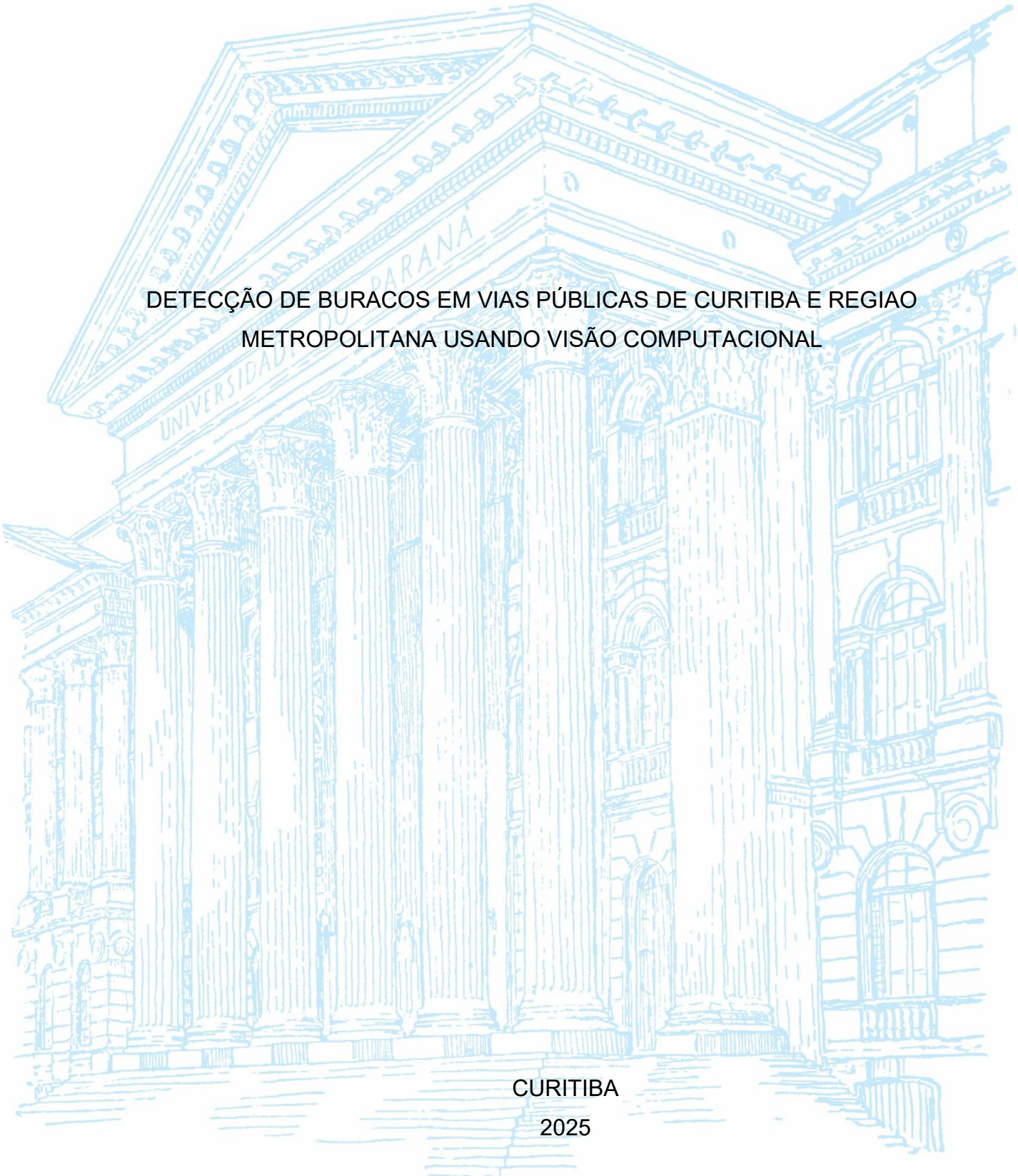
UNIVERSIDADE FEDERAL DO PARANÁ

PIERO SILVA MORAIS



DETECÇÃO DE BURACOS EM VIAS PÚBLICAS DE CURITIBA E REGIAO METROPOLITANA USANDO VISÃO COMPUTACIONAL

CURITIBA

2025

PIERO SILVA MORAIS

Detecção de buracos em vias públicas de Curitiba e região metropolitana usando visão computacional.

TCC apresentado ao curso de Graduação em Engenharia Elétrica, Setor de Tecnologia, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Bacharel em Engenharia Eletrica.

Orientador: Prof. Dr. Eduardo Parente Ribeiro

CURITIBA

2025

AGRADECIMENTOS

Aos alicerces da minha vida, minha família, cujo amor incondicional foi meu porto seguro nos dias de tempestade e minha celebração nos momentos de luz. Vocês me ensinaram que o afeto é a única moeda que nunca perde valor.

À minha esposa, Laís, companheira de todas as horas: esses treze anos ao seu lado me mostraram que o amor não se mede em tempo, mas em gestos silenciosos de apoio. Cada conquista minha carrega suas mãos que me empurram para frente – e nosso filho Vicente é a prova viva disso.

Falando em você, meu pequeno Vicente: seu sorriso desdentado ilumina meus dias como nenhuma estrela consegue. Você me deu um tipo de força que eu nem sabia existir, a força de quem olha para o futuro e vê infinitas possibilidades.

Aos meus pais, Isete e Carlos Morais, meus primeiros heróis:

- Mãe, sua coragem diante das adversidades me ensinou que fé e resiliência são armas invencíveis.
- Pai, seu caráter inquebrantável e sua ética de trabalho moldaram quem eu sou hoje.

Minhas avós, Anilda e Maria Augusta, tecedoras da minha história: o amor com que me embalaram na infância ecoa em cada escolha que faço. Seu legado é o fio dourado que costura minha jornada.

Ao professor Eduardo Parente Ribeiro, meu orientador, agradeço pela paciência de mestre e pela sabedoria em transformar obstáculos em degraus. Sua orientação foi farol em dias de mar revolto.

E aos meus amigos, cúmplices da vida real: obrigado pelas risadas que desarmam o mundo, pelos "pecadinhos" que adoçam a existência e pelas conversas sem pé nem cabeça que, no fim, sempre fazem sentido.

Esta conquista não é minha – é **nossa**.

"Ensinar é despertar mentes inquietas e nutrir uma paixão pela verdade, enquanto o pensamento crítico ilumina os caminhos para a justiça e a compaixão." – Cornel West

RESUMO

Este trabalho explora o uso de visão computacional para a detecção automática de buracos em vias públicas na região de Curitiba, Paraná, empregando a arquitetura YOLO (*You Only Look Once*). Diferentemente de abordagens que utilizam grandes conjuntos de dados, o estudo concentra-se na ideia de *good data*, priorizando a qualidade sobre a quantidade. Para isso, foram aplicadas técnicas de aumento de dados (*data augmentation*), visando melhorar a diversidade e a generalização do modelo. A pesquisa enfatiza estratégias para a obtenção de imagens representativas e bem rotuladas, essenciais para treinar o modelo de forma eficiente e mitigar problemas como *overfitting*. Os resultados indicam que, mesmo com um conjunto reduzido de imagens, é possível alcançar alta precisão na identificação de buracos, contribuindo para soluções automatizadas de manutenção viária.

Palavras-chave: Detecção Automática de Buracos; Arquitetura YOLO; Visão Computacional; Aumento de Dados; *Darknet Framework*.

ABSTRACT

This study explores the use of computer vision for the automatic detection of potholes on public roads in Curitiba, Paraná, employing the YOLO (You Only Look Once) architecture. Unlike approaches that rely on large datasets, this research focuses on the concept of *good data*, prioritizing quality over quantity. To achieve this, data augmentation techniques were applied to enhance the diversity and generalization of the model. The research emphasizes strategies for obtaining representative and well-labelled images, which are essential for efficient model training and for mitigating problems like *overfitting*. The results indicate that, even with a reduced set of images, it is possible to achieve high accuracy in pothole identification, contributing to automated road maintenance solutions.

Keywords: Pothole Detection; YOLO architecture; Computer Vision; Data Augmentation; Darknet Framework.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS OU SIGLAS

YOLO	You Only Look Once
CNN	Rede Neural Convolucional (Convolutional Neural Network)
mAP	Mean Average Precision
IoU	Intersection over Union (sobreposição de caixas)
ReLU	Rectified Linear Unit (função de ativação)
Darknet	Framework para treinamento de YOLO

LISTA DE SÍMBOLOS

∇	Operador gradiente (derivada direcional)
σ	Desvio padrão (filtro Gaussiano)
θ	Ângulo de rotação (em radianos)
$\sum$	Somatório
$\prod$	Produtório
$\Phi(\cdot)$	Função de ativação em redes neurais
W	Vetor/matriz de pesos (redes neurais)
X	Vetor de entrada (redes neurais)
$\hat{y}$	Valor estimado (saída do modelo)
TP	Verdadeiro Positivo
FP	Falso Positivo
FN	Falso Negativo
TN	Verdadeiro Negativo

SUMÁRIO

1 INTRODUÇÃO

A utilização de visão computacional para detectar diversos tipos de objetos tem se popularizado na última década devido ao avanço nos algoritmos de aprendizado de máquina (*machine learning* – ML) e o desenvolvimento tecnológico, especificamente o progresso feito com placas gráficas capazes de efetuar cálculos vetoriais com velocidade e eficiência. Aplicação dessas técnicas permite a automatização de processos e a melhoria da vida na cidade, particularmente, esse trabalho visa utilizar a visão computacional para detectar buracos em pavimentos na cidade de Curitiba e região metropolitana.

1.1 JUSTIFICATIVA

Esse trabalho se justifica devido à necessidade de melhorar a via terrestre na região metropolitana de Curitiba. A melhoria de vida do cidadão e uma motivação que norteia a formação de engenharia, em particular, no aluno da universidade pública.

1.2 OBJETIVOS

1.2.1 Objetivo geral

Criar um modelo usando redes convolucionais que possa identificar buracos em vias públicas em pavimentos de cidade.

1.2.2 Objetivos específicos

Treinar e analisar o desempenho de uma Rede Neural Convolucional (*Convolucional Neural Network* – CNN) na identificação de buracos em Curitiba e na região metropolitana de Curitiba com foco na criação de um banco de imagens da cidade.

Aplicar técnicas para aumentar a variabilidade do banco de dados original, como aumentação de dados (*data augmentation)* ou inclusão de novos cenários, e

analisar o impacto no modelo por meio de métricas quantitativas como precisão, *recall* etc.

Comparar a eficácia do modelo antes e depois da expansão do banco de dados, utilizando métricas quantitativas para verificar melhorias na detecção.

Fornecer de maneira aberta as imagens coletadas e rotuladas para possibilitar pesquisas futuras na área de visão computacional e aprimorar estudos sobre detecção de buracos.

1.3 METODOLOGIA

Esse trabalho se caracteriza como uma pesquisa experimental, com foco na análise e aprimoramento de um modelo de rede neural convolucional (CNN) para a identificação de buracos em vias urbanas, especificamente em Curitiba e região metropolitana. A pesquisa também envolve a coleta e disponibilização de dados para uso acadêmico.

A coleta de dados será realizada na região metropolitana de Curitiba, utilizando dispositivos como câmeras de celular acopladas em veículos. O processo inclui:

- **Rotulagem das imagens:** As imagens coletadas serão classificadas manualmente ou semiautomaticamente, identificando buracos e outros elementos relevantes.
- **Expansão do banco de dados:** Novas imagens serão incorporadas ao banco de dados original, aumentando sua abrangência e variabilidade.
    - Os dados coletados passarão por um processo de preparação para garantir sua qualidade e adequação ao treinamento do modelo. As etapas incluem:
- **Pré-processamento:** Ajustes de contraste, remoção de ruídos, redimensionamento das imagens e normalização.
- **Aumento de variabilidade:** Aplicação de técnicas de aumentação de dados, como rotação, alteração de brilho, adição de ruídos e mudanças de perspectiva, para enriquecer o banco de dados e melhorar a robustez do modelo.

O modelo de CNN será desenvolvido e treinado com base nos dados processados. As etapas incluem:

- **Definição da arquitetura:** Escolha da estrutura da CNN, incluindo número de camadas, funções de ativação e algoritmos de otimização.
- **Treinamento:** O modelo será treinado utilizando os dados originais e, posteriormente, os dados expandidos, com validação cruzada para evitar *overfitting*.
- **Ferramentas:** *DarkNet* será utilizado para a implementação e treinamento do modelo. *DarkMark* será utilizado para rotular imagens.

A eficácia do modelo será avaliada por meio de métricas quantitativas, como:

- **Precisão:** Proporção de identificações corretas entre todas as identificações feitas pelo modelo.
- ***Recall*:** Proporção de buracos corretamente identificados entre todos os buracos existentes nas imagens.
- ***Mean Average Precision* (mAP):** Avaliação baseada na média das precisões obtidas em diferentes níveis de recall, essencial para medir a qualidade da detecção de buracos.
- **Comparação:** O desempenho do modelo será comparado antes e depois da expansão do banco de dados, verificando-se se houve ou não melhorias na detecção.

As imagens coletadas e rotuladas serão disponibilizadas de forma aberta para a comunidade acadêmica, por meio do repositório online GitHub. O objetivo é fomentar pesquisas futuras na área de visão computacional e detecção de buracos.

2 REVISÃO DE LITERATURA

A detecção de buracos em vias públicas exige a revisão de conceitos fundamentais para a criação de um modelo capaz de generalizar a identificação dessas irregularidades na malha viária. Iniciamos com uma revisão sobre visão computacional, abordando a teoria básica de imagens digitais.

2.1 VISÃO COMPUTACIONAL

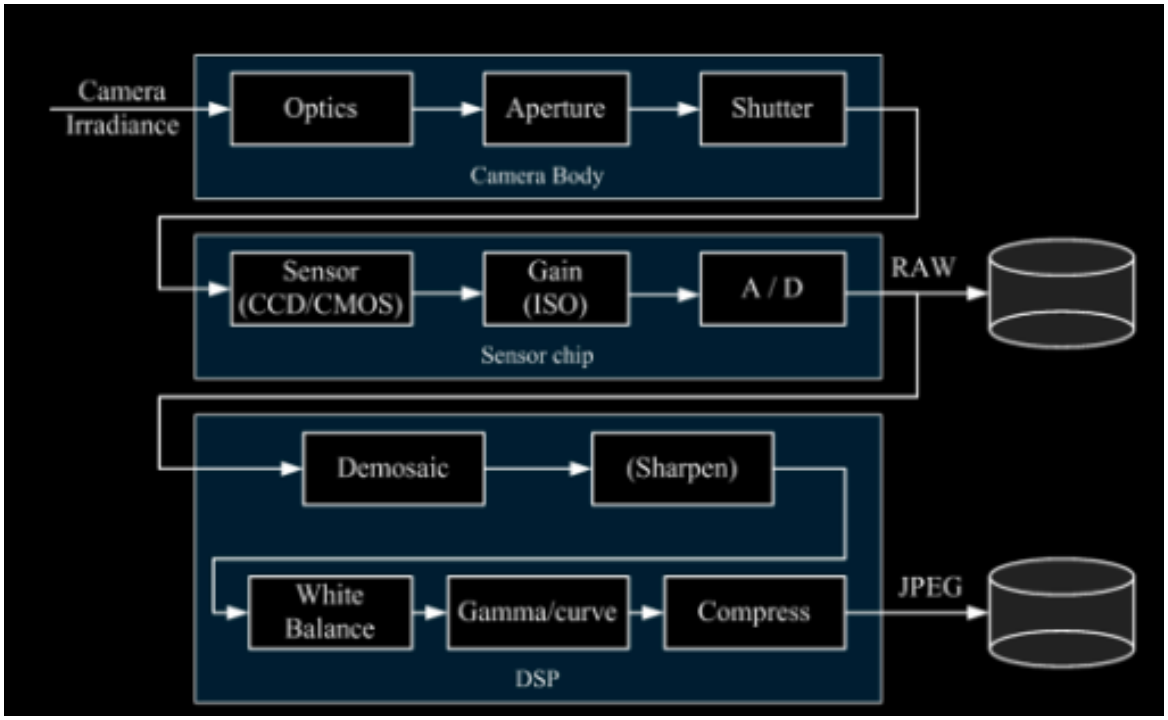
Segundo Szeliski (2011), a visão computacional busca reconstruir o mundo ao nosso redor, permitindo que computadores reconheçam padrões visuais da mesma forma que os seres humanos. Enquanto nós identificamos facilmente rostos em fotografias ou distinguimos imagens reais de geradas por computador com base em nossa experiência e contexto, os algoritmos de visão computacional precisam executar essas tarefas sem um conhecimento prévio do que é ou não real. Primeiro, analisa-se, portanto, a formação de uma imagem na câmera digital.

2.1.1 Imagem digital e câmera digital

Para que haja a formação de uma imagem, faz-se necessária uma fonte de luz, ou seja, o objeto de interesse deve estar devidamente iluminado para que os raios de luz possam refletir na sua superfície e chegar a uma câmera. Uma representação detalhada do processo como luz interage com os objetos e modelos matemáticos, podem ser encontrados em Szeliski (2011).

Segundo o mesmo autor, as câmeras digitais possuem um conjunto de dispositivos que permitem a formação de uma imagem digital. A luz, após passar por uma lente, incide sobre um sensor composto por várias áreas que podem armazenar uma certa intensidade de luz para cada pixel. O sinal gerado é então fornecido a um amplificador de sinal e depois para um conversor analógico-digital, onde se forma uma imagem preliminar. Após alguns processos de correção da imagem inicial, é gerado um arquivo de imagem, como um JPEG, *Joint Photographic Experts Group*, que pode ser compartilhado e aberto por visualizadores de imagem. O modelo simplificado pode ser observado abaixo na FIGURA 1.

FIGURA 1 - MODELO SIMPLIFICADO DE UMA CÂMERA DIGITAL (EM INGLÊS)



FONTE: Szeliski (2011).

Szeliski (2011) ainda fornece alguns conceitos sobre a formação de uma imagem digital que merecem ser comentados aqui.

- **Velocidade do obturador:** Controla o tempo de exposição da luz ao sensor da câmera, impactando diretamente a qualidade e a nitidez da imagem capturada.
- **Amostragem e *Aliasing***: Se dois sinais são amostrados na mesma frequência, pode-se perder informações essenciais, impossibilitando a reconstrução precisa do sinal original. O critério de Nyquist estabelece que a frequência de amostragem deve ser pelo menos duas vezes maior que a frequência máxima do sinal amostrado para evitar distorções.
- **Ruido do sensor:** Ao longo do processo de captura de imagem, diversos tipos de ruído são adicionados, como ruído de amplificação, ruído de quantização etc. A quantidade final de ruído em uma imagem depende dessas variáveis, além da luz incidente, tempo de exposição e ganho do sensor.

Mais detalhes podem ser encontrados em Szeliski (2011). Com uma definição de como se forma uma imagem digital, pode-se considerar outro aspecto importante

na criação de um conjunto de imagens representativas de buracos: o processamento de imagem.

2.1.2 Processamento de imagens

No capítulo 3, Szeliski (2011) detalha o processamento de imagens digitais. Nesta seção, focaremos nas duas primeiras partes, que abordam operações pontuais e filtros lineares que podem ser utilizados para aplicar transformações nas imagens contendo buracos para evidenciar determinadas características.

- Operações pontuais[1] depende exclusivamente do valor do pixel em análise, não sendo necessário conhecer os valores dos pixels da vizinha. Alguns exemplos são ajuste de brilho e contraste.
- Filtros lineares podem ser utilizados para mitigar borrões, melhorar detalhes, acentuar arestas ou remover ruído. Neste trabalho, o filtro de maior interesse recai na detecção de arestas.
- Filtros não lineares podem ter desempenho superior quando comparados com filtros lineares. Esses filtros são obtidos utilizando uma combinação não linear dos pixels da vizinhança.

Os principais métodos de processamento de imagem destacados acima agora são tratados com maior detalhe e com exemplos abaixo.

2.1.2.1 Exemplos de operações pontuais

Os principais exemplos de operações pontuais são o controle de contraste e brilho em uma imagem digital. Sendo $f(i,j)$ uma função que representa os valores de 8 bits de uma imagem digital, pode-se entender que uma operação pontual como uma função $h(x)$ no domínio de f transforma o pixel na posição (i,j) assumindo um novo valor, no contradomínio de g. De forma simplificada, trata-se de uma composição como indicada na equação (1) abaixo.

$$g(i,j) = h(f(i.j)) \tag{1}$$

---

[1] Traducao livre de *Point operations* ver *Szeliski (2011)* para mais

Os processos mais comuns para ajuste de uma imagem são a multiplicação e a adição por uma constante, conforme Szeliski (2011). Esse processo pode ser expresso pela equação (2) abaixo, onde b>0 é chamado de viés (ou *bias*).

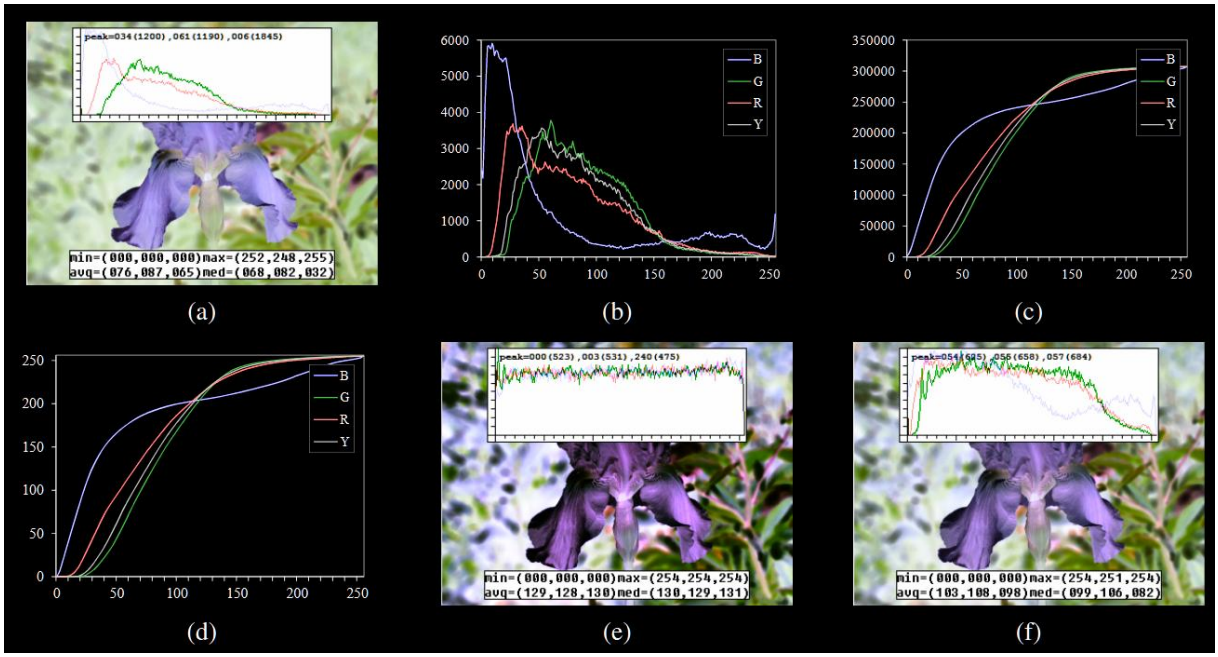
$$g(i,j) = af(i,j) + b \tag{2}$$

A transformação de cores também é bastante comum.

Apesar de melhorar uma imagem através do ajuste de ganho e brilho de forma manual, uma técnica mais interessante seria observar um histograma dos canais de uma imagem digital, que pode mostrar a intensidade de cada pixel e permitir uma melhor redistribuição dos seus valores. Essa técnica é conhecida como equalização por histograma, cuja intenção é encontrar uma função de mapeamento $f(I)$ cujo resultado é um histograma bem distribuído em todos os valores de [0,255] que compõem uma imagem digital. Para encontrar esse histograma equalizado, é necessário integrar o histograma original da imagem e obter uma função cumulativa de probabilidade, como mostrado na equação (3) abaixo, que demonstra a iteração entre os valores de i de h, acumulados ponto a ponto para se obter a função cumulativa de probabilidade. Esse processo pode ser melhor entendido através da Figura 2 abaixo, onde é mostrado esse procedimento sendo efetuado e os histogramas sendo apresentados.

$$c(I) = \frac{1}{N}\sum_{i=0}^{I} h(i) = c(I-1) + \frac{1}{N}h(I) \tag{3}$$

Figura 2 - Equalização de uma imagem através de uma análise de seu histograma



FONTE: Szeliski (2011).

Na Figura 2, o item (a) representa a imagem original com a seu histograma, onde é possível destacar algumas informações importantes como média, mínimos e máximos. O item (b) destaca o histograma da imagem original, nota-se que existe um excesso de *pixels* escuros e claros. O item (c) demonstra uma função de probabilidade cumulativa. O item (d) demonstra a aplicação da cumulativa como uma função que mapeia as cores dos *pixels* com o mesmo tipo de distribuição. O resultado desse processo é mostrado no item (e). Contudo, nota-se que a imagem ficou com um aspecto plano, segundo Szeliski (2011), por falta de contraste. Uma maneira de reverter esse efeito é usar uma equalização parcial como apresentado no item (f).

2.1.2.2 Exemplos de filtros lineares

Os filtros lineares utilizam os valores dos pixels vizinhos para determinar o novo valor do pixel “central” após a transformação. Essa operação pode ser representada pela equação abaixo.

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \tag{4}$$

A convolução pode ser obtida da equação acima, deslocando e invertendo a função f. Essa operação mostra-se fundamental no entendimento das redes neurais convolucionais, e seu entendimento será abordado na seção 2.3. A representação matemática da convolução e apresentada abaixo.

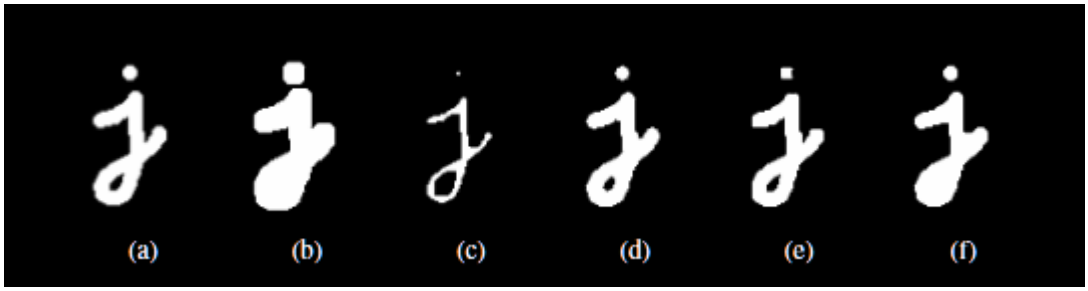
$$g = h * f \tag{5}$$

O filtro de Sobel é de particular interesse para este trabalho. Ele ser construído através de um kernel 3x3 que enfatiza as áreas de transição, regiões com grande gradiente, possivelmente entre uma área recoberta por asfalto em transição para uma área degradada que contém um buraco. Pode ser mostrado, que uma rede convolucional acaba por recriar esses filtros para identificar características fundamentais de imagens Aggarwal (2018a).

2.1.2.3 Exemplos de filtro não lineares

Um exemplo de filtro não linear seria o filtro de mediana utilizado para reduzir o ruído de imagens sem alterar suas bordas (arestas). Além disso, segundo Szeliski (2011), a aplicação de um filtro gaussiano pode atenuar as arestas de uma imagem, enquanto um filtro mediano consegue preservá-las melhor. Outras características podem ser realçadas com outros filtros não lineares, por exemplo, uma imagem binária composta unicamente de 0 e 255 pode ser alterada morfologicamente para se destacar certos objetos da imagem. Devido ao seu caráter binário, pode-se segregar áreas com pixels brancos e pretos de maneira muito fácil, expandindo ou contraindo uma determinada região com base no limite. A Figura 3 abaixo ilustra essa técnica.

Figura 3 - Alteração morfológica de imagens binarias



FONTE: Szeliski (2011)

Apesar das imagens no conjunto não serem binárias, é possível transformá-las através de uma função de limiar. Com isso, realça-se a característica de um buraco em relação ao asfalto.

Outra relação importante entre pixels é se eles estão conectados de alguma maneira, por exemplo, se possuem características em comum. É de se esperar que buracos que são delimitados por uma certa região tenham pixels que apresentem, em média, uma certa intensidade de transição entre buraco e asfalto. Efeitos luminosos podem atenuar essa relação, mas ela pode ser testada para ressaltar buracos.

Todas essas técnicas podem ser aplicadas para tentar aumentar um dataset e melhorar a precisão de um modelo no YOLO.

2.1.3 Detecção de bordas

Segundo Szeliski (2011), a detecção de bordas se baseia na variação local de um pixel em função da sua vizinhança. A variação pode ser medida através de uma derivada direcional, o gradiente, que irá indicar a direção com a maior variação de intensidade. Devido ao seu caráter local, através de uma derivada direcional é possível determinar pontos vizinhos que compõem aquele contorno. Aqui, assume-se contorno como a conexão entre pontos que compõem uma borda.

A rápida variação de intensidade pode ser determinada pela fórmula (6) abaixo.

$$J(x) = \nabla I(x) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial \mathrm{y}}\right)(x) \tag{6}$$

É importante notar que ∇I aponta para a maior variação de intensidade dada por $I(x)$. Segundo Szeliski (2011), aplicar operadores derivativos pode aumentar ruídos em alta frequência, sendo desejável utilizar um filtro passa-baixa antes de derivar a imagem. Ainda segundo o autor, a única opção realmente viável pelas suas características, é um filtro Gaussiano, por ser simétrico circularmente e suave. Trata-se do filtro mais utilizado hoje em dia.

A diferenciação é uma operação linear e pode ser comutada com outros operadores, como por exemplo, o Gaussiano. A fórmula (7) abaixo exemplifica esse processo.

$$J_\sigma = \nabla[G_\sigma * I(x)] = [\nabla G_\sigma](s) * I(x) \quad (7)$$

Essa operação é efetuada de forma discreta através da biblioteca OpenCV.

2.1.4 OpenCV

OpenCV é uma biblioteca de código aberto voltada para visão computacional e aprendizado de máquina criada por Bradski (2000), amplamente utilizada em diversas aplicações, desde reconhecimento facial até detecção de objetos e reconstrução 3D. Desenhada para eficiência, ela permite a execução de algoritmos complexos em tempo real. Implementada em C++, oferece grande eficiência graças ao código compilado e ao controle de memória que a linguagem oferece.

A biblioteca é composta por vários módulos que fornecem funcionalidades importantes, como processamento de imagem, calibração de câmera e análise de movimento.

Além das operações básicas, OpenCV suporta operações mais complexas, como a filtragem de imagens usando filtros Gaussianos, que suavizam a imagem para reduzir ruídos e detalhes desnecessários, como mencionado na seção 2.7. A biblioteca também permite a integração com outras bibliotecas de aprendizado de máquina, como TensorFlow e PyTorch.

2.2 REDES NEURAIS

As redes neurais foram pensadas originalmente como modelos para imitar o cérebro humano. Segundo Aggarwal (2018a), a unidade fundamental de uma rede neural é um neurônio, nome que remonta às origens do desenvolvimento em comparação com a biologia do ser humano. O neurônio, por sua vez, é apenas uma simplificação da sua contraparte biológica, e por muito tempo, acreditou-se que redes neurais não possuem utilidade prática. Acontece que, com o desenvolvimento de técnicas mais eficientes e, também, a evolução dos hardwares mais recentes, as

redes neurais se mostraram capazes de realizar atividades de classificação com alta precisão, geralmente desbancando outros modelos. Nessa seção iremos comentar sobre o funcionamento de redes neurais e os seus conceitos básicos, como fundamentação para a conceituação de redes convolucionais que são a essência do YOLO. Os nomes redes neurais e redes profundas são intercambiáveis, representando o mesmo conjunto de técnicas, essa convenção também se aplicara nesse trabalho.

2.2.1 Neurônio e um exemplo para motivação

O neurônio é a unidade fundamental de uma rede neural e, geralmente, realiza uma operação simples no seu interior, por exemplo uma função afim (linear). Em sua versão mais simples, também é chamado de *perceptron*, nome herdado das origens do desenvolvimento das redes neurais cuja inspiração foi a morfologia humana. Para explicar o seu funcionamento, seguiremos a explicação definida por Aggarwal (2018a). Primeiro, define-se um vetor de entrada $\boldsymbol{X}$ que contém as propriedades ou características a serem modeladas, notadamente $\{x_1, \ldots, x_d\}$, que será a entrada de um neurônio. O vetor de pesos $\boldsymbol{W} = \{w_1, \ldots w_d\}$ modelará o comportamento do fenômeno estudado. Assumindo uma saída binária entre -1 e +1, por simplificação, o valor estimado da saída é calculado como a equação abaixo.

$$\hat{y} = \sum_{i=1}^{d} w_i \, x_i \tag{8}$$

Sendo $\hat{y}$ o valor estimado e não o valor real y. Para esse exemplo, usar a função sinal pode levar os valores reais para as extremidades do problema, particularmente, -1 e +1. Com isso, temos uma nova equação abaixo.

$$\hat{y} = sinal(\boldsymbol{W} \cdot \boldsymbol{X}) = \boldsymbol{sinal}(\sum_{\boldsymbol{i=1}}^{\boldsymbol{d}} \boldsymbol{w_i \, x_i}) \tag{9}$$

Com esses dois valores, pode-se calcular o erro de estimativa, definido como $E = \hat{y} - y$, cujo valor será utilizado para otimizar os pesos do modelo, fazendo com o

erro se aproxime de 0, idealmente. Nesse exemplo acima, a função sinal é chamada de função de ativação cujo objetivo é modelar o sistema em estudo e, também, fornecer a saída com base no valor estimado.

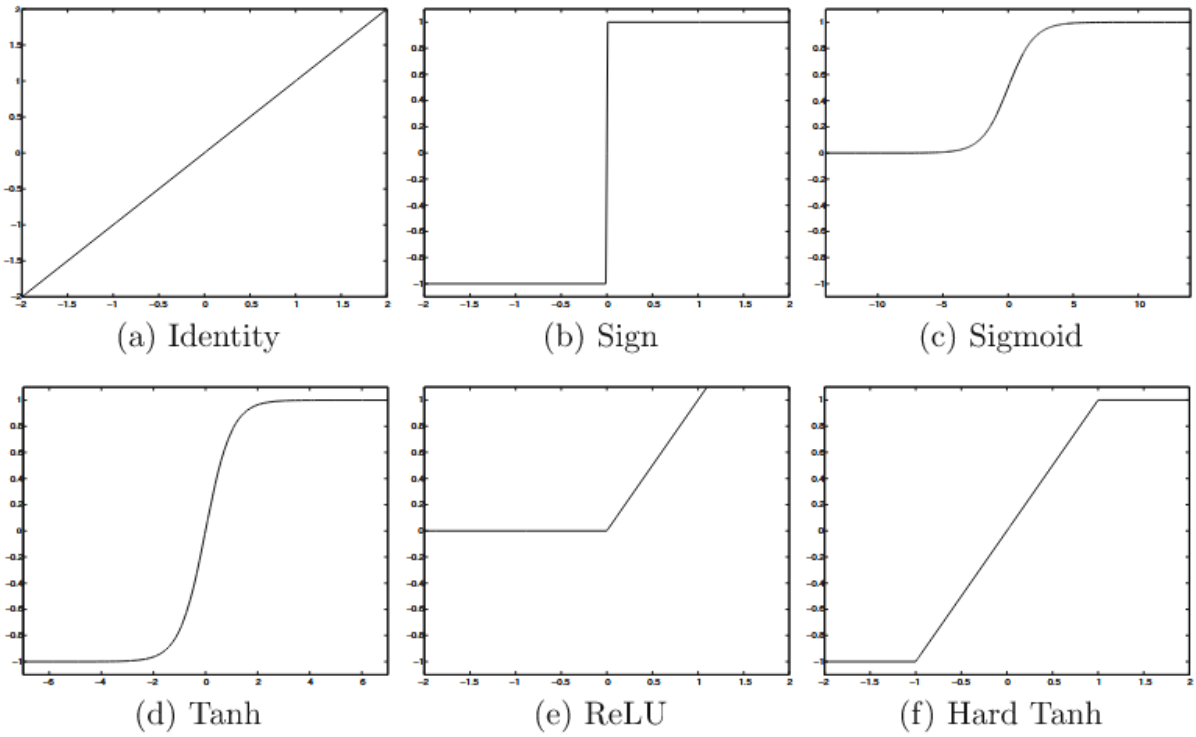
Existe outro parâmetro que pode ser incluída na equação 7 chamado viés. Ele representa uma característica constante que pode ser implementada no modelo. Não a perda de generalidade com a sua omissão na explicação acima.

2.2.2 Função de ativação

Segundo Aggarwal (2018a), a escolha da função de ativação é fundamental para o desempenho da rede neural. No exemplo acima, a escolha da função sinal faz sentido, pois a saída é binária, mas existem casos em que se deseja estimar um número real ou uma porcentagem que busque classificar um objeto ou fenômeno, onde funções de ativação especificas são requeridas.

As funções mais utilizadas como de ativação são a função ativação são: sinal, sigmoide, ReLU (*Rectified Linear Unit*), Tanh Hard. A Figura 4 abaixo apresenta as principais funções de ativação segundo o autor.

Figura 4 - Exemplos de funcão de ativação



FONTE: Aggarwal (2018b)

2.2.3 Escolhendo a função de perda

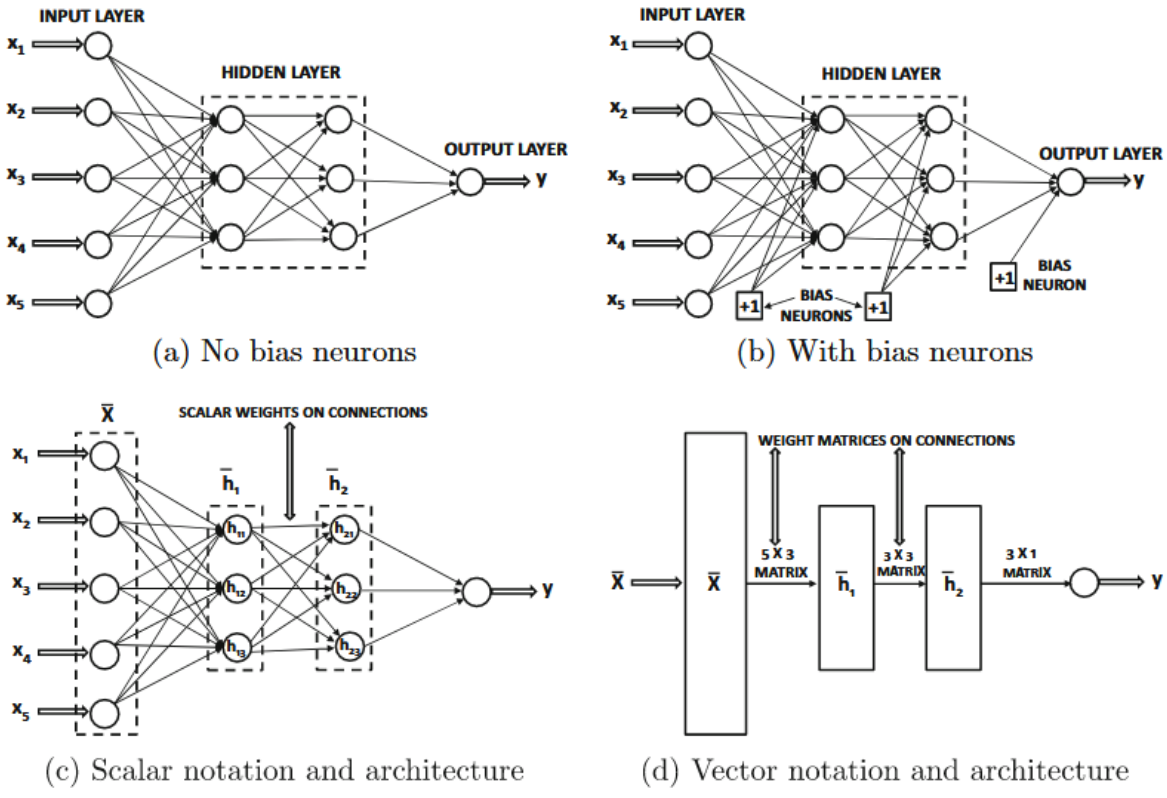
Segundo Aggarwal (2018b), a escolha da função de perda se relaciona diretamente com a problema que se deseja resolver e com a função de ativação. Para valores reais a função de mínimos quadrados é a mais utilizada, por se tratar de uma regressão. Para classificação, a função de perda escolhida depende do número de saídas possíveis: para uma saída binária use regressão logística e para saídas com múltiplas opções, use-se entropia cruzada (*cross-entropy)*.

2.2.4 Redes neurais com múltiplas camadas

As redes neurais com múltiplas camadas são compostas por a inserção de camadas adicionais entre a entrada e a saída de uma modelo. Essas camadas são chamadas de ocultas, por seu método de cálculo ser ‘oculto’ enquanto para redes de camadas únicas, as relações definidas ficam evidentes Aggarwal (2018). Essas camadas são colocadas, tradicionalmente, umas na sequência das outras, gerando um modelo conhecido como propagação para frente (*forward propagation*), segundo Aggarwal (2018).

As camadas ocultas são chamadas de $\overline{h_1}, \ldots, \overline{h_k}$, onde k é a k-ésima camada oculta de uma rede. Chama-se peso ou grau de uma camada oculta, a quantidade de neurônios que a compõe, especificamente para k neurônios, têm-se $p_1, \ldots, p_k$. A Figura 5 a seguir demonstra alguns modelos esquemáticos para redes neurais utilizando a notação acima.

Figura 5 - Modelos de rede neural multi-camada



FONTE: Aggarwal (2018b)

É importante notar que na imagem acima, y pode se tratar de um vetor. As camadas intermediarias não precisam ter o mesmo grau, contudo, por estarem dispostas em sequência, a saída de uma será fornecida a posterior, logo é possível afirmar que as características da camada h_t irá definir as características da camada h_{t+1}. As dimensões dos vetores resultantes em cada etapa dependerão de regras de multiplicação entre matrizes. As equações abaixo regem a dimensão da camada de entrada, da p+1 camada oculta e da camada de saída, respectivamente.

$$\widehat{h_1} = \Phi(W_1^T \bar{x}) \tag{10}$$

$$\widehat{h_{p+1}} = \Phi(W_{p+1}^T \overline{h_p}) \quad \forall p \in (1, \dots, p-1) \tag{11}$$

$$\hat{o} = \Phi(W_{k+1})^T \overline{h_k} \tag{12}$$

Onde $\Phi(\cdot)$ é a função de ativação aplicada de forma recorrente em todas os neurônios da k-ésima camada oculta, W_i é a i-ésima matriz de pesos de uma rede neural, $\hat{o}$ é a camada de saída. As redes neurais em sua maioria são completamente conexas, ou seja, todos os neurônios de uma camada estão conectados ao da seguinte. Segundo Aggarwal (2018b), para grande maioria das aplicações, redes totalmente conectadas são as mais utilizadas, mas existem algumas outras estruturas que podem ser utilizadas, escolhendo propositalmente como os neurônios devem ser conectados ou não. Um exemplo é a rede convolucional que será tratada em seções subsequentes, devido a sua importância na classificação de imagens, e sua utilização no YOLO.

2.2.5 Algoritmo de propagação reversa do erro

O algoritmo de propagação reversa (*error backpropagation*) é uma ferramenta fundamental na explosão recentes das redes neurais. Segundo Aggarwal (2018b), redes com caminhos únicos possuem cálculos de gradiente relativamente simples, pois apenas o cálculo direto da função de perda já permite uma indicação de como os pesos devem ser atualizados ao final de cada interação. Contudo, para redes mais complexas, com múltiplas camadas, o gradiente da função objetivo em relação aos pesos é dependente de uma intrincada teia de pesos entre as camadas ocultas, o que dificulta de forma considerável a complexidade deste processo de otimização. Acontece que é possível calcular de maneira eficiente usando programação dinâmica.

2.3 REDES NEURAIS CONVOLUCIONAIS

Aggarwal (2018) elucida que as redes neurais convolucionais (CNNs) ampliam os conceitos debatidos até aqui sobre redes neurais. As CNNs são utilizadas para identificar padrões em imagens diversas e seu desempenho tem sido atestado em diversas competições de classificação de imagens, onde as CNNs geralmente conseguem alcançar as melhores colocações (Aggarwal (2018).

As CNNs têm como entrada uma rede multidimensional com altura L_q, largura B_q e profundidade d_q. Os valores individuais para a altura e a largura são conhecidos como *pixels* e, eles representam características espaciais da imagem. A profundidade

armazena propriedades adicionais. Por exemplo, para uma imagem, a profundidade pode representar os canais RGB. Geralmente, a profundidade é uma propriedade independente das características espaciais da imagem.

Com isso, a primeira camada, $q = 1$, terá suas configurações definidas com base nas dimensões da entrada, em particular, $L_q \; x \; B_q \; x \; d_q$. Para camadas mais profundas, essa organização irá se manter por todas as suas camadas, contudo, sua profundidade mudará com base nas demais propriedades aprendidas durante o treinamento, que podem ser significativas para a classificação. Para exemplificar como funciona a mudança das dimensões, considera-se uma imagem com $32x32x3$.

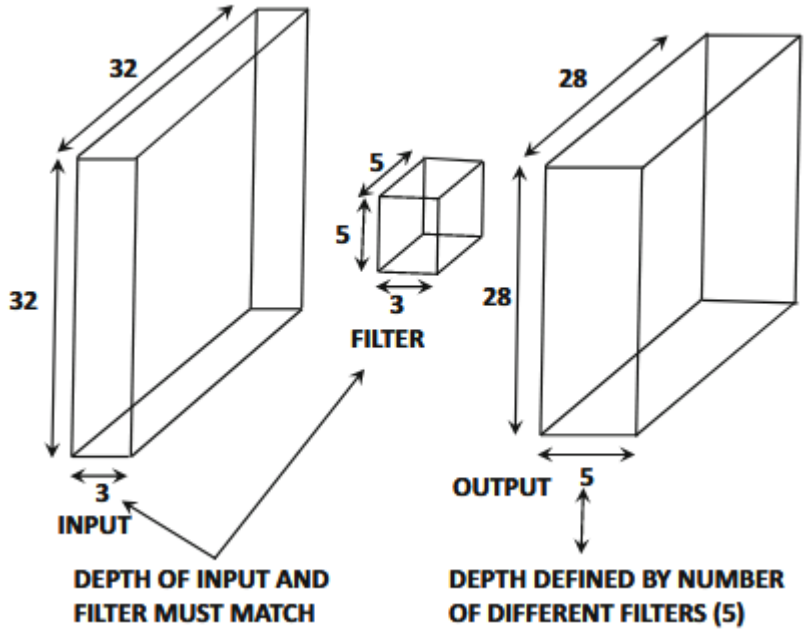
Os filtros aplicados em cada etapa geralmente são quadrados e possuem a mesma profundidade da entrada na respectiva camada. Logo, as dimensões são $F_q \; x \; F_q \; x \; d_q$, onde os valores mais usualmente utilizados são 3 ou 5 Aggarwal (2018b). Na Figura 6 abaixo, nota-se que as dimensões especiais são reduzidas pela aplicação do filtro com dimensões $5x5x3$, onde a imagem resultante tem tamanho $28x28x5$. Pode-se generalizar esse resultado através das equações 13 e 14 abaixo.

$$L_{\{q+1\}} = \; L_q - F_q + 1 \tag{13}$$

$$B_{\{q+1\}} = B_q - F_q + 1 \tag{14}$$

Ademais, nota-se que a profundidade se alterou para a entrada da segunda camada, passando de 3 para 5. Esse valor é determinado pela estrutura das camadas ocultas seguintes, e o seu intuito é identificar outras características mais complexas de uma imagem. A Figura 6 apresenta um exemplo de rede neural convolucional e a aplicação de um filtro que deve estar condizente com a profundidade da imagem de entrada.

Figura 6 - Exemplo de rede convolucional



FONTE: Aggarwal (2018b)

2.4 DARKNET

Segundo Wang et al. (2023), Darknet é um *framework* de código aberto projetado para redes neurais convolucionais, destacando-se por sua eficiência, flexibilidade e suporte otimizado para GPUs (*Graphics Processing Unit)* por meio de CUDA (*Compute Unified Device Architecture*). Escrito em C, o framework permite fácil configuração de arquiteturas e utilização de modelos pré-treinados, como o YOLO, utilizado amplamente para detecção de objetos em tempo real. Sua estrutura leve e modular facilita a instalação em diversos sistemas, enquanto seu desempenho possibilita aplicações em áreas como vigilância, direção autônoma e classificação de imagens, tornando-o uma ferramenta essencial para soluções em visão computacional.

2.4.1 Darkhelp

Darkhelp é uma biblioteca complementar ao Darknet, projetada para simplificar a implementação e interpretação de modelos baseados em YOLO. Ela permite realizar inferências em redes neurais treinadas no Darknet, oferecendo uma interface

amigável para processar imagens e obter resultados, como classes de objetos detectados e caixas delimitadoras, de forma eficiente e rápida.

2.4.2 Darkmark

Darkmark é uma ferramenta projetada para facilitar o treinamento e a validação de redes neurais baseadas no *framework* Darknet. Suportando diversos formatos de anotação de dados, o Darkmark agiliza o processo de preparação de conjuntos de dados para algoritmos como o YOLO, oferecendo uma interface intuitiva para a identificação precisa e rápida de objetos nas imagens. Sua capacidade de gerar visualizações detalhadas das predições, bem como de monitorar métricas de desempenho, torna-o essencial para desenvolvedores que buscam aprimorar suas soluções em visão computacional. Ele também permite processos mais simples de aumentação de dados (*data augmentation*), ou seja, aumentar a variabilidade do conjunto de imagens.

2.5 YOLO

Segundo Redmon; Farhadi (2018), YOLO (You Only Look Once) é um algoritmo de reconhecimento de objetos que se destaca pela sua capacidade de realizar detecção em tempo real. Desenvolvido por Joseph Redmon, o YOLO transforma a tarefa de detecção de objetos em um problema de regressão única, ao invés de depender de técnicas tradicionais de segmentação que envolvem múltiplos passos e processos complexos.

YOLO divide a imagem de entrada em uma grade e, para cada célula da grade, prevê a probabilidade de presença de um objeto. Além disso, o algoritmo estima as coordenadas da caixa delimitadora (*bounding box*) e a classe do objeto. Este método permite que as predições sejam feitas rapidamente e em paralelo, o que é essencial para aplicações de tempo real.

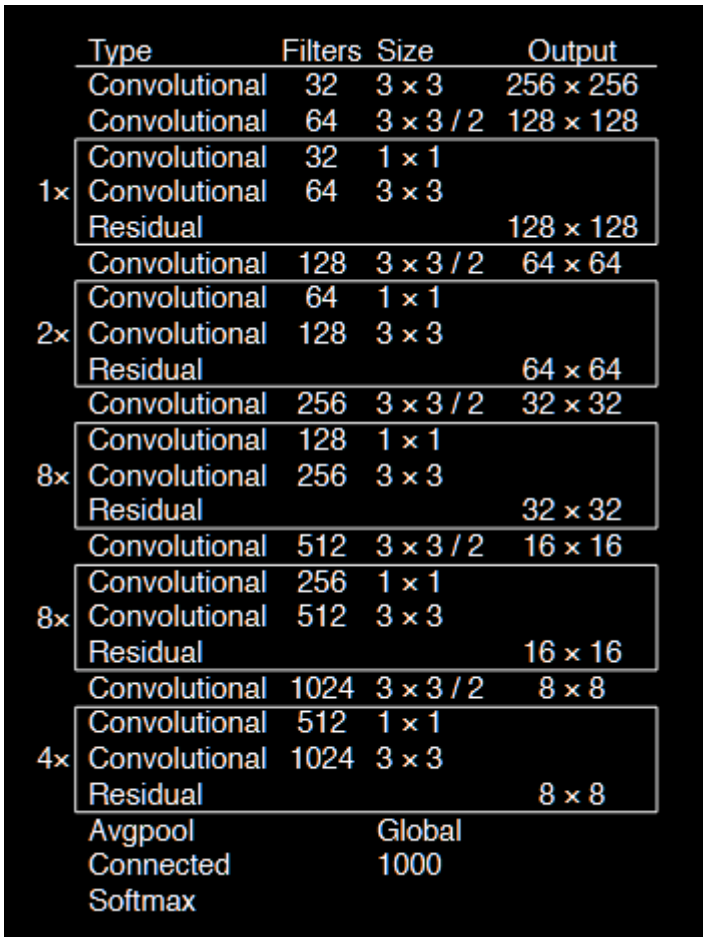
A arquitetura do YOLO é composta por várias camadas de convolução que extraem características da imagem de entrada. Essas camadas convolucionais são seguidas por algumas camadas totalmente conectadas que utilizam os recursos extraídos para prever as caixas delimitadoras e as classes dos objetos. A última

camada da rede neural é responsável por fornece todas as predições necessárias para a detecção.

2.5.1 YOLO V3

Segundo Redmon; Farhadi (2018), a grande diferença entre as versões anteriores, consiste no aumento da complexidade da rede convolucional. Enquanto o YOLO V2 usava uma versão com apenas 19 camadas convolucionais, a sua versão V3 e construída com base no Darknet-53, com 53 camadas convolucionais. Os detalhes são apresentados na Figura 7, abaixo.

Figura 7 - Composição da arquitetura do yolo v3



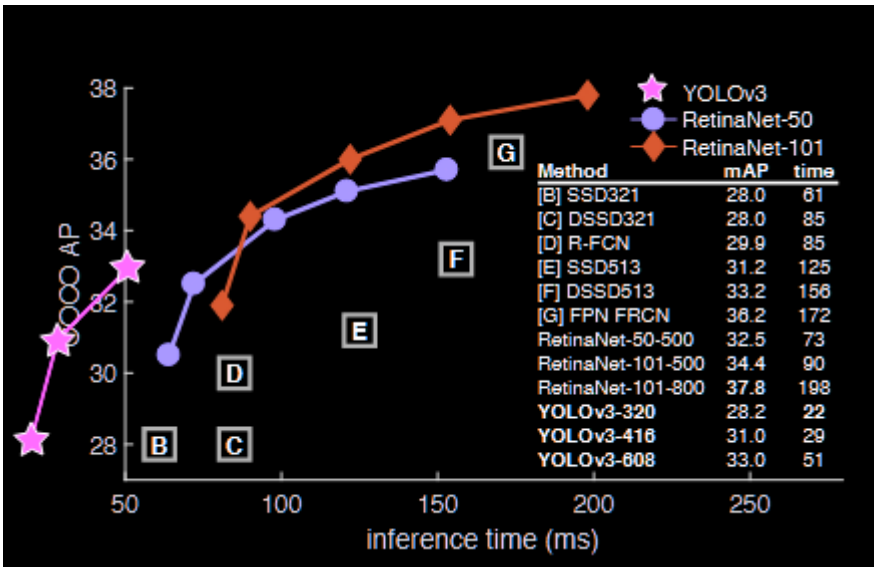
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

FONTE: Redmon; Farhadi (2018)

O aumento da complexidade garante uma melhor acurácia dos modelos criados, contudo existe uma troca no que tange o desempenho na detecção de objetos

em tempo real. Contudo, a troca se justifica, pois, a perda de desempenho é pequena quando comparada com o ganho de precisão obtido. Abaixo, a Figura 8 demonstra esse compromisso, apesar do menor desempenho quando comparado com a arquitetura anterior, o YOLO V3 ainda supera a concorrência na velocidade de detecção.

Figura 8 - Comparação Yolo V3 e outras arquiteturas de CNN



FONTE: Redmon; Farhadi (2018).

2.5.2 YOLO V3-*Tiny*

A versão YOLOv3-*tiny* apresenta uma arquitetura simplificada com apenas 14 camadas convolucionais, interpoladas com camadas de *maxpool*. Devido a essa simplificação, o desempenho para detecção em tempo real aumento, mantendo um bom compromisso entre tempo de detecção e precisão do modelo. A Tabela 1 abaixo, resume esse compromisso entre tempo de detecção, precisão e demanda do hardware.

Tabela 1 - Comparação de desempenho das versões v3 e v3-*tiny*

Versão	Camadas Convolucionais	Velocidade de detecção	Precisão	Compromisso
YOLO V3	53	Menor velocidade em tempo real	Maior precisão	Troca entre desempenho e precisão

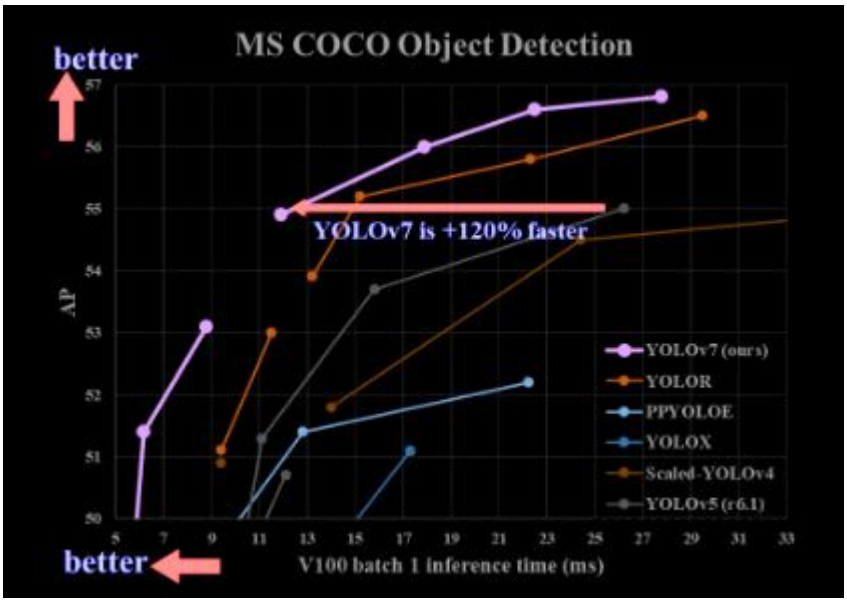
Versão	Camadas Convolucionais	Velocidade de detecção	Precisão	Compromisso
YOLO V3 - *Tiny*	14	Maior velocidade em tempo real	Boa precisão	Equilíbrio entre tempo de detecção e precisão

FONTE: O Autor (2025).

2.5.3 YOLO V7

Devido ao dinamismo da área de detecção de objetos e visão computacional, outra arquitetura foi proposta por Wang et al. (2023). Essa nova metodologia desempenha melhor que as demais em diversos casos, particularmente, um comparativo entre outras redes neurais convolucionais e mostrada na Figura 9 abaixo.

Figura 9 - Comparação entre yolo v7 e demais arquiteturas



FONTE: Wang et al. (2023)

A comparação feita com base no conjunto de dados MS COCO, Lin et al. (2014), onde se percebe que o tempo de inferência da versão V7 é menor que a das demais arquiteturas, mesmo quando comparado com outras versões do YOLO.

2.5.4 YOLO v7-*Tiny*

Existe uma versão com arquitetura simplifica da versão 7 do YOLO. Novamente, um compromisso e feito entre previsão e tempo de inferência. A comparação básica entre as arquiteturas e apresentada na Tabela 2 abaixo.

Tabela 2 - Comparação de desempenho das versões v7 e v7-*tiny*

Versão	**Camadas Convolucionais**	**Velocidade de detecção**	**Precisão**	**Compromisso**
YOLO V7	82	Menor velocidade em tempo real	Maior precisão	Troca entre desempenho e precisão
YOLO V7 - *Tiny*	58	Maior velocidade em tempo real	Boa precisão	Equilíbrio entre tempo de detecção e precisão

FONTE: O Autor (2025).

2.6 COMPARAÇÃO ENTRE YOLO E OUTRAS REDES NEURAIS CONVOLUCIONAIS NA IDENTIFICAÇÃO DE OBJETOS

O R-CNN e suas variantes (Fast R-CNN, Faster R-CNN) são baseados em propostas de regiões, onde um algoritmo externo (como Selective Search) gera candidatos a objetos antes da classificação por CNN. Essa abordagem garante alta precisão, especialmente em cenas complexas com objetos pequenos ou sobrepostos, mas é lenta devido ao processamento individual de cada região. O Fast R-CNN e o Faster R-CNN melhoram a eficiência ao compartilhar computações convolucionais, mas ainda dependem de pipelines em múltiplas etapas, tornando-os menos adequados para aplicações em tempo real, Girshick et al. (2014).

Segundo “SSD” (2016), SSD é um detector single-shot que elimina a necessidade de propostas de regiões, combinando detecção e classificação em uma única rede. Ele opera em múltiplas escalas de *feature maps* para capturar objetos de diferentes tamanhos, oferecendo um bom equilíbrio entre velocidade e precisão. No entanto, seu desempenho em objetos muito pequenos pode ser inferior ao de métodos baseados em regiões (como Faster R-CNN), e sua eficiência depende do ajuste cuidadoso de hiperparâmetros, como o número de *default boxes*.

Para Li et al. (2020), o EfficientDet é uma arquitetura altamente otimizada que combina a EfficientNet (backbone eficiente em FLOPS) com um mecanismo de BiFPN (*Weighted Bi-directional Feature Pyramid Network*) para fusão multiescala

de *features*. Projetado para escalabilidade, ele varia de modelos leves (EfficientDet-D0) a altamente precisos (EfficientDet-D7), adaptando-se a diferentes restrições computacionais. Apesar de sua eficiência, sua implementação pode ser mais complexa que a do YOLO, e seu desempenho em tempo real pode ser inferior em dispositivos *edge* sem otimizações específicas.

A escolha pelo YOLO deve-se à sua implementação mais acessível em comparação com outras arquiteturas. O modelo oferece uma gama versátil de aplicações, tanto do ponto de vista teórico quanto prático. Outro diferencial importante é a padronização no funcionamento, que permite testar múltiplas arquiteturas facilmente, bastando baixar um arquivo .cfg já disponível online.

2.7 DATASET AUMENTADO

Aumentação de dados (*data augmentation*) é uma técnica usada na área de aprendizado de máquina para aumentar a diversidade de um conjunto de dados de imagens sem a necessidade de coletar novas amostras. Isso é feito através de uma série de transformações nas imagens originais, como rotações, reflexões, alterações de brilho, cortes aleatórios, adição de ruído e outros ajustes. Essas modificações ajudam a criar variações nas imagens, permitindo que modelos de aprendizado profundo sejam treinados de forma mais robusta e com maior capacidade de generalização.

Em cenários onde os conjuntos de dados são limitados ou desbalanceados, a aumentação de dados desempenha um papel essencial ao fornecer ao modelo uma variedade maior de exemplos para aprendizado. Além disso, melhora o desempenho do modelo em situações de uso real, onde as condições podem variar significativamente em relação ao ambiente controlado do conjunto de treinamento.

Por exemplo, ao treinar um modelo para detectar buracos em pavimentos, a aplicação permite simular diferentes condições de luz, ângulos de visão ou até mesmo variações no formato e tamanho dos buracos. Isso torna o modelo mais adaptável e preciso na identificação dos defeitos de pavimentação, independentemente das condições em que o sistema é implementado.

2.8 PAVIMENTAÇÃO

Pavimentação é abordada com mais detalhe em Julio (2024). Aqui se faz um resumo sobre os temas sobre pavimentação tratados naquele trabalho, de forma resumida.

Segundo DNIT (2006), pavimento de uma rodovia é uma superestrutura composta por várias camadas de diferentes espessuras, colocadas sobre um subleito, que é a infraestrutura ou terreno de fundação. O subleito deve ser estudado até uma profundidade onde as cargas do tráfego atuam significativamente, geralmente entre 0,60 m e 1,50 m. Devido a razões técnico-econômicas, o pavimento é uma estrutura complexa, onde materiais com diferentes resistências e deformabilidades são combinados, resultando em um cálculo complexo de tensões e deformações causadas pelo tráfego.

2.8.1 Defeitos e falhas

O DNIT (2006) também define os defeitos e falhas que podem ocorrer em um solo pavimentado. Em particular, dois cenários são de interesse deste trabalho: os buracos e os remendos.

2.8.1.1 Buracos

Os buracos são decorrentes da degradação das camadas superiores de uma pavimentação devido a motivos diversos. Essa degradação ocorre de maneira gradual e pode atingir as camadas inferiores do asfalto, retirando as camadas que mantêm as estruturas da pavimentação. Entre os principais motivos para formação de buracos ou panelas estão o tráfego de veículos pesados como caminhões e a precipitação. Os buracos também são conhecidos como panelas. A Figura 10 mostra um exemplo de buraco ou panela.

Figura 10 - Exemplo de buraco

FONTE: Roboflow (2020)

2.9 MÉTRICAS

As redes neurais precisam ser reavaliadas para otimizar os pesos escolhidos inicialmente. Para entender se o modelo criado está sendo capaz de generalizar uma determinado sistema ou situação, utiliza-se métricas que indicam o desempenho do modelo.

2.9.1 Precisão

$$P = \frac{TP}{TP + FP} \tag{15}$$

A precisão (P) é definida como a razão entre o número de verdadeiros positivos (TP) e o total de verdadeiros positivos mais falsos positivos (FP). Um verdadeiro positivo representa a detecção correta, um buraco para esse trabalho. Um falso positivo ocorre quando uma falha é identificada, mas na verdade não há falha alguma.

2.9.2 Recall

$$R = \frac{TP}{TP + FN} \tag{16}$$

O recall (R), conforme definido pela equação 16 acima, é a razão entre verdadeiros positivos e a soma de verdadeiros positivos mais falsos negativos (FN). Para esse trabalho, um verdadeiro positivo seria um buraco, enquanto um falso negativo seria um buraco que não foi identificado pelo modelo.

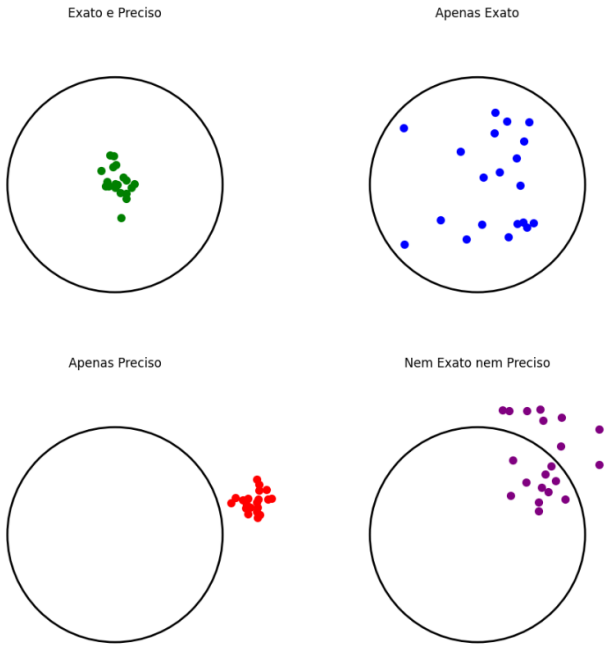
2.9.3 Acurácia

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{17}$$

A exatidão/acurácia (A) mede a capacidade do algoritmo de distinguir entre uma falha e uma não falha para o universo observado/classificado. Mede-se (A) como sendo a razão entre soma de verdadeiro positivos mais verdadeiro negativos (TN) sobre a soma dos verdadeiros positivos, verdadeiros negativos, falso positivos e falso negativos.

Pode existir uma confusão entre os conceitos de precisão e acuraria. A Figura 11 exemplifica de maneira gráfica essa diferença. Considerando os círculos abaixo como sendo alvos, deseja-se intuitivamente acerta com precisão o centro desse círculo. Isso acontece apenas quando os disparos são precisos e exatos. Se os disparos são apenas exatos, eles acertam o alvo, contudo estão espelhados na sua área. Por outro lado, disparos precisos podem não acertar o alvo, mas permanecem pertos uns dos outros. Disparos nem precisos ou exatos, não apresentam nenhum padrão reconhecido.

Figura 11 - Diferença entre precisão e exatidão



FONTE: O autor (2025)

2.9.4 IOU

Intersecção sobre união (*Intersection over Union* - IOU) compara a área do *bounding box* do modelo com a área real. Se o IOU for maior que o *threshold*, geralmente 0.5, a detecção é considerada bem-sucedida. Mesmo se o modelo der um *bounding box* maior que o verdadeiro, a detecção ainda pode ser considerada bem-sucedida.

2.9.5 mAP

$$mAP = \frac{1}{N}\sum_{k=1}^{N} AP_k \tag{18}$$

A métrica mAP (*mean Average Precision*) é calculada utilizando a equação (18) descrita acima, onde N representa o número de classes e AP_k é a Precisão Média para a classe k. O mAP é amplamente utilizado em tarefas de detecção de objetos e classificação em visão computacional, avaliando a precisão dos modelos com base na correspondência entre as predições e os rótulos reais, considerando o conceito de IoU. A Precisão Média AP obtida através do cálculo da área sob a curva de Precisão-*Recall*, e o mAP é a média das APs de todas as classes no conjunto de dados. Quanto

mais próximo o valor do mAP estiver de 1, melhor será o desempenho do modelo na detecção e classificação de objetos.

2.10 TRABALHOS CORRELATOS

2.10.1 Detecção de buracos em pavimentos asfálticos com base em processamento digital de imagens e deep learning

O estudo Tallita Passos et al. (2021) explorou a aplicação de redes neurais convolucionais pré-treinadas, incluindo MobileNet, Inception, e Faster R-CNN com a arquitetura Inception, para a detecção de buracos em pavimentos. Embora tenha sido possível identificar buracos utilizando essas arquiteturas, os resultados revelaram uma precisão insatisfatória. As redes MobileNet e Inception, conhecidas por sua eficiência em dispositivos com recursos limitados, e o modelo Faster R-CNN Inception, que oferece maior complexidade, foram avaliados quanto à sua capacidade de generalização na detecção de falhas em pavimentos asfálticos. A precisão abaixo do esperado sugere desafios na adaptabilidade do modelo para a identificação precisa de buracos, indicando a necessidade de investigações adicionais e ajustes no treinamento do modelo para aprimorar seu desempenho. Este trabalho ofereceu uma base crítica para futuras otimizações visando aperfeiçoar a eficácia dessas redes na detecção confiável de buracos em pavimentos asfálticos.

2.10.2 Detecção e mapeamento de buracos em vias públicas asfaltadas utilizando visão computacional por meio de um sistema embarcado instalado em veículo automotivo.

O estudo Caon (2019) propôs e implementou uma abordagem inovadora para a detecção de buracos em pavimentos asfálticos, incorporando um processo avançado de pré-processamento de imagens. Os resultados obtidos foram altamente positivos, evidenciando uma notável capacidade do sistema em identificar buracos com precisão. Além disso, a solução foi integrada de forma eficaz em um dispositivo Raspberry Pi, demonstrando viabilidade para aplicação em tempo real. O pré-processamento de imagens desempenhou um papel fundamental na melhoria da qualidade dos dados de entrada, contribuindo para o sucesso do modelo na detecção

de buracos. Este trabalho não apenas destaca avanços na identificação de falhas em pavimentos, mas também enfatiza a aplicabilidade prática da solução, promovendo a implementação em dispositivos de baixo custo, como o Raspberry Pi, para aplicações em campo.

2.10.3 Detecção autônoma de trincas e buracos (traduzido)

Este estudo Anand et al. (2018) empregou a arquitetura YOLOv3 para a detecção de buracos em pavimentos na Indonésia, alcançando resultados notáveis e interessantes. A aplicação da rede neural YOLOv3 possibilitou a identificação eficaz e precisa de buracos, mesmo em condições desafiadoras específicas da região indonésia. A abordagem YOLOv3, conhecida por sua eficiência em tempo real e detecção precisa de objetos, demonstrou-se promissora na resolução do desafio específico da identificação de falhas em pavimentos. Esses resultados sugerem um potencial significativo para a utilização prática da tecnologia YOLOv3 na manutenção e monitoramento eficientes das condições das estradas na Indonésia, contribuindo para melhorias na infraestrutura viária. Este estudo destaca a aplicabilidade bem-sucedida do YOLOv3 em um contexto específico, abrindo caminho para futuras implementações e otimizações em projetos de detecção de buracos em nível nacional.

2.10.4 Detecção e mapeamento de defeitos no pavimento de rodovias

Julio (2024) propõe um aplicativo móvel para detectar buracos em rodovias usando YOLOv4-*Tiny*, com imagens capturadas pelo smartphone. A pesquisa surgiu da necessidade de melhorar a manutenção das estradas brasileiras, já que 67,5% da malha rodoviária apresenta problemas. Foram coletadas 1183 imagens para treinar o modelo, que alcançou 86,2% de precisão, mas teve dificuldades na detecção de buracos pequenos e sombras. Nos testes práticos, o desempenho foi afetado em velocidades maiores devido ao desfoque das imagens. Apesar das limitações, o projeto demonstra potencial para otimizar a gestão rodoviária, tornando as manutenções mais rápidas e eficientes, com a necessidade de aprimorar a base de dados e utilizar hardware mais potente para melhorar os resultados.

2.10.5 Foco na qualidade

Este trabalho propõe avaliar como a coleta de imagens de diferentes regiões pode influenciar a precisão e a eficácia do modelo. Para isso, serão coletadas imagens da cidade de Curitiba, complementadas por imagens obtidas em bancos online, que servirão tanto para o treinamento quanto para a validação do modelo. A expectativa é que a ampliação do conjunto de dados inicial proporcione maior diversidade ao modelo, potencializando sua capacidade de inferência. Considerando a grande variedade de buracos com diferentes formatos e condições de iluminação, prevê-se que o uso de técnicas de *data augmentation*, Kaur et al. (2021), seja determinante para melhorar os resultados.

3 MATERIAL E MÉTODOS

3.1 HARDWARE

Os componentes físicos utilizados nesse trabalho são apresentados nas seções seguintes.

3.1.1 Computador para treinamento.

Para o treinamento do modelo em YOLO, utilizou-se a seguinte configuração apresentada no quadro abaixo. Trata-se de um computador para uso pessoal utilizado no trabalho de conclusão de curso.

QUADRO 1 – Hardware utilizado para treinar modelo com YOLO.

Componente	Marca	Modelo	Características
Placa-mãe	Gigabyte	A520M	
Placa de Vídeo	NVIDIA	3060 RTX	Memoria dedicada de 16GB, outras
Processador	Ryzen	5500	3.6 GHz
Memória RAM	Gigabyte	16GB	3200 MHz DDR 4

FONTE: O autor (2025).

3.1.2 Captura de imagem.

Para realizar a captura de imagens necessária para criar um conjunto de dados que contemple as características próprias da região curitibana, foi utilizado um dispositivo móvel Samsung S23, cujas propriedades estão resumidas no quadro abaixo.

QUADRO 2 – Especificações Samsung 23.

Categoria	Detalhes
Câmera principal	50 MP com estabilização ótica de imagem (OIS)
Câmera Ultra Wide	12 MP com ângulo de visão de 120º
Câmera Teleobjetiva	10 MP com zoom ótico de 3x
Câmera Frontal	12 MP com tecnologia Dual Pixel
Processador	Qualcomm Snapdragon 8 Gen 2
Velocidade do Processador	Até 3.36 GHz
RAM	8 GB
Armazenamento	128 GB

FONTE: O autor (2025).

Durante o processo de captura de imagem, foi utilizada a câmera *ultra wide* devido ao seu grande ângulo de captura, para cobrir o máximo possível do terreno gravado. Também foram feitos vídeos de regiões com buracos e sem buracos para verificar como o modelo se aplica às condições da região. Abaixo, a Figura 12 apresenta uma imagem capturada na região de Curitiba.

Figura 12 - Exemplo de buraco em Curitiba

FONTE: O Autor (2025).

3.2 CONJUNTO INICIAL DE IMAGENS

Em um primeiro momento, trabalhou-se com um conjunto reduzido de imagens de buracos de buracos obtidos de Ninja (2025). Esse conjunto foi dividido em treino e validação para testar o desempenho de partido dos modelos Yolo v3-tiny e v7-*tiny* para generalizar a detecção de buracos. Essa metodologia permite identificar problemas na implementação do modelo como *overfitting* e outros problemas

clássicos das redes neurais convolucionais. As proporções estão indicadas na tabela abaixo.

Tabela 3 - Tamanho e proporção do conjunto de imagens inicial

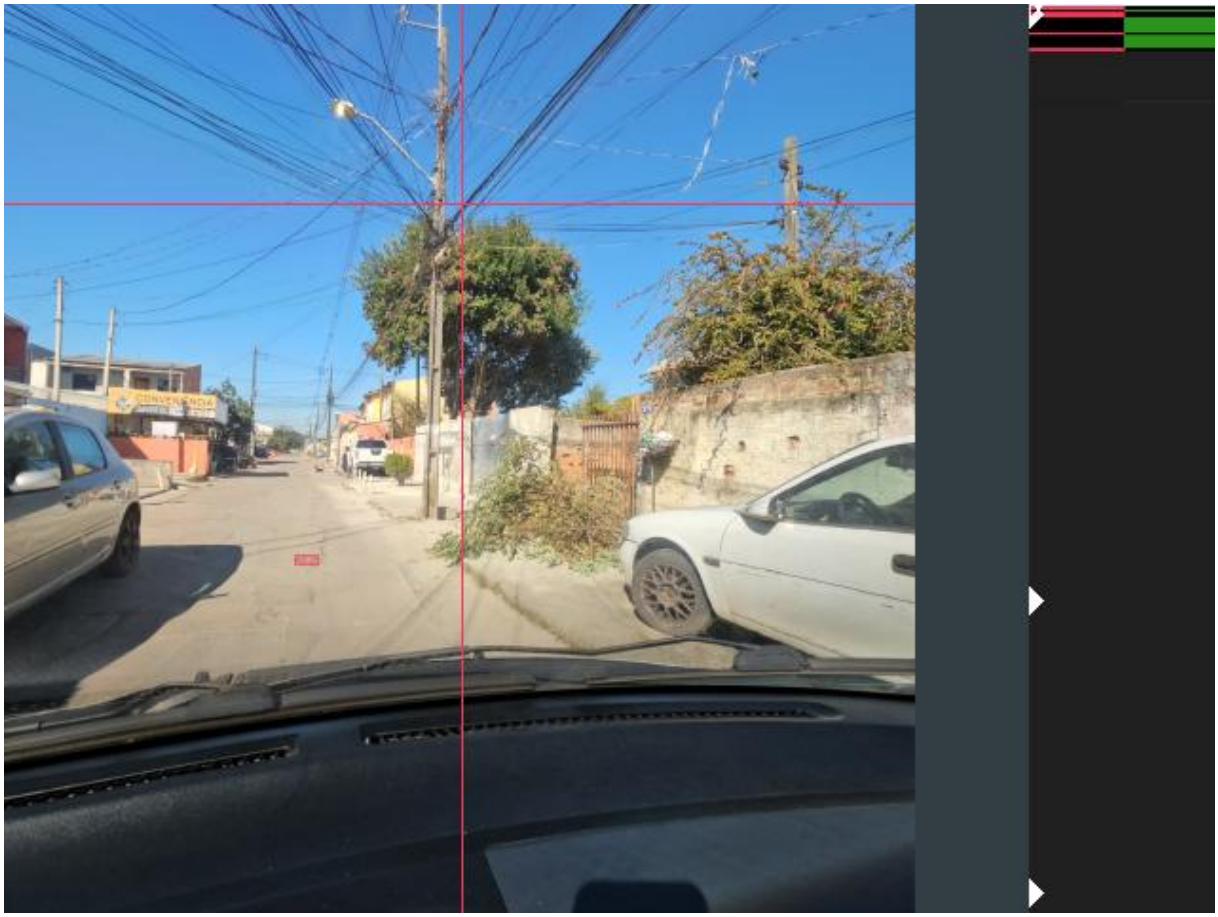
Classe	Tamanho	Proporção (%)
Treinamento	25	80
Teste	6	20

FONTE: O autor (2025).

Foram feitos uma série de testes com ambas as arquiteturas acima, para identificar qual apresentava o melhor desempenho. Este trabalho define desempenho como melhor capacidade de inferência, ou seja, apresentar as melhores métricas durante os testes, notadamente, precisão média (mAP), recall e precisão.

As imagens inicialmente não possuíam rótulos, não podendo serem utilizadas diretamente. A ferramenta DarkMark foi utilizada para marcar as imagens de forma visual, ver seção 2.4.2.

Figura 13 - Exemplo de imagem sendo anotada com DarkMark



FONTE: O autor (2025)

A rotulação de imagens gera arquivos com extensão .txt com as coordenadas normalizadas, geradas automaticamente pela ferramenta. A Figura 14 apresenta um exemplo de arquivo de texto gerado. Ele é composto por diferente n linhas e 5 colunas. As linhas representam diferentes objetos presentes na imagem. A primeira coluna representa a classe de um objeto. As demais colunas estão normalizadas com base no tamanho da imagem, sendo que a segundo e terceira coluna representam a posição do centro do retângulo, e as colunas 4 e 5 representam a largura e altura do retângulo.

Figura 14 - Exemplo de arquivo de texto gerado pelo DarkMark

```
0 0.5529891304347826 0.5981884057971014 0.045108695652173916 0.015217391304347827
0 0.5902173913043478 0.5300724637681159 0.03152173913043478 0.007971014492753623
0 0.6361413043478261 0.5264492753623189 0.027717391304347826 0.006521739130434782
0 0.6364130434782609 0.5376811594202898 0.017391304347826087 0.005797101449275362
0 0.5489130434782609 0.5123188405797101 0.01956521739130435 0.005797101449275362
0 0.5831521739130435 0.5014492753623189 0.009782608695652175 0.002898550724637681
0 0.6505434782608696 0.5666666666666667 0.015217391304347827 0.007246376811594203
0 0.5508152173913043 0.4742753623188406 0.007065217391304348 0.002173913043478261
```

FONTE: O Autor (2025).

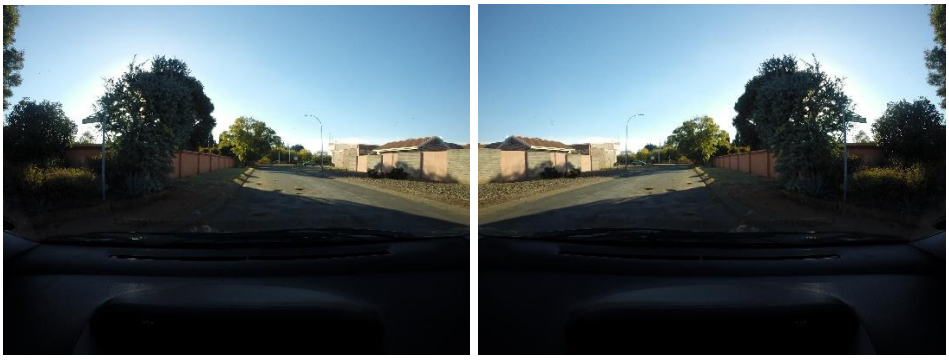
3.3 AUMENTAÇÃO DE DADOS

A seguir, detalhe-se os procedimentos de aumentação de dados aplicados nas imagens utilizadas neste trabalho. Esse procedimento busca aumentar o desempenho do modelo e o seu impacto será medido nas seções posteriores.

3.3.1 Espelhamento

Espelhou-se a imagem no eixo vertical, transferindo pontos do lado direito para o esquerdo e vice-versa. A Figura 15 exemplifica esse procedimento.

Figura 15 - Imagem original (esquerda) e imagem invertida (direita)



FONTE: O autor (ano).

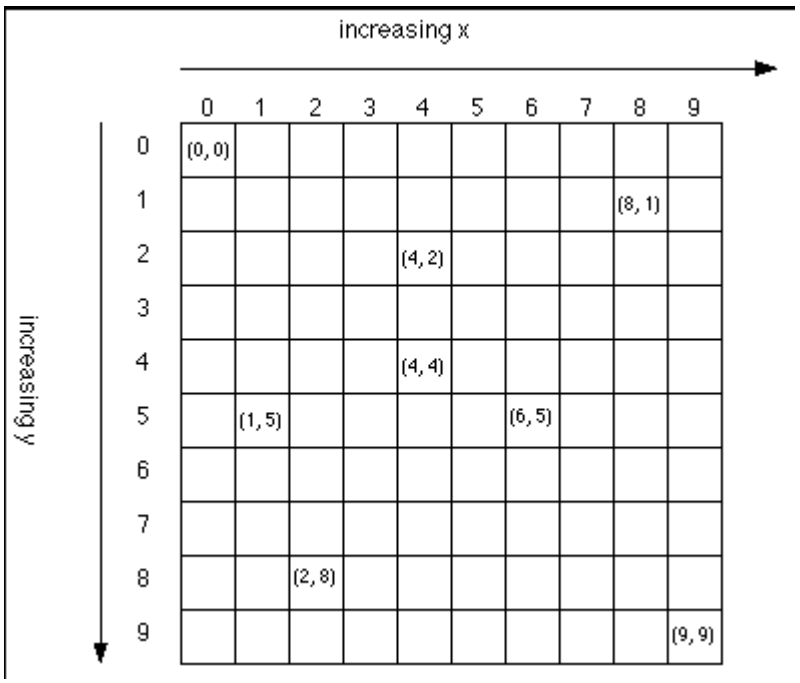
Com isso, de partida, tem-se um conjunto de imagens que é o dobro se comparado com as imagens originais. Outro ponto a se destacar, como as imagens foram geradas automaticamente utilizando o Darkmark, a própria aplicação garante que os arquivos texto sejam modificados para contemplar as novas posições dos buracos.

3.3.2 Rotações

Apesar de grandes variações não fazerem sentido prático para uma grande proporção de imagens usadas durante o treinamento, pequenas rotações podem ser utilizadas para aumentar o conjunto inicial de imagens, sem perda de coerência. Contudo, não possível aplicar esse tipo de transformação diretamente usando o Darkmark. Com isso, desenvolveu um algoritmo utilizando C++, mesma linguagem da aplicação, para criar rotação menores, onde é possível escolher o ângulo de rotação da imagem. Apesar de em teoria, ângulos reais serem possíveis, escolheu-se trabalhar apenas com ângulos inteiros. O código pode ser encontrado no repositório: https://lnk.ink/ZEXIr.

Para isso, utilizou-se a biblioteca Open CV para realizar as transformações nas imagens. O primeiro ponto importante, foi encontrar o centro da imagem para aplicar a rotação em relação ao centro. Grande parte das bibliotecas de visão computacional ou tratamento de imagens, considera como origem o canto superior esquerdo, como na figura abaixo.

Figura 16 - Eixo coordenado para uma aplicação em OpenCV

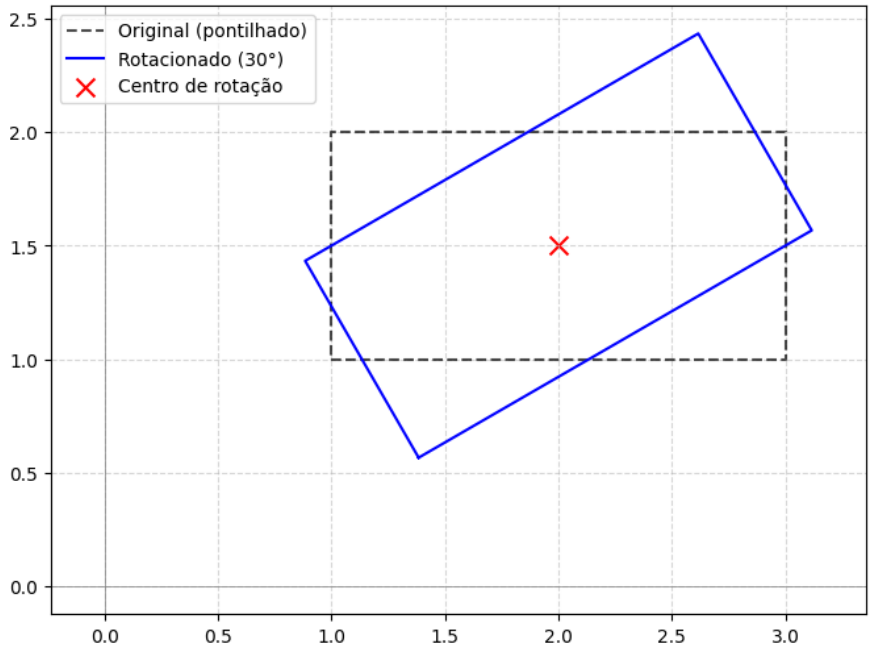


Fonte: Bradski (2000).

Devido a característica discreta do problema, pode ocorrer que não ocorra um centro verdadeiro na imagem. A figura acima exemplifica esse caso. Quando isso acontecer, escolhe-se a melhor aproximação inteira para o centro, arredondando-se para cima ou para baixo dependendo da imagem.

Com o centro identificado, todos os demais pixels foram rotacionados por um ângulo θ. É importante notar que o valor de θ pode variar muito com o problema em questão, mas para o escopo desse trabalho, trabalhou com uma variação não maior que 15 graus em modulo. A Figura 17 abaixo representa esse processo de forma esquemática.

Figura 17 - Representação de uma rotação em relação ao centro

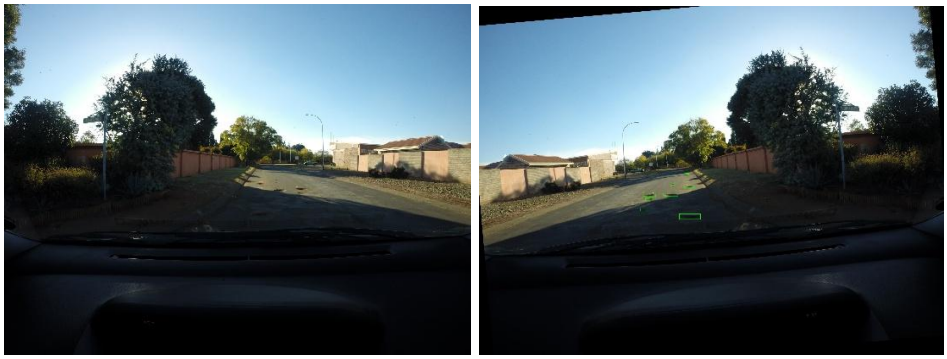


A equação abaixo foi utilizada para rotacionar os pixels da imagem em relação ao centro.

$$(x'y')^T = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} (xy)^T \quad (19)$$

Onde x' e y' são as novas coordenadas do pixel rotacionado, θ e o ângulo de rotação, e x e y as coordenadas originais. Um exemplo de imagem rotacionada e apresentada abaixo.

Figura 18 - Imagem original (esquerda) e imagem rotacionada em 5 graus (direita)



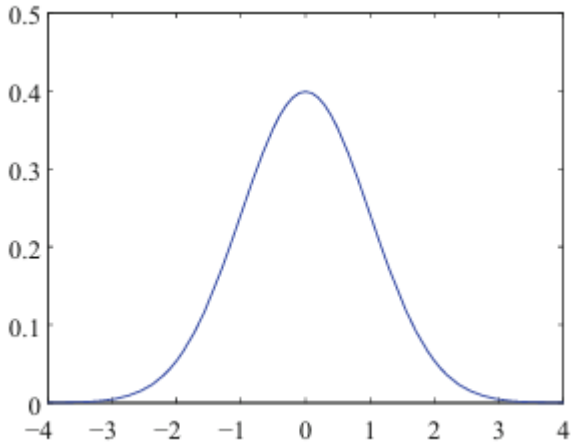
FONTE: O autor (ano).

Nota-se na imagem acima, que a imagem rotacionada perdeu alguns pixels na parte superior esquerda e inferior direita devido a rotação, sendo preenchida por pixels pretos. Durante o processo de rotação, os arquivos originais de texto ainda possuem suas coordenadas originais e, portanto, as caixas de contorno não representam mais a posição exata do buraco. Com isso, criou-se um algoritmo para também rotacionar os arquivos textos e as suas propriedades, garantindo que o YOLO consiga identificar os objetos, buracos, corretamente. Tomou-se o cuidado de codificar a possibilidade de buracos saírem parcialmente, ou totalmente, da imagem quando ela for rotacionada. Neste caso, a imagem e limitada pelas extremidades da imagem.

3.3.3 Aplicação de ruido gaussiano

Outra forma de aumentar o conjunto inicial de imagens e aplicar ruido nos pixels da imagem. O ruido será gerado de forma pseudoaleatória, e aplicado individualmente a cada pixel da imagem original. De forma intuitiva, isso garante que o modelo esteja preparado para variações na intensidade nos canais de cor, pois a adição de ruida aumenta a variabilidade da imagem. A figura abaixo representa uma distribuição normal padrão, ou seja, com média zero e desvio padrão igual a 1.

Figura 19 - Distribuição normal padrão



FONTE: Forsyth (2018)

O primeiro passo foi criar uma matriz com o mesmo tamanho da imagem original a ser transformada, e para cada um de seus pixels, calcular um ruido n_{ij} para uma imagem de tamanho $i \times j$. Essa matriz de ruído N e somada a imagem original

(M) em todos os seus canais, gerando uma imagem com ruído normal. Esse processo é descrito matematicamente na equação (20) abaixo.

$$M_{noise} = M + N \tag{20}$$

Figura 20 - Imagem original (esquerda) e imagem com ruido (direita)

FONTE: O autor (ano).

3.3.4 Automatização via script

Os métodos de aumentação mencionados acima são projetados para serem utilizados na aplicação Darkmark. Os códigos foram desenvolvidos em uma linguagem compatível com a aplicação. Como a aplicação é de código aberto, foi proposto implementar esses métodos para disponibilizá-los à comunidade. A inclusão dos métodos está atualmente em fase de teste.

Para este TCC, foi criado um script em *shell* que permite a aplicação das transformações em todas as imagens de uma pasta específica, evitando assim a aplicação manual do código.

Os códigos desenvolvidos durante esse trabalho de conclusão de curso podem ser encontrados no Github: https://lnk.ink/ZEXIr.

3.4 AVALIAÇÃO DE DESEMPENHO

Diversos cenários foram testados para identificar qual o melhor algoritmo e qual responderia melhor ao processo de aumentação das imagens. Um quadro resume todos os testes feitos com o conjunto de imagens iniciais.

QUADRO 3 – Testes realizados (versão *tiny*)

Arquitetura	Conj. de Imagens
V3 & V7	Original
V3 & V7	Original + Espelhado
V3 & V7	Original + Espelhado + Rotacionado
V3 & V7	Original + Espelhado + Rotacionado + Ruído

FONTE: O autor (2025).

3.5 EXPANSÃO DO CONJUNTO DE IMAGENS INICIAL

Após o período de testes, foi realizada a adição de um maior número de imagens obtidas da internet, às quais foram aplicadas todas as transformações anteriores de aumentação. Esse conjunto ampliado de imagens aumenta significativamente a variabilidade, tornando o processo de *overfitting* menos provável. As transformações para aumentação de dados tornam o procedimento ainda mais robusto.

O desempenho foi avaliado novamente, e o modelo com melhor performance será utilizado para identificar buracos e remendos em Curitiba e região. Até o presente momento deste TCC, não existe nenhum conjunto de imagens específico desta região, o que implica que os modelos treinados com dados de outros países não se adaptam bem às condições locais. Portanto, criou-se um conjunto de imagens para treinar e avaliar o desempenho do modelo desenvolvido neste TCC utilizando YOLO.

Tabela 4 - Tamanho e proporção do conjunto expandido de imagens

Classe	Tamanho	Proporção (%)
Treinamento	694	70
Validação	198	20
Teste	99	10

Fonte: O autor (2025).

3.6 AVALIAÇÃO EM IMAGENS DE CURITIBA

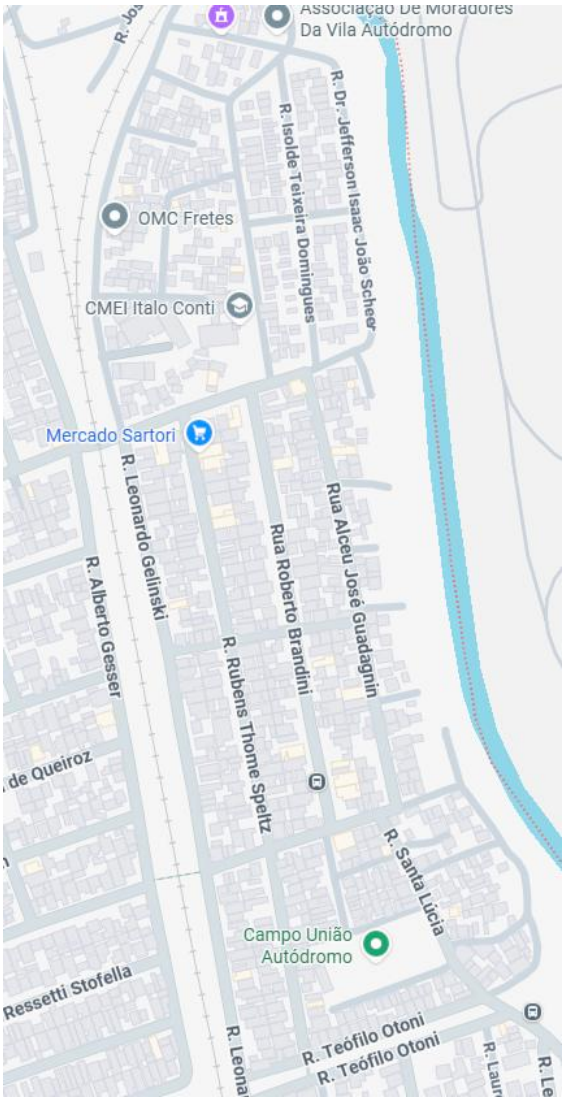
Assume-se que o modelo treinado nos passos anteriores deve possuir uma boa capacidade de identificar buracos nas condições apresentada na cidade. O trabalho anterior de Julio (2024), demonstrou que apenas com imagens retiradas da internet, um modelo capaz de generalizar as propriedades regionais é possível.

Espera-se que os processos de pré-processamento das imagens, gere um modelo ainda mais robusto. As imagens são incluídas no conjunto de dados de validação.

3.7 CAPTURA DE IMAGENS DE CURITIBA E REGIÃO

Apesar do trabalho de Julio (2024) demonstrar a capacidade de generalização sem utilizar dados locais, torna-se importante a criação de um conjunto de imagens da região metropolitana de Curitiba, pois existem especificidades que são características da nossa cidade. A região contemplada pode ser vista na imagem abaixo retirada do *Google Maps*.

Figura 21 - Região da coleta de imagens



Fonte: O autor (2025).

As vias dessa região apresentam um elevado grau de degradação, razão pela qual foi escolhida como local de estudo. As imagens foram capturadas em um dia ensolarado, por volta das 10 horas da manhã, utilizando um veículo e um celular fixado em um suporte no painel do carro. A seguir, é apresentado um exemplo na figura abaixo.

Figura 22 - Imagem da região de Curitiba

Fonte: O autor (2025).

3.8 TREINAMENTO EM IMAGENS DE CURITIBA COM PESOS DE MODELOS ANTERIORES

Espera-se que o desempenho do modelo melhore com a adição de um conjunto de imagens da região curitibana no conjunto de treinamento. Em comparação, o conjunto de dados coletados na região de Curitiba é 80% menor ao retirado na internet, sendo composto de 205 imagens no total, mantendo-se a proporção do conjunto anterior, ou seja, 80-20. Essas imagens foram avaliadas individualmente, buscando criar um conjunto de validação que se representa com

certa confiabilidade o conjunto de treinamento, afinal, trabalha-se com poucas imagens. Não se negligência, contudo, que a ação deliberada de separar manualmente o conjunto de validação possa criar um viés de validação. Não foi utilizado conjunto de validação diretamente, pois nenhuma mudança na arquitetura do Yolo foi feita durante esse trabalho.

O quadro abaixo demonstra um comparativo entre os diversos conjuntos utilizados nesse TCC, com as suas quantidades e porcentagens, divididos em treinamento, validação e teste.

QUADRO 4 – Métricas de performance (melhor)

Conjunto	**Escolha de Arquitetura e avaliação de *data augmentation***		**Conjunto expandido de imagens**		**Conjunto local (Curitiba)**	
Treinamento	25	80%	694	70%	144	70%
Validação			198	20%	41	20%
Teste	6	20%	99	10%	20	10%

FONTE: O autor (2025).

Ao final, o modelo foi retreinado com a totalidade do treinamento tanto do conjunto expandido como do conjunto retirado em Curitiba. Importante notar que durante o processo de escolha de arquitetura, optou-se por não trabalhar com conjunto de dados de validação, afinal deseja-se entender a sensibilidade real.

3.9 REAVALIAÇÃO DO MODELO

Por fim, avaliou-se o desempenho do modelo após o treinamento com imagens adquiridas localmente em comparação com os resultados anteriores. Espera-se uma melhora com a inclusão de novas imagens que representam melhor as características da nossa cidade.

4 RESULTADOS

4.1 ESCOLHA DA ARQUITETURA ADEQUADA

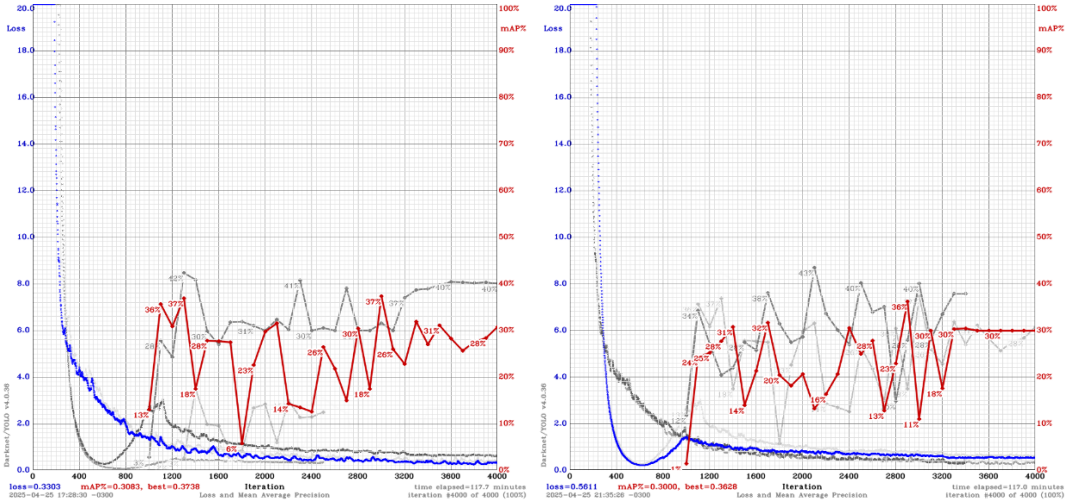
Com base em testes empíricos, determinou-se que a arquitetura que melhor responde ao processo de aumentação. Os resultados são apresentados a seguir para cada cenário aplicado. A Tabela 3 (seção 3.2) apresenta as quantidades e proporções usadas durante o processo de treinamento, validação.

Durante esta etapa do trabalho serão realizados quatro testes para identificar a melhor arquitetura para criar o modelo. Os testes são:

- Teste 1 – Conjunto original
- Teste 2 – Conjunto original + conjunto espelhado
- Teste 3 – Conjunto original + conjunto espelhado + conjunto inclinado
- Teste 4 – Conjunto original + conjunto espelhado + conjunto inclinado + conjunto com ruído

4.1.1 Teste 1

Figura 23 – V3-*tiny* (esquerda) e V7-*tiny* (direita)



FONTE: O autor (2025).

Ambos os modelos apresentaram desempenho semelhante ao usar o conjunto de imagens reduzidas como pode ser visto na Figura 23. O modelo V3-*tiny* alcançou

um desempenho de 32% mAP ao final do treinamento, enquanto o modelo v7-tiny conseguiu 30% mAP ao final do treinamento. Nota-se que apesar da perda do modelo cair rapidamente a partir da interação 400, aproximadamente, o modelo não conseguiu generalizar a identificação dos buracos, o que é evidenciado pela inconstância do gráfico para ambas as arquiteturas. Maiores detalhes podem ser encontrados no quadro abaixo.

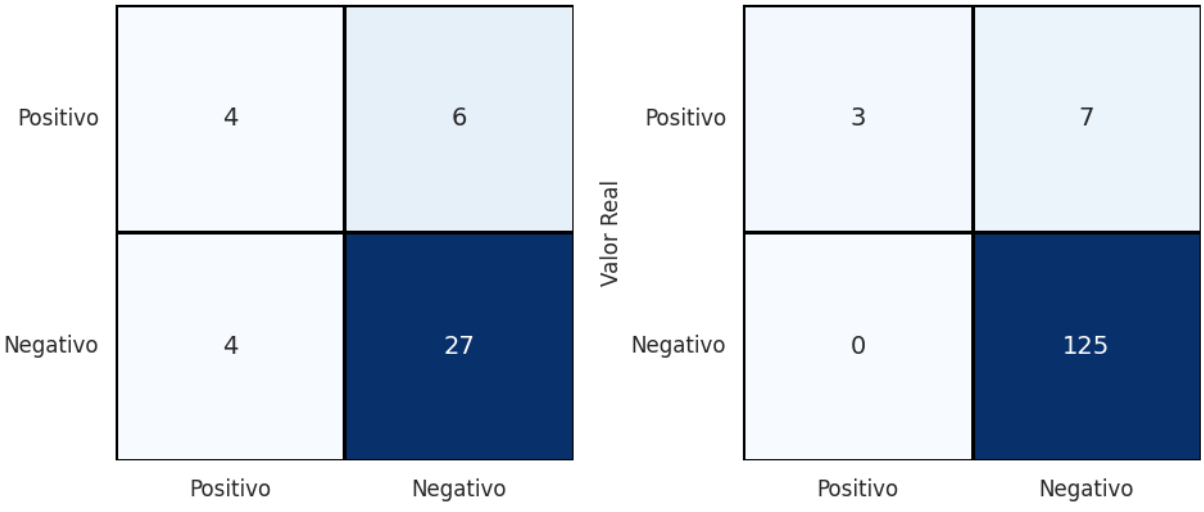
QUADRO 5 – Métricas de performance (melhor)

Métrica	**V3-*Tiny***	**V7-*Tiny***
mAP	36%	36%
Acurácia	75%	95%
Razão de erros	24%	5%
Precisão	50%	100%
Recall	40%	30%

FONTE: O autor (2025).

O quadro acima demonstra que a versão v7-*tiny* apresentou métricas superiores na comparação com a versão v3-*tiny*. Particularmente, o desempenho da versão mais recente do YOLO apresentou o dobro da precisão da versão anterior. Contudo, nota-se que a *Recall* de ambas as versões estão abaixo de 50%, o que pode influenciar na capacidade do modelo em identificar falso negativos. A Figura 24 abaixo apresenta uma matriz de confusão do teste.

Figura 24 - Matriz de confusão V3-*tiny* (esquerda) v7-*tiny* (direita)



FONTE: O autor (2025)

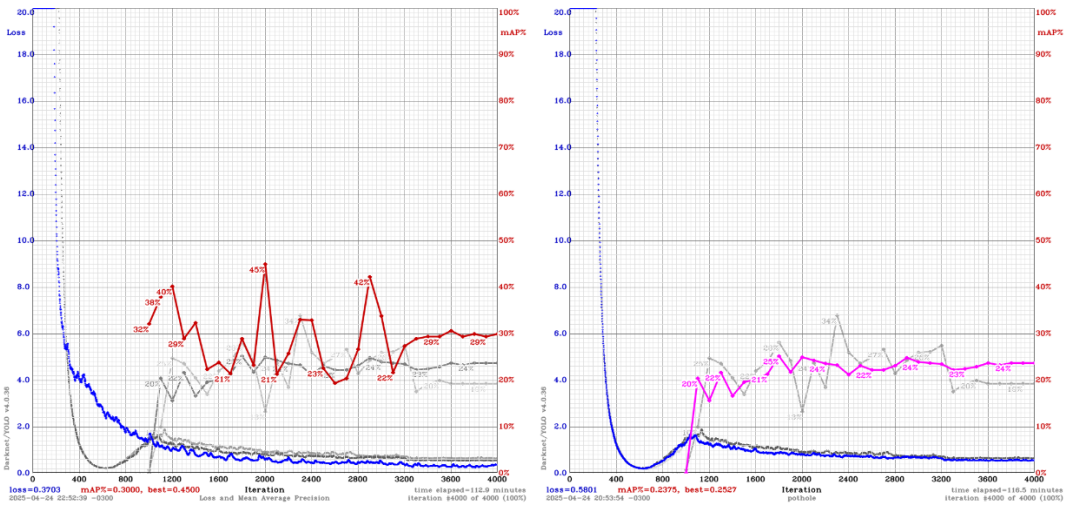
Nota-se na imagem acima, que grande parte das imagens que são fornecidas ao modelo não são compostas de buracos, ou seja, o asfalto apresenta boa qualidade (ausência de buracos), na área que compõe a imagem. No exemplo da Figura 25 abaixo, nota-se que ambos os modelos foram capazes de identificar buracos, contudo, o modelo v7-*tiny* está mais ajustado com a real dimensão do buraco, enquanto o modelo com arquitetura v3-*tiny* apresenta uma área maior, impactando no mAP.

Figura 25 – V3-*tiny* (esquerda) e V7-*tiny* (direita)

FONTE: O autor (2025).

4.1.2 Teste 2

Figura 26 – V3-*tiny* (esquerda) e V7-*tiny* (direita)



FONTE: O autor (2025).

Nota-se na Figura 26, que novamente, nenhuma arquitetura foi capaz de generalizar de forma aceitável a detecção de buracos. O YOLO V3-*tiny*, apresentou ao final do treinamento um desempenho de 30% mAP, enquanto V7-*tiny* manteve-se a 24% mAP. Contudo, a arquitetura v3 conseguiu durante a iteração 2000, aproximadamente, um pico de 50% mAP. Maiores detalhes podem ser encontrados no quadro abaixo.

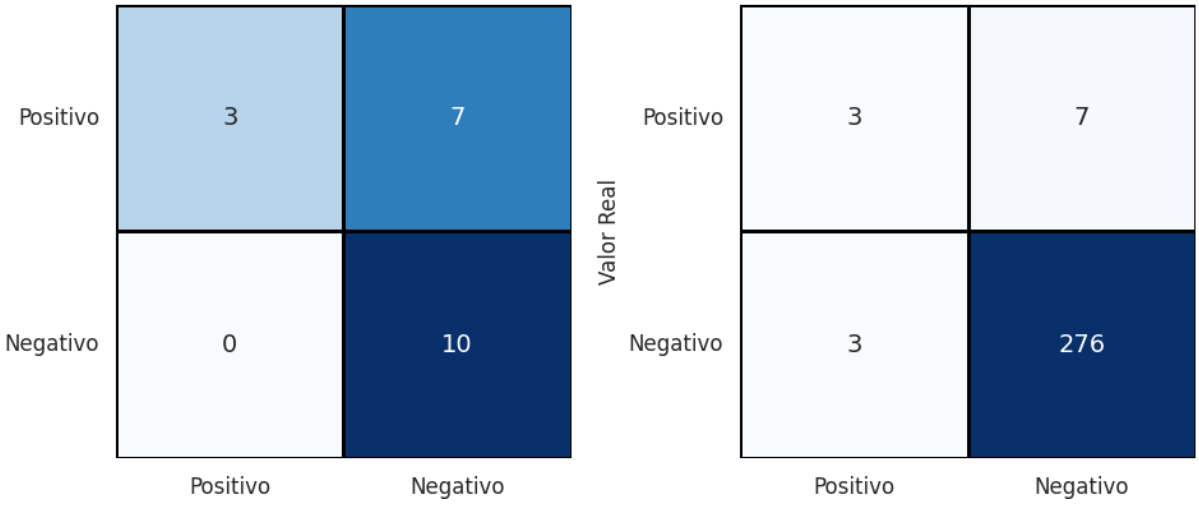
QUADRO 6 – Métricas de performance (melhor)

Métrica	**V3-*Tiny***	**V7-*Tiny***
mAP	50%	25%
Acurácia	65%	95%
Razão de erros	35%	3%
Precisão	100%	50%
Recall	30%	30%

FONTE: O autor (2025).

O quadro acima demonstra que a versão v7-*tiny* apresentou acurácia, razão de erros superiores na comparação com a versão v3-*tiny*. Contudo, mAP da arquitetura v3-*tiny* foi maior do que a versão v7-*tiny*. Este fato decorre do melhor desempenho da versão v3-*tiny* na métrica IoU, onde apresentou 52% enquanto outra arquitetura apresentou apenas 28%. A arquitetura v3-*tiny* apresentou melhor precisão. Ambas apresentaram o mesmo recall.

Figura 27 - Matriz de confusão V3-*tiny* (esquerda) v7-*tiny* (direita)



FONTE: O autor (2025)

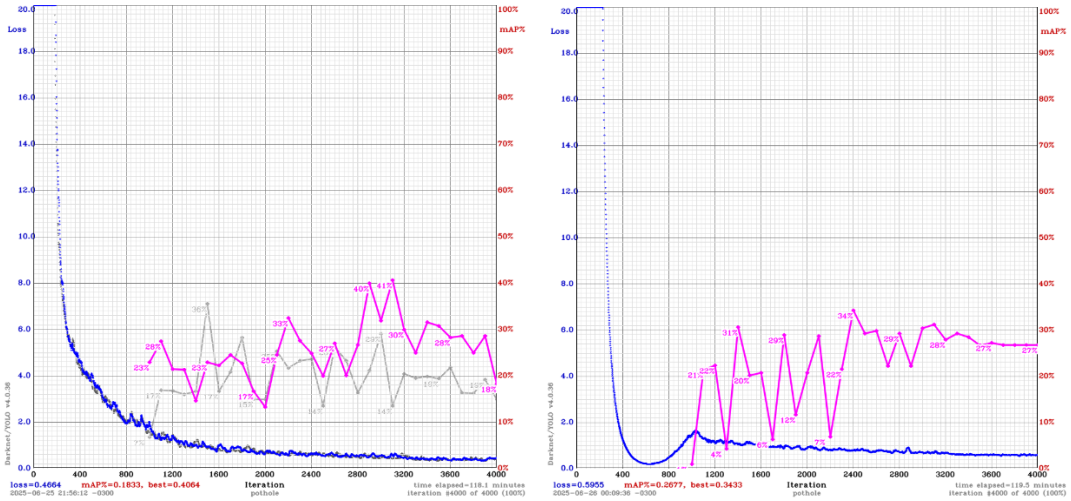
Nota-se na imagem acima, que grande parte das imagens que são fornecidas ao modelo não são compostas de buracos, ou seja, o asfalto apresenta boa qualidade (ausência de buracos), na área que compõe a imagem. No exemplo da Figura 28 abaixo, a arquitetura v3-*tiny* não foi capaz de identificar nenhum buraco, enquanto a arquitetura v7-*tiny* conseguiu, contudo também identificou uma boca de lobo como um buraco, sendo classificado como um falso positivo.

Figura 28 – V3-*tiny* (esquerda) e V7-*tiny* (direita)

FONTE: O autor (2025).

4.1.3 Teste 3

Figura 29 – V3-*tiny* (esquerda) e V7-*tiny* (direita)



FONTE: O autor (2025).

Nota-se na Figura 29, que novamente, nenhuma arquitetura foi capaz de generalizar de forma aceitável a detecção de buracos. O YOLO V3-*tiny*, apresentou

ao final do treinamento um desempenho de 18% mAP, enquanto V7-*tiny* manteve-se a 27% mAP. Contudo, a arquitetura v3-*tiny* conseguiu durante a sua melhor iteração um pico de 37% mAP e a outra arquitetura um pico de 33% de mAP. Maiores detalhes podem ser encontrados no quadro abaixo.

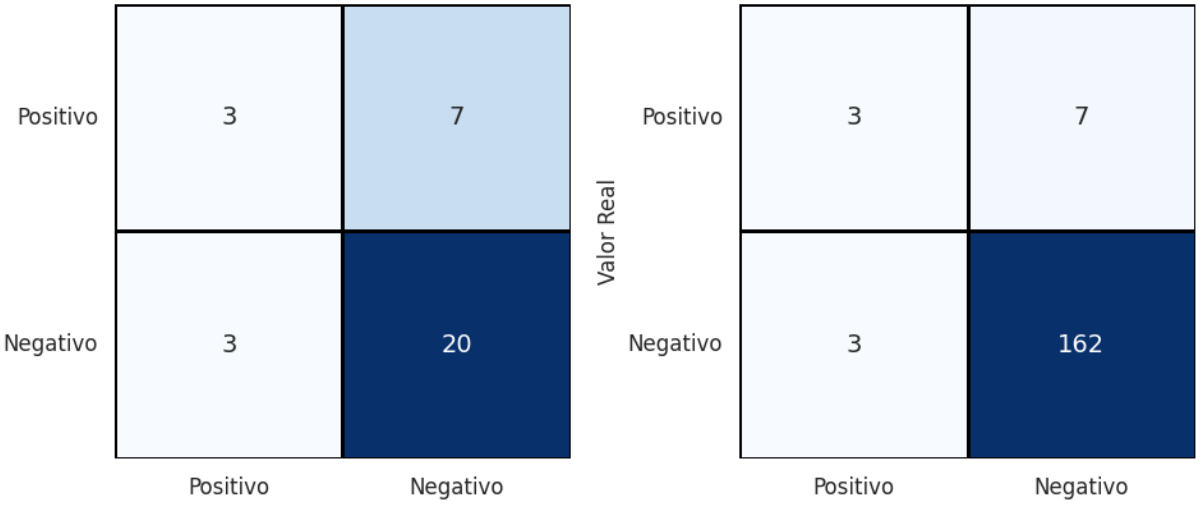
QUADRO 7 – Métricas de performance (melhor)

Métrica	**V3-*Tiny***	**V7-*Tiny***
mAP	37%	33%
Acurácia	69%	94%
Razão de erros	30%	5%
Precisão	50%	50%
Recall	30%	30%

FONTE: O autor (2025).

O quadro acima demonstra que a versão v7-*tiny* apresentou acurácia, razão de erros superiores na comparação com a versão v3-*tiny*. Esse último, apresentou um melhor desempenho em mAP, devido ao seu melhor desempenho em IoU, onde apresentou 32% enquanto outra arquitetura apresentou apenas 30%. A precisão foi a mesma no teste para ambas as arquiteturas. Contudo, nota-se que a *recall* de ambas as versões estão abaixo de 50. A Figura 30 abaixo apresenta uma matriz de confusão do teste.

Figura 30 - Matriz de confusão V3-*tiny* (esquerda) v7-*tiny* (direita)



FONTE: O autor (2025)

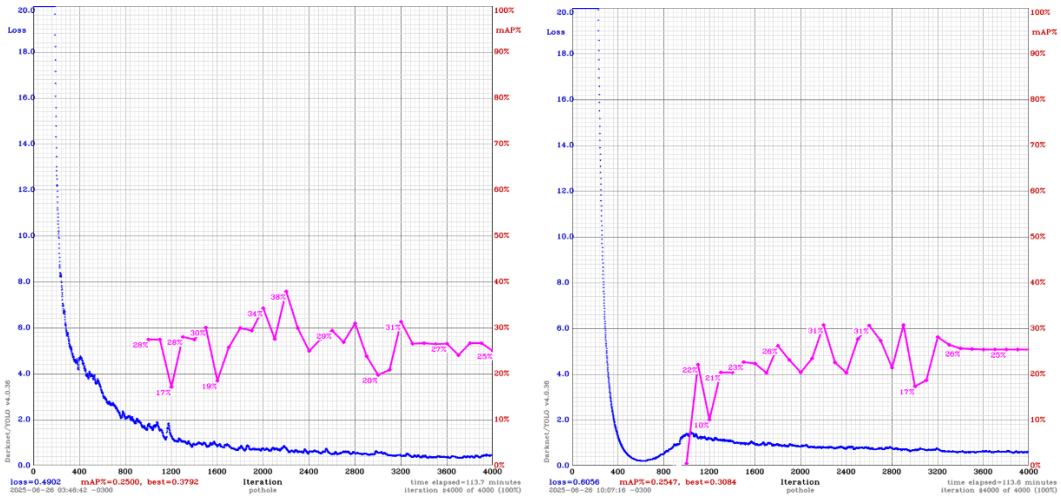
Nota-se na imagem acima, que grande parte das imagens que são fornecidas ao modelo não são compostas de buracos, ou seja, o asfalto apresenta boa qualidade (ausência de buracos), na área que compõe a imagem. No exemplo da Figura 31 abaixo, ambos modelos conseguiram identificar o buraco corretamente.

Figura 31 – V3-*tiny* (esquerda) e V7-*tiny* (direita)

FONTE: O autor (2025).

4.1.4 Teste 4

Figura 32 – V3-*tiny* (esquerda) e V7-*tiny* (direita)



FONTE: O autor (2025).

Nota-se na Figura 32, que novamente, nenhuma arquitetura foi capaz de generalizar de forma aceitável a detecção de buracos utilizando o conjunto reduzido de imagens. Ambas as arquiteturas desempenharam de forma similar quando

comparadas usando mAP. Contudo, a arquitetura v3-*tiny* conseguiu durante a sua melhor iteração um pico de 37% mAP e a outra arquitetura um pico de 32% de mAP. Maiores detalhes podem ser encontrados no quadro abaixo.

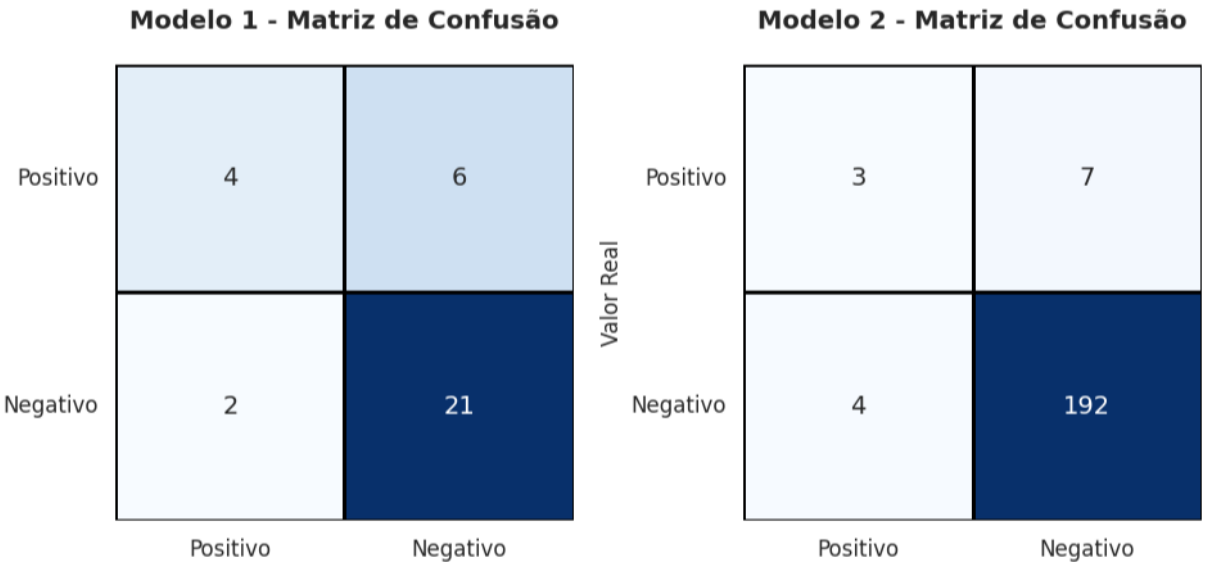
QUADRO 8 – Métricas de performance (melhor)

Métrica	**V3-*Tiny***	**V7-*Tiny***
mAP	37%	32%
Acurácia	76%	95%
Razão de erros	24%	5%
Precisão	67%	43%
Recall	40%	30%

FONTE: O autor (2025).

O quadro acima demonstra que a versão v7-*tiny* apresentou acurácia, razão de erros superiores na comparação com a versão v3-*tiny*. Esse último, apresentou um melhor desempenho em mAP, devido ao seu melhor desempenho em IoU, onde apresentou 43% enquanto outra arquitetura apresentou apenas 28%. A precisão foi a mesma no teste para ambas as arquiteturas. Contudo, nota-se que a Recall de ambas as versões estão abaixo de 50, apesar da melhora na versão v3-*tiny*. A Figura 33 abaixo apresenta uma matriz de confusão do teste.

Figura 33 - Matriz de confusão V3-*tiny* (esquerda) v7-*tiny* (direita)



FONTE: O autor (2025)

Nota-se na imagem acima, que grande parte das imagens que são fornecidas ao modelo não são compostas de buracos, ou seja, o asfalto apresenta boa qualidade (ausência de buracos), na área que compõe a imagem. No exemplo da Figura 31 abaixo, ambos modelos conseguiram identificar o buraco corretamente. Contudo, a versão v7-*tiny* identificou erroneamente dois buracos inexistentes.

Figura 34 – V3-*tiny* (esquerda) e V7-*tiny* (direita)

FONTE: O autor (2025).

4.1.5 Avaliação e escolha da arquitetura

Com os testes acima, deseja-se identificar qual arquitetura apresenta a melhor resposta ao processo de aumento de dados. Isso é particularmente importante para o presente estudo, pois buracos são eventos raros. Ou seja, espera-se que a condição do asfalto na grande maioria da malha viária não apresente falhas.

Os resultados demonstram que ambas as arquiteturas não foram capazes de generalizar bem a identificação de buracos quando trabalhando com o conjunto reduzido de imagens. A arquitetura v3-*tiny* se saiu melhor na métrica mAP, mas sem apresentar grande de margem quando comparada com a concorrente. Já a arquitetura v7-*tiny*, de forma geral, apresentou melhor precisão, acurácia e razão de erros durante os testes. Apenas durante a segunda rodada de avalição, a arquitetura v3-*tiny* apresentou melhor precisão que a arquitetura v7-*tiny*. Levando em consideração o número reduzido de imagens que se trabalhou, e seus eventos que são raros, que a arquitetura v3-*tiny* tenha se saído melhor que a outra em precisão.

QUADRO 9 – Comparação entre testes

	Teste 1		Teste 2		Teste 3		Teste 4	
Métrica	**V3-*Tiny***	**V7-*Tiny***	**V3-*Tiny***	**V7-*Tiny***	**V3-*Tiny***	**V7-*Tiny***	**V3-*Tiny***	**V7-*Tiny***
mAP	36%	36%	50%	25%	37%	33%	37%	32%
Acurácia	75%	95%	65%	95%	69%	94%	76%	95%
Razão de erros	24%	5%	35%	3%	30%	5%	24%	5%
Precisão	50%	100%	100%	50%	50%	50%	67%	43%
Recall	40%	30%	30%	30%	30%	30%	40%	30%

Fonte: O autor (2025)

Os resultados demonstram um cenário de equilíbrio entre as arquiteturas e, portanto, não se podendo definir categoricamente a melhor arquitetura a ser utilizada.

Dado a situação de igualdade para ambas as arquiteturas, definiu-se que a versão v3-*tiny* será utilizada para criar o modelo desse trabalho. Isso se deve por dois motivos principais: o melhor desempenho dessas arquiteturas para mAP e para o *recall*.

4.2 TREINAMENTO COM CONJUNTO EXPANDIDO DE IMAGENS

Essa etapa utilizará a arquitetura v3-*tiny* para treinar um modelo capaz de identificar buracos em Curitiba e na região metropolitana.

4.2.1 Conjunto expandido de imagens

Foi utilizado um conjunto de imagens composto por 480 imagens retiradas do repositório Ninja (2025). Esse conjunto de imagens foi retirada na África do Sul da perspectiva do motorista, situação que foi replicada quando feito o conjunto de imagens para esse trabalho da região metropolitana de Curitiba. Apesar das imagens possuírem anotações, elas não possuem o mesmo padrão utilizado no YOLO, portanto, fez-se necessário anotá-las novamente com a ferramenta DarkMark. Essas imagens anotadas também serão compartilhadas com a comunidade.

Foi feita a aumentação de dados do conjunto inicial, com os processos descritos na seção 3.3. Com isso, o treinamento foi realizado com um total de 10560 imagens. O resumo das transformações é apresentado no quadro abaixo.

QUADRO 10 – Métricas de performance (melhor)

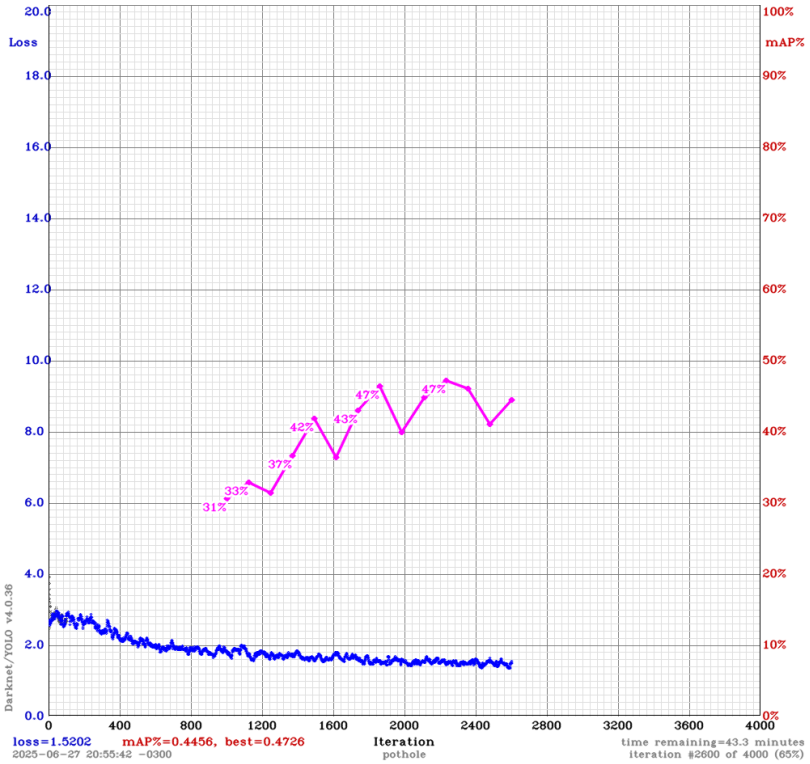
Conjunto de imagens	Quantidade
Original	991
Espelhada	1982
Inclinada	9910
Ruído	19820

FONTE: O autor (2025).

Durante o treinamento dessa nova versão do modelo, utilizou-se os pesos encontrados durante os testes feitos na seção anterior. Novos pesos devem otimizados durante novas épocas com esse novo conjunto de imagens. O desempenho do novo modelo é discutido a seguir.

4.2.2 Desempenho do modelo

Figura 35 – Desempenho V3-*tiny* com conjunto expandido



FONTE: O autor (2025).

Nota-se na imagem acima, que o modelo com mais imagens consegue generalizar melhor o problema proposto de identificar buracos em vias públicas. Enquanto os modelos anteriores, não conseguiam passar de 30% mAP para o conjunto reduzido, o modelo da imagem acima atingiu 45% mAP, momento quando

foi pausado o treinamento para evitar *overfitting*. Essa técnica se chama parada antecipada (*early stopping)* e é muito utilizada para evitar que o modelo decorre as propriedades das imagens ao invés de aprender os padrões desejados.

Os resultados são apresentados no quadro abaixo.

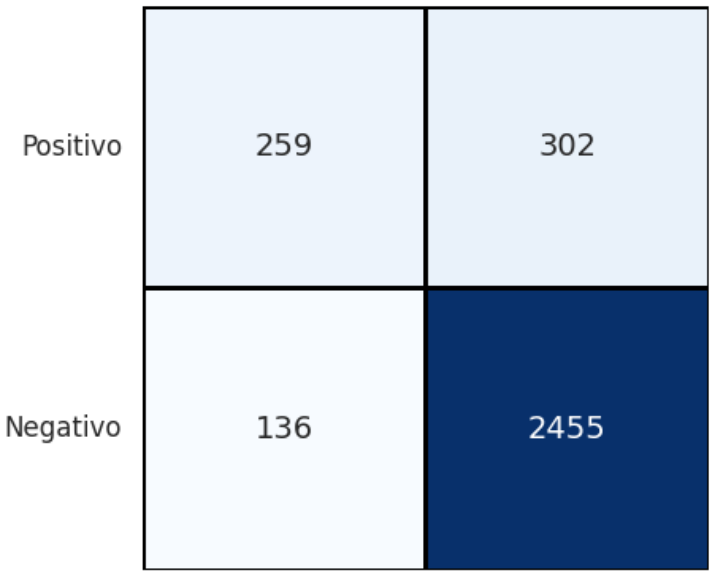
QUADRO 11 – Métricas de performance depois *early stopping*

Métrica	Desempenho
mAP	48%
Acurácia	86%
Razão de erros	13%
Precisão	66%
Recall	46%

FONTE: O autor (2025)

Nota-se que houve uma melhoria na métrica mAP, superando todas as interações dos testes anteriores, com exceção do teste 2 da seção 4.1 cujo melhor resultado foi de 50%. As demais métricas apresentaram melhoria em comparação com os testes anteriores. A seguir apresenta-se a matriz de confusão do treinamento para o melhor resultado do treinamento.

Figura 36 - Matriz de confusão com conjunto expandido de imagens



FONTE: O autor (2025)

A matriz de confusão acima mostra que grande parte das imagens utilizadas apresentam uma proporção baixas em comparação com a área total da imagem. Essa

é composta em sua maioria por uma malha sem danos visíveis, o que explica a grande quantidade de verdadeiro negativos na Figura 36. Agora apresenta-se dois exemplos do conjunto de validação que demostram que o modelo consegue inferir os buracos corretamente, contudo sem conseguir criar uma caixa delimitadora ajustada corretamente.

Figura 37 – Exemplo de inferência conjunto expandido de imagens

FONTE: O autor (2025).

Na Figura 37 os embora os buracos tenham sido classificados, a confiabilidade do modelo permaneceu abaixo de 70% em ambas as imagens. Isso indica que, mesmo em situações evidentes, o modelo ainda apresenta incertezas ao identificar buracos. Agora apresenta-se duas tentativas de inferir buracos em imagens da região de Curitiba.

Figura 38 – Exemplo de inferência conjunto de Curitiba e região

FONTE: O autor (2025).

Nas imagens apresentadas, o modelo demonstrou limitações na correta identificação dos buracos. Na foto à esquerda, dois buracos foram erroneamente inferidos como falso positivos, enquanto os dois buracos reais, claramente visíveis, não foram detectados. Já na foto à direita, nenhum buraco foi identificado, mesmo sendo evidente a presença de um buraco no centro da imagem, configurando um falso negativo.

É importante salientar que apesar do modelo estar sendo usado para inferir imagens na região de Curitiba, nenhuma imagem foi utilizada ainda para treinar a CNN nessa parte do trabalho. Na verdade, isso indica que existe valor na confecção de um conjunto de dados específico para a realidade encontrada na nossa cidade. O quadro a seguir demonstra o desempenho do modelo para todas as imagens de validação da região metropolitana de Curitiba.

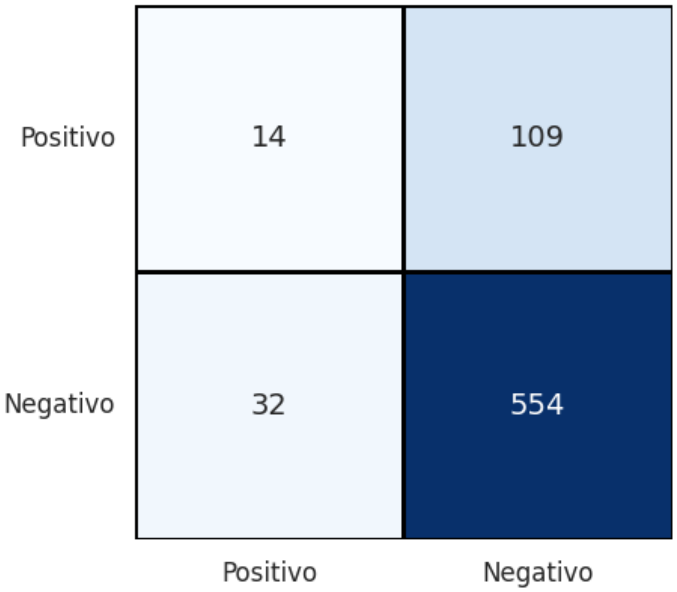
QUADRO 12 – Métricas de performance para imagens de Curitiba

Métrica	**Desempenho**
mAP	7%
Acurácia	80%
Razão de erros	19%
Precisão	30%
Recall	12%

FONTE: O autor (2025)

Nota-se que todas as métricas apresentaram queda, sendo que mAP e a precisão foram os mais afetados, caindo de 48% para 7% e 66% para 30%, respectivamente.

Figura 39 - Matriz de confusão inferência com imagens de Curitiba



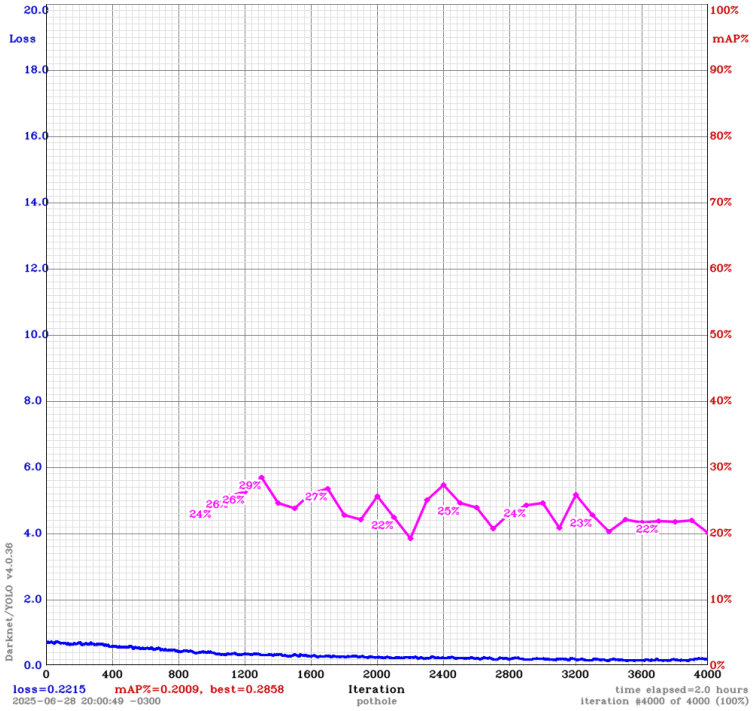
FONTE: O autor (2025)

Na Figura 39, nota-se que a proporção entre verdadeiros positivos diminui em relação ao número de falso negativos e falso positivos, impactando diretamente a precisão e o *recall*.

4.3 TREINAMENTO COM IMAGENS DE CURITIBA E AVALIAÇÃO DO MODELO

Agora as imagens adquiridas foram rotuladas e usadas para treinar o modelo novamente, utilizando a melhor época.

Figura 40 – Desempenho V3-*tiny* com conjunto de imagens de Curitiba treinado com dados obtidos na região



FONTE: O autor (2025).

Figura 40, observa-se que a mAP não conseguiu retornar ao patamar apresentado durante a seção 4.2, contudo nota-se uma melhora em relação ao modelo aplicado as imagens de Curitiba no quadro 11. Embora o processo de treinamento tenha sido concluído, apenas os pesos correspondentes à melhor época serão considerados para a etapa de inferência. O quadro a seguir resume as métricas obtidas durante o treinamento.

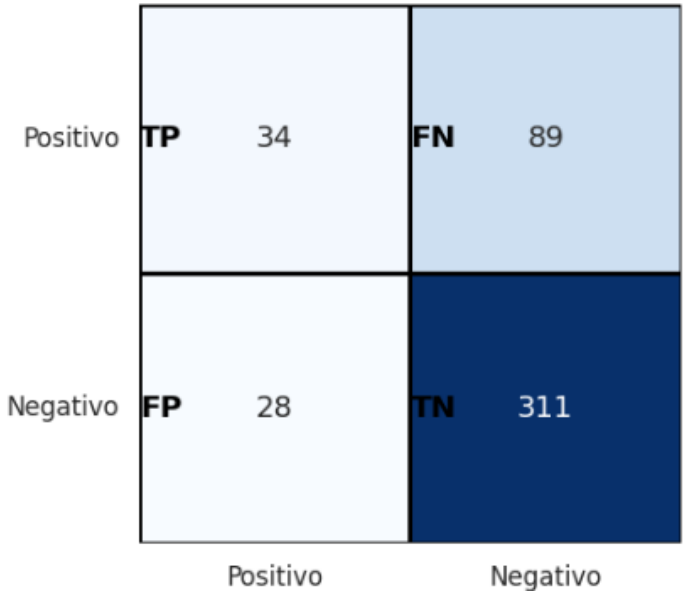
QUADRO 13 – Métricas de performance

Métrica	**Desempenho**
mAP	28%
Acurácia	75%
Razão de erros	25%
Precisão	55%
Recall	28%

FONTE: O autor (2025)

Em relação ao quadro 11, todas as métricas apresentaram uma melhora significativa, com exceção da razão de erros que houve uma pequena piora de 6%.

Figura 41 - Matriz de confusão para conjunto de dados de Curitiba para modelo treinado com dados obtidos na região



FONTE: O autor (2025)

Na Figura 41, observa-se uma alta incidência de falsos negativos, indicando que o modelo enfrenta dificuldades em identificar buracos reais, mesmo após múltiplas iterações. Considerando que para uma aplicação de carro autônomo se exige um *recall* de 95% para objetos fixos, o modelo apresentado nesse estudo não possui esse desempenho. Observa-se que o número de verdadeiros negativos permanece elevado devido à sua predominância nas imagens, enquanto os verdadeiros positivos apresentam números reduzidos, refletindo a raridade desses casos.

Figura 42 – Exemplo de inferência conjunto de Curitiba e região usando modelo treinado com imagens da obtidas

FONTE: O autor (2025).

A Figura 42 demonstra que o novo modelo é melhor em inferir buracos na região metropolitana de Curitiba. Os buracos foram identificados em ambas as imagens, fato que não ocorreu com o modelo puramente treinado com dados coletados da internet. Contudo, ambos apresentam uma confiabilidade baixa, menor que 70%. Além disso, na imagem da direita, apenas um buraco foi identificado corretamente, sendo que o modelo não foi capaz de inferir a segunda ocorrência. Nota-se, também, na imagem da esquerda, que apesar de identificar o buraco corretamente, a caixa delimitadora não está bem ajustado ao tamanho real do buraco.

4.4 COMPARAÇÃO COM TRABALHOS CORRELATOS

É de particular interesse comparar os resultados obtidos por esse trabalho e os de Julio (2024). Primeiro é importante ressaltar as diferenças entre os dois métodos de trabalho.

Apesar de ambos terem como objeto de estudo a identificação de buracos, os materiais utilizados foram distintos. Enquanto Julio (2024) utilizou um sistema em *cloud* para treinar o seu modelo, treinando seu modelo na plataforma Google Colab. Por outro lado, esse trabalho utilizou um computador local com placa de vídeo dedicada. Outra diferença é a arquitetura utilizada, aquele trabalho utilizou a arquitetura v4-*tiny*, enquanto esse utilizou a v3-*tiny*. Outro ponto fundamental, é a

diferença entre o conjunto de imagens utilizados que foi distinto, com origens diferentes. Contudo, pode-se fazer uma comparação quantitativa dos dois conjuntos, como apresentado no quadro abaixo.

QUADRO 14 – Quantidade de imagens: este trabalho (esquerda) e no trabalho correlato (direita)

Este trabalho	Trabalho correlato
1196	1183

FONTE: O autor (2025)

Este trabalho utilizou 1196 imagens, sendo 991 retiradas de Ninja (2025) e outras 205 obtidas no bairro do Cajuru enquanto Julio (2024) trabalha com um total de 1183 imagens obtidas unicamente da internet, através de diversas fontes. O quadro 13 resume esses números.

Escolheu-se um proporção diferente para os conjuntos de teste, validação e treinamento neste trabalho e no trabalho de Julio (2024). Enquanto este reservou 80% do seu conjunto de imagens para treinamento, 10% para validação e 10% para teste; este trabalho optou por separar 70% para treinamento, 20% para validação e 10% para teste. O comparativo quantitativo é apresentado no quadro abaixo.

QUADRO 15 – Métricas de performance (melhor)

Conjunto	Este trabalho		Trabalho correlato	
Treinamento	837	70%	946	80%
Validação	239	20%	118	10%
Teste	120	10%	117	10%

FONTE: O autor (2025).

É importante notar que enquanto esse trabalho escolheu manualmente as imagens dos conjuntos de teste e validação, sem, contudo, fazer uma análise aprofundada das imagens, o trabalho de Julio (2024) fez uma seleção randômica.

O quadro 14 abaixo compara a precisão média, precisão e *recall* em ambos os trabalhos.

QUADRO 16 – Comparação de performance: este trabalho (esquerda) e no trabalho correlato (direita)

	Este trabalho	**Trabalho correlato**
mAP	28%	60%
Precisão	55%	86%
Recall	28%	52%

FONTE: O autor (2025)

Nota-se que o modelo criado por Julio (2024) foi consideravelmente superior em todas as métricas, alcançando picos de precisão média de 60% enquanto este trabalho alcançou no máximo 47%. A precisão foi maior naquele trabalho, alcançando 86% versus 55% obtidos nesse trabalho. Por fim, o *recall* também foi maior no trabalho correlato, chegando a 52% contra 28%.

5 CONSIDERAÇÕES FINAIS

Os resultados obtidos com diferentes arquiteturas do YOLO (v3-*tiny* e v7-*tiny*) e conjuntos de dados variados, incluindo técnicas de aumento de dados (espelhamento, rotação e aplicação de ruído). Os principais achados foram:

Ambas as arquiteturas (v3-*tiny* e v7-*tiny*) apresentaram desempenho semelhante em métricas como mAP, precisão e recall quando testadas com um conjunto reduzido de imagens. A v3-tiny destacou-se em mAP e recall, enquanto a v7-*tiny* teve melhor acurácia e razão de erros.

A escolha pela v3-*tiny* para o modelo final foi baseada em seu melhor desempenho em mAP e *recall*, métricas críticas para a detecção de buracos.

O aumento de dados melhorou a generalização do modelo, especialmente quando combinado com um conjunto expandido de imagens. Inicialmente, o conjunto de treinamento possuía 991 imagens, coletadas principalmente na Africa do Sul, Ninja (2025). Após o processo de aumentação, o modelo foi treinado com 10.560 imagens atingiu 48% de mAP, superando os resultados anteriores.

Técnicas como espelhamento e rotação aumentaram a variabilidade do conjunto de dados, reduzindo o risco de *overfitting*. Não foi possível identificar se houve melhora significativa devido a aumentação de dados para esse estudo.

O modelo inicial, treinado apenas com imagens de outras regiões, teve desempenho insatisfatório em imagens locais (mAP de 7%). Após incluir imagens de Curitiba no treinamento, o mAP subiu para 28%, demonstrando a importância de dados representativos da região.

Apesar da melhora, o modelo ainda apresentou limitações, como falsos negativos e caixas delimitadoras mal ajustadas. O modelo conseguiu identificar buracos em imagens de Curitiba, mas com confiabilidade abaixo de 70%, indicando a necessidade de refinamento.

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Para trabalhos futuros, recomenda-se ampliar a base de imagens de Curitiba e região, contemplando diferentes condições climáticas e horários, a fim de fortalecer a robustez do modelo. Também é sugerido testar novas versões do YOLO, como o

YOLOv8, e explorar arquiteturas alternativas, como o EfficientDet, para avaliar desempenho e eficiência em diferentes cenários. Ajustar hiperparâmetros e empregar técnicas avançadas de aumento de dados, incluindo o uso de GANs (Generative Adversarial Network), pode contribuir para ganhos de precisão. A integração do modelo a aplicativos móveis ou sistemas embarcados em veículos possibilitaria a detecção em tempo real e o mapeamento de buracos, permitindo conexão com plataformas como o Google Maps. Além disso, realizar testes com vídeos dinâmicos capturados em movimento e adaptar o modelo para lidar com desfoques e variações de velocidade reproduz condições reais de uso. Por fim, vale investigar aplicações do modelo por órgãos públicos para auxiliar no planejamento e priorização de reparos em vias urbanas, integrando-o a sistemas de gestão municipal.

Utilizar as imagens obtidas durante esse trabalho para criar um modelo que tenha melhor desempenho para as nossas peculiaridades. As imagens podem ser encontradas em https://lnk.ink/88EKc (github) e os códigos desenvolvidos podem ser encontrados em https://lnk.ink/ZEXlr .

REFERÊNCIAS

AGGARWAL, C. C. Neural Networks and Deep Learning: A Textbook. Cham: Springer, 2018.

DNIT. Manual de pavimentação. 3. ed. Brasília: Departamento Nacional de Infraestrutura de Transportes, 2006.

SZELISKI, R. Computer Vision: Algorithms and Applications. London: Springer, 2011.

BRADSKI, G. The OpenCV Library. Dr. Dobb's Journal of Software Tools, v. 25, n. 11, p. 120-126, 2000.

GIRSHICK, R. et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, p. 580-587, 2014. Disponível em: https://ieeexplore.ieee.org/document/6909475. Acesso em: 7 jul. 2025.

Kaur, P. et al. Data Augmentation for Object Detection: A Review. IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), Lansing, MI, p. 537-543, 2021. Disponível em: https://ieeexplore.ieee.org/document/9531849. Acesso em: 30 jun. 2025.

CAON, F. D. V. Detecção e mapeamento de buracos em vias públicas asfaltadas utilizando visão computacional por meio de um sistema embarcado instalado em veículo automotivo. 2019. Trabalho de Conclusão de Curso (Engenharia da Computação) – Universidade do Vale do Rio dos Sinos, São Leopoldo, 2019.

JULIO, E. D. S. Detecção e mapeamento de defeitos no pavimento das rodovias. 2024. Trabalho de Conclusão de Curso (Engenharia Elétrica) – Universidade Federal do Paraná, Curitiba, 2024.

ANAND, S. et al. Crack-Pot: Autonomous Road Crack and Pothole Detection. In: Digital Image Computing: Techniques and Applications (DICTA), Melbourne, 2018. IEEE, 2018. p. 1-6.

TALLITA PASSOS, B. et al. Detecção de buracos em pavimento asfáltico com base em Processamento Digital de Imagens e Deep Learning. In: Computer on the Beach (COTB), Florianópolis, 2021. UNIVALI, 2021. p. 422-427. Disponível em: https://siaiap32.univali.br/seer/index.php/acotb/article/view/17434. Acesso em: 29 nov. 2023.

LIN, T.-Y. et al. Microsoft COCO: Common Objects in Context. 2014. arXiv:1405.0312. Disponível em: https://arxiv.org/abs/1405.0312. Acesso em: 25 jun. 2025.

NINJA, D. Visualization Tools for Road Pothole Images Dataset. 2025. Dataset. Disponível em: https://datasetninja.com/road-pothole-images. Acesso em: 27 jun. 2025.

REDMON, J.; FARHADI, A. YOLOv3: An Incremental Improvement. 2018. Technical Report. Disponível em: https://pjreddie.com/darknet/yolo/. Acesso em: 1 jun. 2025.

WANG, C.-Y. et al. *YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors*. 2023. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Disponível em: https://ieeexplore.ieee.org/document/10204762. Acesso em: 1 jun. 2025.