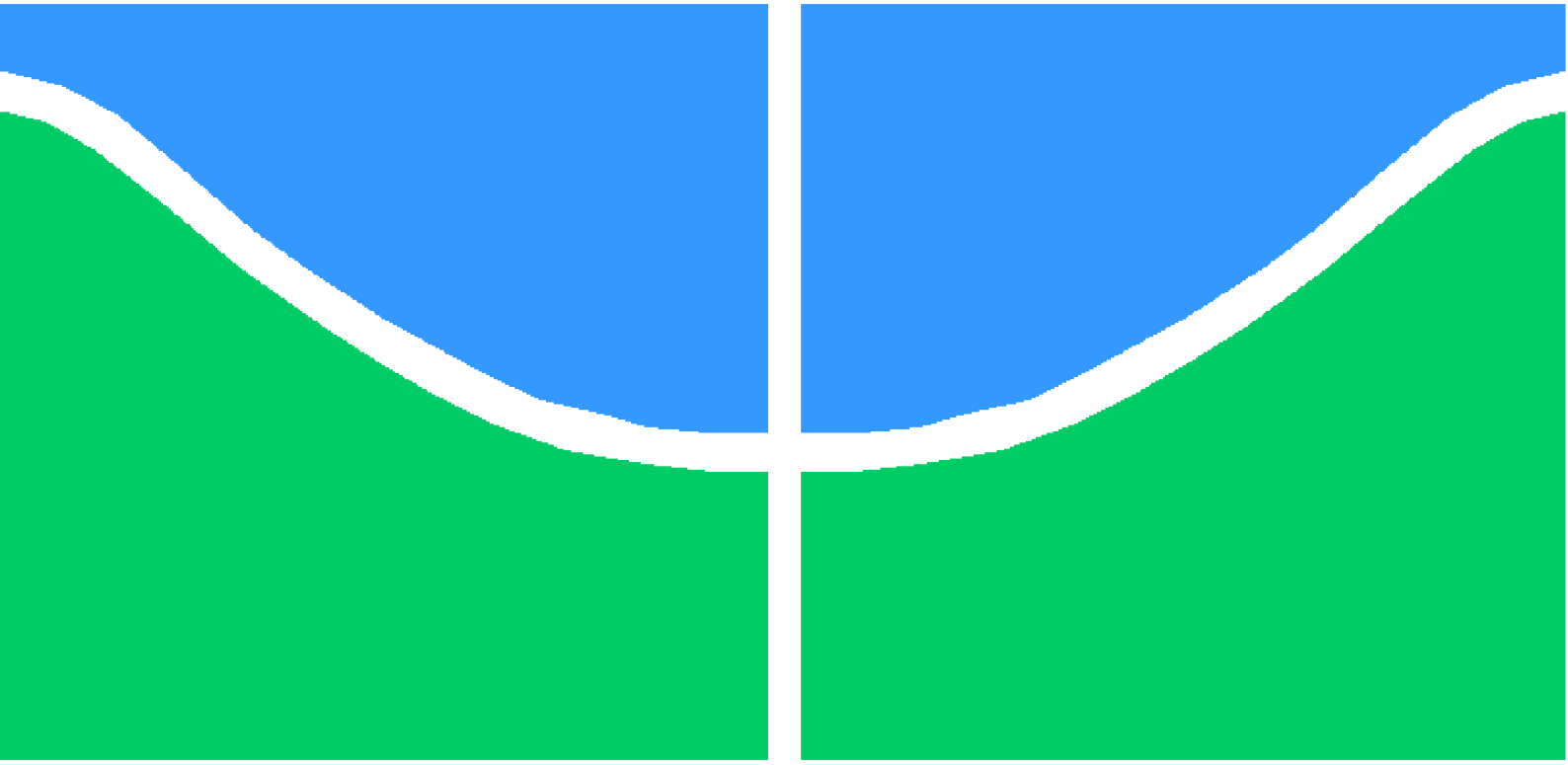


Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Identificação de Defeitos em Pavimentos Utilizando Deep Learning

Autor: Samuel Nogueira Bacelar
Orientador: Prof. Dr. Fabricio Ataides Braz

Brasília, DF
2023

Samuel Nogueira Bacelar

Identificação de Defeitos em Pavimentos Utilizando Deep Learning

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Fabricio Ataides Braz

Brasília, DF

2023

Samuel Nogueira Bacelar

Identificação de Defeitos em Pavimentos Utilizando Deep Learning

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Prof. Dr. Fabricio Ataides Braz
Orientador

Dr. Nilton Correia da Silva
Convidado 1

Dr. Henrique Marra Taira Menegaz
Convidado 2

Brasília, DF
2023

Agradecimentos

Agradeço primeiramente ao meu avô, por ter me ensinado muito durante seu tempo de vida. Agradeço também aos meus pais pelo sacrifício que fizeram, pelo seu amor e por tudo que me proveram. Por fim, agradeço a toda minha família e amigos pelos bons momentos e apoio.

"Por obra e graça da realidade e da natureza da vida,
o homem - todo homem - é um fim em si, existe por si,
e a realização de sua própria felicidade é seu mais
elevado objetivo moral. (John Galt)"

Resumo

Quanto mais desenvolvido é um país, maior é a dimensão de sua malha rodoviária e por consequência maior é a quantidade de recursos gastos com manutenção desta malha. É vital para qualquer entidade responsável pela manutenção de pavimentos, que seja possível identificar buracos em pavimentos o mais rápido possível gastando o mínimo possível. Por conta disso, uma abordagem ideal para auxiliar na identificação rápida e barata é a de utilizar tecnologias de aprendizado de máquina voltadas para a área da visão computacional. Essas tecnologias podem ser utilizadas para criar um modelo de Inteligência Artificial capaz de identificar e classificar imagens de pavimentos que contenham ou não defeitos.

Este trabalho tem como objetivo auxiliar na rápida identificação de defeitos em pavimentos. Para alcançar esse objetivo serão aplicados métodos de *Deep Learning* voltadas para a visão computacional a fim de criar um modelo que seja capaz de fazer essa classificação. Além disso, o modelo será avaliado e possíveis melhorias serão apontadas.

Palavras-chave: Inteligência Artificial; Aprendizado de máquina; Redes neurais; Datasets; Visão Computacional; Redes Neurais Convolucionais; Aprendizado Profundo

Abstract

The more developed a country is, the larger the size of its road network and consequently the greater the amount of resources spent on maintaining this network. It is vital for any entity responsible for the maintenance of roads and highways to be able to identify potholes in the roads as quickly as possible while spending as little as possible. Because of this, an ideal approach to assist in quick and cheap identification is to use machine learning technologies external to the area of computer vision. These technologies are used to create an Artificial Intelligence model capable of identifying and classifying images of roads that contain or do not contain damage.

This work aims to assist in the rapid identification of potholes on roads. For this objective, Deep Learning methods focused on computer vision will be applied in order to create a model that is capable of carrying out this classification. Furthermore, the model will be evaluated and possible improvements will be highlighted.

Key-words: Artificial intelligence; Machine learning; Neural networks; Datasets; Computer vision; Convolutional Neural Networks; Deep Learning

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

DNIT	Departamento Nacional de Infraestrutura de Transportes
IA	Inteligência Artificial
ANN	*Artificial Neural Network*
CNN	*Convolutional Neural Network*
NLP	*Natural Language Processing*
ReLU	*Rectifier Linear Unit*
ResNet	*Residual Network*
YOLO	*You Only Look Once*
IoU	*Intersection over Union*
mAP	*mean Average Precision*

Sumário

1 Introdução

1.1 Contexto

A infraestrutura rodoviária é um componente vital para o desenvolvimento econômico e social de qualquer país. Segundo fontes do governo a malha rodoviária federal do Brasil possui uma extensão total de 75,8 mil km, dos quais 65,4 mil km correspondem a rodovias pavimentadas[1]. Por conta da magnitude da malha rodoviária brasileira a manutenção dos pavimentos é um desafio constante, principalmente devido à detecção oportuna de danos. Atualmente o levantamento de defeitos em rodovias federais do Brasil é realizado pelo Departamento Nacional de Infraestrutura de Transportes (DNIT) através de avaliações de campo para acompanhar a condição da manutenção da malha rodoviária.

Como dito, essa tarefa é realizada manualmente por inspetores, o que pode ser demorado, caro e sujeito a erros humanos. Portanto, surge a necessidade de um modelo de Inteligência Artificial (IA) capaz de identificar automaticamente os defeitos a partir de imagens, proporcionando uma solução mais eficiente e precisa.

A Visão Computacional, tem experimentado avanços significativos nas últimas décadas. Ela se concentra em ensinar máquinas a 'ver' e interpretar imagens e vídeos, semelhante à percepção visual humana. Desde os primeiros estudos na década de 1960 até os recentes avanços com Redes Neurais Convolucionais (CNNs), a visão computacional tem encontrado aplicações em diversas áreas, desde reconhecimento facial até veículos autônomos.

Neste trabalho, pretende-se aplicar técnicas de visão computacional para desenvolver um modelo de aprendizado de máquina que possa identificar defeitos em pavimentos, mais especificamente defeitos do tipo buraco. Acreditamos que tal modelo não só melhorará a eficiência da manutenção da infraestrutura rodoviária, mas também contribuirá para a segurança dos usuários da estrada.

1.2 Problema

A detecção e reparação de danos em pavimentos é uma questão crítica que afeta a segurança dos usuários e a eficiência do transporte. A detecção manual de danos em pavimentos é um processo demorado e trabalhoso, que muitas vezes requer o fechamento

[1] <https://www.gov.br/transportes/pt-br/assuntos/conteudo/rodovias-brasileiras#:~:text=A%20malha%20rodovi%C3%A1ria%20federal%20do,rodovias%20n%C3%A3o%20pavimentadas%20%5B2%5D%20.>

de faixas de tráfego. Além disso, a precisão da detecção depende muito da experiência e julgamento do inspetor.

Os custos de manutenção também são uma preocupação, pois a detecção tardia de danos pode levar a reparos mais caros. Por exemplo, uma pequena rachadura pode se expandir para um buraco maior se não for reparada a tempo. Danos não detectados podem representar um risco significativo para a segurança dos usuários, especialmente para motociclistas e ciclistas. Além disso, danos no pavimento podem causar congestionamentos de tráfego, especialmente se uma faixa tiver que ser fechada para reparos.

A implementação de um modelo de IA para identificar danos em pavimentos pode ajudar a resolver esses problemas. O modelo pode analisar automaticamente imagens de pavimentos, reduzindo a necessidade de inspeções manuais demoradas e melhorando a precisão da detecção. Com detecção precoce e precisa, os reparos podem ser programados antes que os danos se tornem graves, economizando custos de manutenção.

Ao identificar rapidamente os danos, as autoridades podem tomar medidas imediatas para reparar as pavimentos, melhorando assim a segurança dos usuários Além disso, com a detecção e reparação precoce de danos, as interrupções no tráfego podem ser minimizadas.

1.3 Objetivos

1.3.1 Objetivo Geral

O propósito central deste estudo é desenvolver um modelo de Aprendizado de Máquina que possa detectar danos em pavimentos por meio de análise de imagem. Dentro do escopo deste trabalho, a detecção de defeitos é definida como a habilidade do modelo em discernir, com precisão, se uma imagem fornecida contém ou não um dano. Em outras palavras, criar um modelo de classificação de imagens. Este modelo tem o potencial de contribuir significativamente para a manutenção e segurança das infraestruturas rodoviárias. Futuramente há a intenção de aprimorar o modelo para que ele consiga mostrar exatamente onde está o defeito, ou seja, detectar objetos em imagens.

1.3.2 Objetivos Específicos

Para alcançar o objetivo geral temos os seguintes objetivos específicos, que fornecem uma visão detalhada do que deve ser alcançado e como isso será feito:

1. **Coleta e Preparação de Dados**: Coletar um conjunto de imagens de pavimentosm defeitos. Isso pode envolver a coleta de imagens da internet, bancos de dados públicos

ou até mesmo a coleta de imagens próprias. As imagens coletadas devem estar pré-processadas e rotuladas adequadamente.

2. **Seleção do Modelo**: Pesquisar e selecionar um modelo de aprendizado de máquina adequado para a tarefa de classificação de imagens.

3. **Treinamento do Modelo**: Treinar o modelo selecionado usando o conjunto de dados preparado. Isso envolverá a divisão do conjunto de dados em conjuntos de treinamento e teste, a configuração dos parâmetros do modelo e o monitoramento do desempenho do modelo durante o treinamento.

4. **Avaliação do Modelo**: Avaliar o desempenho do modelo treinado em um conjunto de teste independente. Isso pode envolver a utilização de métricas como precisão, recall, F1-score, etc.

5. **Otimização do Modelo**: Com base nos resultados da avaliação, otimizar o modelo ajustando seus parâmetros, escolhendo diferentes arquiteturas ou usando técnicas como *Data Augmentation* para melhorar seu desempenho.

1.4 Organização do Trabalho

Este trabalho está organizado nos seguintes capítulos:

Capítulo 1 - Introdução : Neste capítulo, são apresentados o contexto do trabalho, o problema de pesquisa, os objetivos deste trabalho e uma síntese da metodologia planejada.

Capítulo 2 - Referencial Teórico : Este capítulo descreve os conceitos teóricos que fundamentam este trabalho, incluindo informações sobre processamento de linguagem natural, aprendizado de máquina e técnicas relacionadas.

Capítulo 3 - Materiais e Métodos : Neste capítulo, é apresentado o plano metodológico adotado, com detalhes sobre as etapas envolvidas, como a seleção de dado e a aplicação de modelos de aprendizado de máquina. Também é exposto quais os principais materiais utilizados no trabalho.

Capítulo 4 - Resultados : Este capítulo apresenta os resultados obtidos no desenvolvimento deste trabalho. São descritas as etapas de busca por um conjunto de dados adequado, A seleção do modelo e o treinamento do modelo.

Além dos capítulos mencionados, este trabalho também inclui uma **Discussão**, onde são apresentadas as possíveis direções futuras de pesquisa.

2 Referencial Teórico

Aqui, serão apresentados os conceitos teóricos relevantes para o desenvolvimento deste trabalho. Serão abordados temas relacionados aos objetivos específicos, a fim de fornecer uma base teórica sólida para a compreensão e implementação das etapas propostas.

2.1 Inteligência Artificial

É dito por Oliveira (2021) que existem muitas definições acerca do que é Inteligência Artificial (IA), e parece não haver consenso entre os vários autores que estudam o assunto. Isso porque há muitas maneiras de se definir o que é inteligência sob o ponto de vista humano.

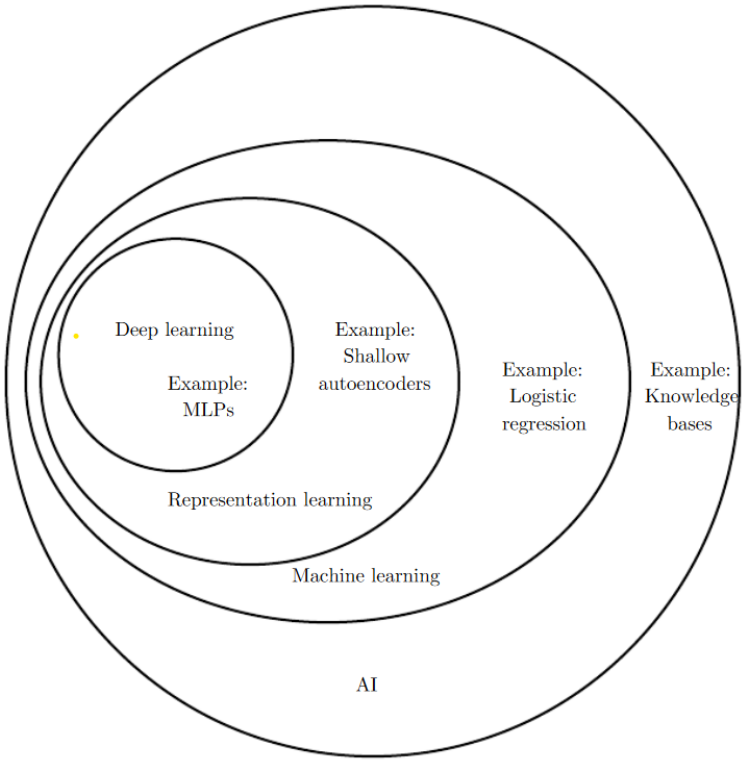
Uma dessas Definições é a de Lima (2017), que afirmam que a IA é o conjunto de ações que, se fossem realizadas por um ser humano, seriam consideradas inteligentes. Entretanto essa definição não é totalmente precisa, uma vez que nem toda ação que um computador consegue realizar se traduz adequadamente em um comportamento inteligente. Um computador consegue realizar cálculos com uma rapidez superior a um ser humano e também consegue guardar e recobrar grandes quantidade de dados, entretanto ambas as ações não são evidências de um comportamento inteligente (OLIVEIRA, 2021).

Já Norvig (2022) afirma que a inteligência artificial é a capacidade dos sistemas cibernéticos de imitar funções cognitivas dos seres humanos. Sendo essas funções cognitivas resumidas em **Resolução de Problemas**, **Aprendizado** e **Percepção**.

A Resolução de Problemas diz respeito a capacidade de realizar tarefas que podem ser consideradas corretas de acordo com algum critério de avaliação. O Aprendizado é a capacidade de um sistema de melhorar o seu desempenho na resolução de problemas por meio da experiência. Já a Percepção é a capacidade de deduzir as situações que ocorrem a nossa volta com a análise das informações que chegam até nós (NORVIG, 2022). Em um sistema cibernético, a percepção é baseada em sensores que coletam objetivamente dados acerca do problema em questão (OLIVEIRA, 2021).

Nos estágios iniciais do desenvolvimento da inteligência artificial, o campo enfrentou e solucionou prontamente questões que são cognitivamente desafiadoras para os seres humanos, mas relativamente simples para os computadores - desafios que podem ser definidos por meio de uma lista de regras matemáticas formais (GOODFELLOW; BENGIO; COURVILLE, 2016). O verdadeiro obstáculo para a inteligência artificial revelou-se a resolução de tarefas que são simples para os indivíduos realizarem, mas complicadas de serem formalmente descritas - questões que abordamos de maneira intuitiva, parecendo

Figura 1 – Diagrama de Venn ilustrando a relação entre Deep Learning, Machine Learning e IA



Fonte:(GOODFELLOW; BENGIO; COURVILLE, 2016)

automáticas, como o reconhecimento de palavras faladas ou de faces em imagens.

Nesse contexto evolutivo, a inteligência artificial (IA) emergiu como um campo amplo, abrangendo diversas subáreas como visto na Figura 1, cada uma focada em aspectos específicos do desenvolvimento de sistemas inteligentes. Uma das principais sub-áreas da IA é o Machine Learning 2.2, que engloba o Deep Learning 2.3, que é um tipo particular de machine learning (GOODFELLOW; BENGIO; COURVILLE, 2016). Abaixo é possível ver uma representação simples de como esses três conceitos se relacionam. Vale notar que os campos englobam mais coisas do que o representado.

2.2 Machine Learning

Segundo Goodfellow, Bengio e Courville (2016) sistemas de IA devem ser capazes de adquirir seu próprio conhecimento por meio de padrões extraídos de um conjunto de dado. Essa capacidade É conhecida por **Machine Learning (ML)**. Ou seja, em vez de seguir instruções programadas, os sistemas de machine learning usam técnicas estatísticas para melhorar sua performance em uma tarefa específica à medida que são expostos a mais dados. Existem três tipos principais de aprendizado de máquina:

- **Aprendizado Supervisionado**: O modelo é treinado com um conjunto de dados rotulado, onde a entrada e a saída desejada são fornecidas. Em cada treinamento, são fornecidos conjuntos de valores de entrada (vetores) juntamente com um ou mais valores de saída designados (O'SHEA; NASH, 2015). O propósito fundamental desse tipo de treinamento é minimizar o erro de classificação geral dos modelos, alcançando o cálculo preciso dos valores de saída para cada exemplo de treinamento durante o processo de treinamento.

- **Aprendizado Não Supervisionado**: O modelo é treinado com dados não rotulados, e o sistema tenta aprender padrões e estruturas por conta própria. O sucesso geralmente é determinado pela capacidade da rede de reduzir ou aumentar uma função de custo associada. (O'SHEA; NASH, 2015).

- **Aprendizado por Reforço**: O modelo aprende através de tentativa e erro, recebendo feedback em termos de recompensas ou penalidades. Ele deve aprender sozinho e escolher a melhor estratégia chamada de política.(GÉRON, 2019).

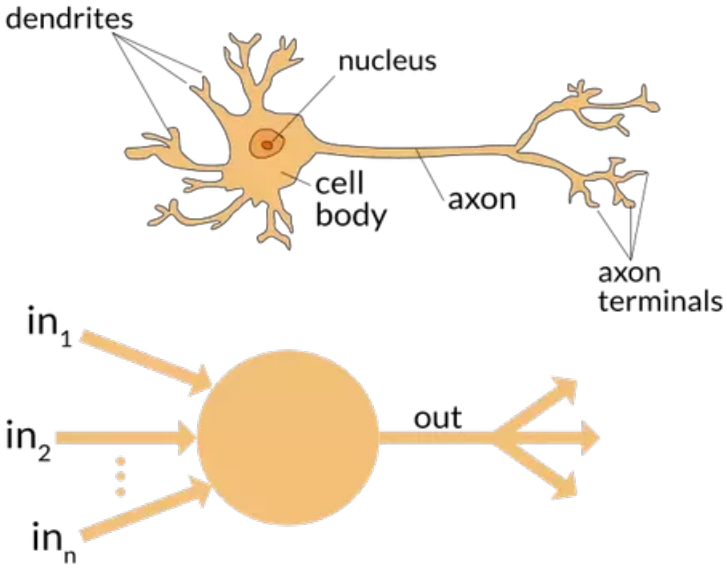
Existem diversas técnicas e métodos de Machine Learning, elas podem ser simples ou complexas e podem até mesmo resolver os mesmos problemas mas de maneiras diferentes. As ***Artificial Neural Networks*** (ANNs) 2.2.1 são modelos inspirados no funcionamento do cérebro humano, compostos por camadas de neurônios interconectados. Elas são notáveis pela capacidade de aprender representações complexas e são amplamente utilizadas em tarefas como reconhecimento de padrões, processamento de imagem e *natural language processing* (NLP). A **Regressão Logística**, por outro lado, é um modelo linear que estima a probabilidade de uma instância pertencer a uma classe específica. Essa técnica é frequentemente utilizada em problemas de classificação binária e fornece interpretabilidade nos resultados. O algoritmo **Naive Bayes** é uma abordagem probabilística que assume independência condicional entre as características. Essa simplicidade torna o Naive Bayes eficaz em tarefas de processamento de linguagem natural, como análise de sentimento e classificação de texto. Cada uma dessas técnicas tem suas próprias vantagens e limitações, e a escolha entre elas depende da natureza do problema e dos dados disponíveis. Para este trabalho haverá um foco maior nas *Artificial Neural Networks*.

2.2.1 Artificial Neural Network

Segundo O'Shea e Nash (2015), *Artificial Neural Networks* (ANNs) são processamentos computacionais inspirados em sistemas nervosos biológicos. Programaticamente são compostos por vários nós computacionais interconectados (conhecidos como neurônios). Inspiradas na estrutura do cérebro, como visto na Figura 2, as ANNs consistem em neurônios artificiais organizados em camadas e interconectados por meio de conexões pon-

deradas. Cada neurônio realiza operações semelhantes aos neurônios biológicos, recebendo entradas, aplicando transformações com funções de ativação e gerando saídas.

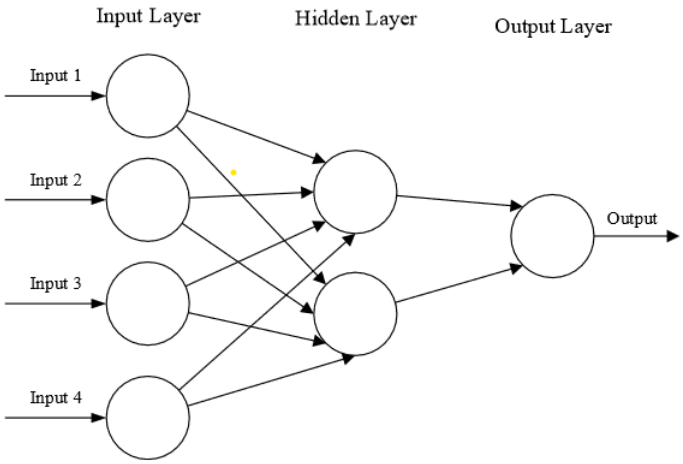
Figura 2 – Estrutura de um neurônio biológico e de um neurônio artificial



Fonte:(HOWARD; GUGGER; CHINTALA, 2020)

O aprendizado em ANNs é alcançado ajustando os pesos dessas conexões, simulando a plasticidade sináptica observada no sistema nervoso biológico. A Figura 3 mostra a estrutura básica de uma Rede neural, ela é composta por três camadas principais, uma camada de *input*, uma camada intermediária, também conhecida como oculta e uma camada de *output*.

Figura 3 – Estrutura básica de uma ANN



Fonte:(O'SHEA; NASH, 2015)

O fluxo básico de uma rede neural segue da seguinte forma. Os dados são carregados na camada *input* utilizando um vetor multidimensional e são distribuídos para a camada

oculta. As camadas ocultas tomarão decisões a partir da camada anterior e avaliarão como uma mudança estocástica em si prejudica ou melhora o resultado final, e isso é conhecido como **processo de aprendizagem** (ver capítulo 2.1 que também trata de aprendizagem). Ter várias camadas ocultas empilhadas umas sobre as outras é comumente chamado de deep learning 2.3 (O'SHEA; NASH, 2015).

A maior limitação das ANNs é que elas tem dificuldade com a complexidade computacional necessária para processar dados vindos de imagens. Com imagens simples e pequenas não há tanto problema, mas quando se considera imagens com maiores detalhes, como cores além de preto e branco e dimensões maiores, a quantidade de pesos na camada oculta aumenta significantemente (O'SHEA; NASH, 2015).

2.2.2 Overfitting

Overfitting é, de modo geral, um problema que pode ser descrito como 'super generalização' de certas características. Géron (2019) explica isso por meio da seguinte analogia. "Digamos que você está em um país estrangeiro, você pediu um taxi e ele cobra um valor absurdo pela viagem. Você pode ficar tentado a dizer que todos os motoristas de táxi daquele país são ladrões."Seres Humanos podem cair nessa armadilha e máquinas também. Já O'Shea e Nash (2015) diz que o *Overfitting*, basicamente é quando uma rede é incapaz de aprender efetivamente de devido a uma variedade de motivos e que é necessário reduzir ao máximo os efeitos do overfitting. Modelos complexos, como redes neurais profundas, podem detectar padrões sutis nos dados, mas se o conjunto de treinamento for ruidoso ou muito pequeno, é provável que o modelo detecte padrões no próprio ruído. Obviamente estes padrões não serão generalizados para novas instâncias.

Em outras palavras o overfitting ocorre quando o modelo é muito complexo em relação à quantidade e ao ruído dos dados de treinamento(GÉRON, 2019). Para contornar esse problema pode-se tomar algumas decisões e adotar algumas técnicas, dentre elas:

- **Simplificar o modelo** selecionando um com menos parâmetros, reduzindo o número de atributos nos dados de treinamento ou restringindo o modelo.
- **Aumentar o conjunto de dados de treinamento**. Quanto mais dados houverem, mais o modelo pode aprender padrões gerais em vez de memorizar exemplos específicos.
- **Reduzir o ruído do dado de treinamento**. Podendo remover dados errôneos ou remover valores discrepantes.
- **Aplicar Dropout** que desativa aleatoriamente um percentual de unidades em cada época, evitando que o modelo se torne excessivamente dependente de unidades específicas.

2.3 Deep Learning

As *Artificial Neural Networks* são a base do *Deep Learning*, oferecendo ferramentas flexíveis e poderosas que podem lidar com tarefas complexas da Aprendizagem de Máquina como classificar imagens, reconhecer fala, recomendar vídeos e até jogar jogos (GÉRON, 2019).

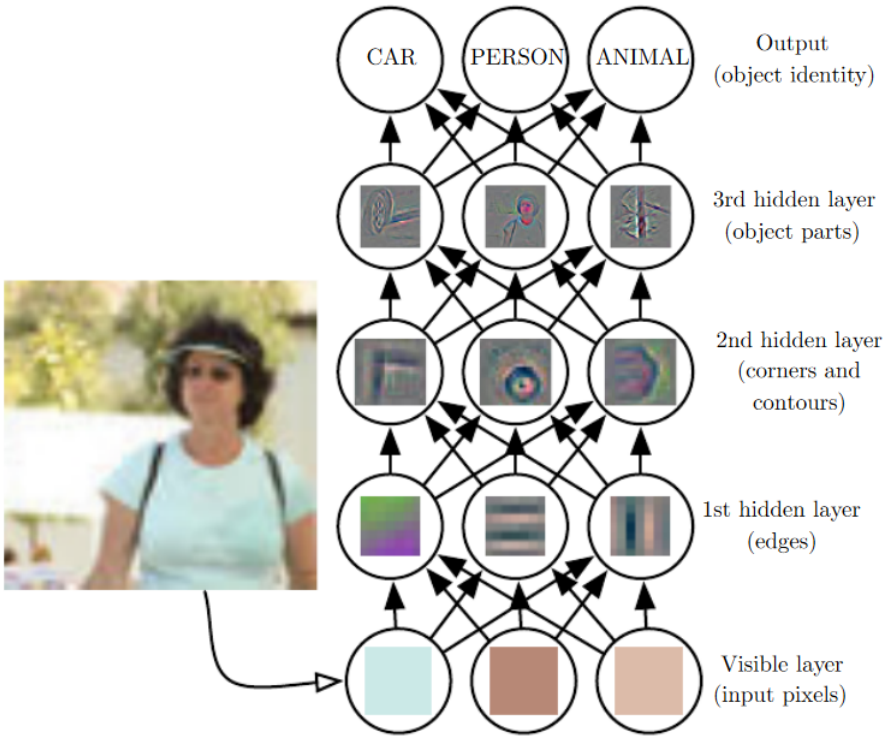
Como dito na seção 2.2, existem diversas técnicas e métodos de machine learning. Os algorítimos simples de machine learning como **Regressão Logística** e **Naive Bayes** dependem muito da **Representação do dado** que lhes é dado. Cada pedaço de informação incluído na representação do dado é conhecido como ***Feature***. O principal problema é que para muitas tarefas de machine learning pode ser difícil de saber quais features devem ser extraídas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para isso surge a *Representation Learning* que é um método que consiste em aprender a própria representação do dado. São métodos que então aprendem as features do dado. Ao projetar recursos ou algoritmos para recursos de aprendizagem, nosso objetivo geralmente é separar os **fatores de variação** que explicam os dados observados (GOODFELLOW; BENGIO; COURVILLE, 2016). Por exemplo, ao analisar a imagem de um carro, os fatores de variação incluem a posição do carro, sua cor e o ângulo e brilho do sol.

Uma grande fonte de dificuldade em muitas aplicações de inteligência artificial do mundo real é que muitos dos fatores de variação influenciam cada dado que somos capazes de observar e isso pode ser muito difícil de se extrair. Quando é quase tão difícil obter uma representação como resolver o problema original, a aprendizagem da representação não parece, à primeira vista, ajudar (GOODFELLOW; BENGIO; COURVILLE, 2016).

A solução para esse desafio fundamental no *representation learning* é oferecida pelo *Deep Learning*, que introduz representações expressas em termos de representações mais elementares. Esse tipo de aprendizado capacita o computador a construir conceitos mais complexos a partir de noções mais simples. A Figura 4 mostra como um modelo de deep learning é capaz de representar uma imagem de uma pessoa por meio de combinação de conceitos mais simples, como cantos e contornos, que por sua vez são definidos em termos de arestas.

Figura 4 – Ilustração de um modelo de aprendizagem profunda.



Fonte:(GOODFELLOW; BENGIO; COURVILLE, 2016)

Explicando de forma mais detalhada a Figura 4, a entrada é apresentada na **Camada visível**, denominada assim por conter as variáveis observáveis. Em seguida, diversas **camadas ocultas** extraem características cada vez mais abstratas da imagem. Estas camadas são denominadas "ocultas"devido ao fato de que seus valores não são fornecidos nos dados; ao contrário, o modelo deve determinar quais conceitos são relevantes para explicar as relações nos dados observados. As imagens apresentadas são visualizações do tipo de recurso representado por cada unidade oculta. Dados os pixels, a primeira camada pode identificar com facilidade as bordas ao comparar o brilho dos pixels vizinhos. Com base na descrição das bordas pela primeira camada oculta, a segunda camada oculta pode buscar eficientemente cantos e contornos estendidos, os quais são reconhecíveis como coleções de bordas. Com base na descrição da imagem pela segunda camada oculta em termos de cantos e contornos, a terceira camada oculta pode detectar partes completas de objetos específicos, identificando coleções específicas de contornos e cantos. Por fim, essa descrição da imagem em termos das partes do objeto que ela contém pode ser utilizada para reconhecer os objetos presentes na imagem (GOODFELLOW; BENGIO; COURVILLE, 2016).

Em conclusão o *Deep Learning* é um tipo particular de aprendizado de máquina que alcança grande poder e flexibilidade ao aprender a representar o mundo como uma hierarquia aninhada de conceitos, com cada conceito definido em relação a conceitos mais

simples e representações mais abstratas computadas em termos de conceitos menos abstratos.

2.3.1 Convolutional Neural Network

As Convolutional Neural Networks (CNNs ou Rede Neural Convolucional) surgiram a partir de estudos do córtex visual do cérebro e têm sido usados no reconhecimento de imagens desde a década de 1980 (GÉRON, 2019). são um tipo especializado de ANN para processamento de dados que possuem uma topologia semelhante a uma grade (ex: imagens). Têm sido tremendamente bem-sucedidas em aplicações práticas. O nome "rede neural convolucional" indica que a rede emprega uma operação matemática chamada **convolução** (GOODFELLOW; BENGIO; COURVILLE, 2016). Uma diferença notável entre as CNNs e as ANNs tradicionais é que as CNNs são usadas principalmente no campo de reconhecimento de padrões em imagens (Visão Computacional). Isso nos permite codificar recursos específicos de imagem na arquitetura, tornando a rede mais adequada para tarefas focadas em imagem ao mesmo tempo em que reduz ainda mais os parâmetros necessários para configurar o modelo (O'SHEA; NASH, 2015).

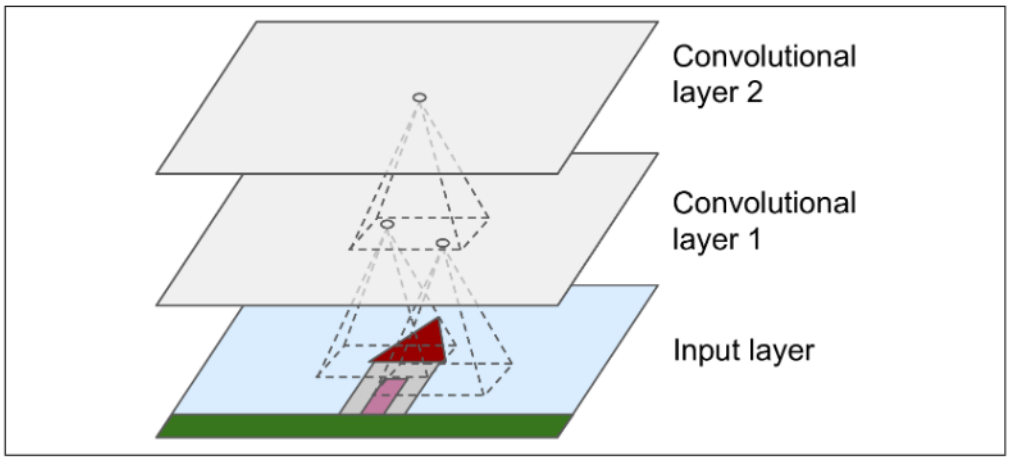
Mas como e porque as CNNs se destacam na parte de Visão Computacional perante as ANNs comuns? Claro, para tarefas que envolvem imagens de dimensões pequenas as ANNs conseguem ser bem úteis. Mas a medida que a dimensão da imagem aumenta, o desempenho da ANN passa a piorar. Com imagens de dimensões em preto e branco de 28 X 28 um único neurônio de uma ANN iria conter 784 pesos (28 X 28). Isso ainda é razoável para a maioria das ANNs. Entretanto se considerarmos imagens coloridas de dimensões de 64 X 64, a quantidade de pesos aumenta para 12.288 (64 X 64 X 3) em um único neurônio (O'SHEA; NASH, 2015).

Isso é um problema pois pode ser computacionalmente caro e não temos nem poder computacional ilimitado nem tempo ilimitado. Também há o risco dessa quantidade de pesos acabarem causando o *overfitting*. Quanto menos parâmetros forem necessários para treinar, menor será a probabilidade de a rede sofrer o overfitting e, claro, melhoramos o desempenho preditivo do modelo (O'SHEA; NASH, 2015).

A base crucial de uma CNN é representada pela camada convolucional (*convolutional layer*). Na primeira camada convolucional, os neurônios não estabelecem conexões com todos os pixels na imagem de entrada, mas sim apenas com os pixels presentes em seus campos receptivos (ver Figura 5). Da mesma forma, cada neurônio na segunda camada convolucional está vinculado exclusivamente aos neurônios situados dentro de um pequeno retângulo na primeira camada. Essa configuração possibilita que a rede se concentre em pequenos atributos de baixo nível na camada oculta inicial e, posteriormente, os integre em características mais amplas de nível superior na camada subsequente e assim por diante. Esta organização em cascata é uma característica comum em imagens do

mundo real, o que explica em parte a eficácia das CNNs no reconhecimento de imagens.

Figura 5 – Camadas CNN com campos receptivos locais retangulares

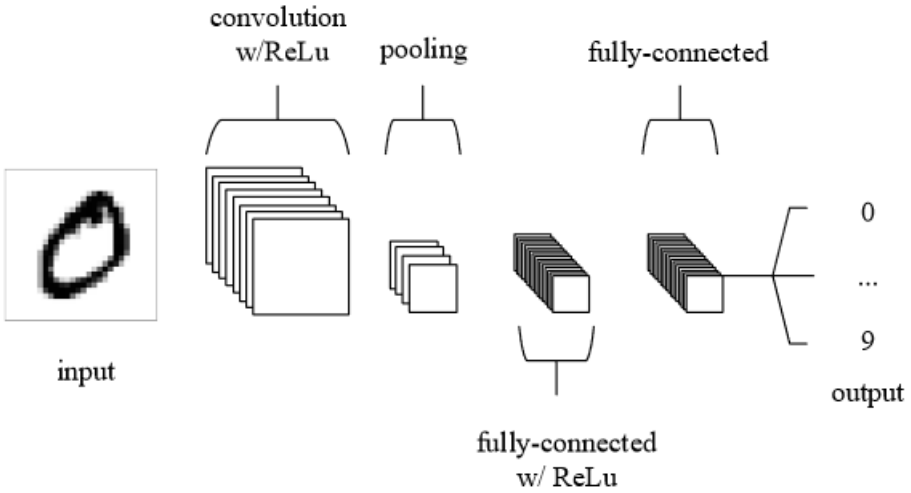


Fonte:(GÉRON, 2019)

2.3.1.1 Arquitetura de uma CNN

As CNNs se concentram principalmente no fato de que a entrada será composta por imagens. Isso concentra a arquitetura a ser configurada da maneira que melhor se adapta à necessidade de lidar com o tipo específico de dados. As CNNs são compostas por três tipos de camadas. São estas: camadas convolucionais, camadas de *pooling* e camadas totalmente conectadas(Figura 6).

Figura 6 – Arquitetura de uma CNN simples



Fonte:(O'SHEA; NASH, 2015)

A camada de convolução é um componente fundamental em uma CNN. Sua função principal é aplicar filtros à entrada por meio de operações de convolução normalmente utilizando a função de ativação *Rectifier Linear Unit* (ReLU) (O'SHEA; NASH, 2015).

Cada filtro é projetado para identificar padrões locais específicos, como bordas, texturas ou características mais complexas. Os parâmetros essenciais incluem o número e tamanho dos filtros, o passo (*stride*) que determina o deslocamento do filtro durante a convolução, e o preenchimento (*padding*) que controla a redução de tamanho.

A camada de *pooling* é empregada para reduzir a dimensionalidade espacial da representação da imagem (O'SHEA; NASH, 2015). Isso é crucial para diminuir a quantidade de parâmetros e a carga computacional na rede. O *max pooling*, extrai o valor máximo de um grupo local de pixels. Os parâmetros incluem o tamanho da janela, que determina a região local considerada, e o *stride* que especifica o deslocamento da janela durante o pooling.

Camadas totalmente conectadas recebem as características aprendidas pelas camadas anteriores e as utilizam para classificação ou regressão. Cada neurônio em uma camada totalmente conectada está conectado a todos os neurônios da camada anterior, funcionando de forma quase idêntica as ANNs (O'SHEA; NASH, 2015), transformando as características aprendidas em uma representação final para tomada de decisões. Os parâmetros essenciais incluem o número de neurônios, que define a dimensionalidade da saída.

Ao longo da passagem dos dados pela rede neural, eles percorrem todas as camadas do modelo, sendo submetidos a diversas transformações. Essas transformações ocorrem por meio de operações como convolução, pooling e camadas totalmente conectadas. Esse processo é fundamental para a capacidade das CNNs de modificar a representação da entrada original camada por camada. Utilizando técnicas convolucionais e de redução de resolução, as CNNs conseguem, de maneira eficaz, gerar pontuações de classe (O'SHEA; NASH, 2015). Essas pontuações são cruciais para tarefas de classificação e regressão, proporcionando uma representação mais refinada e complexa da entrada ao passar pelas diferentes etapas da rede.

Existem várias arquiteturas de CNNs que se destacam por suas contribuições em tarefas de visão computacional. A LeNet-5 , desenvolvida por Lecun et al. (Nov./1998), foi pioneira em aplicar CNNs para reconhecimento de dígitos manuscritos, especialmente no conjunto de dados MNIST[1].

A arquitetura AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), foi revolucionária ao vencer a competição ImageNet[2]. Ela demonstrou a eficácia do deep learning em grandes conjuntos de dados visuais.

A VGGNet, proposta por Simonyan e Zisserman (2015), é conhecida por sua simplicidade e profundidade. Utiliza camadas convolucionais de 3x3, tornando a arquitetura fácil de entender e modificar.

[1] http://yann.lecun.com/exdb/mnist/
[2] https://www.image-net.org/challenges/LSVRC/2012/

O GoogLeNet (SZEGEDY et al., 2015), também conhecido como Inception, em 2014, inovou com módulos de convolução em paralelo chamados "módulos Inception". Essa abordagem eficaz permite a extração de características em diferentes escalas.

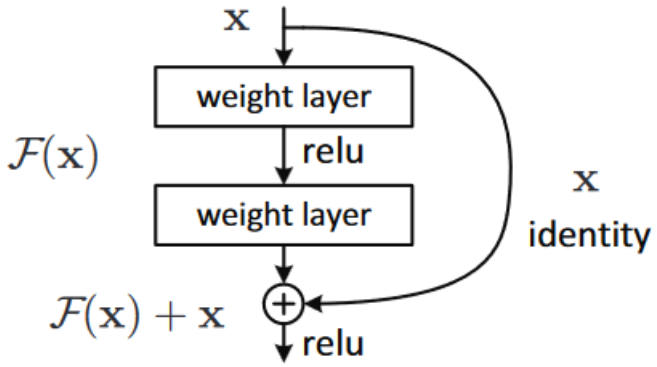
As Redes Residuais (ResNets), introduzidas por He et al. (2015), resolveram o desafio de treinar redes profundas com conexões residuais. Essas conexões diretas ajudam a mitigar o desaparecimento do gradiente.

Essas são apenas algumas das principais arquiteturas de CNNs que moldaram o campo de visão computacional ao longo dos anos. Cada uma trouxe inovações únicas, demonstrando a evolução constante das redes neurais convolucionais para lidar com tarefas complexas de processamento de imagem.

2.3.2 Residual Network

A Residual Network (ResNet) é uma arquitetura de CNN introduzida por He et al. (2015). O conceito chave das ResNets é a introdução de conexões residuais (Figura 7), permitindo o fluxo direto de informações de uma camada para outra. Essas conexões ajudam a mitigar o problema do desaparecimento do gradiente (Problema da degradação) em redes profundas, facilitando o treinamento de arquiteturas significativamente mais profundas do que aquelas previamente viáveis. As ResNets foram fundamentais para avanços em tarefas de visão computacional, ganhando destaque em competições como o ImageNet[3].

Figura 7 – Estrutura básica de uma Residual Network



Fonte:(HE et al., 2015)

Segundo o trabalho de He et al. (2015) a ResNet surgiu a fim de sanar o problema da degradação. Em termos simples, à medida que as redes neurais se tornam mais profundas, espera-se que a precisão melhore. No entanto, o problema da degradação revela o oposto: à medida que a profundidade da rede aumenta, a precisão satura e depois começa a degradar-se rapidamente. Essa degradação é observada apesar de mais camadas

[3] https://image-net.org/challenges/LSVRC/2015/

serem adicionadas ao modelo. Isso ocorre pois os gradientes começam a diminuir exponencialmente à medida que sofrem *backpropagation* durante o treinamento. Isso dificulta a atualização eficiente dos pesos nas camadas mais profundas.

É importante salientar que essa degradação não é atribuído ao *overfitting*. Normalmente, quando um modelo sofre *overfit*, seu desempenho no conjunto de treinamento melhora, enquanto seu desempenho em dados novos e invisíveis piora. No caso do problema de degradação, adicionar mais camadas a um modelo suficientemente profundo leva a um erro de treinamento maior, sugerindo um desafio único no processo de otimização.

A arquitetura ResNet resolveu esse problema introduzindo conexões residuais, ou conexões de salto, que permitem que a rede passe informações diretamente de uma camada para outra. Isso significa que a entrada original é somada diretamente à saída da camada, contornando a aplicação de algumas transformações. Esta inovação ajuda a mitigar o problema do gradiente de desaparecimento associado ao treinamento de redes muito profundas, permitindo o treinamento bem-sucedido de modelos extremamente profundos e, em última análise, melhorando a precisão em tarefas desafiadoras.

Matematicamente, supondo que a entrada da camada seja x e a saída desejada seja $H(x)$, a saída real da camada se torna $F(x) + x$, onde $F(x)$ é a transformação aprendida pela camada. Essa formulação permite que o gradiente flua diretamente através da camada sem ser afetado pela operação $F(x)$.

Por fim, Com essa arquitetura é possível criar redes extremamente profundas, podendo conter até 152 camadas ou mais. Mesmo com a quantidade de camadas a arquitetura mantém uma complexidade menor com relação ao seus pares (como a VGGNet) (HE et al., 2015).

3 Materiais e Métodos

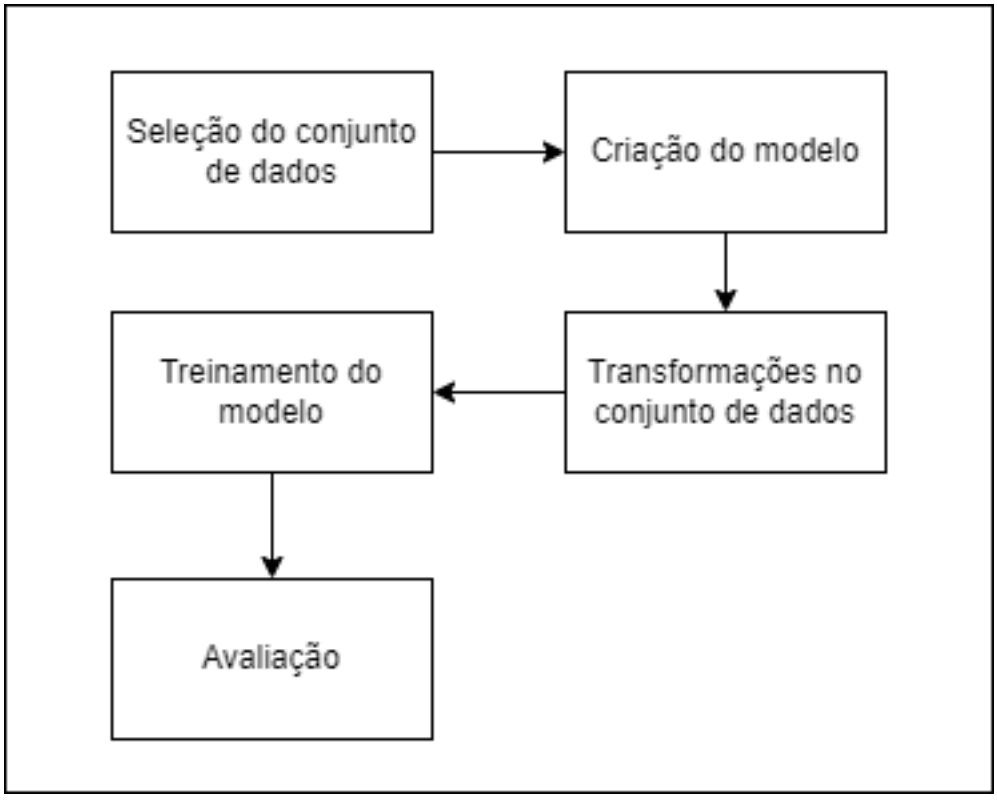
3.1 Considerações Iniciais

Neste capítulo, serão apresentados os materiais e métodos utilizados para alcançar os objetivos propostos neste trabalho. Será descrito o plano metodológico adotado, detalhando as etapas que serão seguidas para aplicar o modelo de aprendizado de máquina que realiza a identificação de defeitos em pavimentos.

3.2 Plano Metodológico

Seguindo o fluxo da Figura 8, temos a sequência de passos que serão realizados neste trabalho.

Figura 8 – Fluxo de atividades planejadas



3.2.1 Seleção do conjunto de dados de imagens pré classificadas

A escolha do conjunto de dados desempenha um papel crucial no desenvolvimento de um modelo de identificação de defeitos em pavimentos utilizando visão computacio-

nal. Esse conjunto deve conter dados pré classificados de pavimentos sem defeitos e com defeitos.

A qualidade e a quantidade de dados presentes no conjunto têm um impacto significativo no desempenho e na precisão da IA. Portanto, é importante obter um conjunto de dados representativo da realidade, com uma amostra adequada de pavimentos sem defeitos e com defeitos, a fim de evitar que o modelo se baseie em uma perspectiva distorcida(ou seja, evitar o *overfitting*). Além disso, o conjunto de dados deve suficiente para que o modelo possa aprender com uma quantidade adequada de exemplos.

3.2.2 Criação do modelo de classificação de defeitos

Nesta etapa crucial do processo, o foco recai sobre o desenvolvimento da arquitetura do modelo de classificação de defeitos em pavimentos estão os pontos relevantes que guiarão essa fase:

- Escolha da Arquitetura: Seleção da arquitetura do modelo, considerando CNNs que são eficazes para tarefas de visão computacional. Dependendo da complexidade do problema, pode-se optar por arquiteturas consagradas, como ResNet, VGG, ou até mesmo modelos mais recentes e especializados.

- Camadas do Modelo: Definição da quantidade e configuração das camadas do modelo. Isso inclui camadas convolucionais para extração de características, camadas de pooling para redução espacial e camadas totalmente conectadas para classificação.

- Função de Ativação: Escolha das funções de ativação, como ReLU (Rectified Linear Unit), para introduzir não-linearidade nas camadas do modelo.

- Regularização: Incorporação de técnicas de regularização, como dropout, batch normalization, ou regularização L1/L2, para prevenir overfitting e melhorar a generalização.

- Construção do Modelo em Framework de Deep Learning: Implementação do modelo utilizando um framework de deep learning, como TensorFlow ou PyTorch, garantindo a eficiência computacional e a facilidade de treinamento.

Ao final desta etapa, espera-se ter um modelo de classificação robusto e bem-configurado, pronto para ser treinado no conjunto de dados selecionado. O sucesso desta fase é fundamental para o desempenho eficaz do modelo na identificação de defeitos em pavimentos

3.2.3 Transformações no conjunto de dados

A manipulação eficaz do conjunto de dados desempenha um papel crucial no treinamento bem-sucedido do modelo. Nesta etapa, é abordado as transformações que, caso necessário, serão aplicadas ao conjunto de dados:

- Data Augmentation: Implementação da técnica de aumento de dados para diversificar o conjunto de treinamento. Isso envolve a aplicação de transformações como rotação, espelhamento, zoom, translação e mudanças de cor em imagens existentes. Essas variações artificiais enriquecem a quantidade e diversidade dos dados de treinamento, ajudando o modelo a generalizar melhor para novas situações.

- Normalização de Dados: Aplicação de técnicas de normalização para garantir que as características das imagens estejam na mesma escala. Isso pode envolver a escala dos valores dos pixels para um intervalo específico (por exemplo, [0, 1]).

- Remoção de Ruídos e Anomalias: Identificação e tratamento de possíveis ruídos ou anomalias no conjunto de dados, utilizando métodos como filtragem de imagens ou remoção de exemplos discrepantes.

- Verificação de Qualidade de Imagem: Avaliação da qualidade geral das imagens no conjunto de dados e tomada de medidas para corrigir problemas, se necessário.

Ao aplicar essas transformações, o objetivo é criar um conjunto de treinamento robusto, diversificado e representativo, permitindo que o modelo aprenda com eficácia padrões relevantes para a identificação de defeitos em pavimentos. Essas práticas contribuem significativamente para a capacidade do modelo de generalizar bem para dados não vistos.

3.2.4 Treinamento do modelo

Na etapa de treinamento do modelo, a atenção é direcionada para a fase em que o modelo é exposto ao conjunto de dados para aprender a mapear padrões específicos para as saídas desejadas. Inicialmente, o conjunto de dados é dividido adequadamente em conjuntos de treinamento, validação e teste. Os pesos do modelo são inicializados conforme as melhores práticas, evitando valores extremos. Durante o treinamento, é escolhido um otimizador apropriado, como o Adam ou o *Stochastic Gradient Descent* (SGD), para ajustar gradualmente os pesos do modelo com base na função de perda definida para a tarefa de classificação de defeitos.

O processo de treinamento é conduzido em lotes de dados para otimização eficiente, monitorando constantemente o desempenho em conjuntos de treinamento e validação. O

modelo é treinado por várias épocas, iterando sobre o conjunto de treinamento e ajustando os pesos para melhorar a performance. Ao concluir esta etapa, espera-se que o modelo esteja apto a realizar predições precisas, incluindo a identificação de defeitos em pavimentos

3.2.5 Avaliação

A fase de avaliação é importante para determinar o desempenho real do modelo e sua capacidade de generalização para dados não vistos. A seguir são listados os passos fundamentais nesta etapa:

- Conjunto de Teste: Utilização do conjunto de teste, que não foi visto pelo modelo durante o treinamento, para avaliar sua capacidade de generalização. Isso proporciona uma avaliação mais realista do desempenho do modelo em condições do mundo real.
- Métricas de Avaliação: Utilização de métricas apropriadas para avaliar o desempenho do modelo. Para tarefas de classificação, métricas como precisão, recall, F1-score e matriz de confusão são comumente empregadas.
- Análise de Erros: Análise detalhada dos erros cometidos pelo modelo, identificando padrões e tipos de defeitos em que o modelo pode ter dificuldades.

A avaliação abrangente não apenas valida o modelo, mas também fornece informações valiosas para possíveis melhorias. O ciclo de treinamento, avaliação e ajuste é iterativo, buscando constantemente otimizar o modelo para a tarefa específica de identificação de defeitos em pavimentos

3.3 Ferramentas e Bibliotecas

3.3.1 Jupyter Notebook

O Jupyter Notebook é uma aplicação web de código aberto que permite criar e compartilhar documentos que contêm código ao vivo, equações, visualizações e texto narrativo (Jupyter, 2023). É amplamente utilizado em campos como ciência de dados e aprendizado de máquina.

Cada desenvolvedor pode dividir o código em partes e trabalhar nelas independentemente da ordem: escrever, testar funções, carregar um arquivo na memória e processar o conteúdo. Isso é possível graças ao conceito de células do notebook, que permitem criar blocos de texto e blocos de código.

O Jupyter Notebook suporta várias linguagens de programação, incluindo Python, Ruby, Julia, Perl, Matlab e R. Ele foi originalmente parte do projeto IPython, mas agora suporta muitas outras linguagens. (Jupyter, 2023)

Para instalar o Jupyter Notebook, pode-se usar o gerenciador de pacotes Python pip ou instalar a plataforma Anaconda, que também inclui Python e várias bibliotecas úteis para ciência de dados.

O Jupyter Notebook pode ser executado localmente no seu computador ou na nuvem. Serviços como o Google Colab permitem que se use o Jupyter Notebook diretamente do navegador, sem necessidade de instalação (GOOGLE, 2019).

Por fim, o Jupyter Notebook é mantido pelo Projeto Jupyter, uma organização sem fins lucrativos que visa desenvolver software de código aberto, padrões abertos e serviços para computação interativa em várias linguagens de programação.

3.3.2 Fast.ai

Fast.ai é uma biblioteca de aprendizado profundo que fornece aos profissionais componentes de alto nível que podem fornecer rapidamente resultados de última geração em domínios padrão de aprendizado profundo (FASTAI, 2023b). Ela também fornece aos pesquisadores componentes de baixo nível que podem ser combinados para criar novas abordagens.

Fast.ai simplifica o treinamento de redes neurais rápidas e precisas usando as melhores práticas modernas. Ela inclui um novo sistema de despacho de tipo para Python, juntamente com uma hierarquia de tipo semântico para tensores, e uma biblioteca de visão computacional (FASTAI, 2023a) otimizada para GPU que pode ser estendida em Python puro.

Por fim, Fast.ai é mantido pelo Projeto Fast.ai, uma organização sem fins lucrativos que visa desenvolver software de código aberto, padrões abertos e serviços para computação interativa em várias linguagens de programação (FASTAI, 2023b).

Por debaixo dos panos o Fast.ai usa PyTorch como seu principal framework de aprendizado profundo. PyTorch fornece a funcionalidade de baixo nível para cálculos de tensores e diferenciação automática. Pelo PyTorch ser o principal framework utilizado é importante explicar com um pouco mais de detalhes o que é o PyTorch

3.3.2.1 PyTorch

É uma biblioteca de aprendizado de máquina de código aberto, projetada com Python em mente e construída para projetos de aprendizado de máquina (PyTorch, 2023). Ele é especializado em diferenciação automática, cálculos de tensores e aceleração de GPU.

Isso o torna mais adequado para aplicativos de aprendizado de máquina de ponta, como aprendizado profundo.

PyTorch é baseado em Torch, um quadro inicial para o aprendizado profundo. PyTorch apenas aproveita o potencial de aprendizado profundo da Torch e a porta para o ambiente Python. Foi desenvolvida pelo laboratório de pesquisa de IA da META em 2016 e, desde então, é adotada nos campos da ciência de dados e ML.

Essa combinação de recursos exclusivos e a simplicidade incomparável do PyTorch o torna uma das bibliotecas de aprendizado profundo mais populares, competindo apenas com o TensorFlow pelo primeiro lugar (O'CONNOR, 2021)

4 Resultados

Nesta seção serão descritos os resultados obtidos em cada etapa da execução do Capítulo 3 que implementa o Plano Metodológico para este trabalho.

4.1 Seleção do conjunto de imagens pré classificadas

Foi realizado uma pesquisa por conjuntos de dados que atendessem as necessidades do projeto em vários sites e com isso foi selecionado o dataset *Pothole Detection Dataset*[1] disponível no Kaggle[2]. O dataset é composto por um total de aproximadamente 700 imagens de duas classes, sendo 352 classificadas como normal e 329 classificadas como contendo buracos (*pothole*). As imagens estão separadas por pastas de acordo com a sua classificação. Na Figura 9 é possível ver um conjunto aleatório de oito imagens selecionadas.

Figura 9 – Conjunto de imagens aleatórias com suas respectivas classificações.

4.2 Criação do modelo de classificação de defeitos

Na criação do modelo de classificação de defeitos, uma série de decisões foram tomadas para garantir uma arquitetura robusta e eficaz. A escolha da arquitetura ResNet, renomada por sua capacidade de treinar redes muito profundas, reflete a busca por

[1] https://www.kaggle.com/datasets/atulyakumar98/pothole-detection-dataset
[2] https://www.kaggle.com/

um modelo capaz de aprender representações complexas e hierárquicas das imagens de pavimentos.

Optou-se por implementar duas variantes da ResNet: uma com 18 camadas e outra com 34 camadas (Figura 10). Essa escolha permite avaliar o impacto da profundidade da rede no desempenho da tarefa de identificação de defeitos. A função de ativação ReLU foi selecionada devido à sua eficiência computacional e à capacidade de introduzir não-linearidades nas camadas do modelo, essenciais para a aprendizagem de padrões complexos.

Além disso, visando mitigar o risco de overfitting, foi aplicada a técnica de regularização Dropout com uma taxa de 0.5. O Dropout, automaticamente incorporado pelo Fast.ai, desativa aleatoriamente unidades durante o treinamento, promovendo uma espécie de ensemble de redes neurais menores e, assim, melhorando a generalização do modelo.

A implementação do modelo foi realizada utilizando o Fast.ai e o PyTorch (Figura 10), aproveitando a eficiência e as facilidades proporcionadas por essas bibliotecas. O Fast.ai, construído sobre o PyTorch, oferece uma abstração de alto nível que simplifica o processo de treinamento, permitindo uma implementação mais rápida e intuitiva.

```
path = Path('pothole_dataset')

block = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    get_y=parent_label,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    item_tfms=[Resize(192, method='squish')]
)

dls = block.dataloaders(path, bs=32, pin_memory=False)

learn_18 = vision_learner(dls, resnet18, metrics=error_rate)
learn_34 = vision_learner(dls, resnet34, metrics=error_rate)
[13]  ✓ 0.6s
```

Figura 10 – Configuração da pipeline de dados, com o carregamento de dados de imagens, e criação de dois modelos de aprendizado profundo baseados na arquitetura ResNet (um com 18 camadas e outro com 34 camadas). O *DataBlock* é uma classe do Fast.ai para definir como os dados serão carregados, transformados e preparados para o treinamentos. Os *dataloaders* carregam os dados com base na configuração definida no bloco anterior. Já o *vision_learner* é uma função do Fast.ai que simplifica a criação de modelos para tarefas de visão computacional, recebendo os dataloaders, as arquiteturas pré-treinadas e também a métrica de erro *error_rate*, que será utilizada para avaliar o desempenho dos modelos.

Essas decisões combinadas representam uma abordagem sólida para a criação de um modelo de classificação de defeitos, considerando fatores como profundidade da rede, função de ativação, técnicas de regularização e ferramentas de desenvolvimento. Ao ajustar essas decisões, espera-se alcançar um equilíbrio ótimo entre complexidade do modelo e capacidade de generalização para identificação precisa de defeitos em pavimentos.

4.3 Transformações no conjunto de dados

Durante a etapa de Transformações no conjunto de dados a principal técnica aplicada foi a de *Data Augmentation*. Com base no *DataBlock* do último passo, é possível gerar um novo conjunto de dados (Figura 11) que passaram por transformações como rotação, espelhamento, zoom, translação, etc. Na Figura 12 é possível visualizar uma parte do conjunto de dados após o processo de *data augmentatio*n.

É importante ressaltar que todas as imagens também sofreram transformação por meio da classe 'Resize' no momento da criação do DataBlock(ver Figura 10). Nessa etapa todas as imagens foram ajustadas para conter 192 pixels e utilizando o método '*squish*', garantiu-se que o tamanho físico das imagens fosse padronizado. Isso é crucial para garan-

Figura 11 – Configuração da pipeline de dados utilizando a função *aug_transforms* para realizar transformações variadas e o *RandomResizedCrop* para escolher um corte em escala aleatória de uma imagem e redimensiona-la para o tamanho especificado (192)

```
block_augmented = block.new(
    item_tfms=RandomResizedCrop(192, min_scale=0.5),
    batch_tfms=aug_transforms(mult=2))
augmented_dls = block_augmented.dataloaders(path, bs=32, pin_memory=False)

learn_augmented_18 = vision_learner(augmented_dls, resnet18, metrics=error_rate)
learn_augmented_34 = vision_learner(augmented_dls, resnet34, metrics=error_rate)
[19]  ✓ 2.0s
```

Figura 12 – Seleção de imagens aleatórias com suas respectivas classificações e que passaram pelo processo de *data augmentation*.

tir que o modelo receba entradas consistentes, independentemente das dimensões originais das imagens no conjunto de dados.

4.4 Treinamento e avaliação dos modelos

Com os conjuntos de dados prontos selecionados e transformados, pode-se então iniciar o trinamento dos modelos. No processo de treinamento dos modelos, foram empregadas duas abordagens utilizando arquiteturas ResNet de 18 e 34 camadas. Em uma abordagem, os modelos foram treinados com o conjunto de dados original, enquanto, na outra, o conjunto de dados passou pela etapa de aumento de dados (data augmentation). Todos os modelos foram treinadas com 10 épocas.

Para garantir a avaliação robusta dos modelos durante o treinamento, foi adotada

uma estratégia de separação entre conjunto de validação e conjunto de testes. Isso é essencial para monitorar o desempenho do modelo em dados não vistos e evitar que o modelo se adapte demais aos dados de treinamento.

A função *RandomSplitter* do Fastai foi empregada para realizar essa divisão aleatória do conjunto de dados, reservando 20% dos dados para validação. A escolha de 20% foi uma decisão arbitrária e poderia ser ajustada conforme necessário, dependendo do tamanho do conjunto de dados e das características específicas da tarefa.

Durante o treinamento, a métrica de erro (*error_rate*) foi utilizada para avaliar a performance dos modelos. A função *error_rate* retorna a taxa de erro do modelo em relação às previsões corretas. É uma métrica comumente utilizada em tarefas de classificação, indicando a porcentagem de previsões incorretas em relação ao total de previsões.

Além disso, é fundamental ressaltar que dois conjuntos de modelos foram treinados, um utilizando o conjunto de dados original e outro utilizando o conjunto de dados aumentado. Os resultados dos treinamentos para os modelos podem ser vistos nos conjuntos de figuras 15 e 18. Essa abordagem visa explorar os benefícios do aumento de dados na capacidade do modelo de generalizar padrões, especialmente quando se tem um conjunto de dados relativamente pequeno.

epoch	train_loss	valid_loss	error_rate	time
0	0.131169	0.167284	0.058824	00:08
1	0.086842	0.172560	0.058824	00:07
2	0.057045	0.149396	0.036765	00:07
3	0.035882	0.163677	0.036765	00:06
4	0.024111	0.144779	0.051471	00:06
5	0.019588	0.203036	0.036765	00:06
6	0.015266	0.182196	0.036765	00:07
7	0.018507	0.155484	0.036765	00:06
8	0.013503	0.147640	0.036765	00:05
9	0.010542	0.154997	0.036765	00:06

Figura 13 – Resultado do treinamento com a resnet18

epoch	train_loss	valid_loss	error_rate	time
0	0.138230	0.170529	0.036765	00:08
1	0.068157	0.168133	0.044118	00:07
2	0.052527	0.079608	0.029412	00:07
3	0.045022	0.090256	0.036765	00:08
4	0.038333	0.129551	0.029412	00:08
5	0.027407	0.119770	0.029412	00:09
6	0.020602	0.069386	0.029412	00:08
7	0.017719	0.070593	0.029412	00:08
8	0.013685	0.089731	0.029412	00:09
9	0.009872	0.079643	0.029412	00:08

Figura 14 – Resultado do treinamento com a resnet34

Figura 15 – Treinamentos dos modelos ResNet 18 e 34 sem *data augmentation*, ambas com 10 épocas

Em resumo, a estratégia de treinamento envolveu a separação adequada dos conjuntos em uma taxa de, 80% treinamento e 20% teste, a escolha de arquiteturas ResNet 18 e 34, e a avaliação do desempenho por meio da métrica *error_rate*. A diferenciação entre conjuntos de dados original e aumentado oferece insights valiosos sobre os impactos do aumento de dados na qualidade do modelo. Por fim, é mostrado na Tabela 1 a taxa

epoch	train_loss	valid_loss	error_rate	time
0	0.285451	0.233166	0.058824	00:09
1	0.177833	0.202907	0.036765	00:09
2	0.121830	0.112667	0.044118	00:09
3	0.107203	0.273933	0.051471	00:08
4	0.095276	0.309916	0.044118	00:07
5	0.070506	0.295366	0.044118	00:06
6	0.069686	0.212871	0.036765	00:07
7	0.057173	0.201195	0.036765	00:08
8	0.046765	0.226527	0.044118	00:07
9	0.036472	0.219135	0.044118	00:06

Figura 16 – Resultado do treinamento da resnet18

epoch	train_loss	valid_loss	error_rate	time
0	0.162569	0.213736	0.044444	00:08
1	0.120984	0.122992	0.037037	00:08
2	0.113165	0.143992	0.051852	00:10
3	0.094536	0.070108	0.022222	00:11
4	0.076832	0.105778	0.037037	00:10
5	0.063352	0.103024	0.029630	00:10
6	0.054232	0.141839	0.037037	00:10
7	0.051180	0.133401	0.037037	00:09
8	0.038618	0.121742	0.029630	00:10
9	0.031404	0.125950	0.029630	00:09

Figura 17 – Resultado do treinamento da resnet34

Figura 18 – Treinamentos dos modelos ResNet 18 e 34 com *data augmentation*, ambas com 10 épocas

Modelo	**Taxa de Erro**
ResNet18	3.6765%
ResNet34	2.9412%
ResNet18 Data Augmentation	4.4118%
ResNet34 Data Augmentation	2.9630%

Tabela 1 – Taxas de erro dos modelos treinados.

de erro de cada modelo[3].

Com o treinamento finalizado é possível avaliar em maiores detalhes onde os modelos erraram. Para isso foi utilizado a matriz de confusão. A matriz de confusão é uma Tabela que é utilizada para descrever o desempenho de um modelo de classificação em um conjunto de dados. Onde cada linha representa as instâncias de uma classe predita, enquanto cada coluna representa as instâncias em uma classe real (ou vice-versa). Ao todo foram gerados quatro matrizes de confusão, uma para cada modelo.

Como pode ser visto no conjunto de figuras 21. temos um total de cinco previsões errôneas para a resnet18 enquanto que o modelo resnet34 obteve uma taxa levemente menor de previsões errôneas, com quatro erros de previsão. A resnet34 conseguiu se sair melhor que a resnet18 prevendo imagens que continham a classificação normal.

Já no conjunto de figuras 24 é possível visualizar uma leve piora nas predições da resnet18 com *data augmentation* com relação resnet18. Enquanto isso a resnet34 com *data augmentation* obteve um desempenho idêntico a resnet34.

[3] É importante lembrar que a Taxa de Erro pode variar para cada treinamento. Isso ocorre pois as imagens contidas nos conjuntos de treinamento e de teste são escolhidos aleatoriamente e com isso não é possível garantir que sempre gere o mesmo resultado.

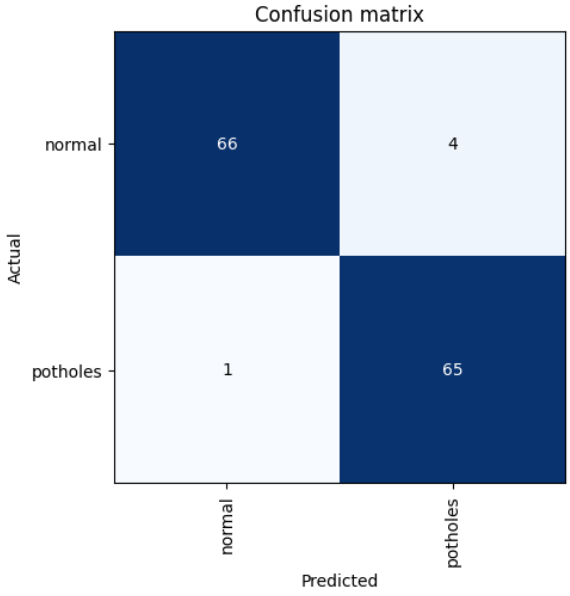


Figura 19 – Matriz de confusão resnet18.

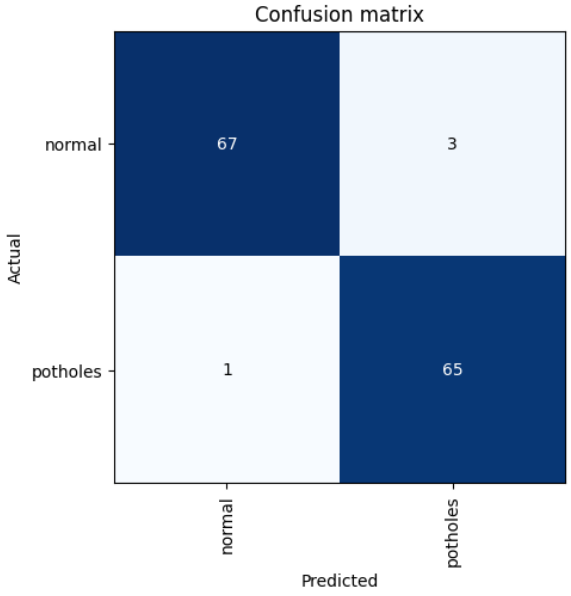


Figura 20 – Matriz de confusão resnet34.

Figura 21 – Matrizes de confusão dos modelos resnet 18 e 34 sem *data augmentation*.

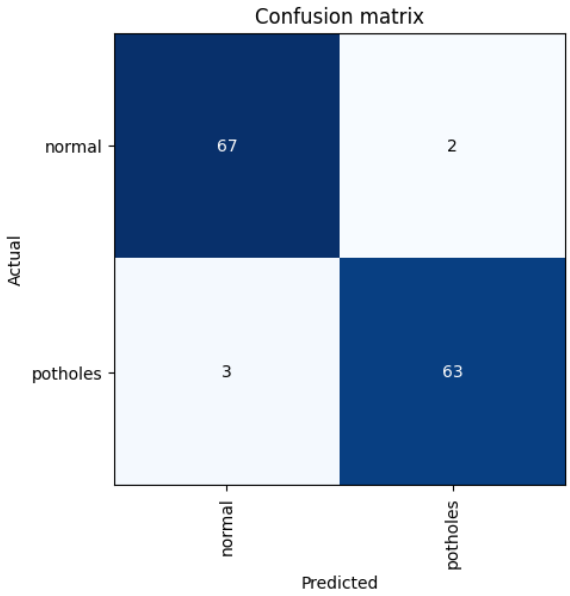


Figura 22 – Matriz de confusão resnet18.

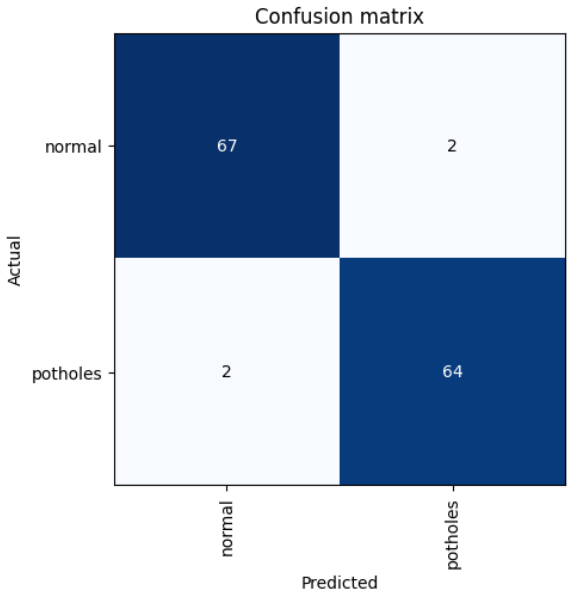


Figura 23 – Matriz de confusão resnet34.

Figura 24 – Matrizes de confusão dos modelos resnet 18 e 34 com *data augmentation*.

Após analisar as matrizes de confusão também é possível observar especificamente quais imagens os modelos erraram em sua predição. Com relação a resnet18 houveram cinco predições errôneas e a Figura 25 mostra quais foram as imagens.

Ná Figura 26 é possível ver que, das cinco imagens, quatro são predições errôneas

Figura 25 – Maiores perdas da resnet18.

Figura 26 – Maiores perdas da resnet34.

e que o conjunto de erros é quase que o mesmo.

Já nas predições dos modelos treinados com *data augmentation* pode-se notar através das figuras 27 e 28 que o conjunto de de erros também é bem similar.

É possível notar também que a predição da imagem que contém o 'carro preto na neve' está presente em todas as figuras e mesmo que na Figura 26 a predição esteja correta a probabilidade não é tão alta (70%). Isso pode ser um indicativo de que o *dataset*

Figura 27 – Maiores perdas da resnet18 com *data augmentation*.

carece de imagens em ambientes de neve e, por consequência, os modelos não conseguem extrair informações o suficiente para generalizar o conhecimento. Além desta imagem, outras podem não ser tão interessantes para o treinamento dos modelos, como a imagem do carro tombado na 28.

Dito isso, algumas estratégias de tratamento de dados podem ser adotadas para reduzir os erros. A adição de novos dados ao *dataset* ou a remoção de dados incoerentes são possibilidades que devem ser levadas em conta. Junto a isso, alguns fatores podem ser determinantes para a adoção das estratégias, por exemplo: caso o modelo seja de uso específico em regiões tropicais, não há sentido em treinar o modelo com imagens que contenham neve, portanto faz sentido remover essas imagens.

Figura 28 – Maiores perdas da resnet34 com *data augmentation*.

Por fim, fora gerado um total de quatro modelos para a tarefa de identificação de defeitos em pavimentos (resnet18, resnet34, resnet18 com *data augmentation* e resnet34 com *data augmentation*). Dentre eles os mais confiáveis são os modelos resnet34 e o resnet34 com *data augmentation*, com taxas de erros de 2.9412% e 2.9630%, respectivamente. Embora os modelos gerados cumpram a tarefa proposta com certa precisão, eles ainda possuem suas limitações. Os modelos tem a capacidade de apenas dizer se há ou não um defeito presente em uma imagem. Entretanto pode ser interessante que o modelo seja capaz de realizar uma detecção de objeto sobre os defeitos e assim possibilitando um melhor levantamento da quantidade de defeitos, bem como a magnitude de cada defeito contido em uma imagem. Todos os passos tomados para a criação do modelo, bem como o modelo gerado podem ser vistos no *GitHub*[4].

4.5 Aprimorando o Modelo

No intuito de melhorar a utilidade do modelo fora necessário mudar a arquitetura do modelo. Com isso, os modelos baseados em resnet deixam de ser usados em favor dos modelos YOLO. Os modelos YOLO são uma das mais novas abordagens na detecção de objetos, onde uma única rede neural é capaz de prever as probabilidades das classes e os contornos dos objetos diretamente da imagem em uma única avaliação (REDMON et al., 2016). Por realizar ambas as tarefas em uma única avaliação essa arquitetura se torna muito rápida, tornando-a ideal para atividades de processamento de imagens e vídeos

[4] Notebook Disponível em:<https://github.com/SamuelNoB/TCC1/blob/main/model.ipynb>

em tempo real. O YOLO possui diversas versões e para este trabalho fora utilizado o YOLOv7[5].

4.5.1 Seleção do conjunto de dados

Para o treinamento do modelo fora necessário a criação de um novo conjunto de dados em que houvesse imagens de defeitos, bem como os rótulos dos defeitos. Para isso, foram utilizados dois datasets distintos que foram mesclados. O primeiro disponível na plataforma *LearnOpenCV*[6]. Este dataset é uma mesclagem de dois outros datasets o *Roboflow Pothole Dataset*[7] e o segundo é o *Dataset of images used for pothole detection* (NIENABER; BOOYSEN; KROON, 2015). O segundo dataset utilizado na mesclagem é o *Potholes Detection for YOLOv4*[8].

A escolha desses conjuntos de dados foi fundamentada pela necessidade de ter uma diversidade de imagens que representassem bem a detecção de defeitos em pavimentos, proporcionando uma base robusta para o treinamento do YOLOv7.

O Dataset da LearnOpenCV continha um total de 1794 imagens, divididas em três subconjuntos: 1275 imagens para treinamento (*train*), 401 imagens para validação (*valid*), e 118 imagens para teste (*test*), seguindo uma proporção de divisão de 70/20/10. Este dataset já se encontrava no formato compatível com o YOLOv7, o que facilitou a sua utilização direta no treinamento do modelo. As anotações estavam organizadas em arquivos de texto, com cada linha representando uma detecção no formato **class x_center y_center width height** como pode ser visto na Tabela 2.

Tabela 2 – Exemplo de anotação para o yolov7

class	**x_center**	**y_center**	**width**	**height**
0	0.459134615	0.165865384	0.15865	0.14423076

Por outro lado, o dataset *Potholes Detection for YOLOv4* consistia em 1984 imagens, dividido em uma proporção de 80/20 para os conjuntos de treinamento (*train*) e teste (*test*), sem incluir um conjunto de validação. Diferentemente do dataset da *LearnOpenCV*, as anotações deste dataset poderiam estar no formato YOLOv4, o que poderia exigir uma transformação para torná-las compatíveis com o formato necessário para o YOLOv7.

Para harmonizar os formatos dos dois datasets, foi necessário realizar algumas transformações nos dados do Kaggle utilizando um *Jupyter Notebook*[9]. Primeiramente, as

[5] <https://github.com/WongKinYiu/yolov7>
[6] <https://learnopencv.com/pothole-detection-using-yolov4-and-darknet/>
[7] <https://public.roboflow.com/object-detection/pothole>
[8] <https://www.kaggle.com/datasets/anugrahakbar/potholes-detection-for-yolov4>
[9] <https://github.com/SamuelNoB/TCC1/blob/main/dataset_merging.ipynb>

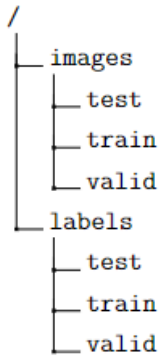


Figura 29 – Estrutura de pastas do dataset final

anotações do *Potholes Detection for YOLOv4* foram avaliadas a fim de validar se o formato se encontrava compatível com o formato de anotações aceitar no YOLOv7. Uma avaliação visual de algumas anotações confirmou que as anotações eram de fato compatíveis. Com isso, imagens e suas respectivas anotações de ambos os datasets foram combinadas em uma estrutura de diretórios como é possível ver na Figura 29 . Isso facilitou o uso direto dos dados durante o treinamento do modelo YOLOv7.

Por fim, a seleção e preparação dos conjuntos de dados foram etapas críticas no desenvolvimento do modelo de detecção de defeitos. A transformação necessária para harmonizar os dados do Kaggle com o formato do YOLOv7 e a combinação dos dois *datasets* garantiram a compatibilidade e qualidade das anotações. Como resultado foi gerado um *dataset* composto de 3777 imagens sendo 2837 imagens de treino 611 imagens de validação e 329 imagens de teste com uma proporção de 75/16/9.

4.5.2 Criação e treinamento do modelo

A criação do modelo de detecção de defeitos foi realizada em um ambiente Jupyter Notebook[10], hospedado no Google Colab. Esta escolha foi motivada pela necessidade de recursos computacionais robustos, fornecidos pela plataforma, essenciais para o treinamento eficaz de modelos de *deep learning*.

O modelo selecionado para este projeto foi o YOLOv7 Tiny, uma versão otimizada e de resolução fixa do YOLOv7, conhecida por sua eficiência e velocidade em tarefas de detecção de objetos. O uso do YOLOv7 Tiny foi estratégico para balancear a precisão da detecção com a demanda por recursos computacionais, possibilitando uma implementação mais rápida e leve.

Para ajustar o modelo às especificidades do *dataset* criado na etapa anterior, foi realizado um *fine tuning* do modelo base. Para indicar ao modelo o local dos arquivos de

[10] <https://github.com/SamuelNoB/TCC1/blob/main/yolov7_pothole_detection_model.ipynb>

Tabela 3 – Parâmetros na última época

Época	**P**	**R**	**mAP**
99	0.742	0.652	0.682

treino, teste e validação fora necessário criar um arquivo *yaml* indicando os caminhos dos dados e também informações sobre as classes.

O treinamento do modelo foi conduzido ao longo de 100 épocas, com um *batch-size* de 64 e 8 *workers*. Este número de épocas foi escolhido para garantir que o modelo tivesse tempo suficiente para aprender as características dos defeitos nas imagens, sem sobreajustar-se aos dados de treino. O tamanho do batch-size e a quantidade de workers fora definida de modo a garantir o maior aproveitamento possível dos recursos disponíveis a fim de acelerar o treinamento do modelo.

Durante o treinamento, os parâmetros de *mean Avarage Precision*(mAP) *Precision* (P) e *Recall* (R) foram monitorados para avaliar o desempenho do modelo e evitar o *overfitting*. A utilização do conjunto de validação foi importante para ajustar hiperparâmetros e realizar avaliações intermediárias, assegurando que o modelo mantivesse uma boa generalização aos dados não vistos. Com isso na última época obteve-se os parâmetros descritos na Tabela 3.

Em resumo, a criação do modelo envolveu a utilização do YOLOv7 Tiny, um Jupyter Notebook no Google Colab para treinamento, a aplicação de *fine tuning* com o *dataset* específico de defeitos em pavimentos, e um treinamento intensivo ao longo de 100 épocas. Este processo detalhado e cuidadoso garantiu a construção de um modelo para a detecção de defeitos, atendendo aos requisitos do projeto.

4.5.3 Resultados do treinamento

Após o treinamento do modelo fora disponibilizado uma série estatísticas a respeito dele durante seu processo de treinamento(Figura 30). Nos dois primeiros gráficos da Figura 30 de *Box Loss* e *Objectness Loss* pode-se perceber uma tendência decrescente ao longo das épocas, isso significa que o modelo está melhorando sua capacidade de prever as caixas delimitadoras (*bounding boxes*) dos objetos nas imagens bem como está ficando melhor em distinguir entre regiões que contêm objetos e regiões que não contêm objetos. Já o gráfico de *Classification Loss* se encontra praticamente vazio pois o modelo fora treinado para detectar apenas uma classe, o que faz o loss de classificação praticamente insignificante, já que não há a necessidade de classificar entre múltiplas classes.

Os dois últimos gráficos da primeira linha da Figura 30 dizem respeito às métricas de avaliação *Precision*, *Recall* e é possível notar seus valores subindo rapidamente nas primeiras épocas e se estabilizando em torno de 0.6 a 0.7 para a *Precision* e 0.6 para o

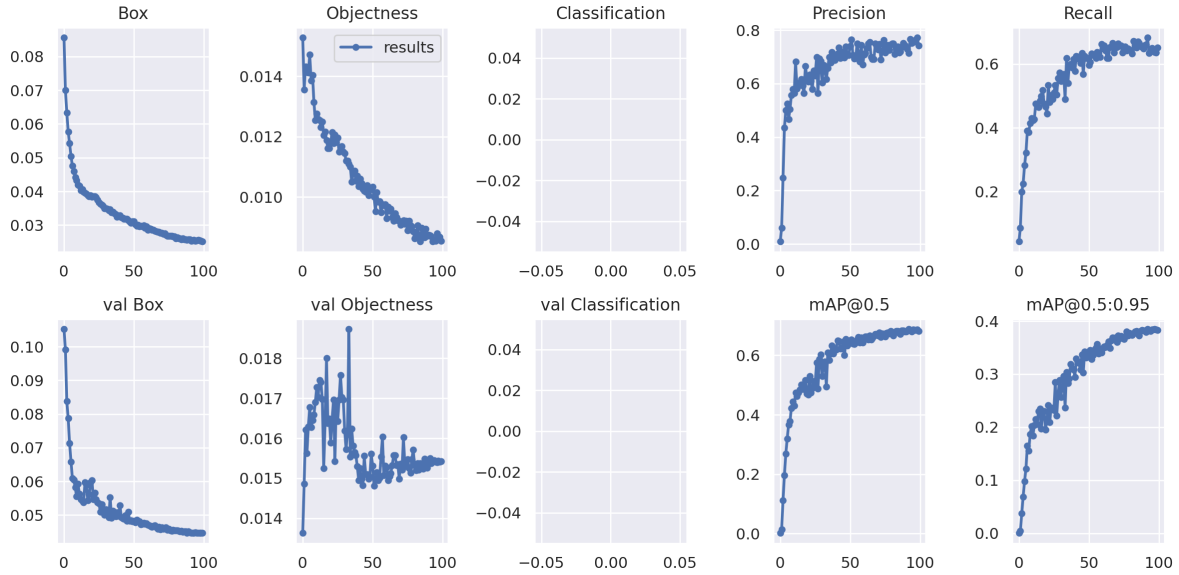


Figura 30 – Resultados do treinamento

Recall. Sendo possível inferir que o modelo estava se tornando mais preciso na identificação dos objetos, com uma taxa relativamente alta de verdadeiros positivos em relação aos falsos positivos. Bem como está detectando a maioria dos objetos presentes nas imagens.

Nos gráficos de validação *val Box*, *val Objectness* observa-se uma tendencia de queda em ambos, embora o *val Objectness* tenha alguma variação. Isso significa que o desempenho do modelo na previsão de *bounding boxes* bem como a capacidade do modelo de distinguir de distinguir entre regiões com e sem objetos estão melhorando também no conjunto de validação. Já o *val Classification* se encontra vazio pois é a mesma situação do *Classification Loss*.

Com relação aos gráficos de mAP (mAP@0.5, mAP@0.5:0.95) da Figura 30 é possível notar no gráfico mAP@0.5 um aumento um aumento contínuo e se estabiliza em torno de 0.6 a 0.7. Nota-se que modelo tem uma boa precisão média em detectar objetos quando se utiliza um limiar de *Intersection over Union* (IoU) de 0.5. O mAP@0.5:0.95 também aumenta, mas atinge valores mais baixos, em torno de 0.3 a 0.4, portanto a precisão média em diferentes limiares de IoU é menor, indicando que o desempenho do modelo pode variar mais com diferentes limiares de IoU.

Com isso conclui-se que os gráficos de *loss* (*Box*, *Objectness*) mostram uma convergência clara, indicando que o modelo está aprendendo durante o treinamento. Tanto a precisão quanto o *recall* aumentam significativamente e se estabilizam, o que é um bom sinal de que o modelo está funcionando bem, mas ainda há espaço para melhorias. A tendência de queda nas perdas de validação (*val Box*, *val Objectness*) sugere que o modelo também está generalizando bem para dados não vistos. Um mAP@0.5 de cerca de 0.6-0.7 é indicativo de um bom desempenho na detecção de objetos, mas o mAP@0.5:0.95 mais

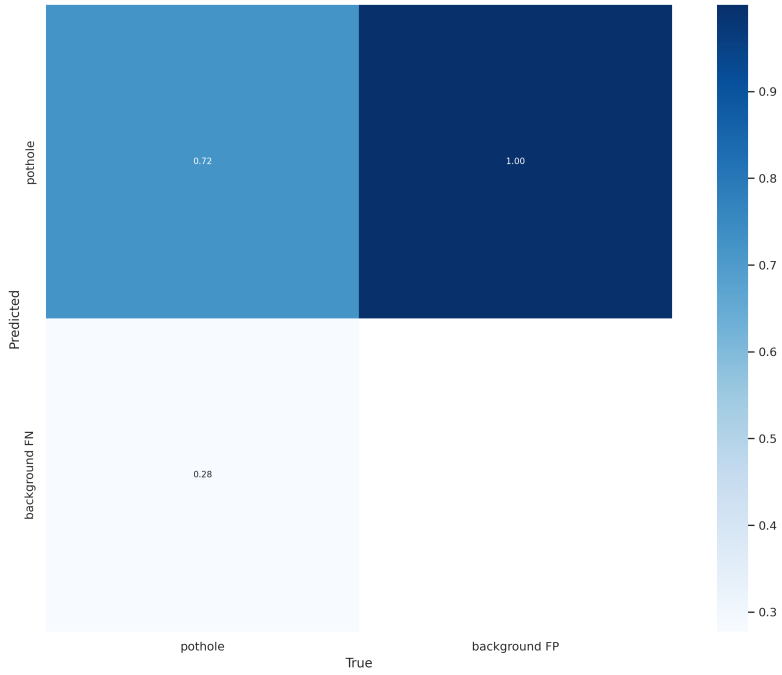


Figura 31 – Matriz de confusão resultante do treinamento

baixo sugere que o modelo pode precisar de ajustes adicionais para melhorar a robustez a diferentes limiares de IoU.

4.5.3.1 Matriz de confusão

Com a conclusão do treinamento também foi possível gerar e visualizar a matriz de confusão que representa o desempenho do modelo no conjunto de dados de treino e validação na Figura 31. Essa matriz mostrou a eficácia do modelo na predição de duas classes principais: "*pothole*"(defeitos) e "*background*"(fundo).

O desempenho do modelo na classe *pothole* foi razoável. Ele conseguiu identificar corretamente os defeitos 72% das vezes, demonstrando um bom desempenho nessa classe. No entanto, houve uma taxa de falsos negativos de 28%, onde o modelo falhou em identificar defeitos, classificando-os como "*background*".

4.5.3.2 Confiança e precisão

Outra métrica importante disponibilizada pelo modelo fora a relação entre confiança e precisão, visível na Figura 32. Ele possibilita entender o desempenho do modelo em diferentes níveis de confiança. Na figura, a curva ascendente indicou que, à medida que a confiança aumentou, a precisão do modelo também aumentou. Inicialmente, com baixa confiança, a precisão foi relativamente baixa, o que significava que muitas das predições de baixa confiança foram incorretas. À medida que a confiança aumentou, a precisão me-

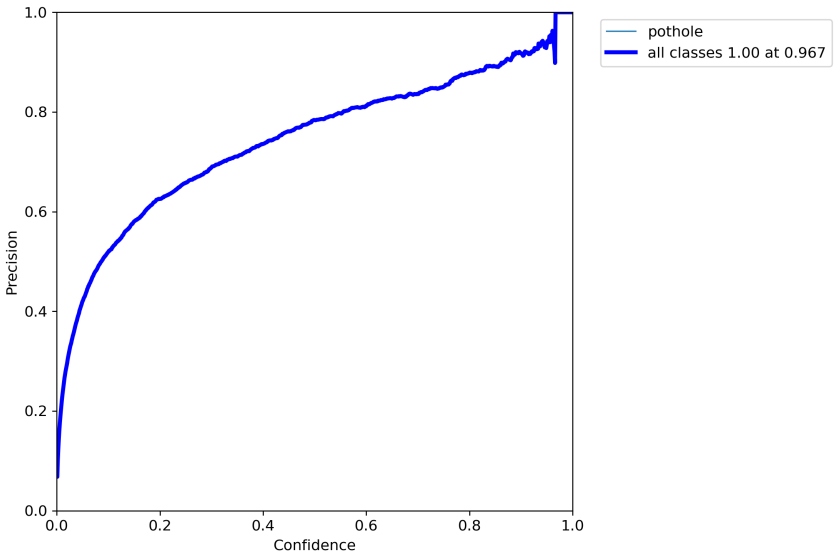


Figura 32 – Gráfico de confiança e precisão

lhorou significativamente, demonstrando que o modelo foi capaz de fazer predições mais precisas quando teve maior certeza.

No ponto em que a confiança alcançou aproximadamente 0.90, a precisão atingiu seu valor máximo, sendo 1.00 para todas as classes. Esse ponto crítico mostrou que, quando o modelo esteve quase totalmente confiante em suas predições, ele não cometeu erros. No entanto, é importante notar que, embora a precisão tenha sido alta, a quantidade de predições feitas pode ter sido reduzida à medida que a confiança aumentou, pois o modelo se tornou mais conservador.

É importante definir um limiar de confiança ideal para o uso prático do modelo, pois definir um limiar de confiança muito baixo poderia resultar em muitas falsas detecções, enquanto um limiar muito alto poderia fazer com que o modelo perdesse detecções importantes.

4.5.3.3 Confiança e Recall

A relação entre *recall* e confiança pode ser vista na Figura 33. O gráfico ajuda a encontrar um equilíbrio entre evitar falsos negativos (perder detecções verdadeiras) e falsos positivos (detecções incorretas). A curva descendente indica que, à medida que a confiança aumenta , menor é a capacidade do modelo de recuperar corretamente todos os exemplos verdadeiros positivos em relação ao total de exemplos positivos presentes no conjunto de dados.

No início da curva, com um nível de confiança próximo de 0, o modelo atingiu um *recall* de aproximadamente 0.85, indicando que, em um cenário com baixa confiança, o modelo foi capaz de detectar a maioria dos verdadeiros positivos. Contudo, à medida

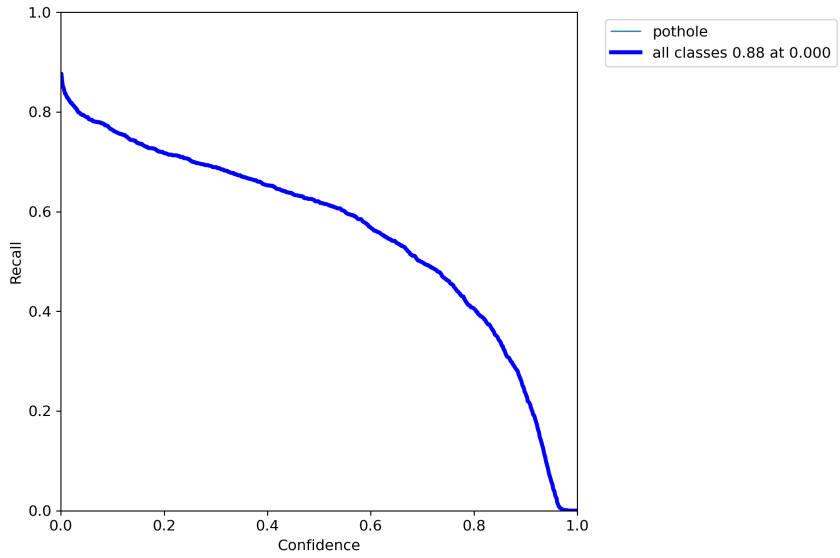


Figura 33 – Gráfico de confiança e recall

que a confiança aumentou, o *recall* diminuiu gradualmente, mostrando uma troca entre *recall* e precisão. Isso significa que, para garantir maior precisão e reduzir falsos positivos, o modelo passou a identificar menos exemplos verdadeiros positivos.

4.5.3.4 Curva F1

O gráfico de curva F1 visível na Figura 34 gerado como resultado do treinamento do modelo apresentou a relação entre confiança (*confidence*) e a métrica F1. A métrica F1 é a média harmônica entre precisão (*precision*) e sensibilidade (*recall*), fornecendo uma medida única de desempenho do modelo que considera tanto a taxa de verdadeiros positivos quanto a taxa de falsos negativos.

A curva de F1 apresentou um comportamento típico em forma de sino. Inicialmente, quando a confiança era baixa, a métrica F1 também era baixa, indicando que, embora o modelo estivesse fazendo muitas predições, muitas dessas predições eram incorretas ou pouco confiáveis. À medida que a confiança aumentou, a métrica F1 subiu, atingindo um pico onde o modelo conseguiu um bom equilíbrio entre precisão e sensibilidade.

O pico da curva, onde a métrica F1 foi de aproximadamente 0.65 com uma confiança de 0.40, indicou o ponto em que o modelo atingiu seu melhor desempenho geral. Esse ponto representou o melhor equilíbrio entre a capacidade do modelo de identificar corretamente as verdadeiras classes (defeitos) e a minimização de erros (falsos positivos e falsos negativos). Em um cenário prático, escolher um limiar de confiança próximo ao ponto de pico da curva de F1 garantiria que o modelo tivesse um desempenho otimizado, equilibrando corretamente precisão e sensibilidade.

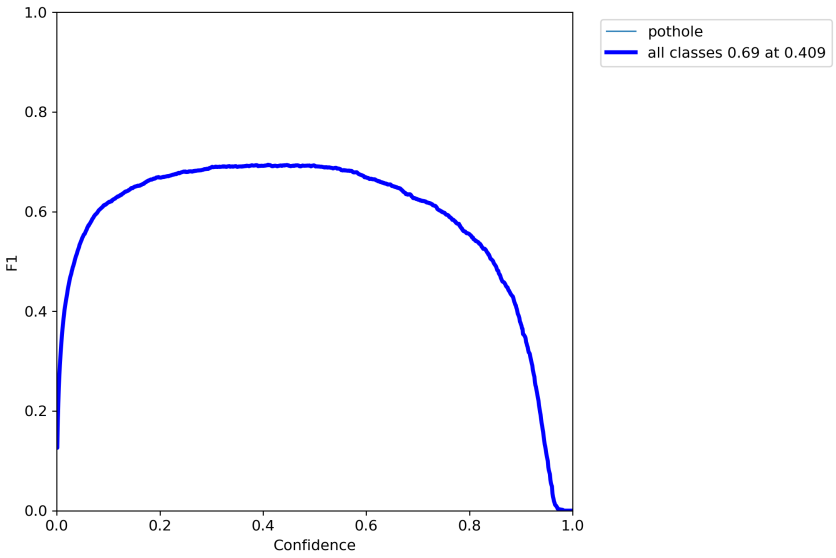


Figura 34 – Gráfico de curva F1

P	**R**	**mAP@.5**	**mAP@.5:.95**
0.762	0.757	0.775	0.479

Tabela 4 – Resultado da validação do modelo sobre o conjunto de teste

4.5.4 Validação do modelo

Para a validação do modelo, foi utilizado o *dataset* de teste de 329 imagens para avaliar o desempenho do modelo treinado. O processo de validação visou fornecer uma análise detalhada da eficácia do modelo na detecção de defeitos nas vias.

Os resultados obtidos e disponíveis na Tabela 4.5.4 revelaram uma precisão de 0.762, indicando que $76,2\%$ das detecções realizadas pelo modelo foram corretas. Esse valor refletiu a capacidade do modelo em minimizar falsos positivos, ou seja, a detecção de defeitos onde não existiam.

O *recall* do modelo foi de 0.757, o que demonstrou que o modelo conseguiu identificar 757% dos defeitos presentes no conjunto de dados de teste. Este valor destacou a eficiência do modelo em detectar verdadeiros positivos, embora ainda houvesse espaço para melhorar a recuperação de todas as instâncias de defeitos.

O mAP@.5 (*mean Average Precision at IoU threshold of 0.5*) foi de 0.775, indicando que o modelo atingiu uma precisão média de 77.5% considerando uma interseção sobre a união (IoU) de 50%. Este resultado refletiu a capacidade geral do modelo de detectar e localizar defeitos com precisão aceitável.

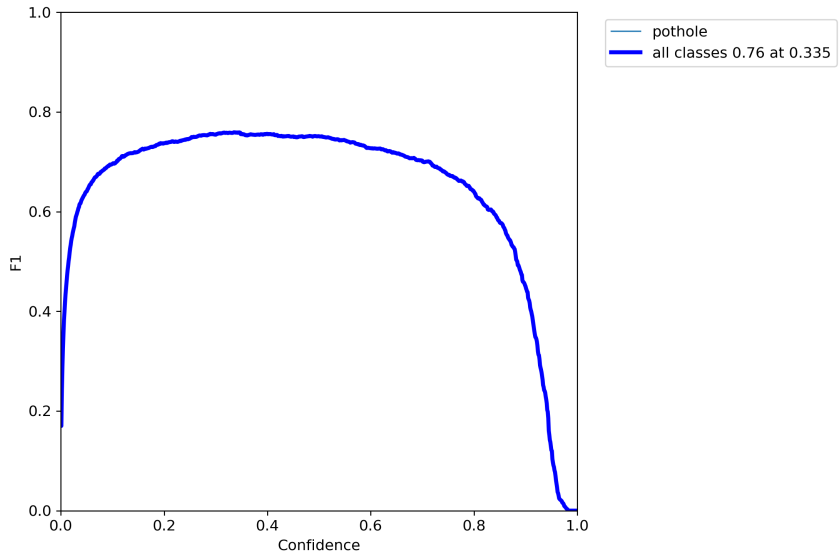


Figura 35 – Gráfico de curva F1 da validação do modelo.

Por outro lado, o mAP@.5:.95 (*mean Average Precision at IoU thresholds from 0.5 to 0.95*) foi de 0.479. Este valor, menor que o mAP@.5, demonstrou que a precisão média do modelo diminuiu quando foram considerados limiares de IoU mais rigorosos. Esse resultado evidenciou a dificuldade do modelo em manter alta precisão ao exigir maior sobreposição entre as detecções e as verdadeiras localizações dos defeitos.

Ao observar o gráfico de curva F1 da Figura 35 é possível notar que o melhor índice de confiança é de 0.335 e gera um F1 de 0.76. Já o gráfico de precisão gerado no processo de validação e visível na Figura 36 indica uma precisão de 1.0 para uma confiança de 0.947.

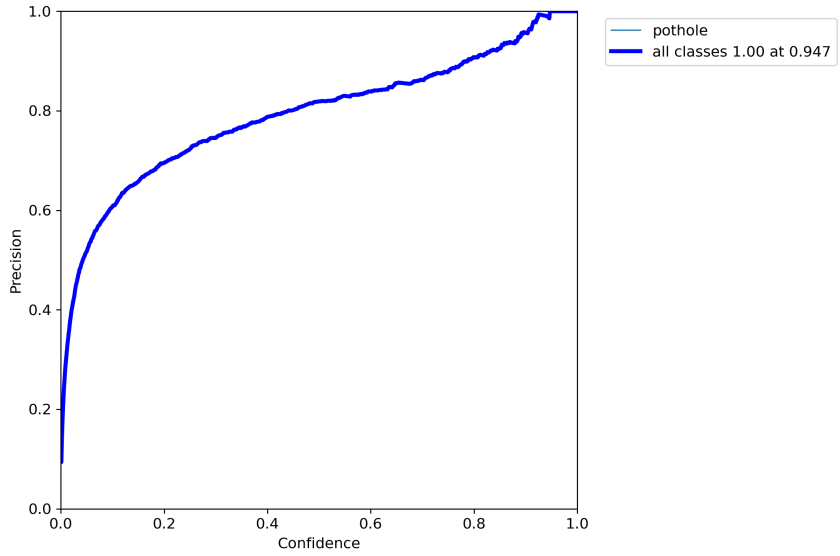


Figura 36 – Gráfico de confiança e precisão da validação do modelo.

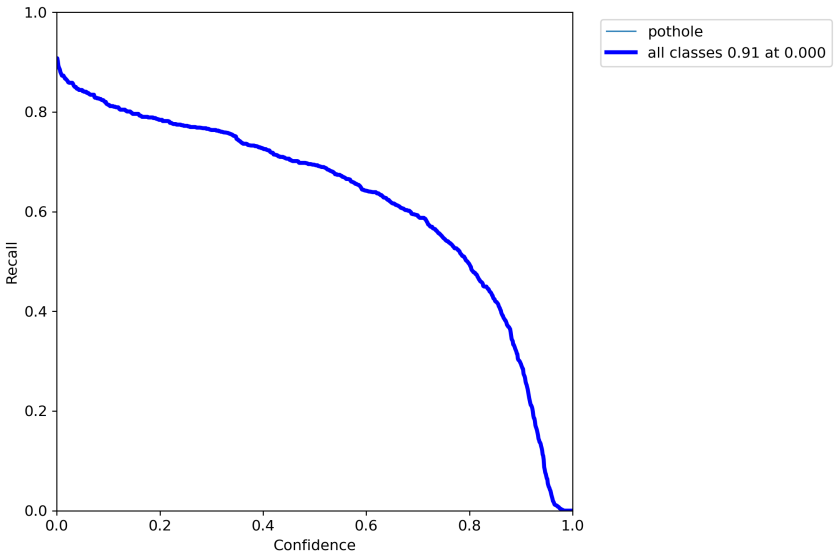


Figura 37 – Gráfico de confiança e recall da validação do modelo.

Figura 38 – Imagens anotadas de um conjunto de teste.

O gráfico de Recall da Figura 37 mostra o oposto do gráfico de precisão 36, com uma confiança de 0.0 o maior *recall* foi de 0.91. À medida que a confiança aumenta, o recall diminui, evidenciando que o modelo se torna mais conservador e menos propenso a identificar corretamente todos os exemplos positivos. Essa relação descendente demonstra o equilíbrio que deve ser alcançado entre a confiança nas predições e a capacidade do modelo de recuperar todas as instâncias relevantes.

Outra informação relevante proveniente do resultado do treinamento foram os *batches* gerados como parte do processo de validação. A Figura 38, mostra um conjunto

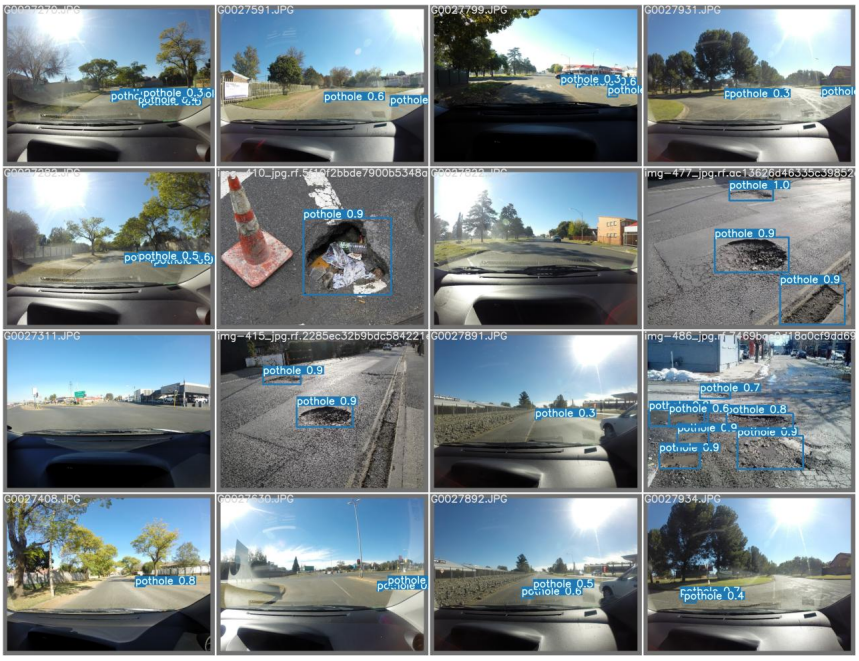


Figura 39 – Imagens com as predições realizada pelo modelo.

de imagens de teste com suas respectivas anotações de defeitos ("*pothole*"). A Figura 39, apresentou as predições realizadas pelo modelo para o mesmo conjunto de imagens. Comparando essas duas imagens, foi possível avaliar visualmente a precisão e *recall* do modelo, observando como as predições correspondem às anotações reais.

Após o processo de treinamento e validação, o modelo foi utilizado em um caso prático com um vídeo que mostra uma estrada com defeitos em seu pavimento. Com o vídeo, o modelo identificou e adicionou os *bounding boxes* sobre os defeitos encontrados. É possível visualizar trechos do vídeo na Figura 40. O vídeo está disponível para visualização no *YouTube*[11]

[11] Modelo sendo usado sobre um vídeo. Disponível em: <https://youtu.be/5wXOu3scNF4>

Figura 40 – Modelo testado sobre vídeo

5 Conclusão

O presente estudo teve como objetivo primordial a aplicação de um modelo de *Deep Learning* capaz de localizar defeitos em pavimentos. Para tanto, diversas etapas foram meticulosamente percorridas, a começar pela seleção e aquisição do conjunto de dados, seguida de sua respectiva análise e tratamento. Posteriormente, procedeu-se com a implementação de modelos Resnet e YOLO, permitindo que alcançássemos o propósito estabelecido.

No processo, conseguimos analisar o desempenho de tais modelos e efetuar uma comparação entre os diferentes modelos ResNet. Foi analisado as diferentes matrizes de confusão dos modelos e pode-se notar que os modelos ResNet com menores taxas de erros foram baseados na ResNet34 e ResNet34 com *data augmentation*. Entretanto os modelos ResNet se limitam a apenas classificar imagens, sem localizar o defeito propriamente dito.

Para que conseguíssemos localizar os defeitos foi preciso mudar o tipo de modelo para o YOLOv7. O modelo foi treinado com um novo *dataset* que foi resultante de uma mesclagem de outros três *datasets*. Com a conclusão do treinamento, verificou-se a precisão do modelo por meio de sua matriz de confusão e o modelo foi testado com um conjunto de teste, obtendo então sua precisão, *recall*, mAP e F1. Por fim o modelo foi utilizado para fazer predições sobre um vídeo.

O resultado do experimento revelou que, apesar dos modelos terem se mostrado eficazes na tarefa proposta, há espaço para aperfeiçoamento. Como primeira sugestão para estudos futuros, recomenda-se incrementar o conjunto de dados do modelo yolov7 com mais imagens rotuladas e também aplicar o *data augmentation*. Como o modelo YOLO utilizado foi o YOLOv7 Tiny com resolução fixa outra sugestão é realizar treinamento do YOLOv7 Tiny com múltiplas resoluções ou utilizar o YOLOv7 normal de Resolução Fixa ou de Múltipla Resolução. Utilizar a versão 8 do YOLO também poderia tornar o modelo ainda mais robusto em detectar defeitos nos pavimentos.

Em resumo, esta pesquisa atingiu seu objetivo de aplicar o *Deep Learning* para detectar defeitos em pavimentos, mas ainda há um vasto campo de possibilidades para aprimoramentos futuros. Com a contínua evolução das técnicas de aprendizado de máquina, espera-se que as futuras iterações deste trabalho possam lidar com a tarefa proposta de maneira ainda mais eficaz.

6 Cronograma

Tabela 5 – Cronograma do projeto

	Setembro	Outubro	Novembro	Dezembro	Janeiro /Março	Abril	Maio	Junho	Julho
Discussão sobre o tema									
Planejamento									
Pesquisa e Desenvolvimento do modelo									
Escrita e apresentação									
Pesquisa por outras soluções									
Aprimoramento do modelo									
Revisão final									
Entrega e Apresentação final									

Referências

FASTAI. *Fastai - Computer Vision Intro*. 2023. Https://docs.fast.ai/tutorial.vision.html. Citado na página 39.

FASTAI. *Fast.Ai - Fast.Ai—Making Neural Nets Uncool Again*. 2023. Https://www.fast.ai/. Citado na página 39.

GÉRON, A. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Second edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2019. (Covid-19 Collection). ISBN 978-1-4920-3264-9 978-1-09-812597-4. Citado 5 vezes nas páginas 25, 27, 28, 30 e 31.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, Massachusetts: The MIT Press, 2016. (Adaptive Computation and Machine Learning). ISBN 978-0-262-03561-3. Citado 5 vezes nas páginas 23, 24, 28, 29 e 30.

GOOGLE. *Google Colaboratory*. 2019. Https://colab.research.google.com/. Citado na página 39.

HE, K. et al. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2015. p. 770–778. ISBN 978-1-4673-8851-1. Citado 2 vezes nas páginas 33 e 34.

HOWARD, J.; GUGGER, S.; CHINTALA, S. *Deep Learning for Coders with Fastai and PyTorch: AI Applications without a PhD*. First edition. Sebastopol, California: O'Reilly Media, Inc, 2020. ISBN 978-1-4920-4552-6. Citado na página 26.

Jupyter. *Project Jupyter*. 2023. Https://jupyter.org. Citado 2 vezes nas páginas 38 e 39.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems*. [S.l.]: Curran Associates, Inc., 2012. v. 25. Citado na página 32.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, Nov./1998. ISSN 00189219. Citado na página 32.

LIMA, I. *Inteligência artificial*. [S.l.]: Elsevier, 2017. ISBN 978-85-352-7808-8. Citado na página 23.

NIENABER, S.; BOOYSEN, M. T.; KROON, S. *Dataset of Images Used for Pothole Detection*. 2015. Citado na página 51.

NORVIG, P. *Inteligência Artificial*. 4. ed. Rio de Janeiro, RJ: Grupo Gen, 2022. ISBN 978-85-951588-7-0. Citado na página 23.

O'CONNOR, R. *PyTorch vs TensorFlow in 2023*. 2021. Https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/. Citado na página 40.

OLIVEIRA, R. F. de. *Inteligência artificial*. [S.l.]: Editora e Distribuidora Educacional S.A., 2021. ISBN 978-85-522-1141-9. Citado na página 23.

O'SHEA, K.; NASH, R. *An Introduction to Convolutional Neural Networks*. [S.l.]: arXiv, 2015. Citado 6 vezes nas páginas 25, 26, 27, 30, 31 e 32.

PyTorch. *PyTorch*. 2023. Https://www.pytorch.org. Citado na página 39.

REDMON, J. et al. *You Only Look Once: Unified, Real-Time Object Detection*. [S.l.]: arXiv, 2016. Citado na página 50.

SIMONYAN, K.; ZISSERMAN, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. [S.l.]: arXiv, 2015. Citado na página 32.

SZEGEDY, C. et al. Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, 2015. p. 1–9. ISBN 978-1-4673-6964-0. Citado na página 33.