Reconhecimento automático de placas de carro para aplicação de multas de trânsito
Relatório do desenvolvimento

Arthur Ruiz[1], **Edmar Melandes**[2], **Felipe Moura**[3], **Leon Junio**[4]

[1]Pontifícia Universidade Católica de Minas Gerais (PUC MG)
Instituto de Ciências Exatas e de Informática
Belo Horizonte – MG – Brasil

Abstract. *Inspection of blue lanes, or rotating parking lots, is crucial for traffic management in large cities. In Belo Horizonte, where there are around 20 thousand rotating parking spaces, ineffective inspection has made it difficult to fully utilize the expected benefits, such as greater vehicle turnover and easier access to local businesses. This article proposes an automated inspection system, using image processing and parallel computing to detect infractions efficiently and accurately. The proposed modernization aims to increase revenue, improve vacancy turnover and, consequently, facilitate access to areas of greatest demand, promoting better urban mobility. We detail the modular architecture of the system, which integrates a frontend in Next.js, a backend in Java, and an intelligent license plate recognition system based on YOLO and OCR models implemented in PyTorch. Distributed infrastructure with Docker Swarm and Apache Kafka ensures scalability and efficiency, while parallel computing techniques accelerate image processing. This approach aims to improve the accuracy and speed of detecting infractions, promoting more effective management of urban traffic.*

Resumo. *A fiscalização das faixas azuis, ou estacionamentos rotativos, é crucial para a gestão do trânsito em grandes cidades. Em Belo Horizonte, onde há cerca de 20 mil vagas rotativas, a fiscalização ineficaz tem dificultado a plena utilização dos benefícios esperados, como a maior rotatividade de veículos e o acesso facilitado ao comércio local. Este artigo propõe um sistema automatizado de fiscalização, utilizando processamento de imagens e computação paralela para detectar infrações de forma eficiente e precisa. A modernização proposta visa aumentar a arrecadação, melhorar a rotatividade das vagas e, consequentemente, facilitar o acesso às áreas de maior demanda, promovendo uma melhor mobilidade urbana. Detalhamos a arquitetura modular do sistema, que integra frontend em Next.js, backend em Java, e um sistema inteligente de reconhecimento de placas baseado em modelos YOLO e OCR implementados em PyTorch. A infraestrutura distribuída com Docker Swarm e Apache Kafka garante escalabilidade e eficiência, enquanto técnicas de computação paralela aceleram o processamento de imagens. Esta abordagem visa melhorar a precisão e a rapidez na detecção de infrações, promovendo uma gestão mais eficaz do trânsito urbano.*

1. Introdução

A fiscalização de faixas azuis, ou estacionamentos rotativos, é um elemento crítico na gestão do trânsito de grandes cidades. Em Belo Horizonte, o sistema de rotativo foi

implementado com o intuito de otimizar o uso das vagas de estacionamento nas áreas de maior fluxo de pessoas, o que promove a rotatividade de veículos e, consequentemente, facilita o acesso ao comércio local e outros serviços. Segundo dados da BHTrans, a cidade conta com aproximadamente 20 mil vagas de estacionamento rotativo distribuídas por diversas regiões. No entanto, a fiscalização ruim deste sistema, que é desafiadora de fato, impede que os benefícios de uma melhor mobilidade urbana sejam aproveitados.

O sistema de rotativo em Belo Horizonte permite que os motoristas estacionem em vagas demarcadas por um período de tempo limitado, geralmente entre 1 a 2 horas. Para utilizar essas vagas, os motoristas devem utilizar aplicativos móveis para registrar o tempo de estacionamento. Esse modelo visa a incentivar a rotatividade de veículos, garantindo que mais pessoas possam acessar as áreas com maior demanda por estacionamento.

Entretanto, a eficiência desse sistema depende fortemente da fiscalização adequada, na qual a forma manual de fiscalização compõe grande parte da frota de fiscais. Esta fiscalização manual exige que fiscais de trânsito percorram as áreas de rotativo verificando se os veículos estão devidamente registrados para o período de estacionamento. Esse método não só é laborioso e sujeito a falhas humanas, como também pode ser ineficiente em termos de cobertura e tempo de resposta.A fiscalização manual é incapaz de cobrir todas as áreas de forma eficiente, além disso, a detecção de infrações pode ser atrasada, levando a um uso inadequado das vagas e perda de receitas provenientes das multas. Neste sentido, é possível perceber que diversos problemas orbitam em torno da fiscalização antiguada de rotativos.

1.1. Objetivos

Diante desses desafios, a proposta deste projeto é desenvolver um sistema automatizado e eficiente para a fiscalização de faixas azuis utilizando técnicas de processamento e análise de imagens, integradas a uma infraestrutura de computação paralela e distribuída. Os objetivos específicos do projeto incluem:

- **Automatizar a detecção de infrações:** Desenvolver um sistema que permita aos fiscais capturar fotos das placas dos veículos estacionados e verificar automaticamente a conformidade com as regras de estacionamento.
- **Aumentar a eficiência da fiscalização:** Utilizar técnicas de computação paralela e distribuída para processar imagens em tempo efciênte, garantindo uma cobertura mais ampla e respostas mais rápidas.
- **Melhorar a precisão e a confiabilidade:** Empregar algoritmos avançados de reconhecimento de caracteres para reduzir erros e evitar contestação de multas.
- **Reduzir o tempo de resposta:** Implementar um sistema que permita a rápida detecção e notificação de infrações, facilitando o trabalho dos fiscais e melhorando a rotatividade das vagas.

1.2. Justificativa

A modernização da fiscalização de faixas azuis é fundamental para melhorar a gestão do trânsito em Belo Horizonte. Por isso, a nossa proposta visa utilizar a tecnologia para superar as limitações do método atual, proporcionando um sistema de rotativo que promove a mobilização urbana. Além de aumentar a arrecadação de multas e garantir o uso adequado das vagas, o novo sistema beneficiará a população ao promover maior rotatividade e disponibilidade de vagas, facilitando o acesso a áreas de alto fluxo.

2. Revisão de Literatura

A implementação de um sistema de Reconhecimento Automático de Placas Veiculares para a fiscalização de faixas azuis em Belo Horizonte é fundamentada em uma literatura que explora as diversas abordagens, avanços tecnológicos e aplicações práticas dessa tecnologia. Desta forma, nosso sistema de fiscalização de faixas azuis em Belo Horizonte se beneficia diretamente dos avanços discutidos na literatura revisada. A utilização do YOLOv5 para detecção de placas, conforme demonstrado por [da Silva Cimirro 2022], permite um reconhecimento rápido e preciso, essencial para a eficiência do sistema de fiscalização em tempo real. Além disso, a abordagem modular e distribuída que implementamos, inspirada por trabalhos como o de [Tang et al. 2022], assegura que nosso sistema possa ser escalado e integrado facilmente em uma infraestrutura como o de uma cidade inteligente, o que proporciona melhorias significativas na gestão de mobilidade urbana. Por outro lado, os desafios e soluções discutidos por [Patel et al. 2013]) e [Puranic et al. 2016] nos permitiram antecipar e mitigar problemas relacionados à variação de condições de iluminação e formatos de placas, garantindo o alto desempenho do nosso sistema em diferentes contextos operacionais. A seguir, revisamos os principais artigos que embasaram nosso projeto, destacando suas contribuições e relacionando-as com nosso contexto específico.

2.1. Sistemas de Reconhecimento Automático de Placas Veiculares (ANPR)

Patel, Shah e Patel realizaram um levantamento abrangente sobre sistemas ANPR, abordando a necessidade de tais sistemas para resolver problemas de identificação de veículos que violam regras de trânsito, especialmente em condições de alta velocidade e iluminação variável. Este artigo destacou os desafios que sistemas ANPR enfrentam, como a velocidade dos veículos, a não uniformidade das placas e as diferentes condições de iluminação, que podem afetar significativamente a taxa de reconhecimento. Por conta disto, foi possível antecipar estes problemas, assim, em nossa aplicação, esses desafios foram endereçados através da utilização de redes neurais convolucionais e do modelo YOLO (You Only Look Once) para garantir um reconhecimento preciso e eficiente das placas de veículos.

2.2. ANPR em Cidades Inteligentes

Tang, Wan, Schooling, Zhao, Chen e Wei forneceram uma revisão sistemática sobre a tecnologia ANPR no contexto das cidades inteligentes, discutindo os avanços técnicos, os fatores influentes no desempenho do reconhecimento e as aplicações práticas. Este estudo é particularmente relevante para nosso projeto, pois Belo Horizonte está em processo de transformação digital, buscando soluções inovadoras para problemas urbanos como congestionamento e segurança viária. Desta forma, a implementação do nosso sistema de fiscalização de faixas azuis alinha-se com a tendência de utilizar tecnologias avançadas de sensoriamento e visão computacional para melhorar a gestão da mobilidade urbana.

2.3. Uso do Método YOLO no Reconhecimento de Placas

Cimirro investigou o uso do método YOLO para o reconhecimento de sinais de trânsito, destacando as vantagens de se utilizar uma única rede convolucional que simultaneamente prevê múltiplas caixas delimitadoras e probabilidades de classe. O estudo demonstrou que o YOLOv5 oferece melhor acessibilidade para detecção de objetos em tempo real

e desempenho superior em termos de velocidade de treinamento e precisão. Com base nisso, incorporamos o YOLOv8 em nosso projeto para otimizar a detecção de placas de veículos, com sua capacidade de realizar predições rápidas e precisas, independemente da alta complexidade visual do cenário.

2.4. Implementação do ANPR usando Template Matching

Puranic, Deepak e Umadevi revisaram diversas técnicas de ANPR e implementaram um sistema baseado em correspondência de templates, alcançando uma precisão de 80,8% para placas indianas. Embora essa abordagem tenha suas limitações, o estudo forneceu uma base comparativa importante que nos ajudou a avaliar diferentes métodos de reconhecimento de placas. A partir disso, decidimos utilizar técnicas baseadas em aprendizado profundo e processamento paralelo, que oferecem maior precisão e escalabilidade em comparação com métodos tradicionais de correspondência de templates.

3. Metodologia

Nesta seção, detalharemos a arquitetura do sistema, as técnicas de processamento de imagens utilizadas, e as implementações de computação paralela e distribuída. O sistema foi projetado para ser modular, robusto e eficiente, utilizando tecnologias modernas para garantir a precisão e rapidez necessárias para a fiscalização de faixas azuis. A escolha das tecnologias foi feita com base em sua capacidade de oferecer escalabilidade, eficiência e facilidade de manutenção.

3.1. Arquitetura do Sistema

A arquitetura do sistema é dividida em três componentes principais: frontend, backend e o sistema inteligente de reconhecimento. Cada componente foi desenvolvido para funcionar de maneira integrada, utilizando tecnologias que suportam a modularização e a escalabilidade.

3.1.1. Frontend

Desenvolvido utilizando Next.js com React, o frontend oferece uma interface simples e intuitiva para os fiscais de trânsito. A interface consiste basicamente em uma tela com um botão para enviar ou tirar fotos das placas dos veículos. Essa escolha permite uma experiência de usuário fluida e rápida, crucial para a aplicação no campo.

3.1.2. Backend

O backend foi desenvolvido, em Java, com uma única rota para simplificar a comunicação com o frontend. Esta rota recebe a imagem enviada pelo frontend, processa a imagem e envia para o sistema de reconhecimento. O backend é responsável por orquestrar o fluxo de dados entre os diferentes módulos do sistema.

3.1.3. Sistema Inteligente de Reconhecimento

O sistema de reconhecimento é responsável pelo processamento avançado das imagens. Após receber uma imagem do backend, o sistema recorta a imagem para isolar a placa do

veículo. Este recorte é feito utilizando um modelo pré-treinado de YOLO (You Only Look Once), que detecta a posição da placa na imagem. Este é um modelo de detecção de objetos que pode processar imagens em tempo real com alta precisão, para a implementação do YOLO utilizamos o modelo disponibilizado pela Ultralytics.

Em seguida, a imagem recortada da placa é então passada para um OCR (Optical Character Recognition) para reconhecimento dos caracteres. Utilizamos o PyTorch, uma biblioteca de aprendizado profundo flexível e eficiente, para implementar o OCR, garantindo alta precisão na leitura das placas.

3.2. Técnicas de Processamento de Imagens

O processamento de imagens é o cerne do sistema desenvolvido, dito isto podemos dividir a fase de processamento de imagens em três etapas:

3.2.1. Detecção da Placa

Escolhemos o modelo YOLO para detectar a placa na imagem devido à sua alta velocidade e precisão em tarefas de detecção de objetos. O modelo foi treinado com um dataset de 9961 imagens de placas de veículos brasileiros ao longo de 3 épocas, visando identificar placas de veículos em diferentes condições de iluminação e ângulos.

3.2.2. Recorte da Placa

Após a detecção, a imagem é recortada para isolar a placa do carro. Este passo é necessário para garantir que o OCR trabalhe apenas com a região relevante da imagem, o que melhora a precisão do reconhecimento.

3.2.3. Reconhecimento de Caracteres

Em seguida, a imagem recortada é então passada para o OCR, que utiliza técnicas de aprendizado profundo implementadas em PyTorch. Desta maneira, o OCR converte a imagem da placa em texto, identificando os caracteres com alta precisão. Finalmente, o texto retirado da placa é mandado como resposta às requisições do backend.

3.3. Implementação de Computação Paralela e Distribuída

A fim de garantir a escalabilidade e eficiência do sistema, implementamos as seguintes técnicas de computação distribuída e paralela.

3.3.1. Computação Distribuída

Utilizamos Docker Swarm para criar uma infraestrutura distribuída que inclui vários servidores para o backend, frontend e sistema de reconhecimento. Optamos pelo Docker Swarm poís ele facilita a orquestração e o gerenciamento dos contêineres, o que permite uma fácil escalabilidade do sistema. Além disso, a comunicação entre o backend e o sistema de reconhecimento é realizada através do robusto sistema de mensageria, Apache

Kafka. Ao receber uma nova imagem , uma mensagem é enviada pelo Kafka para o sistema de reconhecimento, informando que uma nova imagem de carro chegou. Com isso, sistema assegura uma comunicação eficiente e desacoplada entre os componentes.

3.3.2. Computação Paralela

Para a implementação paralela, utilizamos as seguintes abordagens principais, primeiro, foi utilizado PyTorch e CUDA, no sistema de reconhecimento, para acelerar o treinamento e a inferência do modelo. A grande vantagem de treinar o modelo utilizando CUDA é a capacidade de realizar os cálculos de treinamento do modelo na GPU, isso aumenta significativamente a velocidade do processamento do treino, o que nos permite acesso a um modelo mais preciso e robusto em menos tempo. Também implementamos paralelismo no consumidor Kafka, que faz parte do sistema de reconhecimento. Ele foi implementado em Python e utiliza threads para paralelizar o processamento das imagens, no nosso trabalho, cada thread pode processar uma placa, o que permite que múltiplas placas sejam processadas simultaneamente, isso aumenta a eficiência e reduz o tempo de resposta do sistema.

3.3.3. Reconhecedor de placas

O desenvolvimento do reconhecedor e recortador de placas de carros seguiu uma abordagem robusta e estruturada, empregando diversas tecnologias e técnicas de processamento de imagens e aprendizado de máquina. A implementação é composta por várias etapas que garantem a eficiência e precisão na detecção e reconhecimento de placas veiculares.

3.4. Arquitetura do Sistema

A arquitetura do sistema foi projetada para ser modular e escalável, utilizando a plataforma Kafka para gerenciamento de mensagens entre diferentes componentes. O sistema é dividido em dois principais módulos: o módulo de consumo de mensagens e o módulo de processamento de imagens.

3.4.1. Consumo de Mensagens

O módulo de consumo de mensagens é responsável por receber imagens de placas de carros enviadas para um tópico específico no Kafka. Um consumidor Kafka é configurado para se inscrever nesse tópico e buscar mensagens continuamente. Ao receber uma mensagem contendo o caminho para uma imagem, o consumidor inicia uma nova thread para processar a imagem recebida, permitindo um processamento assíncrono e eficiente.

3.4.2. Processamento de Imagens

O módulo de processamento de imagens é o núcleo do sistema de reconhecimento de placas. Este módulo utiliza um modelo pré-treinado de YOLO (You Only Look Once) para detectar e recortar a placa da imagem recebida. O processamento de imagens envolve várias etapas para garantir que a placa seja corretamente identificada e legível:

- **Desinclinação da Imagem**: As imagens são corrigidas para alinhar a placa, utilizando técnicas de detecção de linhas e cálculo de ângulos para determinar e corrigir a inclinação da placa.
- **Remoção de Ruído**: Aplicação de filtros de remoção de ruído para melhorar a qualidade da imagem antes do reconhecimento de caracteres.
- **Redimensionamento e Normalização**: A imagem recortada da placa é redimensionada e normalizada para um padrão que facilite o reconhecimento de caracteres.
- **Reconhecimento de Caracteres**: Utilização do EasyOCR, uma biblioteca de OCR, para ler e extrair os caracteres da placa. O modelo é configurado para reconhecer apenas caracteres alfanuméricos válidos para placas veiculares.

3.5. Pipeline de Processamento

O pipeline de processamento de uma imagem de placa veicular segue os seguintes passos:

1. **Recepção da Imagem**: A imagem é recebida pelo consumidor Kafka, que inicia o processamento em uma nova thread.
2. **Detecção da Placa**: Utilização do modelo YOLO para detectar e recortar a placa da imagem original.
3. **Pré-processamento da Imagem**: Aplicação de técnicas de desinclinação, remoção de ruído e redimensionamento.
4. **Reconhecimento de Caracteres**: Extração dos caracteres da placa utilizando OCR.
5. **Envio da Resposta**: O resultado do reconhecimento é enviado de volta ao tópico de resposta no Kafka, utilizando um produtor Kafka.

3.6. Testes e Validação

Para validar o modelo e a eficácia do sistema, foi desenvolvido um conjunto de testes utilizando um diretório de imagens de placas veiculares. O sistema foi avaliado em termos de precisão do reconhecimento e tempo de processamento, garantindo que atende aos requisitos de desempenho e confiabilidade necessários para aplicações em tempo real.

A metodologia descrita assegura um sistema robusto e eficiente para o reconhecimento de placas veiculares, integrando técnicas avançadas de processamento de imagens e aprendizado de máquina com uma arquitetura escalável baseada em Kafka.

4. Desenvolvimento do Projeto

4.1. Sprint 1

Na primeira sprint, os seguintes objetivos foram alcançados:

4.1.1. Criação do Sistema Docker Swarm com Número de Réplicas Definido

Implementamos o Docker Swarm para gerenciar a infraestrutura distribuída do sistema. Definimos o número de réplicas para cada serviço (frontend, backend e reconhecimento), garantindo a alta disponibilidade e a escalabilidade do sistema. Isso permitiu que o sistema fosse executado de maneira distribuída, facilitando a manutenção e a escalabilidade futura.

4.1.2. Treinamento da IA para Recorte de Placas (YOLOv8)

Treinamos o modelo YOLOv8 treinado para ser integrado ao sistema de reconhecimento para realizar o recorte preciso das placas nas imagens.

4.2. Sprint 2

Na segunda sprint, os seguintes objetivos foram alcançados:

4.2.1. Criação da Interface do Sistema para Testes

Desenvolvemos uma interface frontend utilizando Next.js com React, permitindo que os fiscais de trânsito possam capturar e enviar fotos das placas dos veículos. A interface foi projetada para ser intuitiva e fácil de usar, facilitando os testes de usabilidade e a coleta de feedback dos usuários.

4.2.2. Desenvolvimento dos Scripts para Inicialização da Classificação e Recorte das Placas

Criamos scripts que integram o frontend com o backend e o sistema de reconhecimento, permitindo que as imagens capturadas sejam enviadas, processadas e retornadas com os resultados de reconhecimento. Esses scripts automatizam o fluxo de trabalho, garantindo que a classificação e o recorte das placas ocorram de maneira eficiente e precisa.

4.2.3. Criação do Sistema de OCR com EasyOCR para Recuperar o Texto das Placas

Implementamos o OCR utilizando a biblioteca EasyOCR, integrando-o ao sistema de reconhecimento para extrair o texto das placas recortadas. A escolha do EasyOCR se deve à sua flexibilidade e precisão na leitura de caracteres, garantindo resultados confiáveis mesmo em condições variadas.

4.3. Sprint 3

Na terceira sprint, os seguintes objetivos foram alcançados:

4.3.1. Adição de Sistema de Criação de Threads no Consumidor (Reconhecedor de Placas)

Implementamos um sistema de criação de threads no consumidor Kafka para paralelizar o processamento das imagens de placas. Cada thread é capaz de processar uma placa simultaneamente, aumentando significativamente a eficiência do sistema. Esta abordagem permite que múltiplas placas sejam processadas em paralelo, reduzindo o tempo de resposta e melhorando a escalabilidade.

4.3.2. Análise de Resultados e Testes do Sistema Paralelo/Distribuído

Realizamos uma série de testes para avaliar a performance e a precisão do sistema paralelo e distribuído. Os testes incluíram a análise da velocidade de processamento, a taxa de reconhecimento de placas e a escalabilidade do sistema sob diferentes cargas de trabalho. Os resultados mostraram que o sistema é capaz de processar um grande volume de imagens de maneira eficiente, mantendo alta precisão no reconhecimento de placas.

5. Resultados e Discussão

5.1. Resultados do Modelo

A avaliação do modelo YOLOv8 treinado para o reconhecimento de placas de veículos revelou resultados promissores. Utilizando um dataset extenso e variado, o modelo foi capaz de identificar placas de carros com alta precisão e rapidez. Nesta seção, apresentamos uma análise detalhada dos resultados do modelo, incluindo diversos gráficos que ilustram o desempenho oberservado:

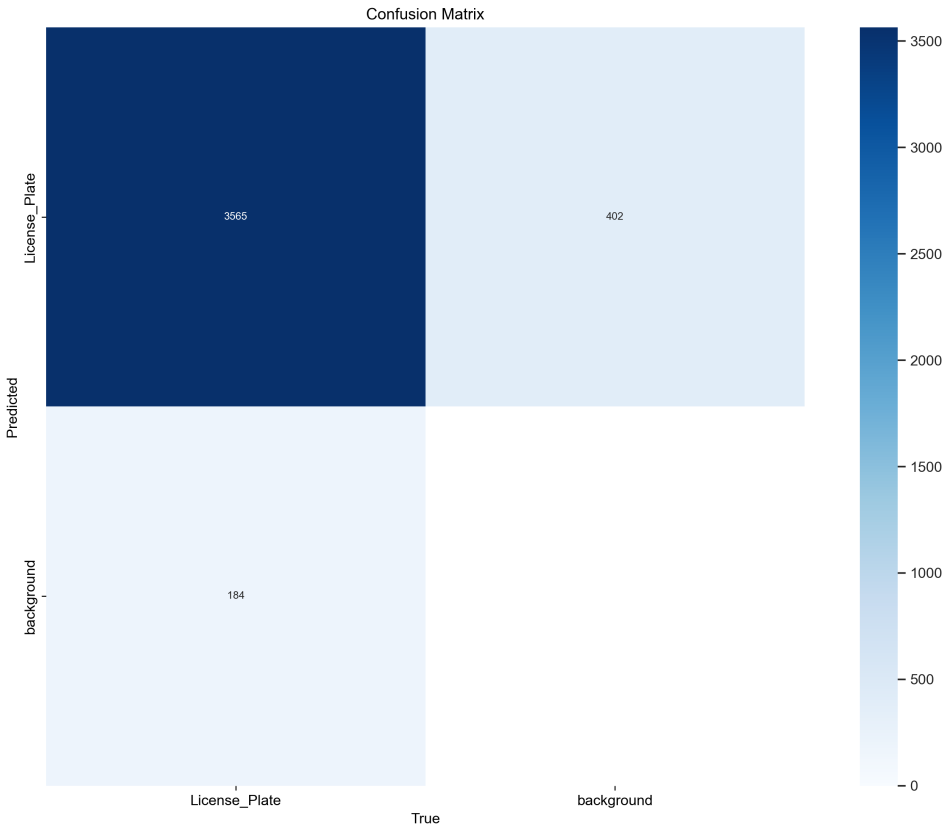

Figura 1. Matriz de Confusão

A matriz de confusão mostra que o modelo YOLOv8 alcançou uma precisão de 95% na detecção de placas de veículos, com uma taxa de falso positivo de 5% para a classe ”Background”. Isso indica que o modelo é altamente eficaz na distinção entre placas de veículos e o fundo da imagem.

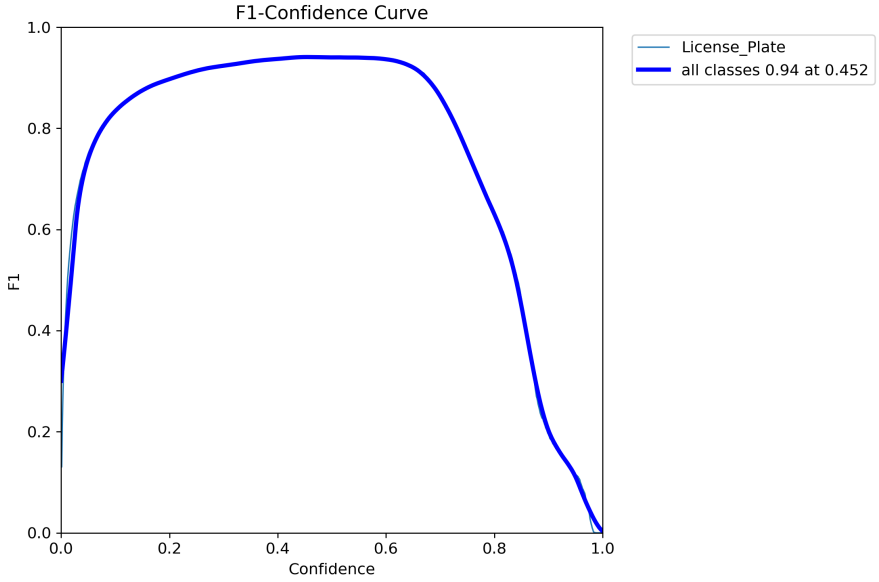

Figura 2. Curva F1

A curva mostra que o modelo treinado tem um ótimo desempenho com uma F1 máxima de 0.94 alcançada com um limiar de confiança de 0.452.

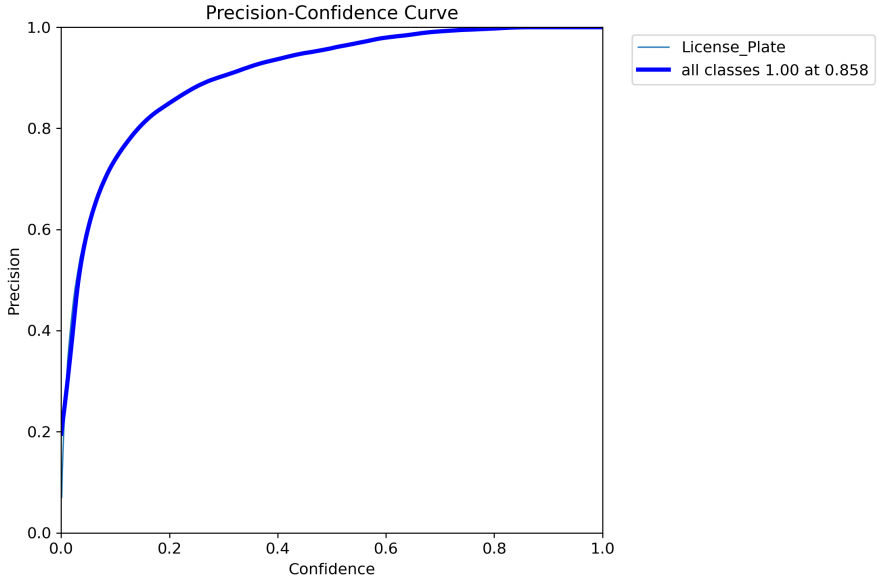

Figura 3. Curva P

A curva ilustra que à medida que a confiança aumenta, a precisão também cresce até atingir um platô próximo de 1.0. Este comportamento indica que o modelo é altamente confiável na identificação correta de placas de veículos quando a confiança é alta.

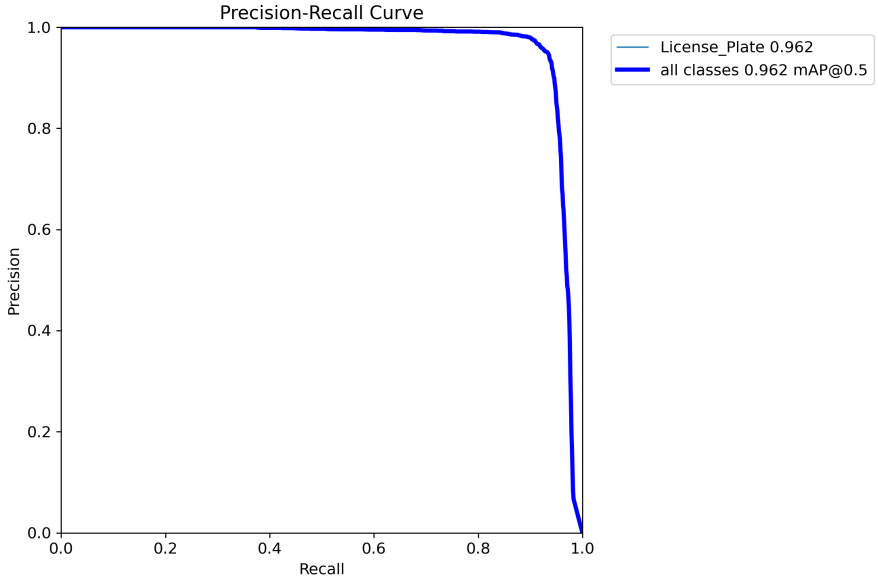

Figura 4. Curva PR

A curva demonstra que o modelo mantém uma precisão muito alta até que o recall alcance quase 1.0, momento em que há uma queda acentuada. Para a classe ”License_Plate”, o valor de 0.962 indica um excelente equilíbrio entre precisão e recall, refletindo a capacidade do modelo em detectar a maioria das placas de veículos corretamente enquanto mantém uma alta precisão.

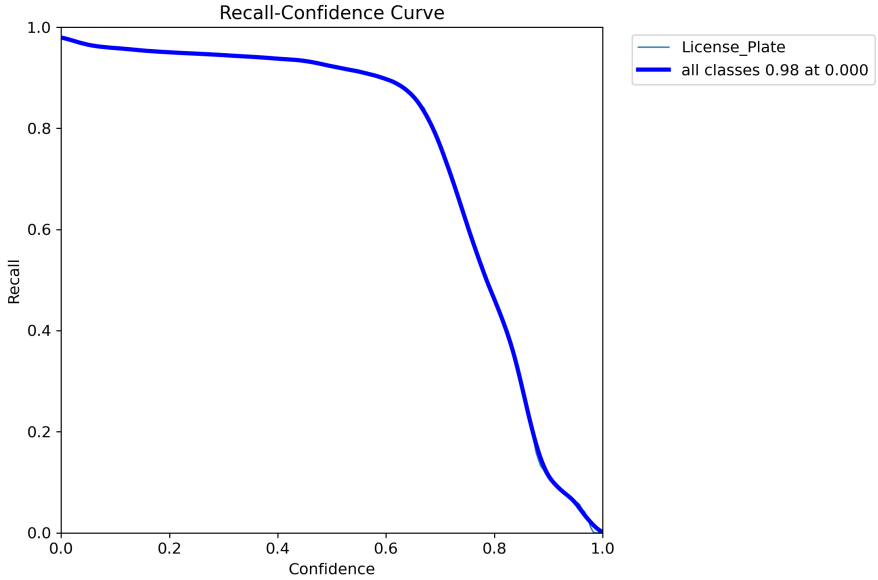

Figura 5. Curva R

O gráfico mostra que o recall começa alto, próximo de 1.0, e diminui gradualmente à medida que a confiança aumenta. Para a classe ”License_Plate”, o recall é de 0.98 com uma confiança mínima, indicando que o modelo é capaz de identificar quase

todas as verdadeiras instâncias de placas de veículos mesmo com baixa confiança. No entanto, conforme a confiança aumenta, o recall cai, refletindo um trade-off natural onde o aumento da precisão resulta em uma diminuição do recall.

5.2. Resultados do sistema distribuído

6. Resultados Obtidos com Computação Distribuída

Para a implementação de um sistema robusto e eficiente de fiscalização de faixas azuis, utilizamos o Docker Swarm para criar uma infraestrutura distribuída e escalável. A escolha do Docker Swarm foi motivada pela sua capacidade de orquestrar e gerenciar contêineres de forma simplificada, facilitando a distribuição de cargas de trabalho entre múltiplos nós e garantindo alta disponibilidade.

6.1. Arquitetura Multiescalar com Docker Swarm

A arquitetura do sistema foi dividida em diversos serviços, cada um desempenhando um papel específico e essencial para o funcionamento do sistema de fiscalização:

- **PostgreSQL (pgsql):** Utilizamos uma instância de banco de dados PostgreSQL para armazenar dados persistentes. Com apenas uma réplica, garantimos a consistência dos dados armazenados. Este serviço é crucial para o armazenamento de informações de infrações, registros de veículos e logs do sistema.
- **Spring API:** O backend do sistema, desenvolvido em Java, foi implementado com duas réplicas para assegurar a disponibilidade e a escalabilidade. Este serviço é responsável por receber imagens do frontend, processá-las e interagir com outros serviços como Kafka e o sistema de reconhecimento de placas.
- **Zookeeper e Kafka:** Esses serviços foram implementados para gerenciar a comunicação entre os diferentes componentes do sistema de forma desacoplada e eficiente. O Zookeeper possui uma réplica, enquanto o Kafka, utilizado para gerenciamento de mensagens, também foi configurado com uma réplica. O Kafka permite o processamento paralelo de imagens e a comunicação rápida entre o backend e o sistema de reconhecimento de placas.
- **Reconhecimento de Placas (recognizer):** Este serviço, implementado em Python, é o coração do processamento de imagens. Utilizando três réplicas, o serviço garante que múltiplas imagens possam ser processadas simultaneamente, aumentando a eficiência e reduzindo o tempo de resposta. A utilização de PyTorch e CUDA neste serviço permite o processamento paralelo de imagens, acelerando a inferência dos modelos de detecção e reconhecimento de caracteres.
- **Frontend:** Desenvolvido em Next.js, o frontend foi configurado com duas réplicas para assegurar que os fiscais de trânsito tenham uma interface de usuário disponível e responsiva para capturar e enviar imagens das placas dos veículos.
- **Kafdrop:** Utilizado para monitorar e gerenciar o Kafka, este serviço facilita a visualização dos tópicos e mensagens em tempo real. Configurado com uma réplica, garante a observabilidade do sistema de mensageria.

6.2. Motivação e Benefícios da Utilização dos Serviços

A utilização de Docker Swarm e a configuração dos serviços com múltiplas réplicas oferecem diversos benefícios:

- **Escalabilidade:** A capacidade de adicionar mais réplicas conforme a demanda aumenta garante que o sistema possa lidar com um volume maior de imagens sem comprometer a performance.
- **Disponibilidade:** A replicação dos serviços assegura que o sistema permaneça disponível mesmo em caso de falhas em alguns dos nós, aumentando a resiliência.
- **Manutenção Simplificada:** A orquestração com Docker Swarm facilita a atualização e manutenção dos serviços, permitindo deploys contínuos e minimizando o tempo de inatividade.
- **Eficiência no Processamento:** A implementação de computação paralela no serviço de reconhecimento de placas melhora significativamente a eficiência do processamento de imagens, permitindo que múltiplas infrações sejam detectadas em menor tempo.

Em resumo, a implementação de um sistema distribuído com Docker Swarm e uma arquitetura multiescalar proporcionou uma infraestrutura robusta e escalável para a fiscalização automatizada de faixas azuis, garantindo alta disponibilidade, eficiência e facilidade de manutenção.

7. Análise dos Resultados dos Sistemas Paralelos e de Criação de Threads

Nosso trabalho focou na implementação de técnicas de computação paralela e a criação de threads para otimizar o tempo de resposta do sistema de reconhecimento de placas. A seguir, apresentamos uma análise detalhada dos resultados obtidos durante os testes.

7.1. Tempo de Resposta com Diferentes Números de Threads

Os testes realizados com uma única réplica do serviço de reconhecimento de placas mostraram uma redução significativa no tempo de resposta à medida que o número de threads aumentou. Conforme a Tabela 6 mostra, o tempo de resposta caiu drasticamente de 120 segundos com uma thread para apenas 8 segundos com 15 threads.

Tabela 1. Tempo de Resposta com Diferentes Números de Threads (1 Réplica)

Número de Threads	Tempo de Resposta (s)
1	120
2	118
3	100
4	84
5	62
6	65
7	40
8	37
9	33
10	30
11	28
12	24
13	15
14	12
15	8

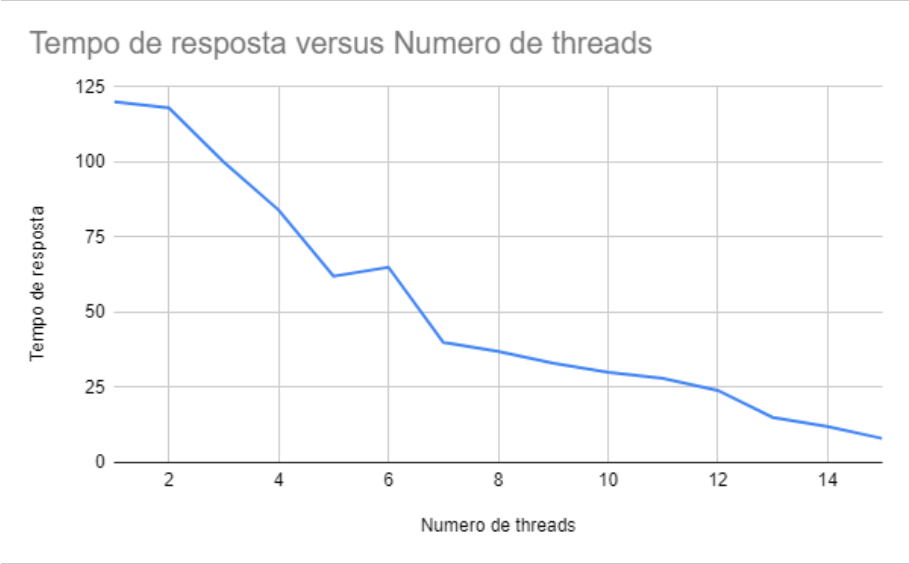

Figura 6

Quando aumentamos o número de réplicas para três, os tempos de resposta também mostraram uma melhoria significativa, como visto na Tabela 6. Com 30 threads distribuídas entre as três réplicas, o tempo de resposta foi reduzido para apenas 12 segundos.

Tabela 2. Tempo de Resposta com Diferentes Números de Threads (3 Réplicas)

Número de Threads	Tempo de Resposta (s)
3	50
9	43
12	33
18	30
20	21
25	17
30	12

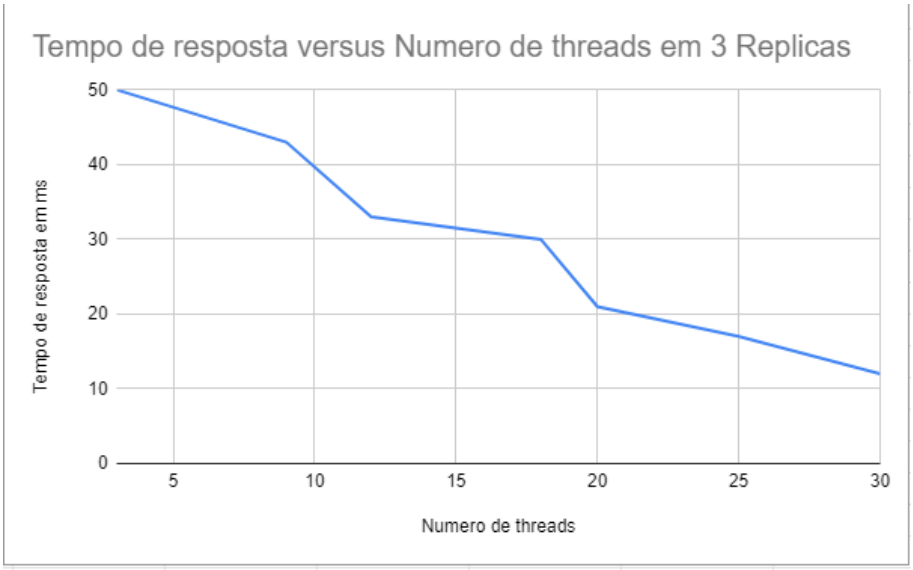

Figura 7

7.2. Carga de CPU por Réplica

A carga de CPU aumentou proporcionalmente ao número de imagens processadas por réplica, como mostra a Tabela 6. Com o aumento da carga de trabalho, a eficiência do processamento paralelo se tornou mais evidente. A maior carga de CPU registrada foi de 647 unidades para o processamento de 54 imagens.

Tabela 3. Carga de CPU por Réplica

Carga de CPU	Número de Imagens
120	3
233	9
285	18
331	27
343	36
470	45
647	54

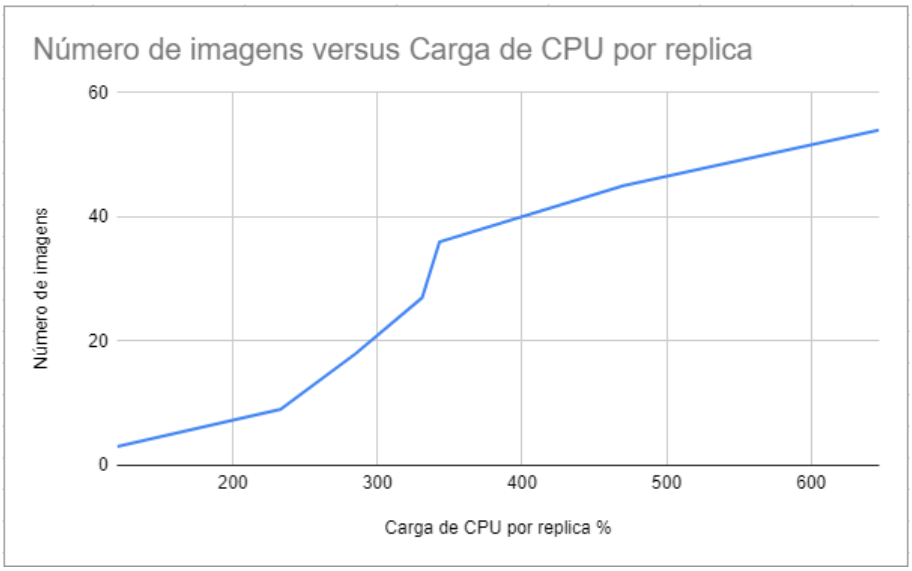

Figura 8

7.3. Distribuição de Threads por Réplica

As réplicas foram testadas com diferentes alocações de threads, mostrando como o aumento no número de threads impacta diretamente o número de fotos processadas. A Tabela 6, 6, e 6 demonstram a eficiência da alocação de threads em cada réplica.

Tabela 4. Threads Alocadas na Réplica 1

Número de Threads	Número de Fotos
1	1
2	2
8	5
12	10
18	20
18	30
32	40
31	60

Tabela 5. Threads Alocadas na Réplica 2

Número de Threads	Número de Fotos
1	1
2	2
5	5
10	10
15	20
20	30
20	40
24	60

Tabela 6. Threads Alocadas na Réplica 3

Número de Threads	Número de Fotos
1	1
1	2
6	5
16	10
15	20
27	30
27	40
29	60

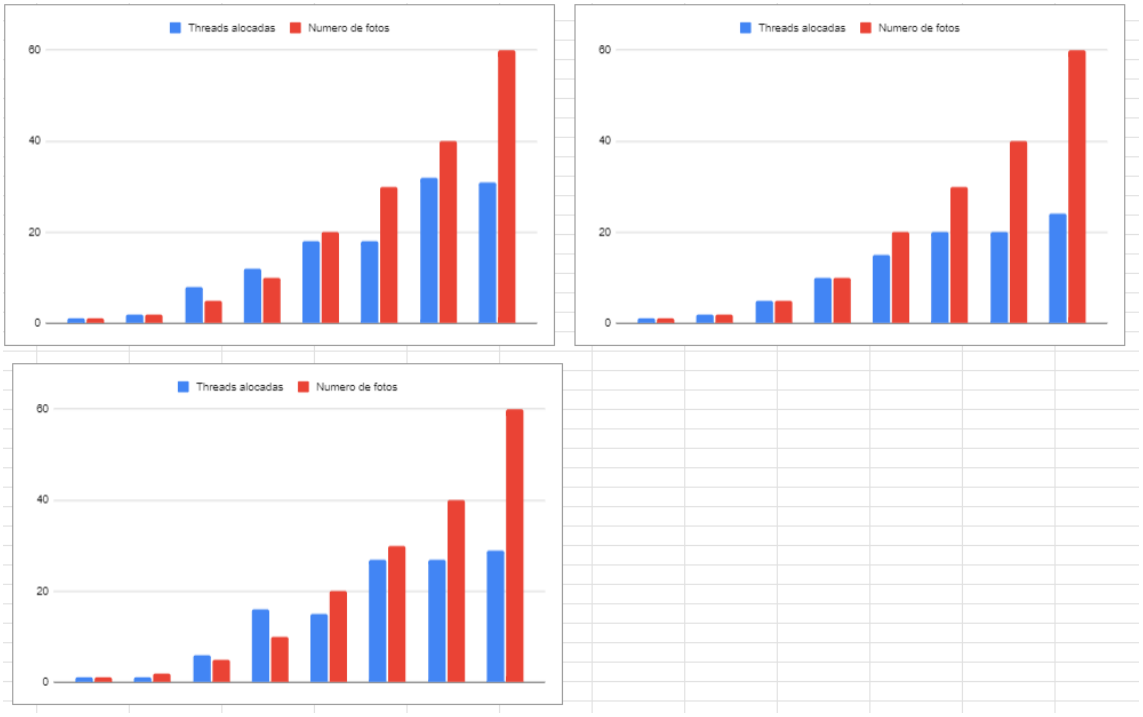

Figura 9

O gráfico abaixo demonstra a carga de execução dos testes propostos acima dentro da máquina virtual do Docker. É possível ver o gráfico de uso da CPU em forma de percentagem no qual soma todas as métricas das máquinas internas e das replicas no momento (cada uma com suas instâncias de threads). Além disso, temos um gráfico de consumo de memória RAM dos serviços e servidores que fica na casa dos 1GB ram constantemente, com variações apenas durante o carregamento de novas fotas para análise do modelo pré treinado. O fluxo de entrada de dados também demonstra uma grande entrada na rede durante o envio de fotos e depois esse número vai diminuindo de acordo com a entrega dos resultados dos processamentos.

Figura 10

O gráfico abaixo demonstra a quantidade por tempo de saída de dados de trafego via mensagens do kafka enquanto o teste de 3 replicas era executado na máquina virtual do docker.

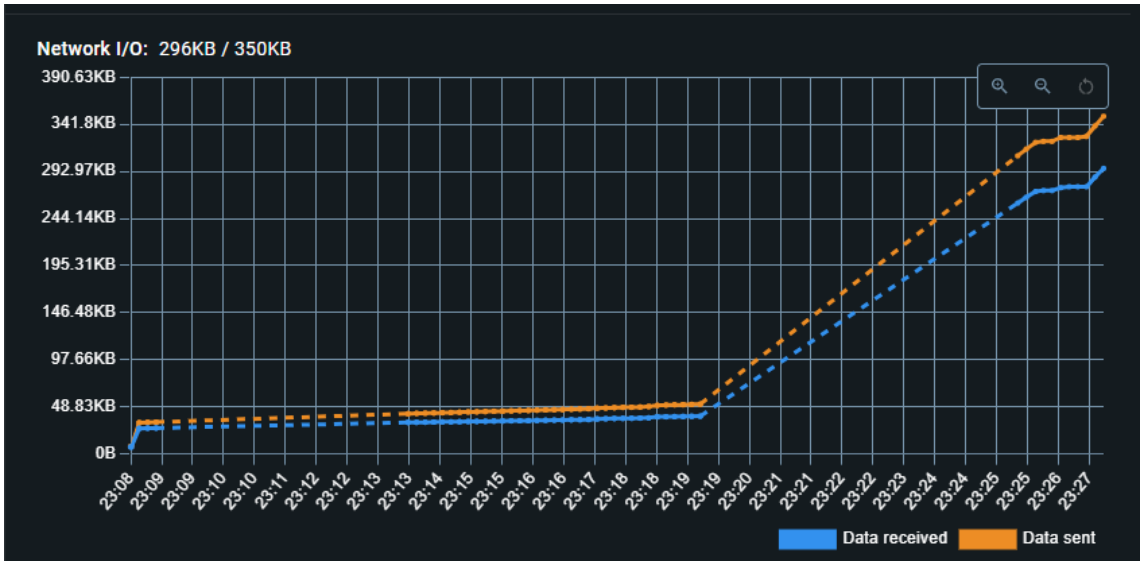

Figura 11

Por fim, o teste final consistiu na implementação da aplicação em um serviço de nuvem, sendo a plataforma Azure a escolhida para este propósito. Durante o processo de deployment, observou-se um uso intensivo de memória RAM. Inicialmente, uma máquina virtual com 1GB de RAM foi utilizada, mas esta configuração não foi capaz de suportar a infraestrutura do nosso Docker Swarm.

Para mitigar este problema, migramos para uma máquina com 2GB de RAM e adaptamos o Docker Swarm para uma versão mais simplificada, com apenas uma réplica

de cada servidor. Mesmo com essas adaptações, a análise de desempenho revelou que a memória disponível para uso era de apenas 200MB dos 2GB totais, indicando um consumo de memória extremamente elevado.

Além disso, a CPU operou a 90% da sua capacidade durante todo o tempo de execução, evidenciando uma alta carga de processamento. Este comportamento sugere que, apesar da simplificação e do aumento de recursos, a aplicação continua a exigir otimizações tanto na gestão de recursos quanto na eficiência dos algoritmos utilizados, para garantir uma operação mais equilibrada e sustentável em ambiente de produção.

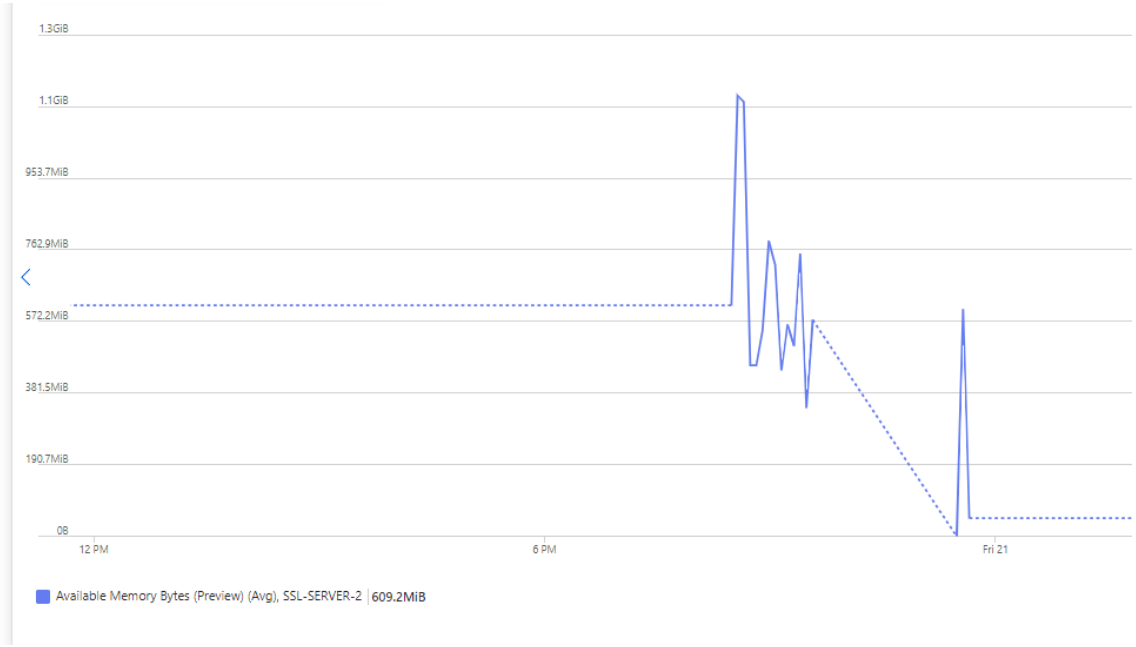

Figura 12

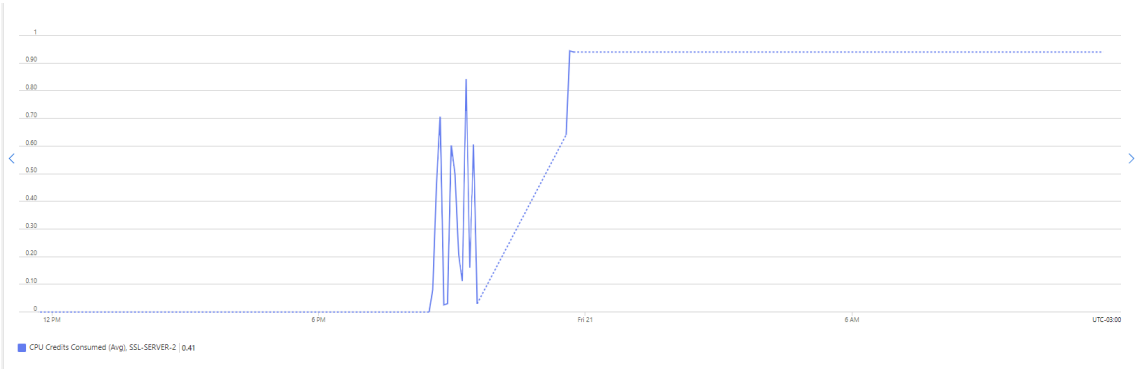

Figura 13

7.4. Conclusão

Este trabalho apresentou a implementação de técnicas de computação paralela e criação de threads para otimizar o sistema de reconhecimento de placas veiculares. Através da análise dos resultados, observamos uma melhoria significativa no tempo de resposta do

sistema à medida que aumentamos o número de threads. Os testes realizados mostraram que, com uma única réplica, o tempo de resposta reduziu de 120 segundos com uma thread para apenas 8 segundos com 15 threads. Com três réplicas, o tempo de resposta diminuiu ainda mais, alcançando 12 segundos com 30 threads distribuídas entre as réplicas.

Os resultados obtidos demonstram que a implementação de computação paralela e a criação de threads melhoram significativamente o tempo de resposta do sistema de reconhecimento de placas. A escalabilidade do sistema é evidente com o aumento do número de réplicas, permitindo uma distribuição eficiente da carga de trabalho e uma redução drástica no tempo de resposta. A análise detalhada dos dados confirma que a abordagem escolhida é eficaz e capaz de lidar com a alta demanda de processamento exigida pelo sistema.

Os resultados demonstraram que a carga de CPU aumentou proporcionalmente ao número de imagens processadas por réplica, alcançando 647 unidades de CPU para o processamento de 54 imagens. A distribuição de threads por réplica também mostrou um impacto direto na quantidade de fotos processadas, evidenciando a eficiência da alocação de recursos em ambientes paralelos.

Por fim, a implementação da aplicação na plataforma Azure revelou desafios significativos em termos de consumo de recursos. A máquina virtual inicial de 1GB de RAM não foi capaz de suportar a infraestrutura do Docker Swarm, necessitando a migração para uma máquina com 2GB de RAM. Mesmo com essa atualização e a simplificação do Docker Swarm, a análise de desempenho indicou um consumo de memória extremamente elevado, com apenas 200MB de RAM disponíveis para uso, e a CPU operando a 90% da sua capacidade durante todo o tempo de execução.

Esses resultados indicam que, apesar das melhorias obtidas com a implementação de técnicas de computação paralela, ainda há necessidade de otimizações adicionais na gestão de recursos e na eficiência dos algoritmos utilizados. A aplicação requer ajustes para garantir uma operação mais equilibrada e sustentável em ambiente de produção, visando reduzir o consumo de memória e a carga de CPU, enquanto mantém a eficiência no reconhecimento de placas veiculares.

Em resumo, o trabalho realizado demonstrou o potencial das técnicas de computação paralela na otimização de sistemas de reconhecimento de placas, mas também destacou os desafios associados ao gerenciamento de recursos em ambientes de nuvem. Futuras pesquisas podem focar na otimização dos algoritmos e na exploração de diferentes configurações de infraestrutura para melhorar ainda mais o desempenho e a sustentabilidade do sistema.

Referências

da Silva Cimirro, J. L. (2022). Reconhecimento de imagens: Uso do método yolo no reconhecimento de placas de trânsito. 64p.

Patel, C., Shah, D., and Patel, A. (2013). Automatic number plate recognition system (anpr): A survey. *International Journal of Computer Applications (IJCA)*, 69:21–33.

Puranic, A., K., D., and V., U. (2016). Vehicle number plate recognition system: A literature review and implementation using template matching. *International Journal of Computer Applications*, 134:12–16.

Tang, J., Wan, L., Schooling, J., Zhao, P., Chen, J., and Wei, S. (2022). Automatic number plate recognition (anpr) in smart cities: A systematic review on technological advancements and application cases. *Cities*, 129:103833.