UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO DE INFORMÁTICA

IAN DOUGLAS ALMEIDA QUEROS

SISTEMA DE INSPEÇÃO DE TALAS DE JUNÇÃO FERROVIÁRIAS VIA COMPUTAÇÃO DE BORDA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2023

IAN DOUGLAS ALMEIDA QUEROS

SISTEMA DE INSPEÇÃO DE TALAS DE JUNÇÃO FERROVIÁRIAS VIA COMPUTAÇÃO DE BORDA

Rail Joint Inspection System Using Edge Computing

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação, do Bacharelado em Engenharia da Computaçãoda Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Heitor Silvério Lopes
Coorientador: Guilherme de Moraes Restani

CURITIBA

2023

IAN DOUGLAS ALMEIDA QUEROS

SISTEMA DE INSPEÇÃO DE TALAS DE JUNÇÃO FERROVIÁRIAS VIA COMPUTAÇÃO DE BORDA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 11/dezembro/2023

Heitor Silvério Lopes
Doutorado
Universidade Tecnológica Federal do Paraná

João Alberto Fabro
Doutorado
Universidade Tecnológica Federal do Paraná

Fabio Kurt Schneider
Doutorado
Universidade Tecnológica Federal do Paraná

CURITIBA
2023

Dedico este trabalho a minha família, pelos momentos de ausência.

AGRADECIMENTOS

Este trabalho não poderia ser terminado sem a ajuda de diversas pessoas e/ou instituições às quais presto minha homenagem. Certamente esses parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre estas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

Em primeiro lugar, gostaria de expressar minha profunda gratidão a Deus. A Ele dedico cada conquista e agradeço por iluminar meu caminho durante essa jornada desafiadora.

Agradeço ao professor orientador, Prof. Dr. Heitor Silvério Lopes por sua inestimável ajuda e dedicação. Sua paciência em me guiar foi crucial para o êxito deste trabalho. Ao coorientador, Guilherme de Moraes Restani, agradeço pelas conversas e pelos importantes auxilios técnicos.

À minha esposa, agradeço por estar ao meu lado a cada passo, encorajando-me nos momentos difíceis e celebrando comigo nas vitórias. Sua paciência e amor foram fundamentais para que eu pudesse superar os desafios.

À minha mãe e ao meu pai, agradeço pelo apoio ao longo de toda a minha trajetória acadêmica. Sua sabedoria e incentivo foram fundamentais para meu crescimento pessoal e profissional.

Minha família, a base da minha vida, agradeço pelo amor, pela compreensão e pelo suporte incondicional.

À empresa 4VANTS[1], pelos dados, recursos de *hardware* e *software* fornecidos. Além do suporte técnico, financeiro e pela oportunidade de realizar este trabalho.

Por fim, aos amigos e professores da universidade, agradeço por compartilharem conhecimento, desafios e momentos de descontração. Cada interação contribuiu para minha formação integral. Também, a todos que, de alguma forma, fizeram parte desta jornada, o meu mais sincero obrigado.

[1] https://4vants.com.br/

It's going to be interesting to see how society deals with artificial intelligence, but it will definitely be cool. (ANGLE, Colin).

RESUMO

QUEROS, Ian Douglas Almeida. **Sistema de Inspeção de Talas de Junção Ferroviárias via Computação de Borda**. 2023. 66 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia da Computação) – Universidade Tecnológica Federal do Paraná. Curitiba, 2023.

Acidentes ferroviários podem causar grandes prejuízos financeiros e ambientais, além de colocar em risco a vida de pessoas. Um objeto de grande importância para a segurança ferroviária são as talas de junção, que são peças de aço ajustadas e fixadas, aos pares, na junta dos trilhos para assegurar a continuidade da superfície de rolamento da via. Porém, sua inspeção é realizada de forma manual, necessitando de considerável alocação de recursos humanos e financeiros. Este trabalho consiste no desenvolvimento de um sistema para identificação e análise de talas de junção ferroviárias por meio de técnicas de visão computacional e aprendizado profundo. O sistema é composto por uma câmera montada à frente da locomotiva, sistema de GPS, e unidade de processamento, todos embarcados na locomotiva, onde ocorre todo o processamento dos dados capturados. Também, diferentes técnicas de otimização de modelos de aprendizado profundo foram utilizadas para possibilitar a implementação do sistema em dispositivos de baixo custo energético. Por fim, o sistema foi validado com dados reais de uma malha ferroviária e obteve-se uma acurácia acima de 90% na localização geográfica das talas de junção e uma acurácia acima de 90% para a análise de erros nas talas de junção rastreadas.

Palavras-chave: Visão computacional. Sistemas embarcados. Inspeção. Malha Ferroviária. Talas de junção.

ABSTRACT

QUEROS, Ian Douglas Almeida. **Rail Joint Inspection System Using Edge Computing**. 2023. 66 p. Bachelor Thesis (Bachelor's Degree in Computer Engineering) – Paraná's Federal University of Technology. Curitiba, 2023.

Railway accidents can cause significant financial and environmental damage, in addition to putting people's lives at risk. A crucial component for railway safety is the rail joint, which are steel pieces adjusted and fixed in pairs at the rail tracks to ensure the continuity of the track's rolling surface. However, their inspection is currently carried out manually, requiring considerable human and financial resources. This project involves the development of a system for the identification and analysis of rail joints using computer vision and deep learning techniques. The system consists of a camera mounted at the front of the locomotive, a GPS system, and a processing unit, all embedded on the locomotive, where all captured data is processed. Additionally, various optimization techniques for deep learning models were employed to enable the implementation of the system on low-energy-cost devices. Ultimately, the system was validated with real data from a railway network, achieving an accuracy above 90% in the geographical localization of the rail joints and an accuracy above 90% for the analysis of errors in the tracked rail joints.

Keywords: Computer Vision. Embedded Systems. Inspection. Railway Network. Rail Joint.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Atualmente a malha ferroviária do Brasil possui uma extensão de cerca de 30.000 Km e é responsável por 20,7% de todo o transporte de cargas realizadas pelo país (CNT, 2019). O DNIT (Departamento Nacional de Infraestrutura de Transporte) realiza inspeções mensais desta malha para avaliar possíveis irregularidades ao longo da via, o que é essencial para garantir a qualidade da malha e impedir possíveis problemas que possam ocorrer caso ela esteja danificada (REDE GLOBO - Santos, 2022).

No entanto, atualmente as inspeções de monitoramento desta malha são realizadas de forma manual, necessitando de considerável contingente humano, tempo, e recursos financeiros. Além disto, o processo é lento, de baixa confiabilidade, e as empresas que fazem a inspeção não possuem tecnologias para realizá-la em tempo real e de forma periódica, o que seria necessário para prevenir eficazmente os possíveis problemas na malha.

Um aspecto da inspeção é a verificação estrutural das talas de junção, que são peças de aço ajustadas e fixadas, aos pares, no ponto de junção de dois segmentos de trilhos. Isto serve para assegurar a continuidade da superfície de rolamento da via, e é essencial que elas estejam em boas condições (DNIT, 2015). Porém, a tala de junção pode acumular problemas com o tempo de uso, como empenamentos, fissuras, trincas, rebarbas, entre outros. Com isto, se torna necessário sua inspeção e manutenção de forma periódica.

Em particular, um dos possíveis problemas das talas de junção é o afrouxamento ou ausência dos parafusos que mantêm a tala fixada ao trilho. Isto pode causar acidentes graves, como o descarrilamento de um trem, caso não seja detectado e corrigido a tempo (REDE GLOBO, 2018).

Porém, é escassa a literatura sobre o tema de inspeção de talas de junção de forma automatizada. Os trabalhos relacionados que fazem o uso de técnicas de visão computacional para este tipo de inspeção requerem uma configuração de hardware muito especializada, ou não lidam especificamente com talas de junção. Além disto, tais trabalhos não lidam com os aspectos práticos da inspeção com algoritmos de visão computacional, como o hardware que será utilizado, otimização dos modelos de aprendizado de máquina para implementação em dispositivos embarcados para inferência, e redução dos dados para transmissão, entre outros.

Considerando o exposto, este trabalho propõe um sistema de captura de dados e imagens, e identificação e análise das talas de junção na malha ferroviária. O sistema será composto por

uma câmera montada à frente da locomotiva, o sistema de GPS, e a unidade de processamento, todos embarcados. Após a captura dos dados, eles serão processados para agregar e resumir em informações úteis como a localização e estado das talas no trajeto percorrido, possibilitando sua transferência pela internet. O estado de cada da tala será representado por um valor inteiro chamado criticidade, no qual um valor maior indica uma maior necessidade de manutenção da tala. A criticidade será definida como uma função relacionada à quantidade de parafusos que a tala deveria possuir em relação à quantidade de parafusos faltantes.

Para validar o trabalho, captura de dados previamente realizadas serão utilizadas e um ambiente semelhante a situação real será simulado. Como esta captura de dados e imagens geram arquivos que necessitam de considerável armazenamento no sistema, é inviável enviá-los para um servidor externo para processamento. Logo, há demanda de grande processamento local destes dados e imagens. Assim, após o processamento local apenas uma quantidade reduzida de informações serão produzidas, possibilitando a fácil transferência das mesmas. Para isso, será utilizado um sistema de computação em borda, que é um sistema que realiza o processamento de dados e imagens próximo ao local de captura, reduzindo a necessidade de transferência de dados e imagens para um servidor externo.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Desenvolver um sistema de identificação e análise de talas de junção em uma malha ferroviária durante o trajeto de uma locomotiva, possibilitando a inspeção automatizada e frequente da via. Assim, o estado das talas pode ser reavaliado com mais eficiência, confiabilidade e com baixo custo financeiro e de recursos humanos.

1.1.2 Objetivos específicos

- Definir os requisitos mínimos para a validação do projeto.
- Considerando a quantidade e qualidade de informações a ser geradas, analisar quais recursos físicos e sensores serão adequados para alcançar uma solução tecnicamente viável.

- Desenvolver um sistema para captura de imagens e informações de GPS em tempo-real como um sistema de computação em borda.
- Desenvolver um sistema de identificação e localização das talas de junção a partir das imagens e informações de GPS.
- Desenvolver um sistema de análise das talas de junção a partir das talas localizadas.
- Testar e gerar métricas para validação dos sistemas desenvolvidos. Analisar os resultados obtidos em relação à situação real da malha ferroviária filmada e em relação às especificações previamente definidas.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em 5 capítulos. O Capítulo 2 apresenta a revisão da literatura, onde são apresentados os principais conceitos teóricos e tecnologias relacionados a este trabalho. O Capítulo 3 apresenta os materiais e a metodologia, como os métodos e técnicas utilizados para o desenvolvimento do trabalho. O Capítulo 4 apresenta os resultados obtidos com os diferentes módulos e objetos criados. Por fim, o Capítulo 5 apresenta as conclusões obtidas com o desenvolvimento do trabalho e possíveis trabalhos futuros.

2 REVISÃO DA LITERATURA

Neste capítulo serão apresentados os principais conceitos teóricos e tecnologias relacionadas a este trabalho.

2.1 SISTEMAS DE COMPUTAÇÃO EM BORDA

Computação de borda se refere ao processamento, análise e armazenamento de dados mais próximos de onde eles são gerados para permitir análises e respostas rápidas, quase em tempo real. Estima-se que até 2025, 75% dos dados serão criados fora dos data centers principais, onde a maior parte do processamento ocorre hoje (Gartner, 2018). Aproximadamente 90% de todos os dados coletados hoje pelas empresas nunca serão utilizados (Forbes, 2019). A computação de borda abre a possibilidade de obter os benefícios dos dados coletados dos dispositivos através de processamento de alto desempenho, conectividade de baixa latência e plataformas seguras.

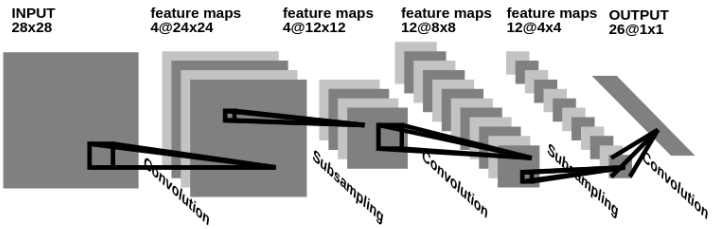
Os principais fatores que contribuem para os desafios na nuvem incluem: latência, largura de banda, segurança, privacidade, conectividade e aplicações de inteligência artificial. Computação de borda aproxima o armazenamento de dados do local do dispositivo, e algoritmos de Inteligência Artificial (IA) processam os dados criados no dispositivo com ou sem conexão com a internet. Isto permite que os dados sejam processados em questão de milissegundos, proporcionando feedback em tempo real. A IA de borda possibilita que as respostas sejam entregues quase instantaneamente. Isto pode ser mais seguro quando significa que alguns dados sensíveis nunca saem efetivamente da borda (Red Hat, 2023).

2.2 REDES NEURAIS CONVOLUCIONAIS

Redes Neurais Convolucionais (CNN do inglês *Convolutional Neural Network*) são uma classe de redes neurais artificiais do tipo *feed-forward* que utilizam camadas de convoluções para extrair características de uma imagem. As características são extraídas por meio de filtros otimizados durante o processo de treinamento (LECUN *et al.*, 1998). Uma rede convolucional típica é mostrada na Figura 1.

Conceituadas por LeCun *et al.* (1998), as CNNs foram inspiradas pela sensibilidade

Figura 1 – Rede Neural Convolucional para processamento de imagem



Fonte: (LECUN *et al.*, 1998)

local e orientação seletiva do cérebro . Depois, Krizhevsky *et al.* (2012) obtiveram um grande destaque na área de visão computacional quando utilizaram uma CNN com 60 milhões de parâmetros para vencer o desafio de classificação de imagens do ImageNet [1], uma competição anual de visão computacional, tendo alcançado uma taxa de erro top-1 e top-5 de 37,5% e 17,0%, respectivamente, o que era consideravelmente melhor do que o estado da arte na época. Desde então, as CNNs têm sido utilizadas em diversas aplicações de visão computacional, como detecção e classificação de objetos, segmentação de imagens, entre outras.

Recentemente, Redmon *et al.* (2015) publicaram o algoritmo YOLO (do inglês *You Only Look Once*), uma nova abordagem para localização e classificação de objetos em imagens, abordando como um problema de regressão para caixas delimitadoras (*bounding-boxes*) espacialmente separadas e a associação destas caixas a uma classe de objeto com certa probabilidade. Uma única CNN infere caixas delimitadoras e probabilidades de classe diretamente de imagens em uma única avaliação. Isto permite a otimização da rede de ponta a ponta, tornando esse tipo de arquitetura extremamente rápida.

Os seguintes conceitos são essenciais para o entendimento de CNNs: camada de convolução, camada de *Pooling*, camada totalmente conectada (*fully connected layer*), função de ativação e função de perda.

2.2.1 Camada de Convolução

A camada de convolução é a principal camada de uma CNN. É responsável por extrair características de uma imagem. É composta por filtros que percorrem a imagem e extraem características de acordo com os pesos e vieses de cada filtro. De uma forma geral, os filtros são responsáveis por detectar características, das mais simples até as mais abstratas, conforme se vai

[1] https://www.image-net.org

aprofundando na rede (GOODFELLOW *et al.*, 2016). A Equação 1 representa esta operação:

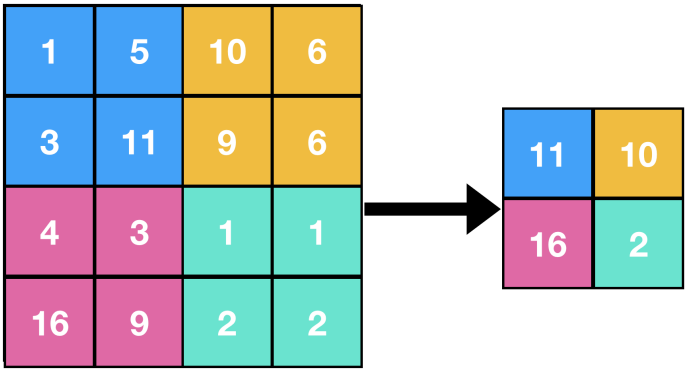
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n) \tag{1}$$

onde I é a imagem de entrada, K é o filtro, S é a saída e $*$ é o operador de convolução.

2.2.2 Camada de *Pooling*

É responsável por reduzir a dimensionalidade da saída da camada de convolução, reduzindo o número de parâmetros e operações da rede, além de tornar a rede mais robusta a pequenas variações na posição dos objetos na imagem. A operação mais comum de *pooling* é o *max-pooling*, que retorna o maior valor de uma região da matriz de entrada (SHARMA, 2023). A Figura 2 ilustra o resultado de uma operação de *max-pooling*.

Figura 2 – Resultado de uma operação de *max-pooling*

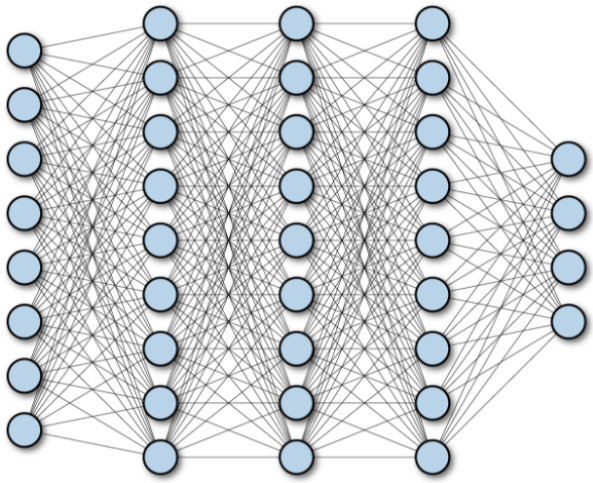


Fonte: (SHARMA, 2023)

2.2.3 Camada Totalmente Conectada

É responsável por realizar a classificação dos valores de entrada nas possíveis classes da rede. Como seu nome indica, cada neurônio desta camada da rede está conectado a todos os neurônios da camada anterior. As saidas das camadas anteriores são transformados em um vetor unidimensional e os valores deste vetor são associados aos neuronios de entrada um a um. Após o cálculo das operações das camadas ocultas da camada totalmente conectada, o resultado é uma classificação entre as possíveis classes da rede. A Figura 3 ilustra camadas totalmente conectadas (as três camadas internas) e a camada de saida.

Figura 3 – Camada totalmente conectada



Fonte: (RAMSUNDAR; ZADEH, 2018)

2.2.4 Função de Ativação

Funções de ativação são utilizadas para introduzir não-linearidades em uma rede neural. A ideia é que as funções de ativação auxiliem a rede em aprender os padrões contidos na imagem, com ativação de neurônios específicos para cada padrão. A escolha da função de ativação modifica tanto o treinamento quanto a inferência da rede. Existem diversas funções de ativação, tais como:

- **ReLU** (*rectified linear unit*): zera os valores negativos da entrada, representada pela função Equação 2. Possui o problema de ocultar parte do resultado.
- **Leaky ReLU**: trata o problema da ReLU diminuindo a inclinação quando os valores de entrada são negativos. Representada pela Equação 3.
- **Sigmoide**: possui uma curva característica de S, normaliza os valores de entrada no intervalo de (0, 1). Representada pela Equação 4. Possui o problema de possívelmente saturar os valores de entrada, o que pode prevenir a convergência para a melhor solução.

$$f(x) = max(0, x) \tag{2}$$

$$f(x, \alpha) = \alpha min(0, x) + max(0, x) \tag{3}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

2.2.5 Função de Custo

Funções de custo são necessárias para calcular o erro das previsões realizadas pela rede neural em relação aos valores esperados. Elas permitem quantificar a qualidade da resposta da rede, permitindo que os pesos e vieses sejam ajustados com o objetivo de diminuir o erro e aumentar a qualidade das previsões calculadas. As funções de custo podem ser divididas em dois grupos, dependendo do tipo de problema a ser resolvido:

- **Problemas de Regressão**: problemas onde o objetivo é prever um valor numérico em um intervalo contínuo.
- **Problemas de Classificação**: problemas onde o objetivo é prever um valor em um conjunto discreto de valores possíveis.

Alguns exemplos de funções de custo para problemas de regressão são:

- **Erro Quadrático Médio** (*Mean Squared Error*): calcula a média do quadrado das diferenças entre os valores previstos $\hat{y}_i$ e os valores esperados y_i para cada padrão de treinamento i. Representada pela Equação 5.
- **Erro Absoluto Médio** (*Mean Absolute Error*): calcula a média do valor absoluto das diferenças entre os valores previstos $\hat{y}_i$ e os valores esperados y_i para cada padrão de treinamento i. Representada pela Equação 6.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{6}$$

Um exemplo de função de custo para problemas de classificação é a Entropia Cruzada (*Cross Entropy*), que indica o quão diferente é a distribuição de probabilidade prevista dos valores de classificação esperados. Representada pela Equação 7.

$$CE = -\frac{1}{n}\sum_{i=1}^{n}(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)) \tag{7}$$

2.2.6 Otimizadores

Uma rede neural precisa que as variáveis de cada camada sejam alteradas de forma que esta funcione melhor no processo de inferência (TAQI *et al.*, 2018). Para isso, os otimizadores são utilizados para minimizar a função de custo, ajustando os pesos e vieses da rede (GOODFELLOW *et al.*, 2016).

O algoritmo de Gradiente Descendente (*Gradient Descent*) é um dos mais populares algoritmos para realizar otimização e também é o mais comumente utilizado. Ele consiste em calcular o gradiente da função de custo em relação aos pesos e vieses da rede, e então atualizar os pesos e vieses de acordo com o gradiente calculado. Para isso, ele minimiza a função $J(\theta)$, onde θ representa os parâmetros da rede, atualizando os parâmetros na direção oposta ao gradiente da função $\nabla_\theta J(\theta)$. A taxa de aprendizado α determina o tamanho do passo dado na direção do mínimo local. (RUDER, 2016). A forma mais simples de Gradiente Descendente é representada pela Equação 8, também chamada de *Batch Gradient Descent*.

$$\theta = \theta - \alpha \nabla_\theta J(\theta) \tag{8}$$

Existem múltiplas variações do Gradiente Descendente, que diferem na forma como os parâmetros são atualizados. Cada uma com suas vantagens e desvantagens.

2.3 PROCESSAMENTOS EM *BATCH*

O processamento em *batch* é uma técnica utilizada para processar um conjunto de dados de uma vez só, ao invés de processar um dado por vez. É uma técnica muito utilizada no treinamento de redes neurais, ela permite que o treinamento seja realizado de forma mais eficiente, pois GPUs são mais eficientes quando o processamento é paralelizável, como é o caso do processamento em *batch*.

2.4 *TRANSFER LEARNING*

Transfer Learning (ou Aprendizado por Transferência) é uma técnica de aprendizado de máquina que consiste em utilizar um modelo de aprendizado de máquina pré-treinado em uma determinada tarefa, em uma segunda tarefa diferente, mas relacionada à primeira. O princípio utilizado é que o conhecimento adquirido durante o treinamento do modelo na primeira tarefa

também possa ser utilizado para melhorar o desempenho do modelo na segunda tarefa.

Transfer Learning permite um progresso muito rápido em problemas de aprendizado de máquina, pois permite que modelos pré-treinados sejam utilizados como ponto de partida para novos problemas, evitando a necessidade de treinar um modelo do zero. Além disto, *Transfer Learning* também permite que modelos pré-treinados sejam utilizados mesmo quando não se tem acesso a uma grande quantidades de dados para treinamento, pois o modelo-base já foi previamente treinado (BROWNLEE, 2019).

2.5 *DATA AUGMENTATION*

O *Data Augmentation* é um conjunto de técnicas cujo objetivo é melhorar a qualidade dos dados, dessa forma, melhorando o desempenho dos modelos de aprendizado. De modo geral, ele processa imagens e aumenta a quantidade de dados de treinamento disponíveis para um modelo, através da criação de novas amostras a partir das já existentes.

Pode ser aplicada de várias maneiras, incluindo o aumento da escala, rotação, corte, espelhamento, mudança de contraste, e outras transformações nas imagens originais. O objetivo é melhorar a capacidade do modelo em reconhecer e generalizar padrões, aumentando a variedade e a diversidade dos dados de treinamento (DISTRITO, 2023).

2.6 *MEAN AVERAGE PRECISION*

Mean Average Precision (mAP) é uma métrica comumente utilizada para avaliar o desempenho de algoritmos de detecção de objetos, como Fast R-CNN, Mask R-CNN e YOLO. Ela representa, de forma geral, a precisão do modelo para a detecção de objetos, de maneira que a localização do objeto inferida pelo modelo esteja próxima da localização especificada no *dataset* utilizado como base para o teste (SHAH, 2022). Para calcular esta métrica, alguns conceitos são essenciais.

2.6.1 Matriz de Confusão

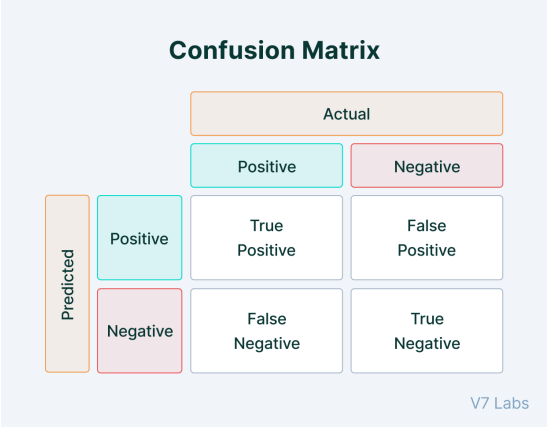
Quatro atributos são necessários para o cálculo da matriz de confusão:

- **Verdadeiro Positivo (TP – *True Positive*)**: Localização e classe do objeto foram corretamente calculados.

- **Falso Positivo (FP – *False Positive*)**: Objeto localizado não existe.
- **Verdadeiro Negativo (TN – *True Negative*)**: Modelo indicou corretamente que o objeto não existe.
- **Falso Negativo (FN – *False Negative*)**: Objeto existente no *dataset* não foi localizado.

A Figura 4 ilustra os campos de uma matriz de confusão.

Figura 4 – Campos de uma matriz de confusão

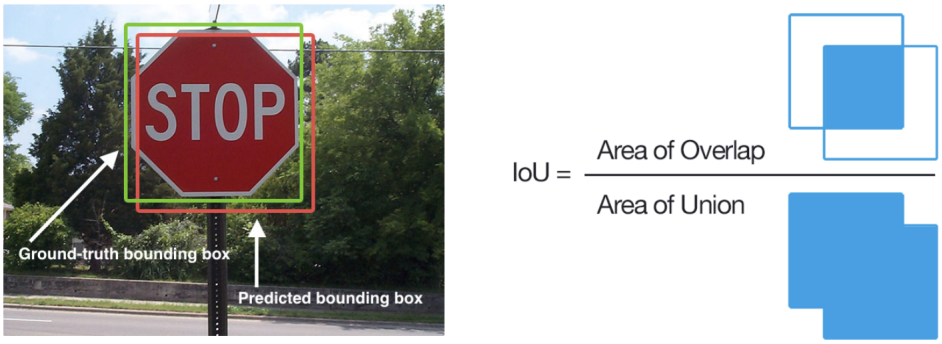


Fonte: Retirado de (SHAH, 2022)

2.6.2 *Intersection Over Union* (IoU)

Considerando os dois retângulos, o IoU é a razão entre a área da intersecção entre os dois retângulos e a área da união entre os dois retângulos. Representa a similaridade de posições de dois objetos geométricos semelhantes. A Figura 5 ilustra o cálculo do IoU.

Figura 5 – Cálculo do IoU



Fonte: Retirado de (SHAH, 2022)

No contexto de detecção de objetos em imagens, o IoU é utilizado para determinar o quanto a localização de um objeto inferida pelo modelo está próxima da localização especificada no *dataset* utilizado como base para o teste.

2.6.3 Curva *Precision-Recall*

A *Precision* é a razão entre o número de verdadeiros positivos e o número de total de previsões realizadas. Representa a proporção de resultados positivos que foram corretamente identificados. Sua formulação é dada pela Equação 9.

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

O *Recall* é a razão entre o número de verdadeiros positivos e o número de objetos existentes no *dataset* utilizado como base para o teste. Representa a proporção de objetos do *dataset* que foram corretamente identificados. Sua formulação é dada pela Equação 10.

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

Para ambas as métricas, é utilizado o valor de IoU para determinar se um objeto foi corretamente localizado. Por exemplo, pode-se definir que se IoU for maior que 0.5, o objeto foi corretamente localizado. Portanto, *Precision* e *Recall* dependem do valor do IoU utilizado.

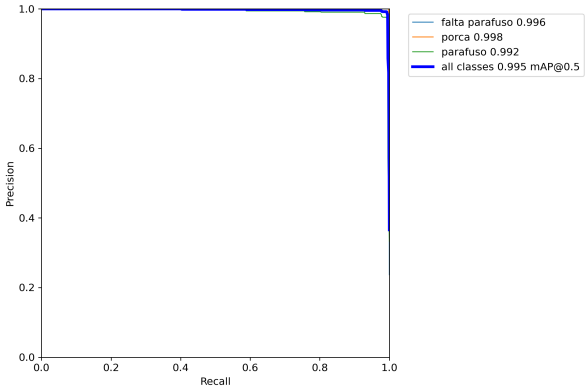
Também, o limiar da confiança do modelo afeta os valores de *Precision* e *Recall*. Se um modelo possui um limiar de confiança baixo, a tendência é que o *Recall* seja alto e a *Precision* seja baixa, pois muitas previsões incorretas serão consideradas. Por outro lado, se um modelo possui um limiar de confiança alto, a têndencia é que o *Recall* diminua e a *Precision* aumente, pois as previsões serão mais corretas, mas a quantidade de objetos corretamente previstos será menor.

Portanto, a curva *Precision-Recall* é relação entre a *Precision* e o *Recall* para diferentes valores de limiar de confiança. A Figura 6 ilustra uma curva *Precision-Recall*.

2.6.4 *Average Precision* (AP)

Average Precision (AP) é a área sob a curva *Precision-Recall*. Representa a precisão média de uma rede considerando os diferentes possíveis valores de *Recall*. Por fim, para calcular o mAP, calcula-se a média dos APs para cada classe de objeto. A Equação 11 representa o

Figura 6 – Exemplo de curva *Precision-Recall*



Fonte: Autoria Própria

cálculo do mAP. É comum também que o mAP seja calculado para uma faixa de valores de IoU, por exemplo, de 50% a 95% com passo de 5%, realizando a média dos valores obtidos, avaliando a rede em sua capacidade de localizar objetos exatamente onde estão.

$$mAP = \frac{1}{n} \sum_{i=1}^{n} AP_i \tag{11}$$

2.7 *NON MAXIMUM SUPPRESSION* (NMS)

É uma técnica utilizada para reduzir a quantidade de caixas delimitadoras geradas por um algoritmo de detecção de objetos. A ideia é que, para cada objeto detectado, apenas a caixa delimitadora com maior confiança seja mantida. Para isto, o seguinte algoritmo é utilizado.

Dada a lista de previsões de caixas delimitadoras B e uma lista vazia de caixas delimitadoras selecionadas S, faça:

1. Selecione a caixa delimitadora b com maior confiança de B e adicione-a a S.
2. Calcule o IoU entre b e todas as caixas delimitadoras restantes em B.
3. Remova todas as caixas delimitadoras de B que possuem IoU maior que um limiar definido.
4. Repita os passos 1, 2 e 3 até que B esteja vazio.

Desta forma, apenas as caixas delimitadoras com maior confiança são mantidas. Este processo é importante para reduzir a um valor aceitável a quantidade de caixas delimitadoras geradas, pois é comum que redes gerem múltiplas caixas delimitadoras para um mesmo objeto.

3 MATERIAIS E MÉTODOS

A seguir serão descritos os materiais de hardware e software utilizados para o desenvolvimento do projeto, bem como os métodos implementados.

3.1 MATERIAIS

3.1.1 CVAT

CVAT[1] é uma ferramenta interativa de anotação de vídeo e imagens, a versão online para visão computacional é gratuita. Atualmente o CVAT é usado por muitas empresas de IA e instituições acadêmicas para anotar rapidamente grandes conjuntos de dados. Neste trabalho o CVAT foi utilizado para anotar as imagens do *dataset* de treinamento e validação, validar as anotações realizadas, e exportar os dados anotados no formato YOLO, que é o formato utilizado para o treinamento do modelo de detecção.

A escolha desta ferramenta para a anotação das imagens foi feita para garantir a criação de um *dataset* de alta qualidade, no qual as anotações fossem feitas de forma rápida e eficiente, e que permitisse a exportação do *dataset* no formato adequado.

3.1.2 Pytorch

PyTorch [2] é um *framework* de aprendizado de máquina de código aberto originalmente desenvolvido pela Meta AI e agora parte da abrangência da Linux Foundation (CHINTALA, 2022).

O Pytorch é baseado na biblioteca Torch e é utilizado em aplicações como visão computacional. O Torch é uma biblioteca de tensores para manipulação de matrizes multidimensionais de dados utilizada em aprendizado de máquina e em muitas outras aplicações que utilizam cálculo de tensores. Ele oferece bibliotecas para manipulação básica de tensores em CPUs ou GPUs, uma biblioteca de redes neurais integrada, utilitários para treinamento de modelos e uma biblioteca de multiprocessamento que pode trabalhar com memória compartilhada (YEGULALP, 2023).

[1] https://www.cvat.ai/
[2] https://pytorch.org/

Este é o framework utilizado para a implementação da rede YOLOv7 e para a execução da inferência nos modelos criados pelos processos de treinamento no projeto.

3.1.3 Python

Python[3] é uma linguagem de programação de alto nível, de compreensão fácil e rápida, *open-source*, orientada a objeto, com tipagem dinâmica e com uma comunidade extremamente ativa.

Esta linguagem foi utilizada para a implementação deste projeto. A escolha desta linguágem foi feita principalmente por possuir uma vasta quantidade de bibliotecas de visão computacional e aprendizado de máquina, sendo considerada o padrão em aplicações nestas áreas, facilitando o desenvolvimento de soluções. Além disto, sua popularidade e facilidade de uso ajudam nos diferentes fatores relacionados à criação de código, tais como: documentação, comunidade, ferramentas de desenvolvimento, etc.

3.1.4 YOLOv7

YOLO[4] é um modelo de detecção de objetos de único estágio, sendo o estado da arte no quesito de detecção em tempo real, tornando-o ideal para implementação em sistemas embarcados.

A família YOLO de modelos consiste em uma única *Convolutional Neural Network* (CNN), na qual a camadas de convolução iniciais extraem características da imagem e as camadas completamente conectadas (do ingles, *Fully Connected Layers*) geram as predições sobre as localizações e probabilidades dos objetos na imagem (REDMON *et al.*, 2015).

O YOLOv7 é a versão oficial mais recente do YOLO, nela, os autores priorizam otimizaçoes no processo de treinamento, com o objetivo de melhorar a acurácia da detecção de objetos, sem aumentar o tempo de inferência. Para isto, eles desenvolveram um conjunto de módulos e algoritmos chamados *trainable bag-of-freebies* (WANG *et al.*, 2022).

O YOLOv7, junto com o TensorRT apresentado na subseção 3.1.5 a seguir, foram as principais ferramentas que viabilizaram a implementação da solução e a realização dos objetivos propostos. Devido à sua performance e acurácia, o YOLOv7 permitiu que os modelos criados

[3] https://www.python.org/
[4] https://pjreddie.com/darknet/yolo/

fossem executados em tempo real, por terem uma baixa quantidade de parâmetros, sem a perda de acurácia, e menor tempo de inferência do que as versões anteriores do YOLO.

3.1.5 TensorRT

Publicado pela NVIDIA, o TensorRT[5] é um *Software Development Kit* (SDK) para alto desempenho para redes de aprendizado profundo. Ele inclui um otimizador de inferência e um *runtime* que oferece baixa latência e alto rendimento para a inferência de redes neurais.

É importante lembrar que as redes neurais são formadas pelos seguintes componentes: entrada, que são os dados alimentados na rede; arquitetura, o grafo que representa a rede neural; pesos, que são os parâmetros da rede; e *kernels*, pedaços de código que implementam operações específicas das camadas, tais como a convolução, ReLU, etc. O TensorRT otimiza a rede neural funcionando em duas parte cruciais da inferência: a compilação, processo em que é decidido quais camadas e *kernels* serão executados, e o *runtime*, processo de execução do grafo da forma mais eficiente possível (DECI, 2023).

Como mencionado na Seção 3.1.4, o TensorRT foi uma das principais ferramentas que viabilizaram a implementação da solução e a realização dos objetivos propostos. Ao permitir a otimização do modelo na GPU específica utilizada no projeto, esta ferramenta foi essencial para garantir o desempenho de inferência em tempo real, sem perdas de acurácia.

3.1.6 OpenCV

OpenCV[6] (*Open Source Computer Vision Library*) é uma biblioteca de software de código aberto para visão computacional e aprendizado de máquina. O OpenCV foi criado para fornecer uma infraestrutura comum para aplicações de visão computacional e para acelerar o uso da processamento de imagens em produtos comerciais. Sendo um produto licenciado pela licença Apache 2, o OpenCV facilita a utilização e modificação de seu código (ITSEEZ, 2015).

O OpenCV foi utilizado para a manipulação de imagens e vídeos, tais como separação/junção de quadros de um vídeo, a leitura de imagens e vídeos, etc. Também auxiliou nas operações de redimensionamento de imagens e conversão entre diferentes formatos.

[5] https://github.com/NVIDIA/TensorRT
[6] https://opencv.org/

3.1.7 Jetson Nano

A Jetson[7] é uma placa de desenvolvimento de hardware da NVIDIA que possui uma GPU integrada. Ela é utilizada para o desenvolvimento de aplicações de inteligência artificial e visão computacional. A Jetson foi utilizada neste projeto porque possui uma GPU integrada, que é utilizada para a execução da inferência dos modelos criados. Desta forma permite um desempenho em tempo real que não seria possível com um hardware sem GPU. Além disto, a Jetson possui um baixo consumo de energia, o que permite que seja alimentada de forma portátil.

3.1.8 QGIS

O QGIS[8] é um Sistema de Informações Geográficas (SIG) de código aberto e amigável ao usuário. O QGIS é um projeto oficial da Fundação Geoespacial de Código Aberto (*Open Source Geospatial Foundation* - OSGeo) e é licenciado sob a Licença Pública Geral GNU. Este software é amplamente utilizado para análise, visualização e manipulação de dados geoespaciais (QGIS Association, 2023).

Neste projeto o QGIS foi utilizado para a visualização e manipulação dos diferentes arquivos geográficos utilizados e gerados no projeto, como arquivos GeoJSON, arquivos de mapa, etc.

3.2 MÉTODOS

Para resolver o problema proposto no Capítulo 1 e desenvolver um sistema capaz de localizar geograficamente e analisar talas a partir de um fluxo de vídeo, o seguinte caminho foi percorrido:

1. Análise do *dataset* disponível.
2. Transformação do *dataset* para o formato exigido pela rede YOLO.
3. Treinamento das redes.
4. Desenvolvimento do código para implementar a solução.

[7] https://www.nvidiastore.com.br/jetson
[8] https://qgis.org/pt_BR/site/

3.2.1 Solução Proposta

Atualmente, a literatura de visão computacional é extensa e há uma grande quantidade de tecnologias que possibilitam o desenvolvimento de soluções. Para definir quais seriam utilizadas, partiu-se da identificação clara dos problemas que deveriam ser resolvidos, que são:

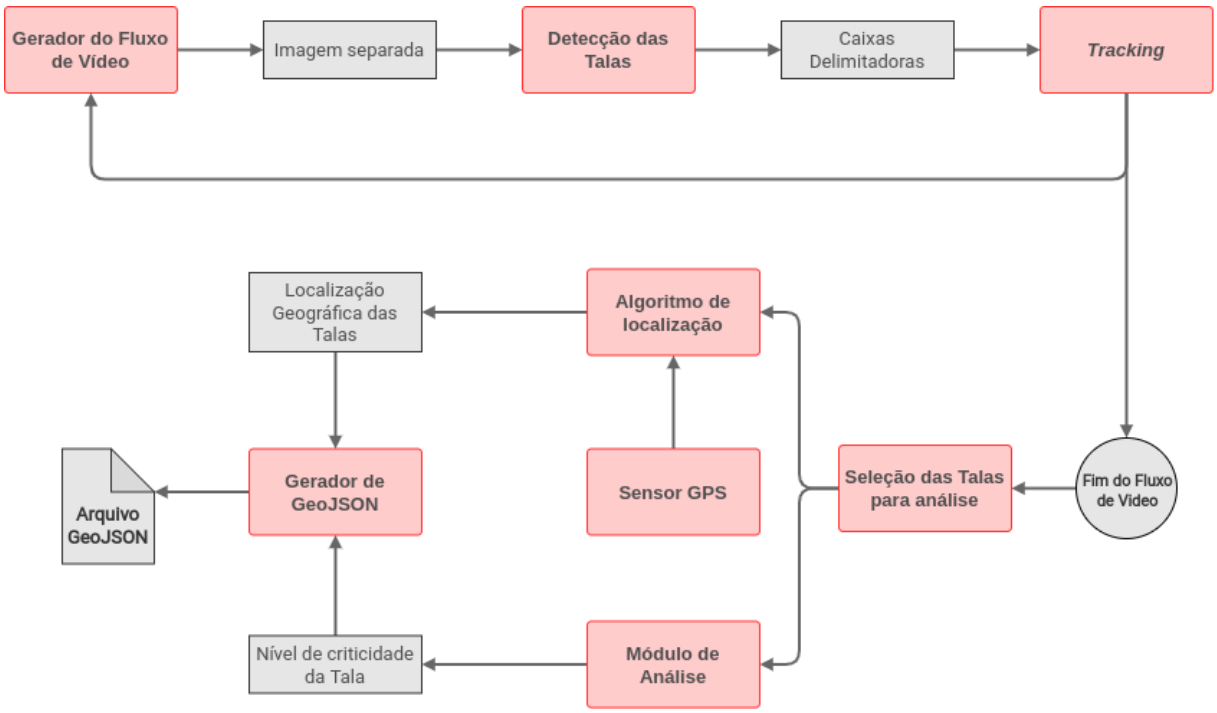
- Detecção das talas nos *frames* do *streaming* do vídeo.
- Transformação do *dataset* para o formato exigido pela rede YOLO.
- Localização geográfica da tala.
- Análise das talas para descobrir seu nível de criticidade.
- Geração de um relatório com as informações encontradas.
- A solução deve ser portável e capaz de ser executada por completo embarcado em uma locomotiva, sem a necessidade de elementos externos.

Com base nestas especificicações, optou-se por utilizar um algoritmo de visão computacional focado em detecção de objetos em imagens com foco em *performance*. Desta forma, foi utilizada uma CNN para a detecção dos objetos, e a arquitetura do modelo escolhida foi a YOLOv7, devido à sua acurácia, desempenho e facilidade de treinamento.

A biblioteca OpenCV foi utilizada para simular um fluxo de vídeo, incluindo a leitura do vídeo e a separação de seus *frames*. Com as imagens dos *frames*, a rede neural é utilizada para a detecção das talas. Tais detecções, representadas por caixas delimitadoras, são adicionadas no módulo de *tracking*, que é responsável por associar as caixas delimitadoras de cada *frame* com a caixa delimitadora da tala nos *frames* anteriores, sendo que as informações salvas são: classe, confiança da inferência, posição da caixa na imagem, *timestamp* do vídeo e número do *frame*. Este processo ocorre até a finalização do *streaming* do vídeo.

Após a finalização do fluxo do vídeo, as melhores detecções são encontradas para cada instância de tala rastreada pelo algoritmo de *tracking*. Das melhores detecções, a posição geográfica da tala é encontrada utilizando o *timestamp* do vídeo e a informação do GPS. Por fim, o módulo de análise das talas é executado sobre um corte das imagens que contém as melhores detecções, gerando um arquivo GeoJSON com as informações das talas encontradas e seu nível de criticidade calculado. A Figura 7 mostra um diagrama de blocos da solução, no qual as caixas vermelhas representam os módulos desenvolvidos e as caixas cinza são as informações geradas.

Figura 7 – Diagrama de blocos da solução proposta



Fonte: Autoria Própria

Como o problema é de domínio muito específico, um *dataset* precisou ser criado para viabilizar o treinamento da rede, o processo de criação do *dataset* será explicado a seguir.

3.2.2 *Dataset*

Foram realizadas duas capturas de dados, que foram posteriormente utilizadas para a criar o *dataset* usado neste trabalho.

Uma das capturas foi feita usando uma câmera AXIS F41. No entanto, o *notebook* usado para a gravação não tinha poder suficiente para processar o vídeo corretamente. Isto resultou em vídeos com problemas na fluidez, onde a taxa de *frames* por segundo estava incorreta. Também houve falhas na sequência de *frames*, levando à perda de alguns *frames* e causando uma reprodução irregular do vídeo. Ainda assim, foi possível separar os *frames* deste vídeo e utilizar aqueles que possuiam talas visíveis para a construção do *frames*.

A outra captura foi feita utilizando uma câmera GoPro, resultando em dados de alta qualidade com uma taxa de 240 *frames* por segundo e resolução em 4K. Esta gravação foi a utilizada para o teste das redes criadas, de forma a separar os *datasets* de treinamento e de teste.

A partir da captura da AXIS F41, separou-se o vídeo em quadros. Destes quadros foram retiradas imagens que continham talas, tentando sempre pegar aquelas que não fossem tão

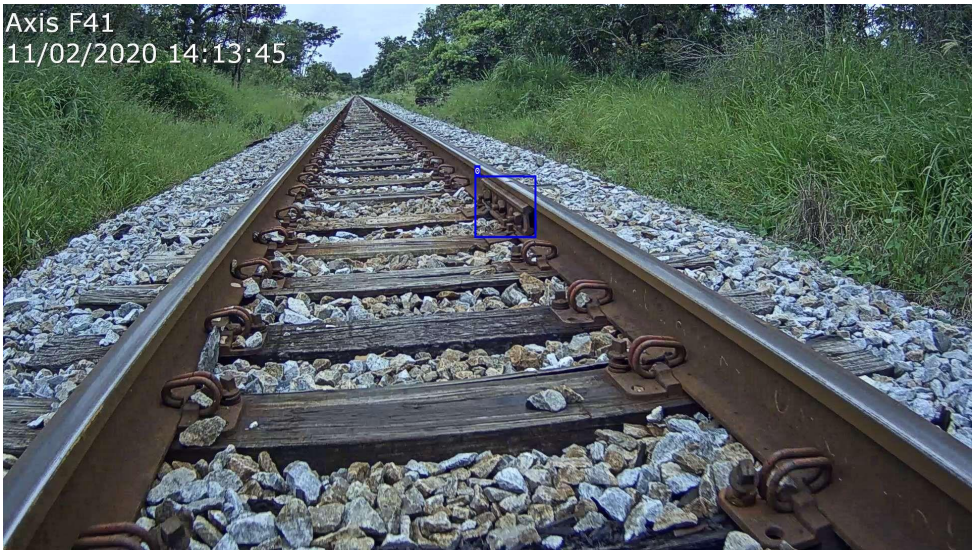
semelhantes entre si, para gerar variabilidade para o treinamento da rede.

O software utilizado para anotar as imagens foi o CVAT, criando uma caixa ao redor dos objetos de interesse, que posteriormente seria utilizado pela rede durante o treinamento para ensiná-la qual a região de interesse que deveria ser buscada para detectar o objeto em questão.

3.2.2.1 *Dataset* para Detecção de Talas

Para a detecção da tala, a partir das imagens obtidas com a câmera AXIS F41, foram selecionadas 776 imagens, contendo 1138 anotações de talas diferentes, com múltiplas anotações da mesma tala em momentos diferentes da captura. Para estas imagens foi criado uma caixa ao redor das talas e a caixa foi classificada como sendo uma tala. Também, foram adicionadas 726 imagens sem nenhuma tala, para ajudar o treinamento da rede com relação ao *background* da imagem, caso não contenha talas, totalizando 1502 imagens. Este *dataset* possui apenas uma classe: tala. Na Figura 8 é mostrado um exemplo de imagem anotada neste *dataset*.

Figura 8 – Exemplo de imagem anotada no *dataset* de detecção de talas



Fonte: Autoria Própria

3.2.2.2 *Dataset* para Detecção de Objetos da Tala

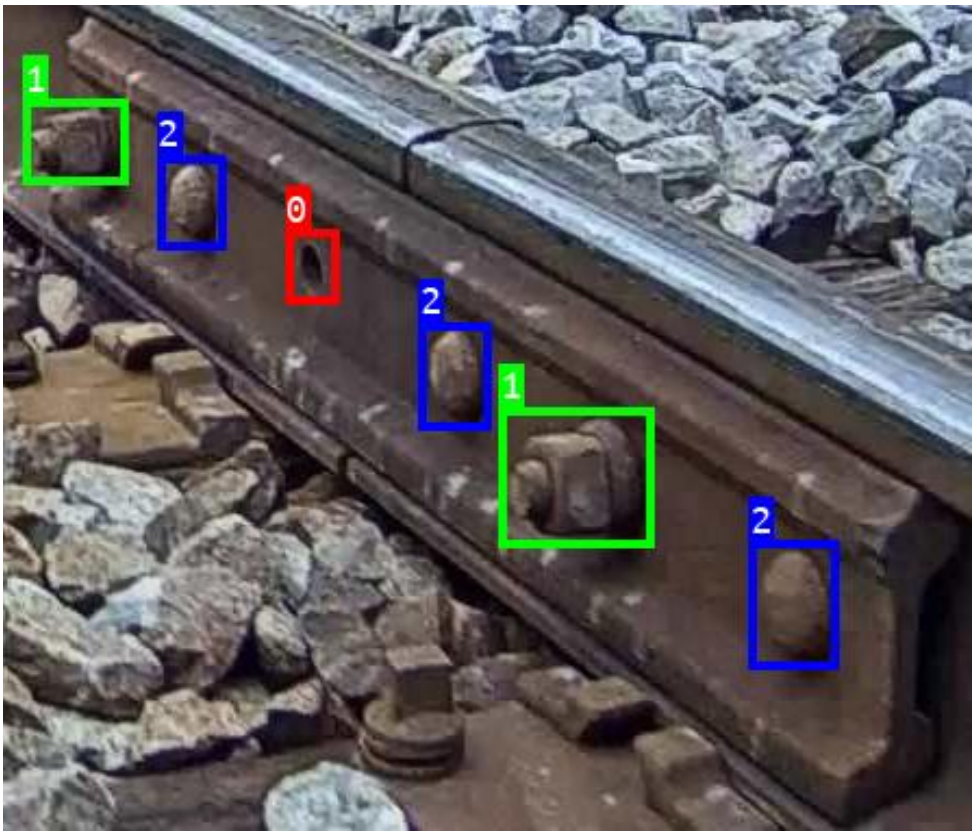
Os objetos de interesse contidos na tala são:

- Parafusos
- Porcas

- Ausência de parafuso ou porca, que, apesar de não ser um objeto, delimita uma região de interesse na tala, que deve ser detectada

A partir das caixas das talas do *dataset* de detecção de talas, um corte foi feito na imagem para separar apenas a região onde a tala era encontrada. Em seguida, esta imagem cortada foi anotada para indicar as posições dos objetos de interesse, de forma que três classes de objeto foram criadas, de acordo com os objetos de interesse mencionados. Este *dataset* possui 1137 imagens com 3286 anotações de porca, 2311 anotações de parafuso e 736 anotações de ausência de porca e parafuso. Na Figura 9 é mostrado um exemplo de imagem anotada neste *dataset*, com um exemplo de região anotada como "falta de parafuso"(vermelho, classe 0), dois exemplos de região anotada como "porca"(em verde, classe 1), e três exemplos de região anotada como "parafuso"(em azul, classe 2).

Figura 9 – Exemplo de imagem anotada no *dataset* de detecção dos objetos da tala



Fonte: Autoria Própria

3.2.3 Treinamento dos Modelos

Após a criação dos *datasets*, foi realizado o treinamento das redes neurais. Esta etapa envolve a separação dos *datasets* em três: o *dataset* de treinamento, utilizado *online* para o ajuste dos parâmetros da rede; o *dataset* de validação, também utilizado *online* durante o treinamento para o teste dos parâmetros calculados e geração das métricas de treinamento; e o *dataset* de teste, utilizado após o treino da rede para testá-la sem os vieses das imagens utilizadas durante o treinamento.

Estas separações devem conter imagens únicas do *dataset* original. Para ambos *datasets*, a separação foi feita com 70% das imagens para treinamento, 20% para validação e 10% para teste. Para o *dataset* de detecção de talas resultou em 1052 imagens para treino, 300 para validação e 150 para teste. Para o *dataset* de detecção de objetos da tala resultou em 797 imagens para treino, 227 para validação e 113 para teste.

Para o treinamento dos modelos é necessário gerar quatro arquivos de configuração, três para especificar a localização das imagens para cada separação do *dataset*, e um arquivo para especificar: as classes que serão treinadas, e a localização dos arquivos de separação do *dataset*, este será o arquivo de configuração do *dataset*.

Além disto, é necessário especificar um arquivo de configuração da arquitetura do modelo que será treinado e um arquivo com a configuração de hiperparâmetros e técnicas de aumento de dados utilizadas, especificadas na subseção 3.2.3.1. Estes arquivos possuem versões com valores padrão do próprio repositório do YOLOv7.

Além dos arquivos de configuração mencionados anteriormente, para realizar o treinamento dos modelos é necessário especificar:

- O dispositivo que será utilizado para o treinamento, como o CPU, uma GPU ou múltiplas GPUs. Isto é feito por meio do ID da GPU.
- O tamanho da imagem que será utilizada como resolução do modelo. Isto afeta a quantidade de memória necessária para o treinamento, tempo de treinamento, acurácia do modelo, tempo de inferência e tamanho do modelo gerado.
- O tamanho do *batch* de imagens que será utilizado durante o treinamento, o que afeta a quantidade de imagens que serão utilizadas para o cálculo do gradiente e, consequentemente, a quantidade de memória necessária para o treinamento. Portanto, este valor

depende da resolução do modelo e da quantidade de memória disponível no dispositivo utilizado.

- Modelo base para realizar a *Transfer Learning*, caso esta técnica seja utilizada.
- Por fim, o nome do modelo gerado.

A principal métrica utilizada para definir qual época foi considerada para o modelo de maior desempenho foi o mAP, uma métrica que a comunidade de visão computacional convencionou para comparar o desempenho de sistemas de detecção de objetos (SOLAWETZ, 2020). Esta métrica também foi utilizada para comparar os diferentes modelos treinados para o projeto e também comparar os modelos com sua versão correspondente após a conversão para o formato TensorRT, de modo a garantir que os modelos convertidos não perderam sua acurácia.

Todos modelos foram treinados na placa de vídeo NVIDIA RTX 3080Ti.

Para testar o desempenho das redes neurais convolucionais em uma plataforma de computação em borda, como a Jetson Nano que foi utilizada, e viabilizar a inferência dos quadros do vídeo de forma que o tempo por quadro fosse menor que a taxa de atualização do próprio vídeo, diferentes modelos de redes neurais foram treinados e testados.

Também, dois diferentes sistemas de inferência foram utilizados, o Pytorch, que é de implementação mais fácil, já que o arquivo do modelo gerado pelo treinamento já está no padrão desse sistema; e o TensorRT, que é otimizado para inferência de modelos de rede neural em hardwares da NVIDIA, porém, a implementação é mais difícil, uma vez que é necessário converter o arquivo do formato Pytorch para uma *engine* do TensorRT e criar um código específico para a inferência na GPU.

Além disto, com os modelos no formato de *engine* do TensorRT, também foi testado o uso uma precisão de parâmetro de rede menor do que a original gerada pelo treinamento, de *float* de 32 bits (FP32) para um *float* de 16 bits (FP16), diminuindo o custo computacional da inferência em troca de uma eventual queda na acurácia da rede.

Para garantir que estas conversões mantivessem a qualidade da rede, os modelos foram testados com o *dataset* de validação, que não foi utilizado no treinamento. As métricas obtidas para comparação do modelo foi o mAP (*Mean Average Precision*) com um IoU de 50% e o mAP com um IOU (*Intersection Over Union*) indo de 50% até 95%, que são as métricas utilizadas também no treinamento.

Por fim, foi realizado o teste de velocidade de processamento, quantificando quantos quadros por segundo (FPS – *Frames Per Second*) cada modelo era capaz de processar. Este valor é utilizado para definir a viabilidade do modelo para processamento em tempo-real, conforme inicialmente proposto.

3.2.3.1 *Data Augmentation*

Por padrão, no modelo YOLOV7, são utilizadas técnicas de aumento de dados durante treinamento. As técnicas e parâmetros para o aumento de dados utilizados neste projeto foram:

- Modificação da faixa de cores HSV em até 1,5%, 70% e 40% respectivamente, abstraindo as possíveis variações de iluminação.
- Translação da imagem em até 20% da sua largura e altura, abstraindo o modelo em relação à posição dos objetos nas imagens, dado que os objetos treinados podem aparecer em qualquer posição da imagem.
- Mudança de escala da imagem em até 50%, abstraindo o modelo em relação ao tamanho do objeto na imagem, dado que os objetos em relação ao vídeo podem possuir tamanhos variados.
- 50% de chance de espelhar horizontalmente a imagem, pois os objetos podem aparecer tanto na esquerda quanto na direita da imagem. Neste caso, vale notar que foi retirado o espelhamento vertical, pois os objetos não podem aparecer de cabeça para baixo no vídeo.
- 100% de chance das imagens serem parte de um mosaico de imagens, para aumentar a quantidade de objetos em uma imagem, dado que os objetos podem aparecer em grupos no vídeo.

Uma outra técnica de aumento de dados que não foi utilizada é a rotação de imagens, mesmo sendo comumente implementada no treinamento de modelos. A razão disso é que os objetos sempre aparecem em uma orientação específica no vídeo. Logo, a rotação de imagens não se faz necessária para o domínio que está sendo considerado.

A Figura 10 mostra um exemplo de um conjunto de imagens utilizado para o treinamento. Neste exemplo, pode-se observar que há imagens com diferentes tamanhos, cores, iluminação e posição dos objetos.

Figura 10 – Exemplo de imagens que o modelo utiliza durante o treinamento

Fonte: Autoria Própria

3.2.3.2 *Transfer Learning*

Para utilizar a técnica de *Transfer Learning*, é necessário especificar um modelo base para o treinamento. Neste projeto, foram utilizados os modelos disponibilizados no próprio repositório do YOLOv7, que são modelos pré-treinados com o *dataset* COCO[9], que contém 80 classes de objetos diferentes. Dois modelos pré-treinados foram utilizados para o *Transfer Learning* no treinamento das redes do projeto: o modelo base YOLOv7 e o modelo YOLOv7 *Tiny*. Cada modelo correspondente deve ser utilizado para o treinamento, então para o treinamento da arquitetura base do YOLOv7, o modelo pré-treinado com a mesma arquitetura base é especificado, e o mesmo ocorre com a arquitetura YOLOv7 *Tiny*.

[9] https://cocodataset.org/

3.2.4 Módulos do Sistema

A seguir, os diferentes módulos necessários para o funcionamento do sistema serão apresentados em suas motivações e métodos de implementação.

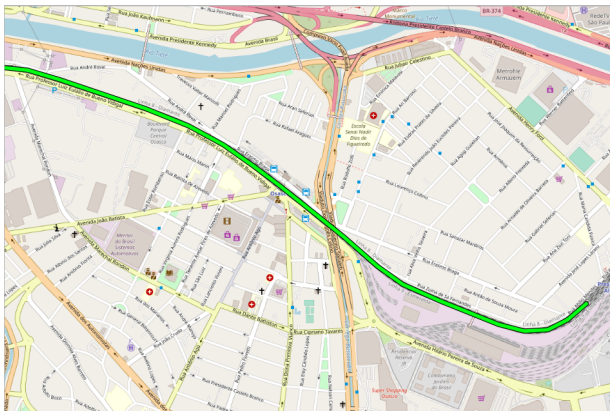
Em sua base, a entrada do sistema consiste de um simulador de *streaming* de vídeo associado a um simulador de sensor de GPS. O processamento do sistema consiste em:

- Separar os quadros do vídeo,
- Realizar a detecção e rastreamento das talas nestes quadros,
- Associar as instâncias das talas rastreadas a um ponto temporal que será calculado como uma posição geográfica utilizando o sensor de GPS,
- Encontrar as melhores detecções de cada tala e executar a inferência para a análise da tala, gerando um GeoJSON com as informações calculadas.

3.2.4.1 Simulação de GPS

Para a localização geográfica da tala, um sistema de GPS é necessário. Como os dados de captura de vídeo não continham as informações de GPS embutidas, foi necessário desenvolver um simulador de geração de sinais de GPS. Para isto, utilizou-se o arquivo GeoJSON base de uma ferrovia e foi feita a simulação de leituras de dados de longitude e latitude para uma determinada velocidade de percurso nesta ferrovia, e a uma dada frequência de leitura. Na Figura 11, é mostrado um exemplo de um arquivo GeoJSON com a linha verde representando uma ferrovia.

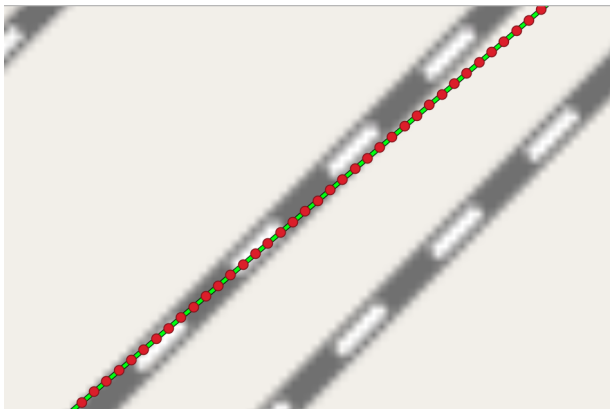
Figura 11 – Trajeto da ferrovia como uma linha no arquivo GeoJSON



Fonte: Autoria Própria

A Figura 12 mostra a simulação das leituras do sensor de GPS, representadas por pontos vermelhos, realizadas nesta linha.

Figura 12 – Leituras do sensor de GPS simulado, representadas por pontos vermelhos na linha do trajeto.



Fonte: Autoria Própria

3.2.4.2 *Tracking*

Para analisar o desempenho da solução de *tracking* implementada, tem-se algumas métricas que podem ser utilizadas. Primeiro, de maneira bem simples e fácil de visualizar, cria-se um vídeo no qual as instâncias únicas de talas são evidenciadas com cores diferentes, ou seja, caso caixas de detecção representem a mesma tala, elas serão desenhadas em um vídeo gerado com a mesma cor. Desta forma, pode-se visualizar se o rastreamento está funcionando como esperado. Também é possível visualizar os erros do algoritmo, por exemplo, quando talas diferentes são consideradas como sendo a mesma.

Uma outra métrica passível de ser utilizada para verificar a qualidade do algoritmo de *tracking* é a quantidade de verdadeiros positivos encontrados, ou seja, quais talas foram corretamente encontradas pelo algoritmo de *tracking*. Para isto, é necessário realizar uma anotação de quais são as diferentes talas do vídeo e comparar manualmente com as instâncias de talas encontradas pelo mesmo.

3.2.4.3 Localização Geográfica da Tala

Para localizar geograficamente a tala são necessários dois passos. O primeiro é rastrear as detecções do modelo de detecção que se referem a uma mesma tala em diferentes quadros do vídeo. Para isto, é necessário realizar um *tracking* sobre as detecções inferidas, calculando-se

associações entre detecções de diferentes quadros do vídeo. O algoritmo de rastreamento utilizado é simples para diminuir o consumo de recursos de hardware e aumentar o desempenho da solução proposta. Também, como em geral a captura é em um ambiente relativamente controlado, o rastreamento não precisa ser sofisticado. O segundo passo é calcular o *timestamp* da provável localização da tala em relação ao vídeo e associar a um *timestamp* do sensor de GPS.

Para cada quadro do vídeo de entrada, é realizada a inferência com o modelo de detecção de talas para localizar na imagem as posições das talas. Após a detecção das talas no quadro atual do vídeo, as detecções são associadas às detecções dos quadros anteriores de acordo com as seguintes regras:

1. O ponto central da caixa de detecção atual deve estar a uma distância máxima pré-definida de uma detecção de um quadro anterior.

2. Entre um quadro e outro, deve ter corrido no máximo uma certa quantidade pré-definida de tempo.

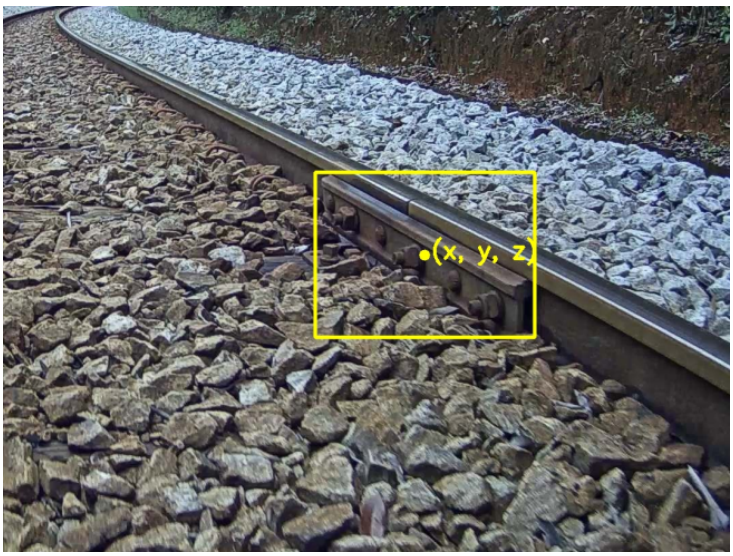
3. Caso mais de uma detecção atual atenda as regras 1 e 2, a detecção associada será a que possuir o ponto central com a menor distância do quadro anterior.

Ao final do processo de rastreamento, uma lista de detecções foi associada a cada instância de tala, caso uma tala possua menos de 3 detecções associadas, ela é desconsiderada ao final do processo. Isto é feito para diminuir os casos em que a rede realiza um falso positivo. Então, as regras descritas delimitam a implementação do algoritmo de rastreamento.

Ao final do rastreamento das talas, tem-se listas com detecções das talas como instâncias únicas ao longo do trajeto da ferrovia. A partir desta lista de detecções, seleciona-se as três detecções de maior pontuação para representar a tala, tanto para o cálculo de sua localização geográfica quanto para análise subsequente. A pontuação de uma detecção é calculada como o produto de sua altura, largura e confiabilidade, resultando, assim, na escolha das detecções de maior dimensão e confiabilidade em média.

Para obter a localização geográfica da tala das três detecções selecionadas, são utilizadas as detecções com maior e menor *timestamp* como base. Depois, a partir destas detecções, são criados dois vetores 3D nos quais cada valor do vetor tem o seguinte significado: x é a posição horizontal do centro da detecção, y é a posição vertical do centro da detecção, e z é o *timestamp* da detecção. Esta consideração é visualizada na Figura 13.

Figura 13 – Representação da detecção da tala como coordenadas 3D



Fonte: Autoria Própria

Para descobrir a posição geográfica da tala, basta descobrir o *timestamp* da tala no momento que ela passar pelo sensor de GPS. Considerando que o sensor de GPS está na mesma posição da câmera, em relação à imagem capturada por ela, uma heurística simples que pode ser adotada é que a seção que representa a posição do sensor é a linha horizontal que toca na parte inferior da imagem, como mostrado na Figura 14, com a borda inferior verde.

Figura 14 – Borda verde indicando a posição estimada do sensor de GPS em relação à imagem

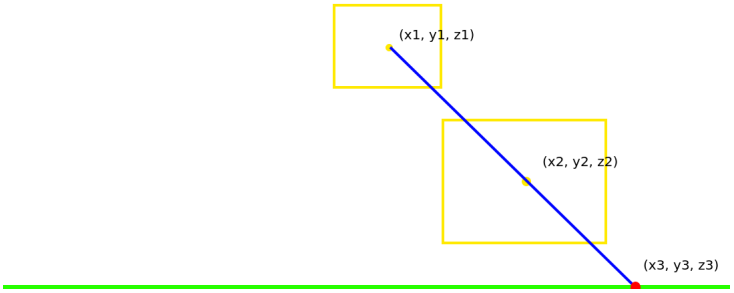
Fonte: Autoria Própria

É importante notar que a heurística anterior, que a borda inferior da imagem representa a posição da câmera, só pode ser adotada uma vez que a câmera está próxima dos trilhos e direcionada para baixo. Desta forma, o erro real em relação a posição de GPS é muito pequeno, podendo ser desconsiderado.

Então, em relação aos vetores criados anteriormente, basta calcular o timestamp no

momento em que a linha criada pelos dois vetores intercepta a linha horizontal que toca a parte inferior da imagem, como mostra a Figura 15.

Figura 15 – Representação do cálculo para encontrar o *timestamp* da tala



Fonte: Autoria Própria

Para efetivamente realizar este cálculo, é necessário realizar o seguinte procedimento:

1. Seja a tala definida pelo vetor (x, y, z), onde:
   - x e y representam a posição horizontal e vertical do centro da detecção. Ambos são normalizados no intervalo $[0..1]$. O eixo y está invertido, ou seja, o topo da imagem possui $y = 0$ e o fundo $y = 1$, como é comum na representação de imagens.
   - z representa o *timestamp* da detecção, ou seja, o momento que ela ocorreu no vídeo.
2. Tem-se duas talas, representadas por $\vec{v_1} = (x_1, y_1, z_1)$ e $\vec{v_2} = (x_2, y_2, z_2)$, e se deseja calcular um terceiro vetor $\vec{v_3} = (x_3, y_3, z_3)$ tal que $y_3 = 1$ e que este vetor intercepte a linha criada pelos vetores 1 e 2. Para isto, encontra-se o parâmetro t que representa a proporção do eixo y entre os vetores 3 e 1 e os vetores 2 e 1 através da Equação 12:

$$t = \frac{y_3 - y_1}{y_2 - y_1} \tag{12}$$

3. Em seguida, o vetor 3 é calculado pela Equação 13:

$$\vec{v_3} = \vec{v_1} + t * (\vec{v_2} - \vec{v_1}) \tag{13}$$

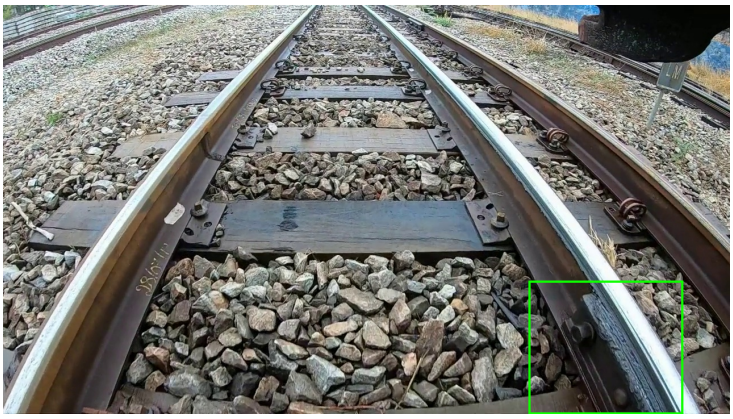
4. Para calcular o valor de z_3 (*timestamp* da tala), utiliza-se a Equação 14:

$$\textit{Timestamp} \text{ da tala} = z_1 + \frac{y_3 - y_1}{y_2 - y_1} * (z_2 - z_1) \tag{14}$$

5. Para obter a localização geográfica aproximada da tala, basta associar o *timestamp* obtido com as duas leituras mais próximas do sensor de GPS, de forma semelhante ao que é feito no cálculo do *timestamp* da tala.

Um grande desafio em relação à avaliação do módulo de cálculo da localização geográfica da tala é a inexistência de um *dataset* que pudesse ser considerado como *ground-truth* para a localização verdadeira das talas. Isto é, não se tem a localização real das talas que aparecem nas capturas de vídeo que estavam disponíveis. Portanto, se faz necessário gerar dados com localização aproximada das talas. Para isto, os vídeos foram anotados em relação ao quadro que seria o mais provável onde uma tala específica estivesse, logo abaixo da câmera, de forma que sua localização seria a mais próxima possível do sensor de GPS. A Figura 16 mostra um exemplo em que dois quadros à frente, a 30 FPS, uma tala, evidenciada pela caixa verde, estaria bem abaixo da câmera.

Figura 16 – Momento em que a tala estaria próximo ao sensor GPS



Fonte: Autoria Própria

Ao realizar estas anotações ao longo das capturas de vídeos, obtêm-se os *timestamps* de todas as talas que se encontram neles. Desta forma, ao relacionar estes *timestamps* com as localizações geográficas do simulador do sensor do GPS, da mesma forma que foi explicado acima, a localização geográfica aproximada destas anotações é obtida e um arquivo GeoJSON com tais informações é gerado. Isto permite a comparação com o resultado da localização geográfica calculada pelo sistema.

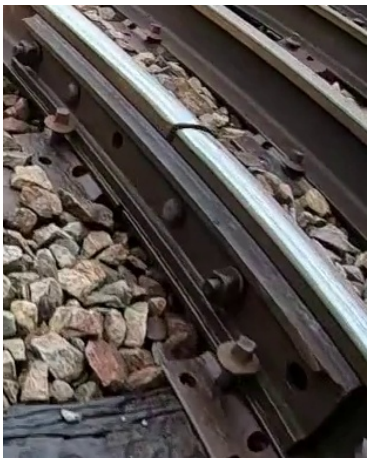
Um ponto importante a ser enfatizado em relação a estes dados gerados é que, apesar do processo de anotação dos vídeos ser trabalhoso e não apresentarem a posição geográfica real da tala, eles são essenciais para realizar uma avaliação imparcial dos resultados obtidos com o cálculo das localizações geográficas obtidas a partir do vídeo.

Para realizar a associação entre o arquivo GeoJSON gerado e o calculado pelo sistema, a distância entre os pontos de ambos os arquivos é calculada e os pontos mais próximos são considerados como representantes de uma mesma tala, caso estejam a uma distância menor do que 5 metros entre si.

3.2.4.4 Análise da Tala

Ao final do processamento, é necessário analisar a tala para definir se ela está em boas condições ou não. Para realizar esta análise, a rede de detecção de objetos na tala é utilizada. A partir das três detecções com as melhores pontuações achadas anteriormente para cada tala no trajeto ferroviário, um recorte é feito na imagem original do quadro correspondente à detecção que será analisada. Um exemplo deste recorte é mostrado na Figura 17 .

Figura 17 – Recorte da imagem original para análise da tala



Fonte: Autoria Própria

Este recorte será utilizado como entrada para a rede de detecção de objetos na tala, que identificará os furos contidos na tala de junção representada por esta detecção. Então, os seguintes critérios serão utilizados para definir a conformidade da tala:

- Caso não possua furos, está adequada e com nível de criticidade 0.
- Caso possua um furo, está não adequada e com nível de criticidade 1.
- Caso possua dois ou mais furos, está não adequada e com nível de criticidade 2.

Para melhorar o resultado da análise da tala, a inferência e classificação de criticidade é feita nas três detecções com maior pontuação encontrada. Assim, o resultado da análise geral é

definido por duas ou mais detecções com análise com mesmo nível de criticidade. Por exemplo, se duas detecções mostrarem criticidade 1 e uma mostrar criticidade 0, a criticidade da tala será calculada como 1. Caso as três detecções mostrarem valores distintos de criticidade, o valor de criticidade será considerado como 1.

A análise da tala é equivalente a um problema de classificação, tanto no caso binário (talas adequadas e não-adequadas), quanto no caso de multi-classe (de acordo com o grau de criticidade). Assim, a forma de avaliar o módulo é utilizando uma matriz de confusão. Para isto, é necessário anotar manualmente as talas encontradas pelo sistema, de forma que se saiba qual o nível de não conformidade da tala e comparar com o resultado obtido pelo sistema.

Por fim, ao finalizar a análise de todas talas encontradas no vídeo de entrada, um arquivo GeoJSON é gerado com as informações geográficas da tala e seu nível de criticidade. Desta forma, é possível visualizar o resultado do processamento facilmente com qualquer software de visualização de mapas.

4 RESULTADOS E DISCUSSÃO

Neste capítulo serão mostrados os resultados e como eles foram obtidos. Primeiramente, serão mostrados os resultados e métricas dos diferentes modelos de rede neural convolucional. Depois, dos diferentes módulos do sistema. Por fim, os resultados obtidos com o sistema como um todo, além das possíveis falhas que ocorreram.

4.1 MODELOS

Todos os resultados foram obtidos com mesmo hardware, a Jetson Nano (subseção 3.1.7).

4.1.1 Detecção de Tala

Os seguintes modelos foram criados para a detecção de tala:

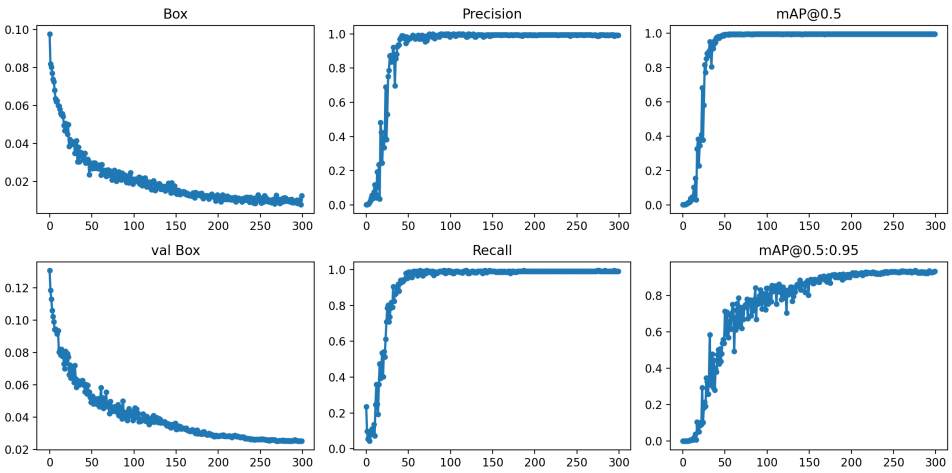
1. Modelo com arquitetura base do YOLOv7 e resolução de 640x640, treinado por 300 épocas.
2. Modelo com arquitetura YOLOv7 Tiny e resolução de 640x640, treinado por 209 épocas.
3. Modelo com arquitetura YOLOv7 Tiny e resolução de 448x448, treinado por 212 épocas.
4. Modelo com arquitetura YOLOv7 Tiny e resolução de 416x416, treinado por 300 épocas.

Destes modelos, as métricas discutidas nesta Seção se referem ao modelo 4, com o qual foram obtidos os melhores resultados. Os resultados e discussão para o outro modelo estão disponíveis no apêndice A.1.

A Figura 18 mostra as métricas obtidas com o treinamento do modelo de detecção de tala utilizado. A primeira coluna se refere à curva de *Box* e seu equivalente do *dataset* de validação, que representa a precisão das caixas delimitadoras obtidas. Na segunda coluna, tem-se *Precision* e *Recall*, métricas clássicas na avaliação de detectores de objetos. Estas métricas refletem a proporção de verdadeiros positivos em relação aos exemplos positivos previstos, e a proporção de verdadeiros positivos em relação a todos os exemplos positivos existentes, respectivamente. Por fim, na última coluna, tem-se as métricas de mAP@0,5 e mAP@0,5:0,95,

que calculam o mAP considerando um IOU de 0,5 e entre as faixas de IOU a partir de 0,5 até 0,95 com passos de 0,05, respectivamente.

Figura 18 – Métricas do modelo de detecção de tala com o modelo YOLOv7 Tiny 416x416



Fonte: Autoria Própria

Observa-se na Figura 18 que a métrica que representa os erros em relação às caixas delimitadoras propostas (*Box*) converge para 0, e as métricas que representam a eficiência do modelo em encontrar o objeto proposto (*Precision* e *Recall*) convergem para 1. Por fim, as métricas de mAP alcançam valores maiores do que 0,9, mostrando que a rede aprendeu a generalizar os dados a partir do treinamento realizado.

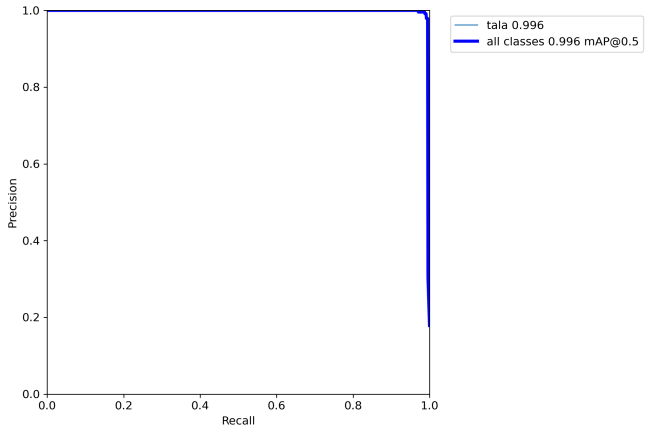
O gráfico da Figura 19 mostra a relação entre *Precision* e *Recall*, para o qual foi obtida uma área sob a curva de 0,996, indicando que o modelo atinge resultados muito bons para a relação entre verdadeiros positivos e falsos positivos, representado visualmente na Figura 19 pela proximidade da curva ao canto superior a direita do gráfico.

4.1.1.1 Métricas para o sistema embarcado

Para a escolha do modelo específico para viabilizar o sistema e obter os resultados das próximas Seções, foi realizado o teste de velocidade de processamento (FPS) para todos os modelos gerados pelo treinamento inicial, tanto os convertidos para TensorRT e quanto os com precisão reduzida. A Tabela 1 mostra os resultados obtidos.

Na Tabela 1, é observa-se que o modelo com arquitetura YOLOv7 Tiny, resolução 416x416, utilizando sistema de inferência TensorRT e com precisão numérica reduzida, evidenciado em negrito, é capaz de processar 36 FPS, o que é suficiente para o sistema proposto.

Figura 19 – Curva de *Precision* em função do *Recall* para o modelo de detecção de tala YOLOv7 Tiny 416x416



Fonte: Autoria Própria

Tabela 1 – Métricas dos modelos de detecção de talas

Modelo	Resolução	Biblioteca	Precisão Numérica	mAP@0.5	mAP@0.5:0.95	FPS
YOLOv7	640x640p	PyTorch	fp32	1	**0.9624**	1.2
YOLOv7 Tiny	640x640p	PyTorch	fp32	0.9908	0.8909	6.5
YOLOv7 Tiny	448x448p	PyTorch	fp32	1	0.9184	11.1
YOLOv7 Tiny	416x416p	PyTorch	fp32	0.9895	0.9033	**12.5**
YOLOv7	640x640p	TensorRT	fp32	1	**0.9598**	1.8
YOLOv7 Tiny	640x640p	TensorRT	fp32	0.9909	0.8987	10.8
YOLOv7 Tiny	448x448p	TensorRT	fp32	1	0.9165	22.2
YOLOv7 Tiny	416x416p	TensorRT	fp32	1	0.9357	**23.6**
YOLOv7	640x640p	TensorRT	fp16	1	**0.9603**	3.0
YOLOv7 Tiny	640x640p	TensorRT	fp16	0.9909	0.9012	16.5
YOLOv7 Tiny	448x448p	TensorRT	fp16	1	0.9165	35.2
YOLOv7 Tiny	**416x416p**	**TensorRT**	**fp16**	1	0.9355	**36.0**

Fonte: Autoria própria.

Portanto, foi o modelo utilizado nos próximos experimentos e no sistema final. É interessante notar que o modelo equivalente, mas com resolução 448x448, possui um FPS de 35,2, o que também é suficiente para o sistema proposto. No entanto, mesmo com uma resolução maior, o modelo com resolução 416x416 possui uma mAP maior.

4.1.2 Detecção de Objetos na Tala

Os seguintes modelos foram criados para a detecção de objetos na tala:

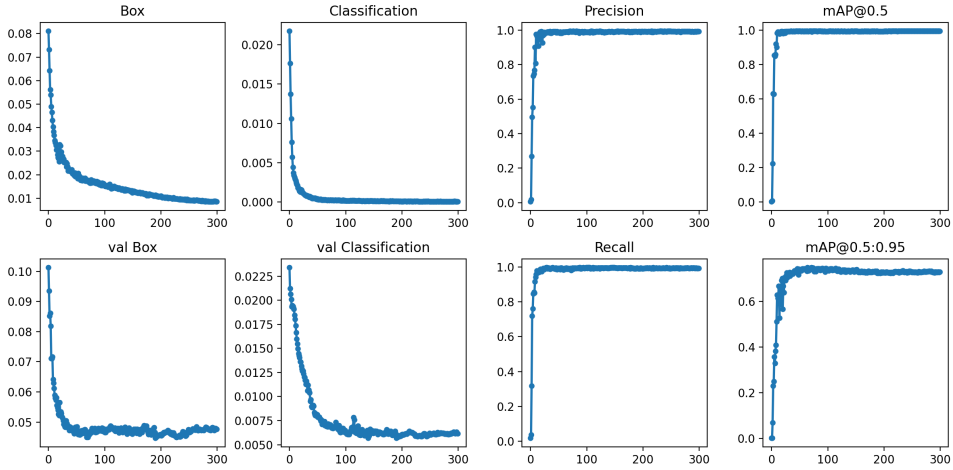
1. Modelo com arquitetura base do YOLOv7 e resolução de 416x416, treinado por 300 épocas.

2. Modelo com arquitetura YOLOv7 Tiny e resolução de 416x416, treinado por 300 épocas.

Destes modelos, as métricas discutidas nesta seção se referem ao modelo 1. Os resultados e discussão dos outros modelos estão disponíveis no apêndice A.2.

A Figura 20 mostra as métricas obtidas com o treinamento do modelo de detecção de objetos na tala utilizado. Esta Tabela difere da Tabela 1 na segunda coluna, que contém a curva de *Class* e seu equivalente do *dataset* de validação, que avalia a precisão da classificação do objeto contido na caixa delimitadora proposta. Este valor é importante pois o modelo possui mais de uma classe.

Figura 20 – Métricas do modelo de detecção de objetos na tala YOLOv7 416x416



Fonte: Autoria Própria

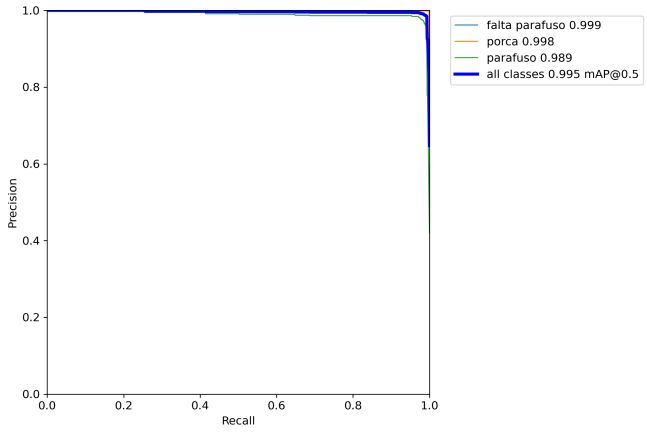
Com este modelo é possível observar, assim como no modelo de detecção de talas, uma convergência das métricas para valores bastante satisfatórios. Um ponto interessante de se notar em relação à métrica de mAP@0.5:0.95, é que ela possui um pico e depois diminui, o que pode indicar um eventual *overfitting* do modelo para épocas maiores de 100.

O gráfico da Figura 21 mostra a relação entre *Precision* e *Recall*, para a qual foi obtida uma área sob a curva geral de 0,995, 0,999 para classe **falta de parafuso**, 0,998 para classe **porca**, e 0,989 para classe **parafuso**, de forma semelhante a Figura 19, a qualidade do resultado é representado visualmente na Figura 21 pela proximidade da curva ao canto superior a direita do gráfico.

4.1.2.1 Métricas no sistema embarcado

Assim como no caso da detecção de talas, para a escolha do melhor modelo para o sistema e obter os resultados das próximas seções, foi realizado o teste de velocidade de

Figura 21 – Curva de *Precision* em função do *Recall* para o modelo de detecção de objetos na tala YOLOv7 416x416



Fonte: Autoria Própria

processamento (em FPS) dos modelos gerados e convertidos. A Tabela 2 mostra os resultados obtidos.

Tabela 2 – Métricas dos modelos de detecção de de objetos na talas

Modelo	Resolução	Biblioteca	Precisão Numérica	mAP@0.5	mAP@0.5:0.95	fps
YOLOv7	416x416p	PyTorch	fp32	**0.9914**	**0.7392**	2.1
YOLOv7 Tiny	416x416p	PyTorch	fp32	0.9897	0.7323	**12.4**
YOLOv7	416x416p	TensorRT	fp32	**0.9902**	**0.7543**	4.3
YOLOv7 Tiny	416x416p	TensorRT	fp32	0.9843	0.7333	**23.9**
YOLOv7	**416x416p**	**TensorRT**	**fp16**	**0.9902**	**0.7545**	6.2
YOLOv7 Tiny	416x416p	TensorRT	fp16	0.9843	0.7333	**35.9**

Fonte: Autoria própria.

Para a análise da tala, o FPS não tem grande importância, pois isto só será feito apenas para as três melhores detecções rastreadas para cada instância de tala encontrada. Então, a métrica mais importante é a mAP@0.5:0.95, que é a que melhor representa a acurácia do modelo. De acordo com a Tabela 2, o modelo com arquitetura YOLOv7, resolução 416x416, utilizando sistema de inferência TensorRT e com precisão numérica reduzida possui o maior valor de mAP@0.5:0.95 evidenciado em negrito. Portanto, foi o utilizado no sistema final.

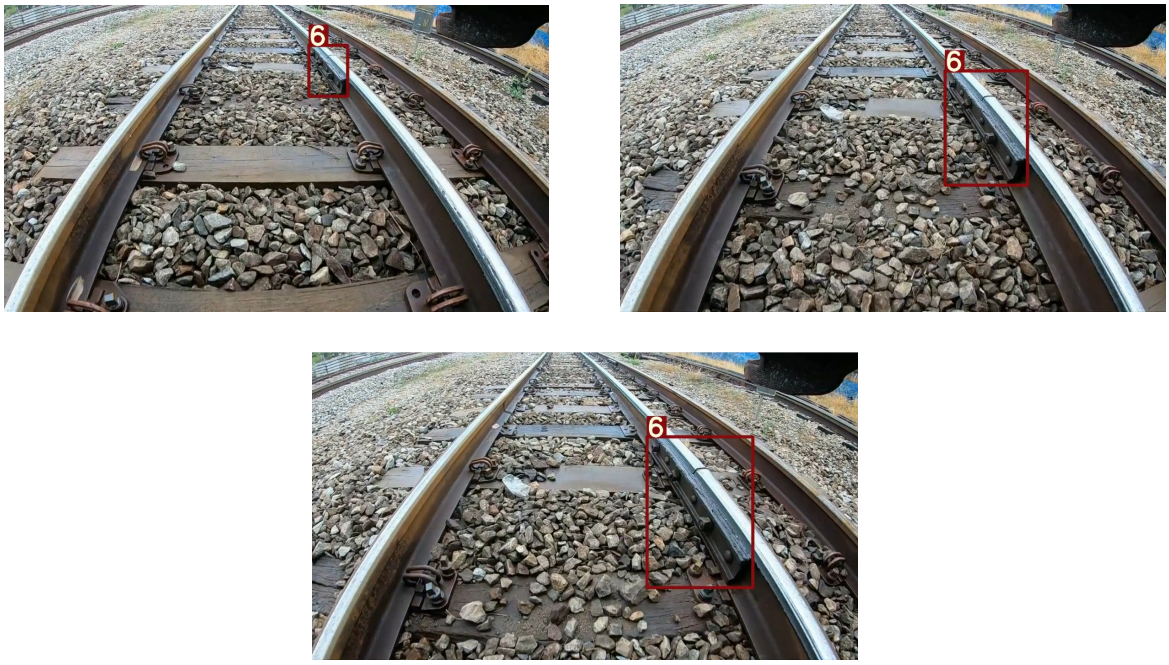
4.2 MÓDULOS

Os principais desafios do sistema foram o rastreamento da tala, o cálculo de sua posição geográfica e sua análise. Portanto, tem-se a necessidade de obter resultados e métricas que indicassem a qualidade das soluções desenvolvidas. As Seções a seguir detalham os resultados obtidos com a implementação de cada módulo do sistema.

4.2.1 *Tracking*

Para exemplificar o rastreamento da tala, a Figura 22 mostra três quadros onde diferentes detecções foram consideradas como uma mesma tala.

Figura 22 – Exemplo do *tracking* de talas



Fonte: Autoria Própria

Nas capturas de vídeo utilizadas, há 46 talas diferentes, o algoritmo de *tracking* em conjunto do modelo realizaram 41 rastreamentos corretos e 4 falsos positivos (instâncias consideradas talas que na verdade não eram). Com base nestes dados, pode-se concluir que o módulo de *tracking* possui uma acurácia de 93,18% e precisão de 91,11% nas capturas obtidas.

Para as talas não rastreadas, tem-se diferentes situações. A Figura 23 mostra exemplos de talas que não foram rastreadas pois estavam oclusas por pedra brita. Nesta figura, em alguns casos a tala está completamente oclusa, sendo dificil até a detecção dela por humanos. Em outros casos, como na Figura 23(a) a tala está parcialmente oclusa, necessitando de mais exemplos deste tipo de problema no *dataset* de treinamento para que a rede considere como uma tala.

A Figura 24 mostra um caso de erro no algoritmo de *tracking* onde a tala está se movendo muito rápido em relação ao vídeo para ser detectada e rastreada corretamente. Uma das possíveis correções para este problema é aumentar o FPS da câmera, o que, no contexto do projeto proposto, não é viável sem o aumento do custo do sistema embarcado utilizado. Outra possível solução seria a anotação de mais exemplos de talas distantes da câmera, a Figura 25

Figura 23 – Exemplo de talas não rastreadas devido à oclusão.

(a) Tala parcialmente oclusa **(b) Tala completamente oclusa**

Fonte: Autoria Própria

mostra um exemplo do momento em que uma tala é detectada pela rede, notavelmente próxima da câmera. É um problema comum dos modelos de detecção de objetos, que não conseguem detectar objetos muito pequenos ou muito distantes da câmera, e existem múltiplas técnicas que poderiam ser implementadas para aumentar esta distância de detecção.

Figura 24 – Erro no rastreamento devido à velocidade da tala no video.

Fonte: Autoria Própria

Por fim, tem-se a situação que o modelo gera um falso positivo. A Figura 26 mostra um exemplo de detecção que não possui tala. É um problema comum em redes neurais e diz respeito à robustez da rede, sendo que o problema pode ser diminuído com a implementação de melhorias no processo de treinamento.

Figura 25 – Exemplo de *tracking* de talas

(a) Distante da câmera

(b) Momento antes da detecção

(c) Primeiro frame do rastreamento

Fonte: Autoria Própria

Figura 26 – Detecção de falso positivo

Fonte: Autoria Própria

4.2.2 Localização Geográfica da Tala

A Figura 27 mostra um exemplo com algumas associações entre pontos do arquivo GeoJSON anotado e do arquivo GeoJSON calculado pelo sistema. Nesta figura, os pontos verdes representam a localização correta da tala e os pontos vermelhos são os calculados pelo sistema. Também, as linhas pretas representam a associação entre os pontos.

Das 46 talas contidas nas capturas de vídeo, 40 foram corretamente associadas, o que representa uma acurácia de 86,95% e precisão de 88,88% nas capturas obtidas.

Figura 27 – Associação entre pontos do arquivo GeoJSON anotado e do arquivo GeoJSON calculado pelo sistema

Fonte: Autoria Própria

Os problemas em relação ao sistema de localização são semelhantes aos problemas de rastreamento das talas, portanto, a discussão sobre estes resultados obtidos e as soluções implementáveis são as mesmas.

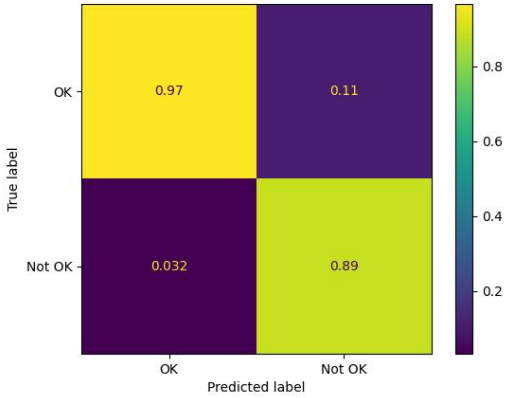
4.2.3 Análise da Criticidade da Tala

Das 40 talas avaliadas, a Figura 28 mostra a matriz de confusão que representa a comparação entre o resultado obtido pelo sistema e o anotado manualmente, considerando o caso de classificação binária, onde a diagonal principal representa os acertos e as outras células representam os erros.

A Figura 29 mostra a matriz de confusão que representa a classificação multi-classe, de acordo com o nível de criticidade da tala. As talas não-adequadas com os níveis de criticidade 1 e 2 são discutidas na Seção 3.2.4.4.

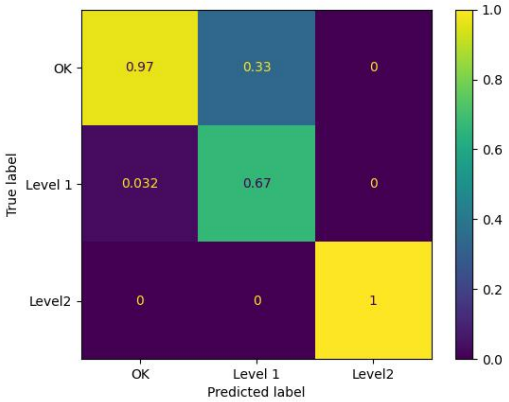
A partir dos resultados apresentados, é possível observar que a classificação binária entre tala adequada e não-adequada, é obviamente mais simples de ser realizada. Isto decorre da natureza das redes neurais e da captura utilizada, sendo comum casos onde existe uma dificuldade para detectar os objetos contidos nas talas. Diferentes situações do ambiente e das detecções podem causar tais erros, como na Figura 30 onde são mostrados exemplos de talas que foram

Figura 28 – Matriz de confusão para o caso de classificação em conformidade e não conformidade



Fonte: Autoria Própria

Figura 29 – Matriz de confusão para o caso de classificação multi-classe pelo nível de criticidade



Fonte: Autoria Própria

classificadas de forma errada pelo sistema. Alguns dos motivos são: posição da tala em relação à câmera, que dificulta a detecção dos furos na tala; oclusão da tala por outros objetos; caixa de delimitação cortando uma parte da tala, que pode esconder furos contidos nela; velocidade da locomotiva e taxa de atualização da câmera, o que torna a imagem borrada, dificultando a classificação.

Figura 30 – Exemplos possíveis problemas na análise da tala

(a) Em posição desfavorável

(b) Cortada no momento da inferência

(c) Tala oclusa por outros objetos

(d) Tala borrada devido à velocidade da locomotiva

Fonte: Autoria Própria

5 CONCLUSÃO

A malha ferroviária brasileira é uma das maiores do mundo, com mais de 30 mil quilômetros de extensão. Ainda assim, é um setor que carece de pesquisas e tecnologias que possam melhorar a eficiência e a segurança do transporte ferroviário. Mesmo com o recente crescimento das tecnologias de visão computacional por aprendizado de máquina, ainda são poucos os trabalhos que aplicam tais tecnologias no setor ferroviário. Desta forma, este trabalho teve como objetivo desenvolver um sistema de detecção de anomalias em talas de junção, que são peças de aço que unem dois trilhos de uma ferrovia, utilizando técnicas modernas de visão computacional e aprendizado profundo.

O sistema proposto foi idealizado para viabilizar sua utilização em sistemas embarcados de baixo custo. Para alcançar este objetivo, o sistema que utiliza redes neurais convolucionais, especificamente a arquitetura YOLOv7, para detectar talas de junção em vídeos de câmeras colocadas na parte frontal de uma locomotiva. Todos os aspectos da solução foram desenvolvidos, desde a anotação das capturas de vídeo, o treinamento de redes neurais e a criação de um sistema que utiliza as redes treinadas para retirar as informações desejadas das imagens capturadas.

Também, diferentes técnicas de otimização foram utilizadas para permitir que o sistema desenvolvido pudesse ser embarcado, tais como a utilização do sistema de inferência com o TensorRT, que otimiza a rede neural para uma GPU específica, a utilização de um algoritmo para *tracking* de baixo custo computacional, e o uso da versão *Tiny* da arquitetura YOLOv7. O sistema foi testado e validado em uma plataforma Jetson Nano, um dos modelos mais simples da família Jetson. Isto evidencia a viabilidade do projeto como um produto embarcado completo.

Conforme mostrado no Capítulo 4, os resultados obtidos foram condizentes com os requisitos especificados do projeto. Em resumo, o sistema desenvolvido foi capaz de: encontrar as talas contidas na captura de vídeo, identificar sua localização, e classificar seu estado de acordo com a quantidade de parafusos faltantes. Além disto, o sistema foi capaz de realizar essas tarefas em tempo real, com um FPS acima da taxa de atualização da captura do vídeo, e de forma embarcada, utilizando uma Jetson Nano. Portanto, todos os objetivos delimitados na Seção 1.1 foram alcançados.

5.1 TRABALHOS FUTUROS

Embora o projeto tenha alcançado sucesso, algumas melhorias poderão ser feitas, tais como:

- Algoritmos de visão computacional por apendizado de máquina são sensíveis à quantidade de dados utilizada para treinamento. Assim, diferentes capturas de dados poderiam ser utilizadas tanto para o treinamento quanto para o teste, utilizando diferentes dispositivos de captura ou diferentes condições de iluminação. Com isso, poderia ser desenvolvida uma solução com maior robustez, podendo ser utilizada em situações mais realistas e com diferentes especificações. Por exemplo, utilizando diferentes dispositivos de captura, funcionando em diferentes condições climáticas e conseguindo detectar os diferentes tipos de talas de junção que são utilizadas na malha ferroviária.

- Diferentes *backbones* poderiam ser utilizados além do YOLOv7 padrão, objetivando uma solução mais eficiente e que demande menos recursos computacionais.

- O uso de algumas heurísticas no sistema de *tracking* poderiam melhorar ainda mais a acurácia da solução, como a suposição que a tala sempre está associada aos trilhos. Além disto, poderiam ser testadas outras técnicas de pré-processamento e pós-processamento das imagens e caixas delimitadoras inferidas.

- Os dados gerados são arquivos de diferentes formatos, tais como imagens e arquivos de GeoJSON. Um sistema de armazenamento e visualização dos dados poderia ser criado para facilitar o tratamento e a análise dos dados gerados. Inclusive adicionando um aspecto temporal aos resultados, de forma que se pudesse inferir a evolução do estado dos trilhos ferroviários ao longo do tempo.

REFERÊNCIAS

BROWNLEE, Jason. **A Gentle Introduction to Transfer Learning for Deep Learning**. 2019. Disponível em: https://www.redhat.com/en/topics/edge-computing/what-is-edge-ai. Acesso em: 15 de novembro de 2023.

CHINTALA, Soumith. **PyTorch strengthens its governance by joining the Linux Foundation**. 2022. Disponível em: https://pytorch.org/blog/PyTorchfoundation/. Acesso em: 15 de outubro de 2023.

CNT. **Boletim Estatístico de Fevereiro de 2019**. 2019. Disponível em: https://www.cnt.org.br/boletins. Acesso em: 28 de outubro de 2022.

DECI. Understanding nvidia tensorrt for deep learning inference optimization. 2023. Disponível em: https://deci.ai/blog/tensorrt-framework-overview/. Acesso em: 20 de novembro de 2023.

DISTRITO. **Data augmentation: o que é e como usar essa técnica?** 2023. Disponível em: https://distrito.me/blog/data-augmentation-o-que-e-e-como-usar-essa-tecnica/. Acesso em: 13 de dezembro de 2023.

DNIT. **Procedimentos de Inspeção de Materiais – PIM 02 – Tala de junção para trilhos**. 2015. Disponível em: https://www.gov.br/dnit/pt-br/ferrovias/instrucoes-e-procedimentos/procedimentos-para-inspecao/pim-002-tala-de-juncao.pdf. Acesso em: 28 de outubro de 2022.

Forbes. **What You Need To Know About Dark Data**. 2019. Disponível em: https://www.forbes.com/sites/tomtaulli/2019/10/27/what-you-need-to-know-about-dark-data/?sh=649811242c79. Acesso em: 20 de novembro de 2023.

Gartner. **What Edge Computing Means for Infrastructure and Operations Leaders**. 2018. Disponível em: https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders. Acesso em: 20 de novembro de 2023.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. *[S.l.]*: MIT Press, 2016. http://www.deeplearningbook.org.

ITSEEZ. **Open Source Computer Vision Library**. 2015. Disponível em: https://github.com/itseez/opencv. Acesso em: 15 de outubro de 2023.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. *In*: PEREIRA, F.; BURGES, C.J.; BOTTOU, L.;

WEINBERGER, K.Q. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, 2012. v. 25, p. 1–9. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf. Acesso em: 17 de novembro de 2023.

LECUN, Yann; BENGIO, Yoshua *et al.* Convolutional networks for images, speech, and time series. **The handbook of brain theory and neural networks**, MIT Press, v. 3361, n. 10, p. 255–258, 1998.

QGIS Association. **QGIS Geographic Information System**. 2023. Disponível em: http://www.qgis.org. Acesso em: 15 de outubro de 2023.

RAMSUNDAR, Bharath; ZADEH, Reza Bosagh. Fully connected deep networks. *In*: **TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning**. *[S.l.]*: O'Reilly Media, 2018. cap. 4.

Red Hat. **What is edge AI?** 2023. Disponível em: https://www.redhat.com/en/topics/edge-computing/what-is-edge-ai. Acesso em: 20 de novembro de 2023.

REDE GLOBO. **Trem carregado com combustível descarrila após parafusos serem retirados da linha férrea**. 2018. Disponível em: https://g1.globo.com/sp/bauru-marilia/noticia/trem-carregado-com-combustivel-descarrila-apos-parafusos-serem-retirados-da-linha-ferrea.ghtml. Acesso em: 28 de outubro de 2022.

REDE GLOBO - Santos. **Trem carregado com grãos descarrila e vagões tombam em Cubatão**. 2022. Disponível em: https://g1.globo.com/sp/santos-regiao/noticia/2022/10/27/trem-carregado-com-graos-descarrila-e-vagoes-tombam-em-cubatao-video.ghtml. Acesso em: 28 de outubro de 2022.

REDMON, Joseph; DIVVALA, Santosh Kumar; GIRSHICK, Ross B.; FARHADI, Ali. You only look once: Unified, real-time object detection. **CoRR**, abs/1506.02640, 2015. Disponível em: http://arxiv.org/abs/1506.02640.

RUDER, Sebastian. An overview of gradient descent optimization algorithms. **CoRR**, abs/1609.04747, 2016. Disponível em: http://arxiv.org/abs/1609.04747.

SHAH, Deval. **Mean Average Precision (mAP) Explained: Everything You Need to Know**. 2022. Disponível em: https://www.v7labs.com/blog/mean-average-precision. Acesso em: 15 de novembro de 2023.

SHARMA, Priyanshi. **Everything about Pooling layers and different types of Pooling**. 2023. Disponível em: https://iq.opengenus.org/pooling-layers/. Acesso em: 15 de novembro de 2023.

SOLAWETZ, Jacob. **What is Mean Average Precision (mAP) in Object Detection?** 2020. Disponível em: https://blog.roboflow.com/mean-average-precision/. Acesso em: 15 de outubro de 2023.

TAQI, Arwa Mohammed; AWAD, Ahmed; AL-AZZO, Fadwa; MILANOVA, Mariofanna. The impact of multi-optimizers and data augmentation on tensorflow convolutional neural network performance. *In*: **2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)**. *[S.l.: s.n.]*, 2018. p. 140–145.

WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. **arXiv preprint arXiv:2207.02696**, 2022. Acesso em: 15 de outubro de 2023.

YEGULALP, Serdar. **Facebook brings GPU-powered machine learning to Python**. 2023. Disponível em: https://www.infoworld.com/article/3159120/facebook-brings-gpu-powered-machine-learning-to-python.html. Acesso em: 15 de outubro de 2023.

APÊNDICES

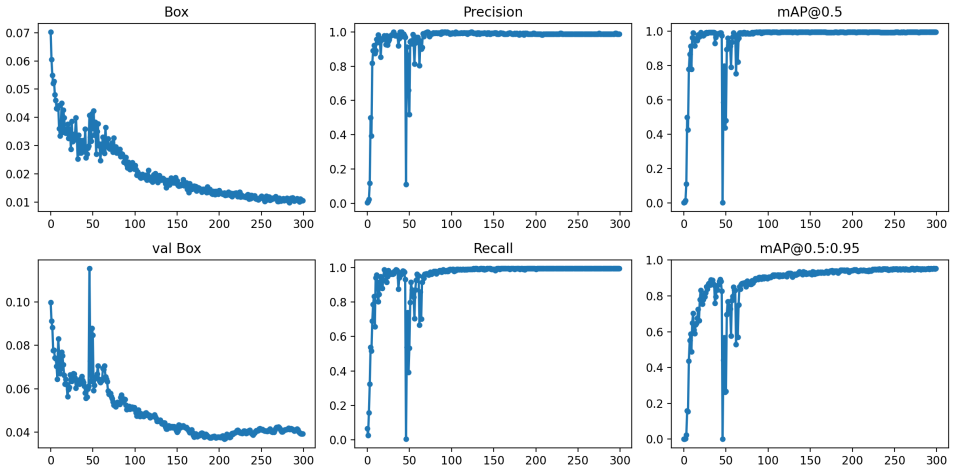
APÊNDICE A – RESULTADOS DOS MODELOS NÃO UTILIZADOS NO SISTEMA FINAL

A seguir, serão detalhados os resultados obtidos com as diferentes redes que foram treinadas para o sistema de identificação das talas de junção.

A.1 MODELOS DE DETECÇÃO DE TALA

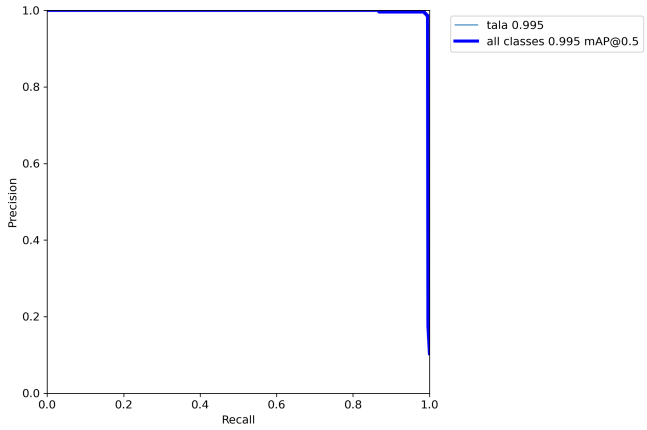
A Figura 31 mostra as métricas obtidas com o treinamento do modelo de detecção de tala utilizando a arquitetura base do YOLOv7 e resolução de 640x640. O gráfico da Figura 32 mostra a relação entre *Precision* e *Recall* deste modelo.

Figura 31 – Métricas do modelo de detecção de tala com o modelo YOLOv7 640x640



Fonte: Autoria Própria

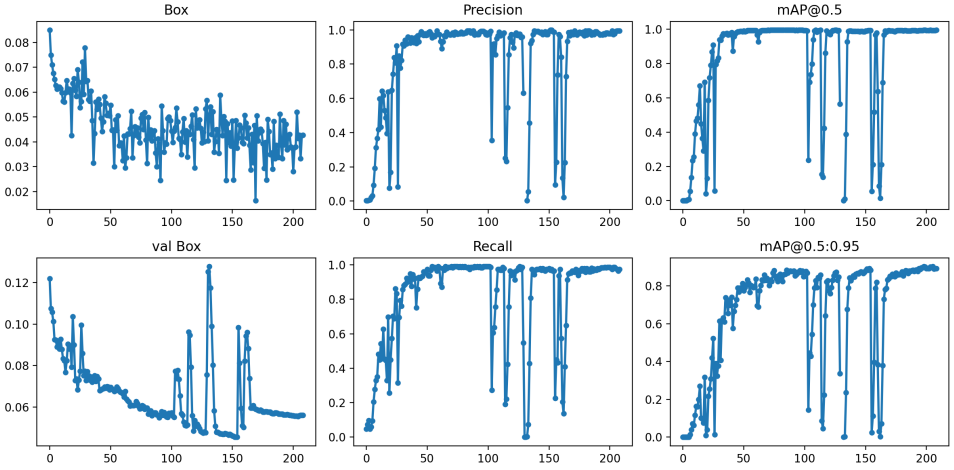
Figura 32 – Curva de *Precision* em função do *Recall* para o modelo de detecção de tala YOLOv7 640x640



Fonte: Autoria Própria

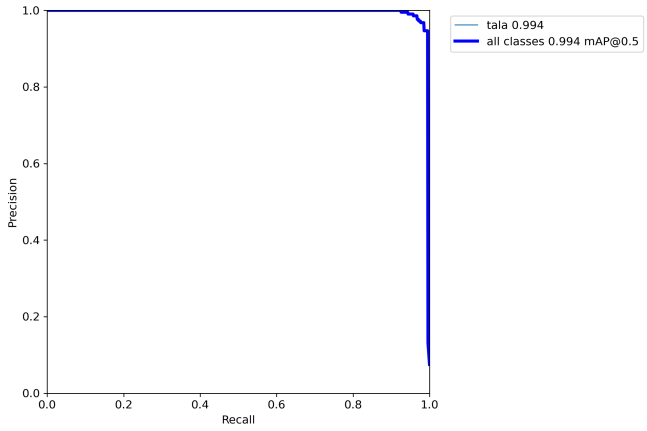
A Figura 33 mostra as métricas obtidas com o treinamento do modelo de detecção de tala utilizando a arquitetura YOLOv7 Tiny e resolução de 640x640. O gráfico da Figura 34 mostra a relação entre *Precision* e *Recall* deste modelo.

Figura 33 – Métricas do modelo de detecção de tala com o modelo YOLOv7 Tiny 640x640



Fonte: Autoria Própria

Figura 34 – Curva de *Precision* em função do *Recall* para o modelo de detecção de tala YOLOv7 Tiny 640x640

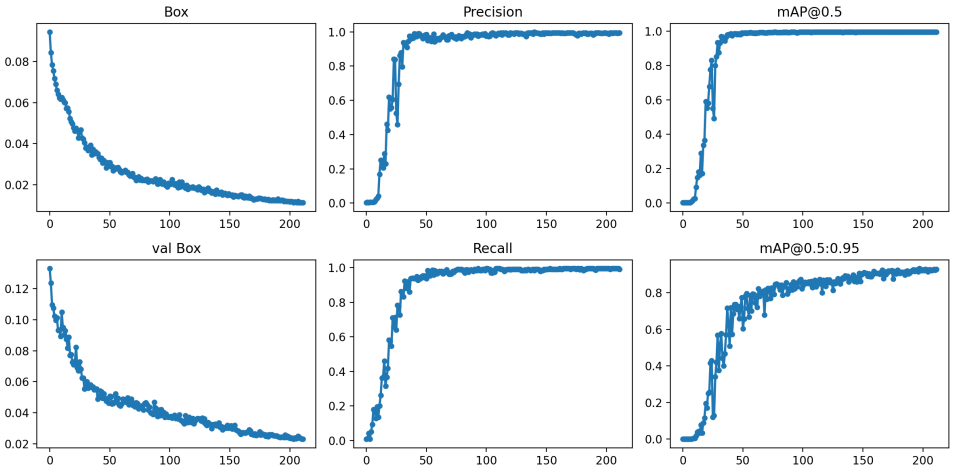


Fonte: Autoria Própria

A Figura 35 mostra as métricas obtidas com o treinamento do modelo de detecção de tala utilizando a arquitetura YOLOv7 Tiny e resolução de 448x448. O gráfico da Figura 36 mostra a relação entre *Precision* e *Recall* deste modelo.

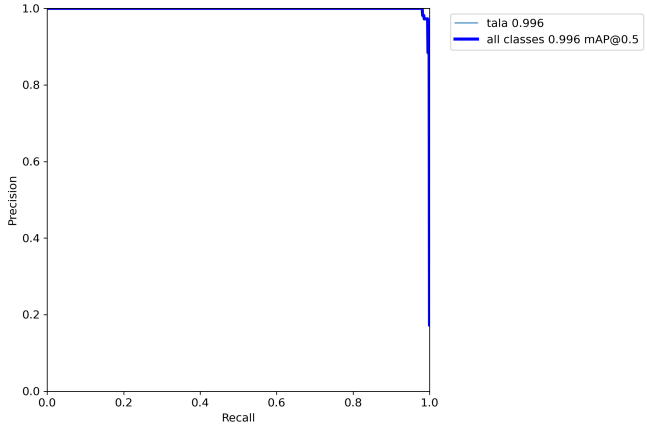
Observa-se que todos os modelos obtiveram resultados muito semelhantes e com alta qualidade. Naturalmente, o modelo com a arquitetura base do YOLOv7 e resolução de 640x640 obteve os melhores resultados e seria o mais adequado para o sistema final. No entanto, devido aos requisitos delimitados, ele não é viável para a solução final.

Figura 35 – Métricas do modelo de detecção de tala com o modelo YOLOv7 Tiny 448x448



Fonte: Autoria Própria

Figura 36 – Curva de *Precision* em função do *Recall* para o modelo de detecção de tala YOLOv7 Tiny 448x448



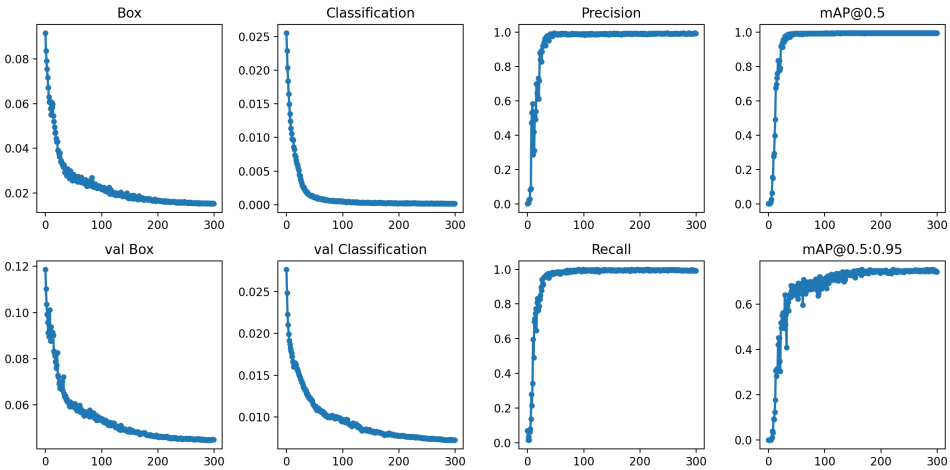
Fonte: Autoria Própria

A.2 MODELOS DE DETECÇÃO DE OBJETOS NA TALA

Apenas um modelo adicional foi treinado para a detecção de objetos na tala, utilizando a arquitetura YOLOv7 Tiny e resolução de 416x416. A Figura 37 mostra as métricas obtidas com o treinamento e gráfico da Figura 38 mostra a relação entre *Precision* e *Recall* deste modelo.

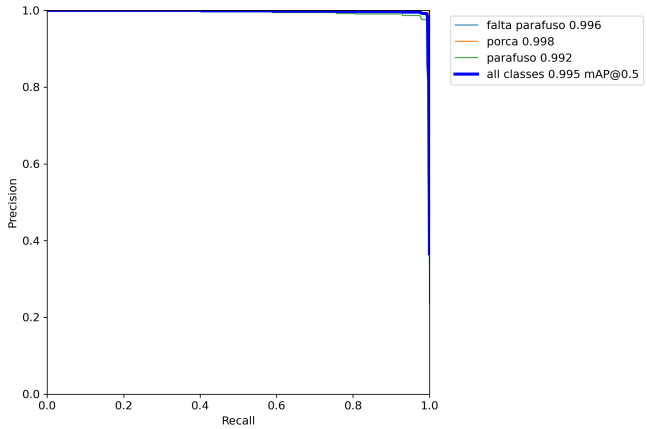
Por fim, observa-se que este modelo também alcançou resultados satisfatórios, mas ligeiramente piores que o modelo com arquitetura base do YOLOv7 e resolução de 640x640. Portanto, como o FPS não possui grande importância neste caso, este modelo não foi utilizado no sistema final.

Figura 37 – Métricas do modelo de detecção de objetos na tala YOLOv7 Tiny 416x416



Fonte: Autoria Própria

Figura 38 – Curva de *Precision* em função do *Recall* para o modelo de detecção de objetos na tala YOLOv7 Tiny 416x416



Fonte: Autoria Própria