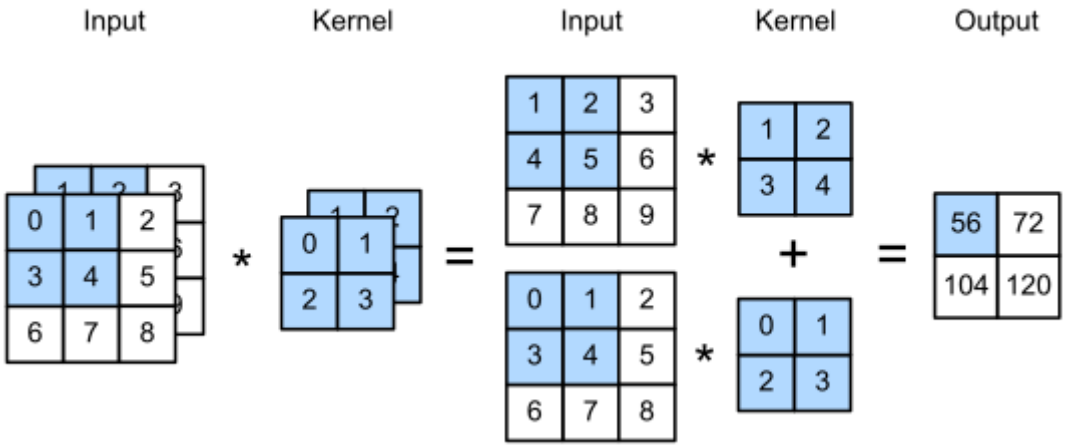
> "Suponha-se que o número de canais para os dados de entrada seja c_i, o número de canais de entrada do kernel de convolução também precisa ser c_i. Se a forma da janela do nosso kernel de convolução for $k_h \times k_w$, então quando $c_i = 1$, podemos pensar em nosso kernel de convolução apenas como um tensor bidimensional da forma $k_h \times k_w$" (ZHANG, 2023).

Em exemplos em que $c_i > 1$, conforme (ZHANG, 2023),

> "[...] ao considerarmos $c_i > 1$, há a necessidade de um núcleo que contenha um tensor de forma $k_h \times k_w$ para cada canal de entrada. Concatenar esses tensores c_i produz um núcleo de convolução de forma $k_h \times k_w \times c_i$. Como o núcleo de entrada e o núcleo de convolução possuem canais c_i, podemos realizar uma operação de correlação cruzada no tensor bidimensional da entrada e no tensor bidimensional do núcleo de convolução para cada canal, somando os resultados c_i (soma-se os canais) para produzir um tensor bidimensional. Assim, temos o resultado de uma correlação cruzada bidimensional entre uma entrada multicanal e um kernel de convolução de canal multe entrada" (ZHANG, 2023).

Um exemplo de aplicação deste conceito pode ser visto na figura 11.

Figura 11 - Operação validação cruzada para entradas com múltiplos canais.



Fonte: (ZHANG, 2023).

2.2.4.5. Saídas com Múltiplos Canais

Além dos tensores de entrada com múltiplos canais, temos também a possibilidade do tensor de saída ter múltiplos canais. No contexto de CNN, é comum a medida em que há a adição de mais camadas na arquitetura, aumentar o número de canais no tensor de saída (ZHANG, 2023).

Conforme (ZHANG, 2023),

> "Denota-se por c_i, e c_o o número de canais de entrada e saída, respectivamente, e seja k_h e k_w a altura e a largura do núcleo. Para obter uma saída com múltiplos canais, cria-se um tensor de núcleo da forma $k_h \times k_w \times c_i$ para cada canal de saída. No passo seguinte, há a concatenação desses tensores de núcleo na dimensão do canal de saída, de modo que a forma do kernel de convolução seja $k_h \times k_w \times c_i \times c_o$. Nas operações de correlação cruzada, o resultado em cada canal de saída é calculado a partir do kernel de convolução correspondente a esse canal de saída e recebe a entrada de todos os canais no tensor de entrada." (ZHANG, 2023)

2.2.4.6. Camadas de Pooling

Por fim, um outro conceito importante em arquiteturas de CNN são as camadas de pooling. Neste contexto, à medida que uma imagem é processada por uma CNN, há a necessidade de se reduzir gradativamente a resolução espacial, com a agregação das informações para que quanto mais profundo na rede, maior seja o campo receptivo, na entrada. (ZHANG, 2023)

Conforme (ZHANG, 2023),

> "Como exemplo, dados uma tarefa em que a pergunta global sobre a imagem, seja, ela contém um gato? Normalmente, as unidades da nossa camada final devem ser sensíveis a toda a entrada. Assim, ao agregar gradualmente informações, produz-se mapas cada vez mais grosseiros, em que alcança-se o objetivo de, em última análise, aprender uma representação global, enquanto há a manutenção de todas as vantagens das camadas convolucionais nas camadas intermediárias de processamento." [Tradução nossa](ZHANG, 2023)

Ainda sobre isso, conforme (ZHANG, 2023),

> "[..] ao detectar características de nível inferior, como arestas, muitas vezes há a necessidade de que as representações sejam um tanto invariantes à translação. Por exemplo, ao tomar-se uma imagem X com um delineamento nítido entre preto e branco e deslocar-se a imagem inteira um pixel para a direita, ou seja, Z[i, j] = X[i, j + 1], então a saída para a nova imagem Z pode ser muito diferente. A borda terá sido deslocada em um pixel. Na realidade, os objetos dificilmente ocorrem exatamente no mesmo lugar, mesmo com um tripé e um objeto estacionário, a vibração da câmera devido ao movimento do obturador pode mudar tudo em um pixel ou mais (câmeras

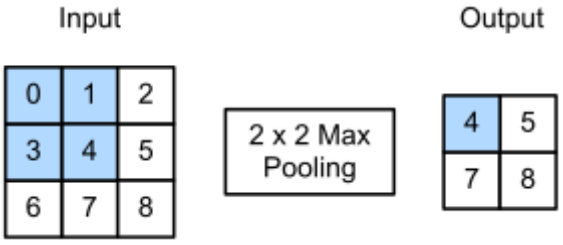
> de última geração são carregadas com recursos especiais para resolver esse problema)" [Tradução nossa] (ZHANG, 2023).

Desta forma, temos que os objetivos das camadas de pooling em CNN são: mitigação a sensibilidade de camadas convolucionais para localização e de reduzir a representação espacial dos tensores de entrada (ZHANG, 2023).

De maneira semelhante às camadas convolucionais, os operadores de pooling consistem em uma janela de formato fixo que se desliza sobre todas as regiões do tensor de entrada de acordo com seu *stride*, com a computação de uma única saída para cada local atravessado pela janela. Embora seja semelhante às camadas convolucionais, nas camadas de pooling não há parâmetros, não há *kernel*, porém os conceitos de *padding* e *stride* se aplicam de maneira semelhante (ZHANG, 2023).

Nas camadas de pooling, os seus operadores são determinísticos, normalmente calcula-se o valor máximo ou médio dos elementos na janela de pooling, chamadas de pooling máximo (abreviadamente max-pooling) e pooling médio, respectivamente. Na figura 12, vemos um exemplo de uso da camada de pooling, com o pooling máximo (ZHANG, 2023).

Figura 12 - Exemplo de uso da camada de pooling, com o pooling máximo.



Fonte: (ZHANG, 2023).

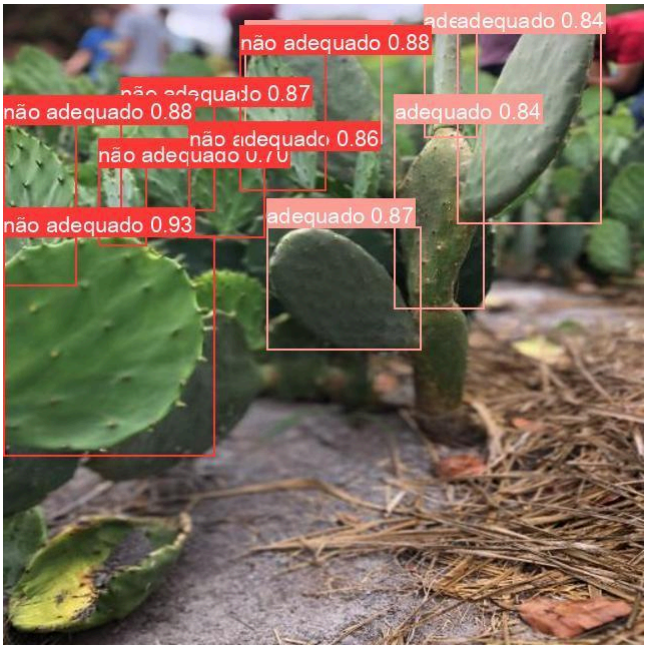
2.2.5. Detecção de Objetos

2.2.5.1. Visão Geral

No tópico 2.2.4 introduzimos as redes convolucionais, com a continuação da abordagem de redes neurais e aprendizado profundo de limitar a explicação apenas para a tarefa de classificação de objetos. Na classificação de objetos em imagens, há a pressuposto de a um objeto mais predominante na imagem. Contudo, nas imagens há geralmente múltiplos objetos de interesse. Neste caso, não queremos apenas saber as suas categorias, queremos

saber também as suas posições na imagem, e isso em visão computacional chamamos de detecção de objetos (ZHANG, 2023).

Figura 13 - Exemplo de detecção de objetos.



Fonte: De própria autoria.

Na detecção de objetos, os seus diversos modelos como R-CNN, Fast R-CNN, Faster R-CNN e YOLO usam redes convolucionais para classificar os objetos e uma rede de regressão para predizer precisamente as coordenadas das caixas delimitadoras. (NUHL, 2024)

Os algoritmos de detecção de objetos podem ser englobados em duas categorias: detecção de objeto de etapa única e detecção de objeto em duas etapas. Na detecção de objeto de etapa única, os algoritmos preveem a caixa delimitadora e a classe em uma única etapa, isto é, em uma única passagem para a frente da rede, a presença de um objeto e a caixa delimitadora são previstas simultaneamente. (NUHL, 2024)

Por outro lado, nos algoritmos de detecção de duas etapas, usam um processo de duas etapas para detectar objetos. O primeiro passo envolve propor uma série de caixas delimitadoras que poderiam potencialmente conter um objeto. Isso geralmente é feito usando um método chamado proposta de região. O segundo passo envolve a execução dessas regiões propostas através de uma rede neural convolucional para classificar as classes de objetos dentro da caixa. (NUHL, 2024)

2.2.5.2. Rede de Detecção de Objetos YOLO

Para a detecção das placas dos veículos, treinamos o algoritmo de detecção objetos para tempo real YOLO 11 com a biblioteca da Ultralytics, para detectar as placas, como vemos na figura 14.

Figura 14 - Detecção de placas de veículos.

Fonte: De própria autoria.

A rede YOLO é uma família de algoritmos de detecção de objetos do tipo de etapa único. Com a primeira versão sendo publicada por Redmon em 2015 no artigo intitulado You Only Look Once: Unified, Real-Time Object Detection, esta família passou por diversas evoluções até chegar na versão 11, com contribuição de diferentes grupos de pessoas e organizações. Na figura 15 podemos visualizar uma linha do tempo das versões da rede YOLO.

Figura 15 - Linha do Tempo da Rede YOLO.

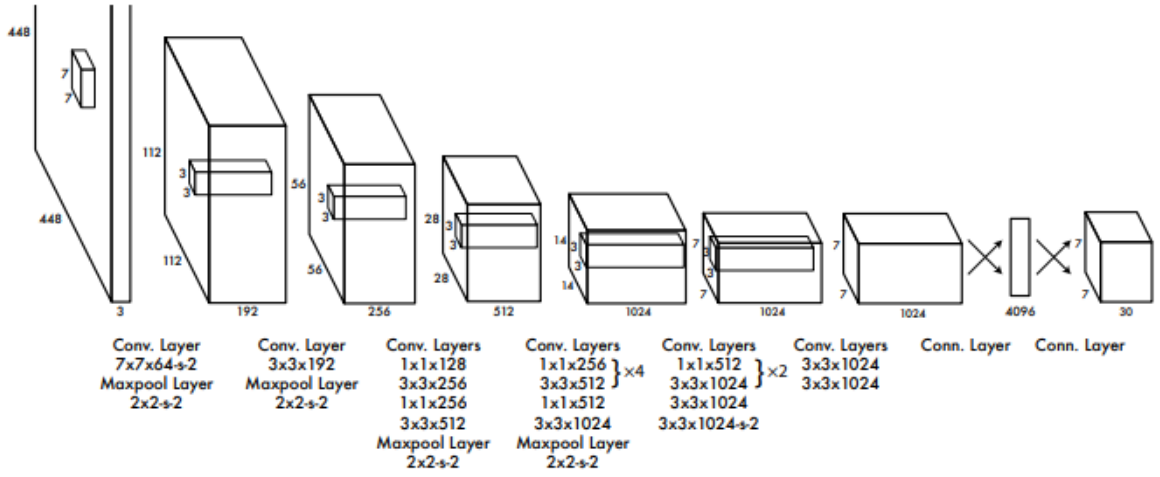
2016	2017	2018	2020	2022	2023	2024
YOLO v1	YOLO v2	YOLO v3	YOLOv4 e YOLOv5	YOLO v6 e YOLO v7	YOLO v8	YOLO v9, YOLO v10 e YOLO v11

Fonte: De própria autoria.

Antes de introduzir a detecção de objetos YOLO, no campo de detecção de objetos as abordagens predominantes eram baseadas em duas etapas como a R-CNN e Fast R-CNN. Essas abordagens eram lentas e exigiam muitos recursos, porém o surgimento dos modelos YOLO trouxeram grandes avanços para a detecção de objetos. Quando o primeiro YOLO foi desenvolvido, ele superou a maioria dos problemas com algoritmos tradicionais de detecção de objetos, com uma arquitetura nova e aprimorada. (BOESCH, 2024)

A arquitetura YOLO original, conforme a figura 16, consistia em 24 camadas convolucionais seguidas por 2 camadas totalmente conectadas. Nesta arquitetura, as camadas convolucionais iniciais da rede extraem recursos da imagem enquanto as camadas totalmente conectadas preveem as probabilidades e coordenadas de saída. Com isso, tanto as caixas delimitadoras quanto a classificação acontecem em uma etapa.

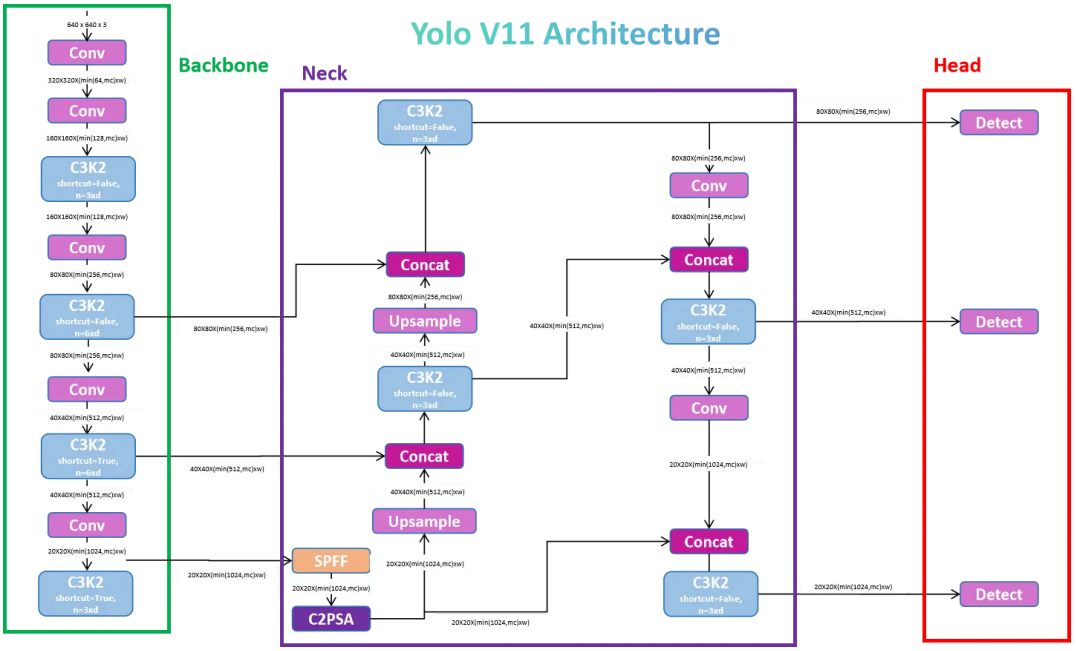
Figura 16 - Arquitetura da Rede YOLO original.



Fonte:(REDMON, *et. al*, 2016).

Contudo, como dito anteriormente, a rede YOLO passou por diversas evoluções ao longo do tempo, através de suas versões até chegar a versão 11. Conforme a figura 17, vemos a arquitetura da versão 11, que consiste em três partes principais: *backbone*, *neck* e *head*.

Figura 17 - Arquitetura da Rede YOLO 11.



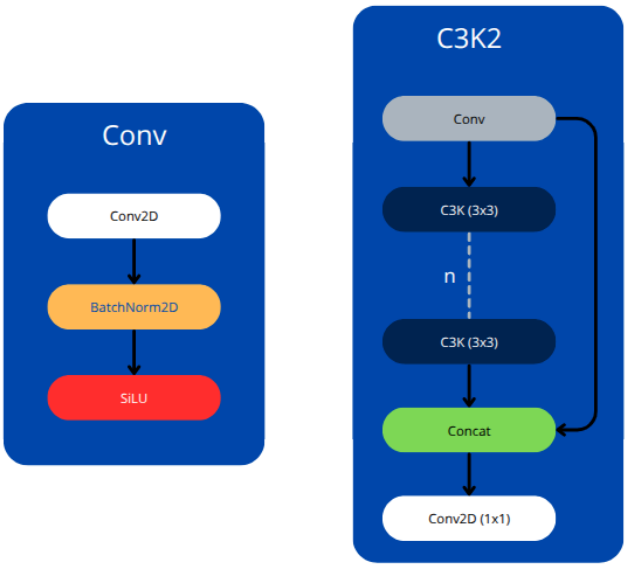
Fonte: (RAO, 2024).

Conforme (RAO, 2024),

> "A arquitetura do YOLO 11 foi projetada para otimizar tanto a velocidade quanto a precisão, com base nos avanços introduzidos em versões anteriores do YOLO, como YOLO 8, YOLO 9 e YOLO 10. As principais inovações arquitetônicas no YOLO 11 giram em torno do bloco C3K2 , do bloco SPFF e do bloco C2PSA, todos os quais aprimoram sua capacidade de processar informações espaciais enquanto mantêm a inferência de alta velocidade." (RAO, 2024)

Na versão 11 da YOLO, o *backbone* é constituído de blocos Convolucionais e blocos C3K2, que conforme a figura 18 podemos visualizar o seu funcionamento. Os blocos Convolucionais são compostos de uma camada de convolução, seguida por uma camada de normalização e por último a aplicação da função de ativação SiLU. Nos blocos C3K2 temos, um bloco convolucional, seguido por *n* blocos C3K, que por último é concatenado com o bloco convolucional do início e por último temos uma camada de convolução com filtro 1x1.

Figura 18 - Blocos Convolucionais e C3K2.

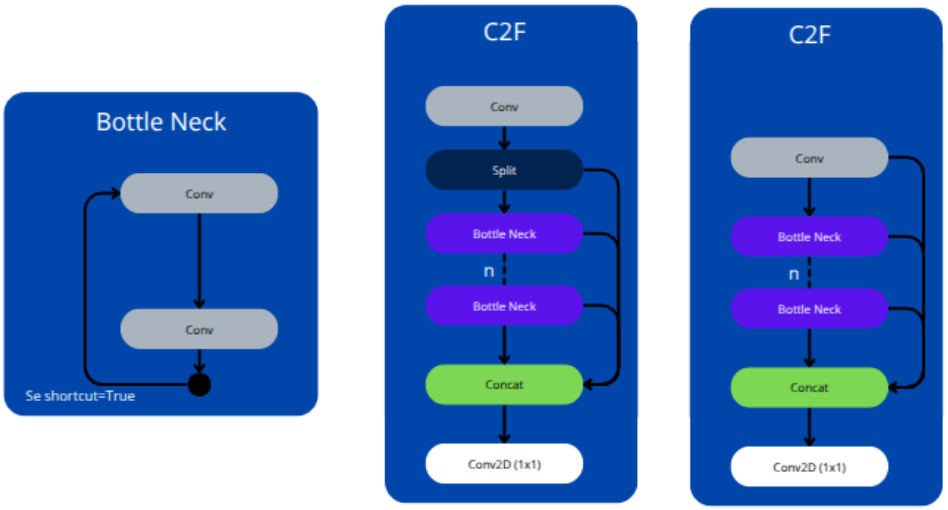


Fonte: Adaptado de (RAO, 2024).

O algoritmo YOLO 11 usa blocos C3K2 para manipular a extração de recursos em diferentes estágios do *backbone*. Estes blocos são uma evolução dos blocos CSP, introduzido em versões anteriores, assim os blocos C3K2 otimiza o fluxo de informações pela rede com a realização da divisão dos mapas de recursos e com a aplicação de uma série de convoluções menores do núcleo, de tamanho 3x3, que são mais rápidas e computacionalmente mais baratas do que convoluções maiores do núcleo. Com o processamento dos mapas de recursos menores e separados e mesclá-los após várias convoluções, o bloco C3K2 melhora a representação de recursos com menos parâmetros em comparação aos blocos C2F do YOLO 8. (RAO, 2024)

O bloco C3K, que está contido no bloco C3K2, contém uma estrutura semelhante ao bloco C2F, porém não é realizada nenhuma divisão. A entrada é passada por um bloco Convolucional, seguido por n camadas de *Bottle Neck* com concatenações e termina com o bloco Convolucional final. Na figura 19 podemos visualizar os blocos C2F, C3K e *Bottle Neck*.

Figura 19 - Blocos C2F, C3K e *Bottle Neck*.

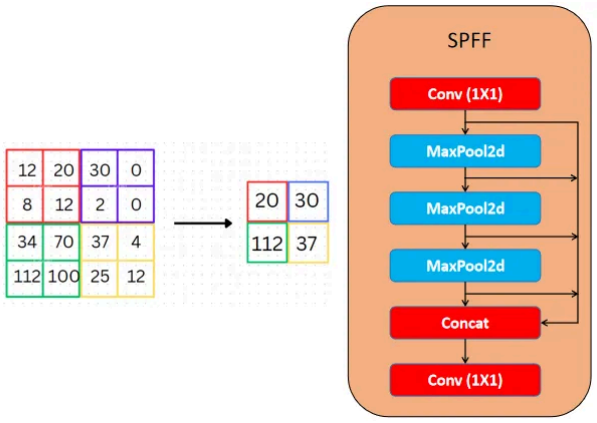


Fonte: Adaptado de (RAO, 2024).

No *neck*, o YOLO mantém o bloco SPFF (Agrupamento de pirâmide espacial rápida), que foi projeto para reunir recursos de diferentes regiões de uma imagem de em várias escalas, isto permite a rede melhorar a capacidade de captura de objetos de diferentes tamanhos, inclusive objetos pequenos. (RAO, 2024)

Esses recursos são agrupados, através do bloco SPFF, com o uso de diversas camadas de max-pooling, com núcleos de tamanhos variados, para agregar informações contextuais em várias escalas. A inclusão deste bloco no YOLO 11 o permite manter a velocidade em tempo real enquanto aprimora sua capacidade de detectar objetos em várias escalas. Na figura 20, podemos visualizar o bloco SPFF.

Figura 20 - Bloco SPFF.

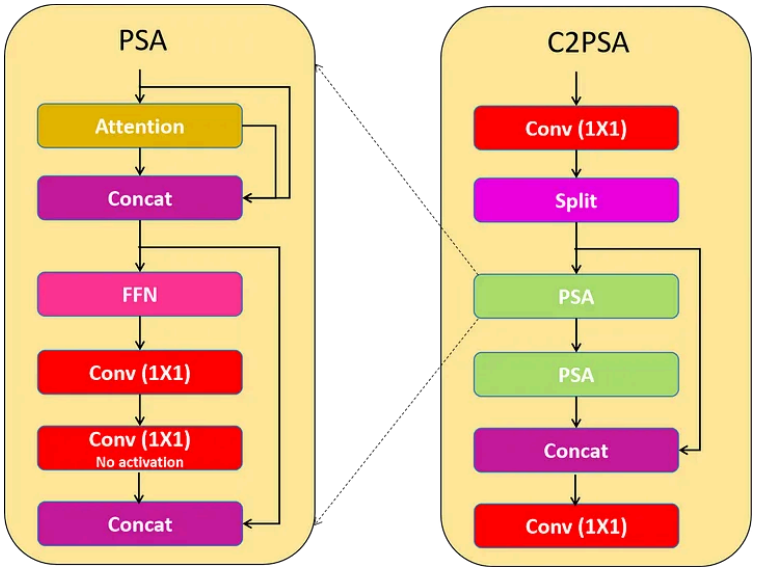


Fonte: (RAO, 2024).

Ainda na parte que compreende o *neck*, o Bloco C2PSA (Estágio de cruzamento parcial com atenção espacial), como mostrado na figura 21, introduz mecanismos de atenção que melhora o foco do modelo em regiões importantes dentro de uma imagem, como objetos menores ou parcialmente ocluídos, enfatizando a relevância espacial nos mapas de recursos. (RAO, 2024)

O bloco C2PSA usa dois blocos PSA (Atenção Espacial Parcial), que operam em ramificações separadas do mapa de recursos e são posteriormente concatenados. Essa construção garante o foco em informações espaciais por parte do modelo, com a manutenção do equilíbrio entre custo computacional e precisão de detecção. O bloco C2PSA refina a capacidade do modelo de focar seletivamente em regiões de interesse aplicando atenção espacial sobre os recursos extraídos. (RAO, 2024)

Figura 21 - Bloco C2PSA.



Fonte: (RAO, 2024)

Além das informações vistas anteriormente, o bloco de atenção sensível à posição (PSA) encapsula a função de aplicar atenção sensível à posição e redes de *feed-forward* a entrada, o que aprimora as capacidades de extração e processamento de recursos. Essas camadas incluem o processamento da camada de entrada com a camada de atenção e a concatenação da entrada e da saída da camada de atenção, que então é passada por uma Rede Neural de Feed forward seguida por uma camada convolucional, com filtro 1x1, e por uma camada convolucional,com filtro 1x1 e sem aplicar função de ativação, por último é concatenada com a saída da primeira concatenação. (RAO, 2024)

Por último, na parte de *head* da arquitetura da rede, temos a detecção, de forma semelhante às versões anteriores do YOLO, o YOLO 11 usa um cabeçote de previsão multiescala para detectar objetos de diferentes tamanhos, que emite caixas de detecção para três escalas diferentes, baixa, média e alta, com o uso dos mapas de recursos gerados pelo backbone e pelo pescoço. (RAO, 2024)

2.2.6. Reconhecimento Óptico de Caracteres

2.2.6.1. Visão Geral

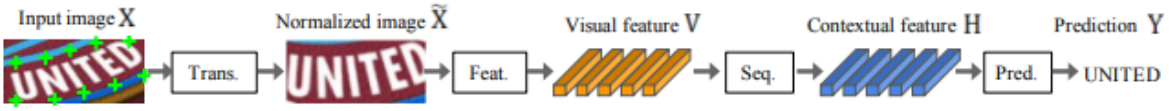
O reconhecimento óptico de caracteres, conhecido pela sigla (OCR) é a conversão de imagens digitalizadas de texto impresso, manuscrito ou datilografado em texto digitalizado. Esta tecnologia permite que caracteres sejam reconhecidos automaticamente através de um mecanismo óptico. No caso dos seres humanos, os nossos olhos são mecanismos ópticos, onde a imagem vista por nossos olhos representam uma entrada de informação para o cérebro. De uma maneira semelhante, os algoritmos de OCR buscam extrair de diferentes tipos de documentos, como documentos digitalizados, PDF e imagens capturadas, o seu conteúdo textual. (JANA et. al, 2014)

Embora podemos destacar que as técnicas atuais ainda enfrentam alguns desafios, especialmente quando lidamos com imagens de baixa qualidade, marcas em cima de outras letras, sombras e condições de iluminação inadequadas, os algoritmos de OCR vêm sendo aplicados em diversas áreas em nosso dia a dia. Como por exemplo, leitura de código de barras, verificação de passaporte, reconhecimento de rótulos de produtos em lojas, verificação de documentos legais, tradução, rótulos de medicamentos, reconhecimento de placas de veículos e entre outros. (SHAIP, 2025)

2.2.6.2. Modelo de Reconhecimento de Caracteres

Para este sistema, utilizamos o framework de reconhecimento de texto da cena de quatro estágios abordados por (BAEK et. al, 2019), no artigo intitulado *What Is Wrong With Scene Text Recognition Model Comparisons?*, em que os quatro estágios são: transformação, extração de características, modelagem sequencial e predição. Na figura 22 podemos visualizar o fluxo de reconhecimento de texto em quatro estágios.

Figura 22 - Fluxo de reconhecimento de texto em quatro estágios.



Fonte: (BAEK et. al, 2019).

No estágio de transformação, a imagem de entrada é transformada na imagem normalizada. Por exemplo, imagens de texto em cenas naturais podem ser capturadas em diversas formas, com os textos mostrados de forma curva ou inclinada, caso estas imagens de entrada não forem transformadas, o estágio subsequente de extração de características precisará aprender uma representação invariante com relação a tal geometria. Para isso a transformação com a *Thin-Plate Spline* (TPS), pode ou não ser aplicada. (BAEK et. al, 2019)

No estágio de extração de características, é aplicado uma CNN que abstrai uma imagem de entrada e gera um mapa de características. Essas características são usadas para estimar o caráter em cada campo receptivo. No artigo, foram estudadas três arquiteturas de VGG, RCNN e ResNet. A VGG em sua forma original consiste em múltiplas camadas convolucionais seguidas por algumas camadas totalmente conectadas. A RCNN é uma variante de CNN que pode ser aplicada recursivamente para ajustar seus campos receptivos dependendo das formas dos caracteres. Por último, a ResNet é uma CNN com conexões residuais que facilita o treinamento de CNNs relativamente mais profundas. (BAEK et. al, 2019)

Para a modelagem sequencial, as características extraídas do estágio anterior são remodeladas para serem uma sequência de características. No entanto, essa sequência pode sofrer com a falta de informações contextuais (BAEK et. al, 2019). Dessa forma, (BAEK et. al, 2019), para a modelagem sequencial permite a seleção ou não seleção do uso de LSTM bidirecional, dado pela sigla BiLSTM, que é uma proposta existente na literatura anterior.

No último, o estágio de predição, temos a predição de uma sequência de caracteres, em que (BAEK et. al, 2019), ao resumir trabalhos anteriores, propôs duas opções para predição: classificação temporal conexionista, dada pela sigla CTC e predição de sequência baseada em atenção. O CTC permite a previsão de um número não fixo de uma sequência, mesmo que um número fixo de recursos seja fornecido. Por outro lado, a predição de sequência baseada em atenção captura automaticamente o fluxo de informações dentro da sequência de entrada para prever a sequência de saída, ele permite que um modelo STR

aprenda um modelo de linguagem de nível de caractere que representa dependências de classe de saída. (BAEK et. al, 2019)

Por fim, no trabalho de (BAEK et. al, 2019) ao realizar diferentes combinações, das possibilidades destes quatro estágios, a combinação que obteve melhores resultados foi: TPS (para o estágio de transformação), ResNet (para extração de características), BiLSTM (para o estágio de modelagem sequencial) e Atenção (para o estágio de predição). Desta maneira, esta combinação foi a utilizada neste trabalho.

3. DESENVOLVIMENTO

3.1 Pipeline para o Reconhecimento de Placas de Veículos

Após o desenvolvimento, a *pipeline* permitirá o monitoramento dos veículos, através das seguintes etapas, conforme mostrado na figura 23.

- Aplicar o modelo de detecção de placas para identificar as regiões das placas dos veículos;
- Recortar a imagem nas regiões das caixas delimitadoras, que são retornadas pela detecção de objetos;
- Redimensionar a imagem para 100 de largura e 30 de altura;
- Aplicar o modelo de OCR na imagem recortada para extrair o conteúdo textual.

Figura 23 - Pipeline de Reconhecimento de Placas de Veículos.

Fonte: De própria autoria.

3.2. Conjunto de Dados

Para ambos os modelos da pipeline, o modelo de detecção de placas e o modelo para o reconhecimento dos caracteres, o conjunto de dados utilizado para o treinamento dos modelos foi o UFPR-ALPR, do Laboratório de Visão, Robótica e Imagem da Universidade Federal do Paraná, disponibilizado publicamente para objetivos de pesquisa. (LAROCA et al., 2018)

Este conjunto é composto por um total de 4500 imagens, para o qual foram divididos em 150 *tracks*, com cada *track* com 30 imagens, divididos da seguinte maneira: 60 *tracks* para o treinamento, 30 *tracks* para validação e 60 *tracks* para o teste. Cada *tack* corresponde a uma sequência de frames de um vídeo de um mesmo veículo.

Na estrutura deste conjunto de dados, para cada imagem temos um arquivo .txt com as seguintes informações, conforme vemos na figura 24: camera (câmera utilizada para a

captura), position_vehicle (posição do veículo), type (tipo do veículo), make (fabricante), model (modelo do veículo), year (ano do modelo do veículo), plate (placa do veículo), corners (posição da placa do veículo), char 1 (posição do primeiro caractere da placa do veículo), char 2 (posição do segundo caractere da placa do veículo), char 3 (posição do terceiro caractere da placa do veículo), char 4 (posição do quarto caractere da placa do veículo), char 5 (posição do quinto caractere da placa do veículo), char 6 (posição do sexto caractere da placa do veículo) e char 7 (posição do sétimo caractere da placa do veículo).

Figura 24 - Exemplo de um arquivo .txt do conjunto de dados.

```
camera: GoPro Hero4 Silver
position_vehicle: 808 311 301 277
    type: car
    make: Suzuki
    model: Grand Vitara
    year: 2013
plate: AXV8804
corners: 912,526 986,531 983,553 912,550
    char 1: 915 534 8 14
    char 2: 923 534 9 14
    char 3: 932 535 9 13
    char 4: 945 536 9 13
    char 5: 954 536 8 13
    char 6: 963 537 8 12
    char 7: 971 537 8 13
```

Fonte: De própria autoria.

Na aplicação deste conjunto de dados para este trabalho optamos pela divisão do conjunto de dados da seguinte maneira: 80% dos *tracks* para os treinamentos e 20% para o teste, que corresponde a 120 *tracks* (3600 imagens) para o treinamento e 30 *tracks* (900 imagens) para o teste.

Com o objetivo de facilitar o processamento para a construção dos rótulos de cada um dos modelos (pois cada um dos modelos precisa que as informações dos rótulos estejam em um formato específico), convertemos os arquivos .txt de cada imagem para um arquivo .json como vemos na figura 25.

Figura 25 - Exemplo de um arquivo .json do conjunto de dados.

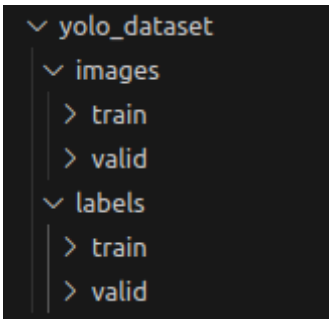
Fonte: De própria autoria.

3.3. Detecção de Placas

3.3.1. Processamento do Conjunto de Dados

O modelo YOLO requer que conjunto de dados seja estruturado de uma forma bem definida para que possa ser passado para o modelo, conforme a figura 26. Onde você possui dois diretórios, um chamado *images*, que armazena as imagens, e outra chamada de *labels*, que armazena os rótulos das imagens. Para cada uma dois diretório, temos duas pastas, uma chamada *train*, que armazena as imagens/rótulos de treinamento, e a outra chamada *valid*, que armazena as imagens/rótulos de validação.

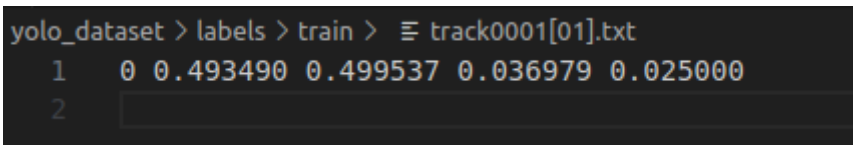
Figura 26 - Organização do conjunto de dados.



Fonte: De própria autoria.

Além das imagens, uma informação fundamental para o treinamento do modelo são os rótulos das imagens. No conjunto de dados do modelo YOLO, os rótulos devem ser armazenados em um arquivo com o mesmo nome da imagem correspondente salvo em formato .txt. Dentro do arquivo .txt do rótulo temos que em cada linha representa um objeto na imagem com cinco informações: identificador da classe (representado por um número inteiro), posição x e y do centro da caixa delimitadora e largura e altura da caixa delimitadora. Todas essas informações, com exceção do identificador da classe, devem ser normalizadas em relação ao tamanho total da imagem, conforme podemos ver na figura 27.

Figura 27 - Exemplo de um arquivo .txt para os rótulos de uma imagem.



Fonte: De própria autoria.

Assim, a partir dos arquivos .json gerados na seção 3.2, passamos por uma etapa de pré-processamento dessas informações para gerar os arquivos dos rótulos do modelo como se segue:

1º Percorrer cada um dos arquivos.json;

2º Acessar o campo *corners*, que armazena as informações das coordenadas de localização da placa do veículo;

3º Conversão das coordenadas da placa de veículo que estão no formato de quatro pontos (x,y), que formam um retângulo, em formato YOLO.

4º Salvar as informações em formato. txt da seguinte maneira: **0 x_centro y_centro largura altura**, onde 0 é o identificador da classe de placa de veículos.

3.3.2. Treinamento

Para o treinamento do modelo YOLO11m, foi utilizado a linguagem Python e a biblioteca Ultralytics, na plataforma Google Colab, em um ambiente de execução com uma GPU A100. O arquivo args.yaml que contém os parâmetros utilizados para o modelo está disponível no apêndice A, com esse arquivo é possível carregar um modelo com os mesmos parâmetros que foram utilizados neste trabalho.

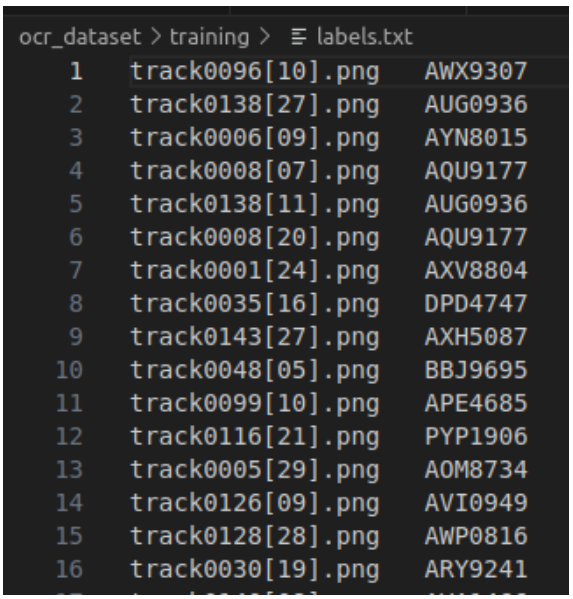
3.4. Reconhecimento de Caracteres

3.4.2. Processamento do Conjunto de Dados

No conjunto de dados para o modelo de reconhecimento de caracteres, a partir da informação contida no campo *cornes* dos arquivos .json, foram extraídos os recortes nas imagens da placa dos veículos. Após a extração dos recortes todas as imagens foram redimensionadas para o tamanho 100 de largura e 30 de altura.

Além do redimensionando das imagens para padronizá-las em um mesmo tamanho, para o modelo de OCR selecionado também é necessário inserir em cada um dos diretórios de imagens, do treinamento e do teste, um arquivo .txt em que cada linha representa uma das imagens do diretório e em cada linha temos duas informações separadas por espaço o nome do arquivo e o texto contido na imagem, conforme a figura 29.

Figura 29 - Arquivo .txt para o treinamento do OCR.



Fonte: De própria autoria.

3.4.2. Treinamento

Para o treinamento do modelo de reconhecimento de caracteres, foi utilizado a plataforma Google Colab em um ambiente de execução com uma GPU T4, seguindo as instruções no repositório do Github dos autores do artigo[1].

[1] Link para o repositório do Github: https://github.com/clovaai/deep-text-recognition-benchmark

4. RESULTADOS

4.1. Detecção de Placas

4.1.1. Métricas de Avaliação

A interseção sobre união (IoU) é uma métrica comum para avaliação de desempenho de algoritmos de detecção de objetos. Ela mede a sobreposição entre a caixa delimitadora prevista pelo algoritmo e a caixa delimitadora real para aquela imagem. Assim, a IoU é calculada como a área de intersecção dividida pela área de união das duas caixas, conforme a equação 4.1. Ao final, a pontuação da IoU varia de 0 a 1, onde 0 indica nenhuma sobreposição e 1 indica uma combinação perfeita. (NUHL, 2024)

$$IoU = \frac{\text{área da caixa prevista} \cap \text{área caixa real}}{\text{área caixa prevista} \cup \text{área caixa real}} \tag{4.1}$$

A Precisão Média (AP) é outra métrica importante usada na detecção de objetos. Ele resume a curva de precisão-recall que é criada pela variação do limiar de detecção, conforme a equação 4.2. A precisão é a proporção de detecções positivas verdadeiras entre todas as detecções positivas, dada pela equação 4.3, enquanto o recall é a proporção de detecções positivas verdadeiras entre todos os positivos reais na imagem, dada pela equação 4.4. (NUHL, 2024)

$$AP = \text{área sob a curva de precisão e recall} \tag{4.2}$$

$$\text{precisão} = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Positivos} \tag{4.3}$$

$$recall = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Negativos} \tag{4.4}$$

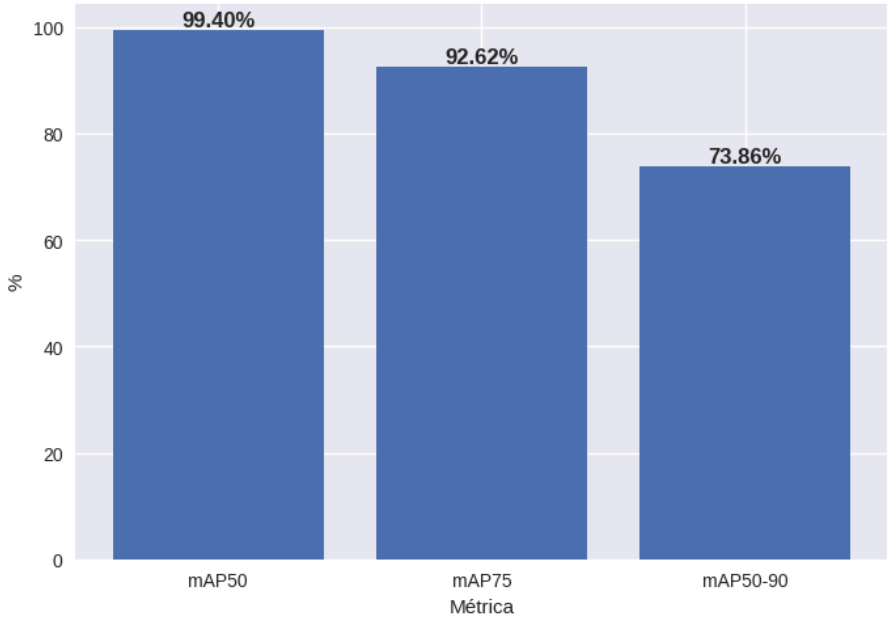
Derivada do AP, como nos algoritmos de detecção de objetos podemos ter múltiplas classes, temos o mAP que é a média do AP de cada classe. Neste trabalho, por se tratar de uma única classe de objetos, AP = mAP.

Além disso, o modelo YOLO retorna as métricas de mAP considerando alguns limiares de IoU, sendo estes mAP50 (para limiar de IoU de 50%), mAP75 (para limiar de IoU de 75%) e mAP50-90 (que são para limiares de IoU em uma variação de 50% a 90%).

4.1.2. Resultados do Treinamento

No resultado final do treinamento, o modelo que foi treinado em 50 épocas, com lote de 32 e com os outros parâmetros *default*. Assim, as métricas atingidas pelo modelo podem ser vistas na figura 30.

Figura 30 - Métricas mAP do Modelo de Detecção das Placas dos Veículos.



Fonte: De própria autoria.

Além das métricas acima, para a precisão e recall o modelo atingiu 98.10% e 98.40% respectivamente.

4.2. Reconhecimento Óptico de Caracteres

4.2.1. Métricas de Avaliação

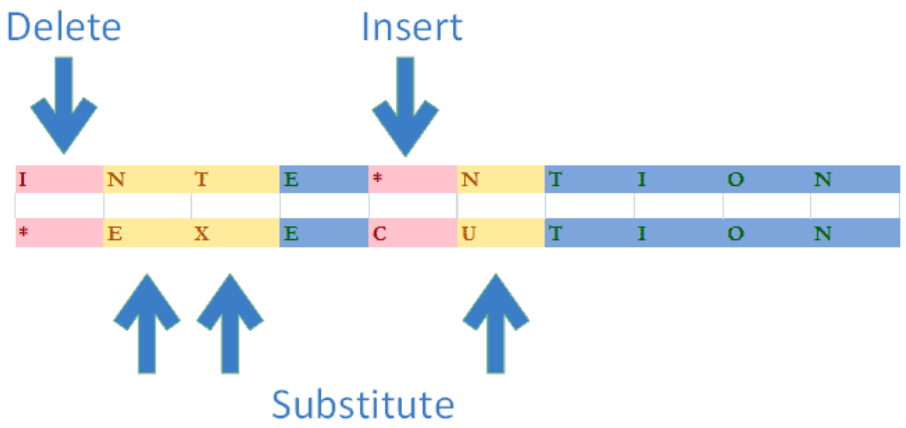
Accuracy é uma métrica que mede a relação entre as predições corretas do modelo sobre o total de predições, conforme a equação 4.5. Assim, aplicamos esta métrica neste trabalho com o intuito de obter a relação de quantos placas o modelo acertou em todos os caracteres.

$$Accuracy = \frac{Predições bcorretas}{Totalde\ predições} \quad (4.5)$$

Outra métrica relevante para o reconhecimento de caracteres é a distância de levenshtein é uma métrica de distância aplicada entre duas palavras, de forma a obter o menor número de operações de edição necessárias para transformar uma palavra em outra. Nesta métrica, as operações de edição incluem inserções, exclusões e substituições. (GRASHCHENKO, 2024)

De maneira prática, podemos exemplificar a distância de levenshtein, considere duas palavras, *Intention* e *Execution*. Para transformar *Intention* na palavra *Execution* são necessárias 5 operações, como mostrado na figura 31. Assim, a distância de levenshtein entre *Intention* e *Execution* é igual a 5. (GRASHCHENKO, 2024)

Figura 31 - Exemplo da Distância de Levenshtein.



Fonte: (GRASHCHENKO, 2024).

Para a forma normalizada é realizada com a divisão da distância pelo número de caracteres da maior palavra, dada pela equação 4.6.

$$normED = 1 - \frac{ED(palavra01, palavra02)}{max(Tamanho(palavra01), Tamanho(palavra02))} \quad (4.6)$$

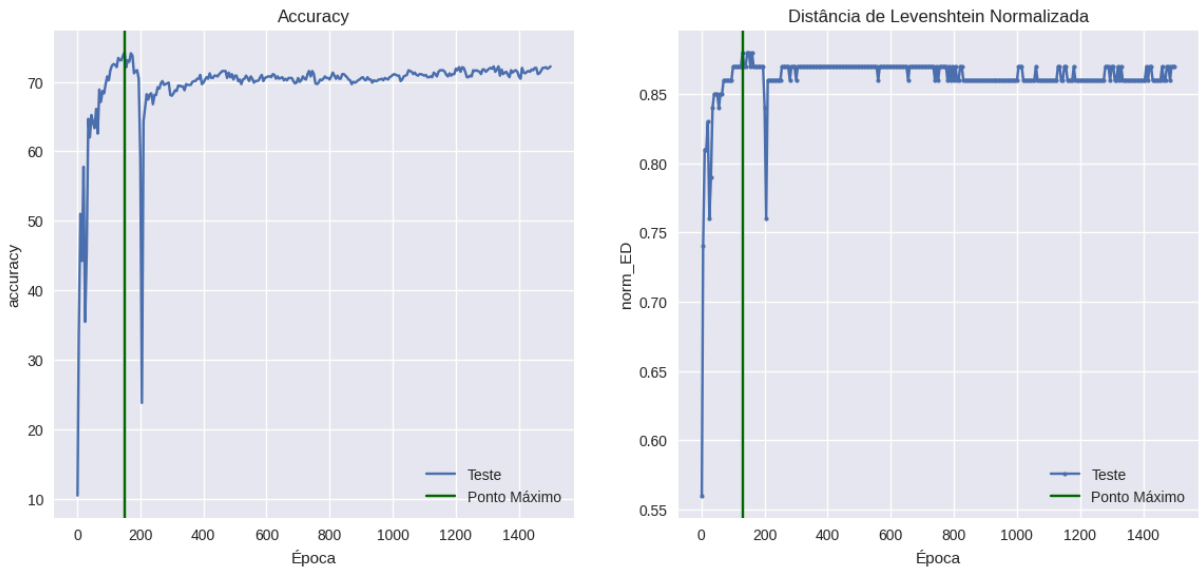
No contexto deste trabalho, a distância de levenshtein normalizada é usada para calcular a diferença de caracteres das placas geradas pelo modelo de OCR com os caracteres

reais, desta maneira, quanto mais próximo de 1, maior o número de caracteres corretos previstos.

4.2.2. Resultados do Treinamento

No resultado final do treinamento, o modelo que foi treinado em 1500 épocas, com lote de 32 e com os outros parâmetros *default*, atingiu as seguintes métricas, conforme a figura 32.

Figura 32 - Evolução das Métricas *Accuracy* e Distância de Levenshtein do Conjunto de Teste.



Fonte: De própria autoria.

Embora o modelo tenha sido treinado ao longo de 1500 épocas, ele atingiu os valores máximos de ambas as métricas antes da época 200, 74,22% para o *accuracy* e 88% para a distância de Levenshtein normalizada, o modelo treinado conta com um *checkpoint* para salvar o modelo sempre que obtém o melhor resultado, com um modelo sendo para cada uma das métricas. Isto é um fato importante, pois a estabilização dos valores das métricas neste mesmo patamar ao longo das épocas poderia indicar uma possibilidade de *overfitting* do modelo, porém checkpoint foi atingido em um momento antes que isso pudesse acontecer.

5. CONCLUSÃO

Dado que o objetivo deste trabalho foi propor uma pipeline de identificação de veículos para a Universidade Estadual do Sudoeste da Bahia, por meio de aplicação de técnicas de visão computacional identificar as placas dos veículos. Neste sentido, as conclusões deste trabalho são baseadas em dois pontos principais:

- A capacidade do modelo YOLO 11 para detecção de objetos identificar e localizar as placas dos veículos; e
- A capacidade do modelo de reconhecimento de caracteres identificar o conteúdo textual das placas.

Sobre o modelo de detecção de objetos, ao observar os resultados de suas métricas, vemos que para um limiar de IoU de até 75% o modelo obteve um mAP acima de 90% o que demonstra um ótimo resultado, porém para o mAP50-90 mostra que o modelo teve algumas dificuldades para detecções mais difíceis, embora ainda se mantenha com um bom resultado. Estas dificuldades podem ter como explicação alguns fatores, como distância do carro para a câmera de captura, ângulo da captura e problemas de iluminação, como sombras, brilho e reflexão, que podem trazer dificuldades para a detecção da placa.

Ao analisar os resultados obtidos pelo modelo, vemos que ele atingiu, no melhor resultado, 74,22% de *accuracy* e 88% para a distância de Levenshtein normalizada, podemos concluir que o modelo obteve uma boa capacidade de generalização, pois embora o modelo tenha atingido 74,22% de *accuracy*, o que representa um resultado regular, o fato de ter atingido 88% para a distância de Levenshtein significa que o modelo está acertando grande parte dos caracteres, o que significa que os erros que trazem impacto ao *accuracy* são de poucos caracteres.

Desta forma podemos concluir que a pipeline de reconhecimento de caracteres proposta obteve uma boa capacidade de reconhecimento e identificação de placas veiculares para a maioria das situações, com a ressalva de que em uma possível implementação em um sistema real, além da expansão do conjunto de dados para suportar o modelo de placas do MERCOSUL, alguns pontos em relação ao posicionamento das câmeras sejam vistos, como: altura, ângulo de captura e a iluminação.

REFERÊNCIAS

AUGUSTO, Abraham. **Redes Neurais Artificiais: Uma Introdução Prática**. 2014. Apresentação de Slides em PDF. Disponível em: <https://www.passeidireto.com/arquivo/68347211/ica-2014-intro-redes-neurais>. Acesso em: 2 de mar. de 2025.

BAEK, Jeonghun; et. al. What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis. **arXiv**. 2019. Disponível em: <https://arxiv.org/pdf/1904.01906>. Acesso em: 28 de fev. de 2025.

CASTILHO, Rubens. O que é sinapse, quais os tipos e como ocorrem. **Enciclopédia Significados.** 2025. Disponível em: <https://www.significados.com.br/sinapse/#:~:text=A%20sinapse%20%C3%A9%20a%20regi%C3%A3o,alguma%20a%C3%A7%C3%A3o%20espec%C3%ADfica%20no%20corpo.>. Acesso em: 15 de dez. de 2023.

CONCI,Aura; AZEVEDO, Eduardo; LETA, Fabiana R. **Computação Gráfica: teoria e prática**. 2ª Reimpressão. Rio de Janeiro: Elseivier. 2008. v.2.

GRASHCHENKO,Sergey. Levenshtein Distance Computation. **Baeldung**. 2024. Disponível em:<https://www.baeldung.com/cs/levenshtein-distance-computation>. Acesso em 27 de fev. de 2025.

HAYKIN, Simom. **Redes Neurais: princípios e prática**. 2ª edição. Porto Alegre: Bookman. 2001.

JUNIOR, Odemir D. CRISP-DM: é útil mesmo?. **DataV**. 2023. Disponível em: <https://www.dataviking.com.br/post/crisp-dm-e-util-mesmo#:~:text=O%20CRISP%2DDM%20(Cross%2D,uma%20estrutura%20clara%20e%20flex%C3%ADvel.>. Acesso em: 2 de abr. de 2025.

JUNIOR, Pedro. CSI em São José dos Campos: saiba como a tecnologia aumentou a segurança. **i9vale**. 2023. Disponível em: <https://blog.i9vale.com.br/csi-em-sao-jose-dos-campos-saiba-como-a-tecnologia-aumentou-a-seguranca/>. Acesso em: 2 de abr. de 2025.

LAROCA, Rayson; et. al. A robust real-time automatic license plate recognition based on the YOLO detector. In: International Joint Conference on Neural Networks (IJCNN), 2018, Rio de Janeiro. Anais eletrônicos [...]. Rio de Janeiro: **IEEE Xplore**, 2018. p. 1-10. Disponível em: <https://ieeexplore.ieee.org/document/8489629/authors#authors>. Acesso em: 3 de mar. de 2025.

LAROCA, Rayson; et. al. A First Look at Dataset Bias in License Plate Recognition. In: SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), 2022, Natal. Anais eletrônicos [...]. Natal: **IEEE Xplore**, 2018. p. 234-239. Disponível em: <https://ieeexplore.ieee.org/document/9991768>. Acesso em: 3 de mar. de 2025.

LAROCA, Rayson; et. al. Do We Train on Test Data? The Impact of Near-Duplicates on License Plate Recognition 2023 International Joint Conference on Neural Networks (IJCNN),

2023, Gold Coast. Anais eletrônicos [...]. Austrália: **IEEE Xplore**, 2023a. p. 1-8. Disponível em: <10.1109/IJCNN54540.2023.10191584>. Acesso em: 3 de mar. de 2025

LAROCA, R.; et al. Leveraging model fusion for improved license plate recognition. In: VASCONCELOS, V.; DOMINGUES, I.; PAREDES, S. (org.). Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. **CIARP 2023**. Coimbra: Springer, 2023b. p. 60-75. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-031-49249-5_5>. Acesso em: 3 de mar. de 2025.

MURALHA digital: prefeitura informa vias com radares em funcionamento a partir desta quarta-feira 1 de janeiro. **Prefeitura Municipal de Vitória da Conquista - PMVC**. 2025. Disponível em: <https://www.pmvc.ba.gov.br/muralha-digital-prefeitura-informa-vias-com-radares-em-funcionamento-a-partir-desta-quarta-feira-1o-de-janeiro/>. Acesso em: 2 de abr. de 2025.

NUHL, Nikolaj. YOLO Object Detection Explained: Evolution, Algorithm, and Applications. **Encord**. 2024. Disponível em: <https://encord.com/blog/yolo-object-detection-guide/>. Acesso em 25 de fev. de 2025.

OZGUR, Atilla. What is the difference between a neural network and a deep neural network, and why do the deep ones work better?. **Stack Exchange**. 2018. Disponível em: <https://stats.stackexchange.com/questions/182734/what-is-the-difference-between-a-neural-network-and-a-deep-neural-network-and-w>. Acesso em: 08 de dez. de 2023.

RAO, Nikhileswara. YOLOv11 Architecture Explained: Next-Level Object Detection with Enhanced Speed and Accuracy. **Medium**. 2024. Disponível em: <https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71>. Acesso em: 28 de fev. de 2025.

REDMON, Joseph; *et. al*. You Only Look Once: Unified, Real-Time Object Detection. **arXiv**. 2016. Disponível em: <https://arxiv.org/pdf/1506.02640>. Acesso em: 1 de mar. de 2025.

SZELISKI, Richard. **Computer Vision: Algorithms and Applications**. 2ª edição. Springer. 2022.

VADAPALLI, Pavana. Biological Neural Network: Importance, Components & Comparison. **upGrad**. 2021. Disponível em: <https://www.upgrad.com/blog/biological-neural-network/>. Acesso em: 07 de dez. de 2023.

ZHERZDEV, Sergey; GRUZDEV, Alexey.LPRNet: License Plate Recognition via Deep Neural Networks. **arXiv**. 2018. Disponível em: <https://arxiv.org/pdf/1806.10447v1>. Acesso em: 12 de abr. de 2025.

APÊNDICES

APÊNDICE A - Arquivo args.yaml com os parâmetros de treinamento do modelo de detecção de placa de veículos.

task: detect
mode: train
model: yolo11m.pt
data: data.yaml
epochs: 50
time: null
patience: 100
batch: 32
imgsz: 640
save: true
save_period: -1
cache: false
device: 0
workers: 8
project: null
name: train
exist_ok: false
pretrained: true
optimizer: auto
verbose: true
seed: 0
deterministic: true
single_cls: false
rect: false
cos_lr: false
close_mosaic: 10
resume: false
amp: true
fraction: 1.0

profile: false
freeze: null
multi_scale: false
overlap_mask: true
mask_ratio: 4
dropout: 0.0
val: true
split: val
save_json: false
save_hybrid: false
conf: null
iou: 0.7
max_det: 300
half: false
dnn: false
plots: true
source: null
vid_stride: 1
stream_buffer: false
visualize: false
augment: false
agnostic_nms: false
classes: null
retina_masks: false
embed: null
show: false
save_frames: false
save_txt: false
save_conf: false
save_crop: false
show_labels: true
show_conf: true
show_boxes: true
line_width: null

format: torchscript
keras: false
optimize: false
int8: false
dynamic: false
simplify: true
opset: null
workspace: null
nms: false
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 7.5
cls: 0.5
dfl: 1.5
pose: 12.0
kobj: 1.0
nbs: 64
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
bgr: 0.0
mosaic: 1.0

mixup: 0.0
copy_paste: 0.0
copy_paste_mode: flip
auto_augment: randaugment
erasing: 0.4
crop_fraction: 1.0
cfg: null
tracker: botsort.yaml
save_dir: runs/detect/train

APÊNDICE B - Arquivo opt.txt com os parâmetros de treinamento do modelo de reconhecimento de caracteres.

---------- Options ----------
exp_name: TPS-ResNet-BiLSTM-Attn-Seed1111
train_data: ./ocr_lmdb_dataset/training
valid_data: ./ocr_lmdb_dataset/validation
manualSeed: 1111
workers: 0
batch_size: 32
num_iter: 1500
valInterval: 5
saved_model: TPS-ResNet-BiLSTM-Attn.pth
FT: False
adam: False
lr: 1
beta1: 0.9
rho: 0.95
eps: 1e-08
grad_clip: 5
baiduCTC: False
select_data: ['/']
batch_ratio: ['1.0']
total_data_usage_ratio: 1.0
batch_max_length: 7
imgH: 32
imgW: 100
rgb: False
character: 0123456789abcdefghijklmnopqrstuvwxyz
sensitive: False
PAD: False
data_filtering_off: True
Transformation: TPS
FeatureExtraction: ResNet

SequenceModeling: BiLSTM
Prediction: Attn
num_fiducial: 20
input_channel: 1
output_channel: 512
hidden_size: 256
num_gpu: 1
num_class: 38

---------- Options -------------
exp_name: TPS-ResNet-BiLSTM-Attn-Seed1111
train_data: ./ocr_lmdb_dataset/training
valid_data: ./ocr_lmdb_dataset/validation
manualSeed: 1111
workers: 0
batch_size: 64
num_iter: 1500
valInterval: 5
saved_model: TPS-ResNet-BiLSTM-Attn.pth
FT: False
adam: False
lr: 1
beta1: 0.9
rho: 0.95
eps: 1e-08
grad_clip: 5
baiduCTC: False
select_data: ['/']
batch_ratio: ['1.0']
total_data_usage_ratio: 1.0
batch_max_length: 7
imgH: 32
imgW: 100
rgb: False

character: 0123456789abcdefghijklmnopqrstuvwxyz
sensitive: False
PAD: False
data_filtering_off: True
Transformation: TPS
FeatureExtraction: ResNet
SequenceModeling: BiLSTM
Prediction: Attn
num_fiducial: 20
input_channel: 1
output_channel: 512
hidden_size: 256
num_gpu: 1
num_class: 38

---------- Options -------------
exp_name: TPS-ResNet-BiLSTM-Attn-Seed1111
train_data: ./ocr_lmdb_dataset/training
valid_data: ./ocr_lmdb_dataset/validation
manualSeed: 1111
workers: 0
batch_size: 48
num_iter: 1500
valInterval: 5
saved_model: TPS-ResNet-BiLSTM-Attn.pth
FT: False
adam: False
lr: 1
beta1: 0.9
rho: 0.95
eps: 1e-08
grad_clip: 5
baiduCTC: False
select_data: ['/']

batch_ratio: ['1.0']
total_data_usage_ratio: 1.0
batch_max_length: 7
imgH: 32
imgW: 100
rgb: False
character: 0123456789abcdefghijklmnopqrstuvwxyz
sensitive: False
PAD: False
data_filtering_off: True
Transformation: TPS
FeatureExtraction: ResNet
SequenceModeling: BiLSTM
Prediction: Attn
num_fiducial: 20
input_channel: 1
output_channel: 512
hidden_size: 256
num_gpu: 1
num_class: 38

---------- Options ----------
exp_name: TPS-ResNet-BiLSTM-Attn-Seed1111
train_data: ./ocr_lmdb_dataset/training
valid_data: ./ocr_lmdb_dataset/validation
manualSeed: 1111
workers: 0
batch_size: 32
num_iter: 1500
valInterval: 5
saved_model: TPS-ResNet-BiLSTM-Attn.pth
FT: False
adam: False
lr: 1

beta1: 0.9
rho: 0.95
eps: 1e-08
grad_clip: 5
baiduCTC: False
select_data: ['/']
batch_ratio: ['1.0']
total_data_usage_ratio: 1.0
batch_max_length: 7
imgH: 32
imgW: 100
rgb: False
character: 0123456789abcdefghijklmnopqrstuvwxyz
sensitive: False
PAD: False
data_filtering_off: True
Transformation: TPS
FeatureExtraction: ResNet
SequenceModeling: BiLSTM
Prediction: Attn
num_fiducial: 20
input_channel: 1
output_channel: 512
hidden_size: 256
num_gpu: 1
num_class: 38
