UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO
ENGENHARIA DE COMPUTAÇÃO

JOÃO VITOR ALMEIDA DE OLIVEIRA

DESENVOLVIMENTO DE UM SOFTWARE IDENTIFICADOR DE INCÊNDIOS UTILIZANDO UM MODELO DE MACHINE LEARNING

GOIÂNIA - GOIÁS
2023

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): João Vitor Almeida de Oliveira

Título do trabalho: Desenvolvimento de um software identificador de incêndios utilizando um modelo de Machine Learning

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento

[X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Carlos Galvão Pinheiro Júnior**, **Vice-Diretor**, em 15/08/2023, às 12:22, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro

de 2020.

Documento assinado eletronicamente por **João Vitor Almeida De Oliveira**, **Discente**, em 17/08/2023, às 21:14, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **3964534** e o código CRC **52AFA017**.

Referência: Processo nº 23070.024199/2023-38 SEI nº 3964534

JOÃO VITOR ALMEIDA DE OLIVEIRA

DESENVOLVIMENTO DE UM SOFTWARE IDENTIFICADOR DE INCÊNDIOS UTILIZANDO UM MODELO DE MACHINE LEARNING

Trabalho apresentado como requisito para conclusão do Curso de Graduação em Engenharia de Computação pela Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás.

Orientador: Prof. Dr. Carlos Galvão Pinheiro Júnior.

GOIÂNIA - GOIÁS
2023

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Almeida de Oliveira, João Vitor
Desenvolvimento de um software identificador de incêndios utilizando um modelo de Machine Learning [manuscrito] / João Vitor Almeida de Oliveira. - 2023.
XIX, 19 f.

Orientador: Prof. Dr. Carlos Galvão Pinheiro Júnior.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia da Computação, Goiânia, 2023.
Bibliografia.

1. Machine Learning. 2. Sistema. 3. Gerenciamento. 4. Combate à incêndio. 5. Tempo real. I. Galvão Pinheiro Júnior, Carlos, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

ATA DE AVALIAÇÃO DE PROJETO FINAL

Curso

( ) Eng Elétrica	( ) Eng Mecânica	( ) Eng Computação PFC 1 ( ) PFC 2 (X)

Título do Trabalho

Desenvolvimento de um software identificador de incêndios utilizando um modelo de Machine Learning

Banca Avaliadora

Membro 1	Carlos Galvão Pinheiro Júnior
Membro 2	Cássio Dener Noronha Vinhal
Membro 3	Gélson da Cruz Júnior

Discente

Matrícula	**Nome**
201611867	João Vitor Almeida de Oliveira

NOTAS										
	Membro 1			**Membro 2**			**Membro 3**			
Matrícula	**NPT**	**NTE**	**NAA**	**NPT**	**NTE**	**NAA**	**NPT**	**NTE**	**NAA**	**Média***
201611867	10	8,5	9	10	8	9	10	8	9	8,7

NPT – Nota plano de trabalho;
NTE – Nota do trabalho escrito;
NAA – Nota de apresentação e arguição
Para Eng. Elétrica, Mecânica e PFC2 da Eng. Da Computação: NF = 0,1 x NPT + 0,45xNTE + 0,45 x NAA
Para PFC1 da Eng. Da Computação: NF = 0,3 x NPT + 0,7 x NAA

* A APROVAÇÃO DO(S) ALUNO(S) ESTÁ CONDICIONADA À APRESENTAÇÃO DO TRABALHO FINAL AO ORIENTADOR COM TODAS AS CORREÇÕES SUGERIDAS PELA BANCA.

OBSERVAÇÕES:

Preencher com modificações solicitadas, caso existam. Em caso de reprovação, informar a justificativa.

Documento assinado eletronicamente por **Carlos Galvão Pinheiro Júnior**, **Vice-Diretor**, em 14/08/2023, às 18:02, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Cassio Dener Noronha Vinhal**, **Professor do Magistério Superior**, em 14/08/2023, às 21:19, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Gelson Da Cruz Junior**, **Professor do Magistério Superior**, em 16/08/2023, às 14:48, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **3962054** e o código CRC **6CB5A668**.

Referência: Processo nº 23070.024199/2023-38 SEI nº 3962054

Desenvolvimento de um *software* identificador de incêndios utilizando um modelo de *Machine Learning*

João Vitor Almeida de Oliveira[1]. Universidade Federal de Goiás (UFG) - Escola de Engenharia Elétrica, Mecânica e de Computação (EMC) - Goiânia, Goiás, Brasil 74601-010, e-mail: joao.almeida200@discente.ufg.br[1]

***Resumo* — O presente artigo visa apresentar um sistema de monitoramento em tempo real para detectar incêndios em estabelecimentos e nas ruas da cidade. Esse sistema faz uso de técnicas de *Machine Learning*, mais especificamente o YOLOv5, para detecção de fogo em imagens advindas de câmeras IP via protocolo RTSP. Para o treinamento foi criada uma base de dados com anotações especificamente para esse projeto. O treinamento foi realizado em uma máquina local e o resultado foi um modelo com uma mAP50 de 0,577 para o conjunto de validação e 0,413 para o conjunto de teste. No conjunto de teste foi obtida uma precisão de 0,514 e sensibilidade de 0,448, o que faz com que possua muitas ocorrências de falsos positivos e falsos negativos nas detecções. Por fim, o sistema de gerenciamento desenvolvido – servidor, aplicação web e estrutura do banco de dados – conta com funcionalidades como cadastro de usuários, autenticação, gerenciamento de câmeras IP, visualização das imagens da câmera e a resposta em tempo real para detecção de fogo.**

***Palavras-chaves* — machine learning; sistema; gerenciamento; incêndio; combate a incêndios; tempo real.**

***Abstract* — This article aims to present a real-time monitoring system to detect fires in establishments and on city streets. This system makes use of Machine Learning techniques, more specifically YOLOv5, for fire detection in images coming from IP cameras via RTSP protocol. For the training, a database was created with annotations specifically for this project. Training was performed on a local machine and the result was a model with a mAP50 of 0.577 for the validation set and 0.413 for the test set. In the test set, a precision of 0.514 and a sensitivity of 0.448 were obtained, which means that it has many occurrences of false positives and false negatives in the detections. Finally, the developed management system – server, web application and database structure – has features such as user registration, authentication, IP camera management, camera image viewing and real-time response to fire detection.**

***Keywords* — machine learning; software; management; fire; firefighting; real time.**

I. INTRODUÇÃO

Incêndio é qualquer fogo fora de controle. Os incêndios representam uma ameaça à segurança pública e ao meio ambiente, causando devastação, perda de vidas e danos materiais. Dessa forma, o combate a incêndios é imprescindível para a segurança de forma geral. A tecnologia tem um papel importante nessa área, pois são muitos os esforços para criar alternativas para combater incêndios. Dentre as alternativas estão o uso de drones para extinguir incêndios [1] e até sensores inteligentes que enviam alertas [2].

Assim sendo, o presente artigo tem como objetivo o desenvolvimento de um sistema de gerenciamento, utilizando um modelo YOLOv5 para realizar as detecções de incêndios em tempo real. Dessa forma, é possível entender o uso da solução proposta como uma alternativa promissora para melhorar a segurança geral das ruas da cidade e dos estabelecimentos. A abordagem mostra esse potencial, realizando a integração do sistema de gerenciamento – desenvolvido em Python e *React* – com um modelo de *Machine Learning*.

A próxima seção refere-se à fundamentação teórica e apresenta conceitos de *Machine Learning* e do algoritmo YOLOv5. Ainda, as métricas referentes ao algoritmo são abordadas para serem utilizadas nas discussões das seções seguintes. Em seguida, a metodologia é apresentada, falando sobre o desenvolvimento do sistema. Essa seção expõe o processo para preparação do banco de imagens, assim como fala sobre o treinamento do modelo e o desenvolvimento do sistema de gerenciamento. Por fim, os resultados acerca do modelo e do sistema de gerenciamento são retratados na seção subsequente.

II. FUNDAMENTAÇÃO TEÓRICA

A. *Machine Learning*

Na computação é comum a escrita de programas para realizar as mais diversas tarefas por meio de algoritmos. Dentre essas tarefas, existem aquelas que são soluções para problemas reais complexos e que costumam requerer o auxílio de especialistas de um dado domínio. No entanto, o processo de obtenção de conhecimento para resolver esses problemas possui várias limitações, como a subjetividade advinda do uso da intuição do especialista para a tomada de decisão. Assim sendo, vê-se a necessidade de ferramentas computacionais mais sofisticadas e autônomas, a fim de reduzir a intervenção humana e a dependência de especialistas. Dessa forma, essas ferramentas deveriam ser capazes de criar uma hipótese, ou função, com base na experiência passada e é esse processo de indução que origina o nome Aprendizado de Máquina (AM, *Machine Learning* ou ML) [3].

Em *Machine Learning*, computadores são programados para aprender com a experiência passada, como por meio da aplicação do princípio de inferência – também conhecido como indução – no qual conclusões gerais são extraídas de um

conjunto particular de exemplos. Em suma, algoritmos de *Machine Learning* aprendem a generalizar hipóteses ou funções capazes de resolver um problema a partir de dados que representam instâncias do problema que está sendo resolvido [3].

1) Indução de Hipóteses

A relação entre *Machine Learning* e indução de hipóteses pode ser elucidada imaginando um conjunto de dados composto de pacientes de um hospital. Em cada dado ou objeto desse conjunto, existem características ou atributos referentes a um único paciente, que o discriminam com base em suas informações, como identificação, nome, idade, sexo, estado de origem, sintomas e resultados de exames clínicos. Um algoritmo de *Machine Learning* tem como objetivo aprender, a partir de um subconjunto dos dados (ou conjunto de treinamento), um modelo ou hipótese capaz de relacionar os valores dos atributos de entrada de um dado do conjunto de treinamento ao valor de seu atributo de saída – chamado de atributo alvo ou atributo meta – determinado previamente como sendo um de seus atributos [3].

Suponha-se que o objetivo do algoritmo de *Machine Learning* para o conjunto de dados do hospital seja diagnosticar um paciente, relacionando atributos de entrada com os resultados de exames clínicos. Alguns atributos, como identificação e nome não são entradas relevantes para determinar o diagnóstico de uma doença, pois não possuem relação alguma. A finalidade do algoritmo é inferir uma hipótese capaz de fazer diagnósticos corretos para novos pacientes diferentes daqueles que foram utilizados no conjunto de treinamento para aprender a regra de decisão. Essa propriedade de uma hipótese continuar a ser válida para novos dados é conhecida como capacidade de generalização da hipótese. Ainda, uma hipótese com boa capacidade de generalização é considerada útil em suas aplicações. Por fim, se a capacidade de generalização da hipótese for baixa, a razão pode ser que ela está superajustada aos dados (*overfitting*). Também pode ser dito que a hipótese memorizou ou se especializou no conjunto de dados de treinamento. O caso inverso também é possível, em que o algoritmo induz hipóteses com baixa taxa de acerto no subconjunto de treinamento, configurando uma condição de subajustamento (*underfitting*) [3].

2) Viés Indutivo

O resultado do aprendizado de um algoritmo de *Machine Learning* a partir de um conjunto de dados de treinamento será uma hipótese, encontrada dentro do espaço de possíveis hipóteses, descrevendo as relações entre os objetos e que melhor se adequa aos dados de treinamento. A representação da hipótese varia com o algoritmo. Para exemplificar, redes neurais artificiais representam uma hipótese por um conjunto de valores reais, que se associam aos pesos das conexões da rede. Por outro lado, árvores de decisão se baseiam em associações de cada nó interno a uma pergunta que se refere ao valor de um atributo e de cada nó externo a uma classe. Cada representação define sua própria preferência ou viés (*bias*) de representação do algoritmo, limitando o conjunto de hipóteses induzidas pelo algoritmo [3].

Algoritmos de *Machine Learning* também possuem um viés de busca. Esse viés pode ser definido como a busca de uma hipótese que melhor se ajusta ao conjunto de dados de treinamento, definindo como o espaço de hipóteses será delimitado. Em suma, cada algoritmo de *Machine Learning* possui um viés de representação e um de busca, permitindo o aprendizado e a generalização de hipóteses [3]. Segundo Mitchell, um algoritmo sem viés não é capaz de generalizar o conhecimento obtido na fase de treinamento e sua aplicação a novos dados não teria utilidade [4].

3) Tarefas de Aprendizado

Existem diversas tarefas em que um algoritmo de *Machine Learning* pode atuar. Essas tarefas podem ser subdivididas de acordo com diferentes critérios. Um critério a ser levado em consideração é o paradigma de aprendizado a ser utilizado para resolver uma determinada tarefa. Com base nesse critério, as tarefas de aprendizado podem ser segmentadas em preditivas e descritivas [3].

Tarefas preditivas têm como meta encontrar uma hipótese, a partir dos dados de treinamento, que possa ser aplicada a um novo dado para obter uma previsão de saída ou valor que o discrimine com base nos atributos de entrada. Dessa forma, cada objeto do conjunto de treinamento deve possuir atributos de entrada e saída. Os modelos resultantes seguem o paradigma de aprendizado supervisionado – a saída desejada é conhecida, portanto é dito que na fase de aprendizado existe um "supervisor externo" [3].

Já as tarefas descritivas possuem o objetivo de explorar ou descrever um conjunto de dados. O atributo de saída não se faz necessário para esses algoritmos. Dessa forma, como a saída é desconhecida, seguem o paradigma de aprendizado não supervisionado. Suas aplicações são variadas, como realizar o agrupamento de objetos semelhantes em um conjunto de dados e até mesmo encontrar regras de associação para relacionar um grupo de atributos a outro grupo de atributos [3].

Na Fig. 1 demonstra-se uma hierarquia de acordo com os tipos de tarefas de aprendizado apresentadas. No topo da hierarquia há o aprendizado indutivo, subdividido nos tipos de aprendizado com supervisão e sem supervisão. As tarefas dos nós subsequentes são exemplos de suas aplicações [3].

Fig. 1. Hierarquia de aprendizado [3].

B. Algoritmo YOLOv5

YOLO é a abreviação para *You Only Look Once*. Foi o primeiro algoritmo de detecção de objeto de estágio único proposto por Joseph Redmon. O que possibilita ter apenas um estágio é a unificação da etapa de extração de possíveis caixas delimitadoras (*bounding boxes*) – caixas mínimas que englobam o objeto em questão – com a etapa de classificação em um problema de regressão. Em outras palavras, o objetivo

desse algoritmo é "olhar" para a entrada (uma imagem) apenas uma vez, obtendo uma maior velocidade de processamento quando comparada a outros algoritmos de detecção com dois estágios ou mais [5].

O primeiro estágio do processo é dividir a imagem em uma malha S × S. Cada célula dessa malha tem a meta de fazer a predição uma classe e de uma quantidade B de caixas delimitadoras, levando a um total de S × S × B. Em cada caixa delimitadora existem cinco parâmetros: coordenadas do ponto central do alvo (x e y), largura do alvo (w e h) e a confiança de ter o alvo contido. Ainda, a confiança da caixa delimitadora é multiplicada com a predição da classe para obter uma pontuação final, determinando a probabilidade de a caixa delimitadora conter um objeto específico [5]. Por fim, as caixas delimitadoras são filtradas por um processo de *non-maximum suppression* (NMS), responsável por reduzir a redundância nas detecções, eliminando múltiplas detecções sobrepostas, mantendo as detecções mais relevantes e resultando nas previsões finais [6]. A Fig. 2 exemplifica esse processo visualmente.

Fig. 2. Exemplo de aplicação de NMS [7].

Em junho de 2020 o YOLOv5 foi lançado por Glenn Jocher. Até os dias atuais ainda é considerado um dos modelos oficiais de última geração, com excelente suporte de produção e facilidade de uso [8]. Vale ressaltar que modelos de detecção de estágio único são compostos por três componentes: Espinha dorsal (*backbone*), pescoço (*neck*) e cabeça (*head*) (Fig. 3). O primeiro consiste em uma rede pré-treinada que extrai recursos importantes para as imagens, reduzindo a resolução espacial da imagem e aumentando o número de canais. O pescoço do modelo é responsável por extrair pirâmides de recursos, ajudando o modelo a generalizar objetos em diferentes tamanhos e escalas. Por fim, a cabeça do modelo realiza as operações finais, aplicando as *anchor boxes* em mapas de recursos e renderizando a saída final com as classes, a confiabilidade e as caixas delimitadoras [9].

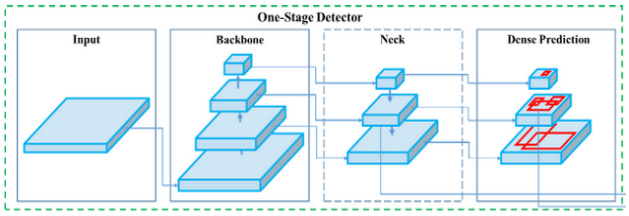

Fig. 3. Arquitetura de um modelo de detecção de único estágio [9].

O algoritmo YOLOv5 usa CSP-Darknet53 como sua espinha dorsal. Ela é construída a partir da rede convolucional Darknet53 usada pelo YOLOv3 mais a aplicação da estratégia de rede CSP (*Cross Stage Partial*). YOLO é uma rede neural profunda [9]. Em redes neurais profundas, existe um fenômeno em que os gradientes usados para atualizar a rede se tornam excessivamente pequenos ou desaparecem conforme são retropropagados das camadas de saída para as camadas de entrada, denomina-se como problema de dissipação do gradiente (*vanishing gradient problem*) [10]. Dessa forma, são utilizados blocos densos e residuais para superar o problema de dissipação do gradiente, permitindo também o fluxo de informações para as camadas mais profundas da rede. Contudo, os blocos densos e residuais trazem o problema de gradientes redundantes. Por sua vez, a rede CSP ajuda a reduzir a quantidade de gradientes profundos [9].

Para lidar com o mapa de recursos, o YOLOv5 utiliza a estratégia CSPNet (CSP Network) para realizar o particionamento da camada base em duas partes e faz a união por meio de uma hierarquia de estágio cruzado. Aplicar essa estratégia permite reduzir o número de parâmetros, aumentando a velocidade de inferência. O aumento de velocidade é essencial para algoritmos de detecção em tempo real, então é de suma importância para o algoritmo YOLOv5 [9].

Na sequência, o pescoço do YOLOv5 é composto por uma variação da camada de agrupamento *Spatial Pyramid Pooling* (SPP) e pela rede *Path Aggregation Network* (PANet). A PANet é um mapa de recursos em pirâmide que foi utilizado no YOLOv4 para melhorar o fluxo de dados. A aplicação no YOLOv5 conta com o uso da estratégia CSPNet. Por sua vez, a variação SPP utilizada foi o SPPF, responsável por agregar as informações recebidas e disponibilizar uma saída com tamanho fixo, aumentando a velocidade da rede [9].

A cabeça do algoritmo YOLOv5 é a mesma implementada no YOLOv3 e YOLOv4. Sua composição conta com três camadas de convolução e é responsável por predizer a localização das caixas delimitadoras junto com os seus parâmetros. Por fim, existem duas funções de ativação que foram utilizadas no YOLOv5. A primeira é a SiLU (*Sigmoid Linear Unit*) e a outra é a *Sigmoid*. A SiLU é usada nas operações de convolução das camadas ocultas, enquanto a *Sigmoid* é usada nas operações de convolução da camada de saída, normalizando a saída para o intervalo de 0 a 1 [9].

A escolha do algoritmo para o presente artigo se deve a dois motivos. O primeiro é pela facilidade de implementação com Python, visto que foi a linguagem utilizada para a construção da API. Dessa forma, realizar a preparação do modelo ao iniciar o servidor torna-se uma tarefa simples, assim como processar as imagens para detectar fogo. O segundo motivo deve-se ao modelo base utilizado para realizar o treinamento, também obtido com o algoritmo YOLOv5 por um usuário do Github a fins didáticos [11].

Nos algoritmos de detecção é comum utilizar a métrica AP (*average precision*) para mensurar a acurácia dos modelos, independente do algoritmo. Essa métrica calcula o valor de precisão médio para o valor de sensibilidade do modelo (*recall*) de 0 a 1. Para entender como essa métrica funciona, é necessário falar sobre precisão, sensibilidade e IoU (*Intersection Over Union*) [12]. Além disso, é essencial entender a métrica F1 e o possível resultado de uma predição. Este pode ser um verdadeiro positivo (VP ou predição positiva correta), verdadeiro negativo (VN ou predição negativa correta), falso

positivo (FP ou predição positiva incorreta) ou falso negativo (FN ou predição negativa incorreta).

1) Precisão (precision)

A precisão informa a proporção de predições corretas em relação a todos os itens classificados como positivos [12]. Traduzindo para uma fórmula, tem-se:

$$Precisão = VP / (VP + FP) \quad (1)$$

2) Sensibilidade (recall)

A sensibilidade do modelo indica a proporção de predições corretas em relação a todos os positivos. A fórmula correspondente, dada por (2) é parecida com a de precisão, porém é utilizada a quantidade de falsos negativos (FN) no lugar dos falsos positivos (FP) [12].

$$Sensibilidade = VP/ (VP + FN) \quad (2)$$

3) F1

Com a combinação da precisão e a sensibilidade através de (3) é possível obter a métrica F1. O valor obtido indica a qualidade geral do modelo, ou seja, quando maior o valor melhor o modelo [13].

$$F1 = \frac{2 * Precisão * Sensibilidade}{Precisão + Sensibilidade} \quad (3)$$

4) IoU (Intersection Over Union)

IoU é a métrica que mede a sobreposição entre 2 caixas delimitadoras. Ou seja, é o limite utilizado para definir se uma previsão é um verdadeiro positivo ou falso positivo [12].

$$IoU = Área\ da\ Interseção / Área\ da\ União \quad (4)$$

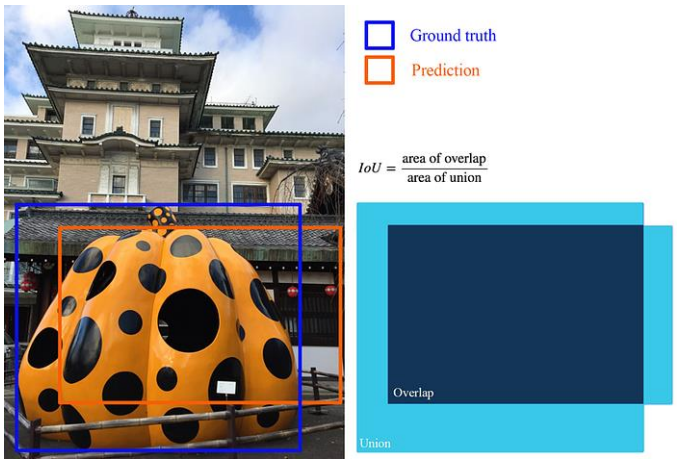

Fig. 4. Exemplo da definição de IoU [12].

5) AP (average precision) e mAP (mean average precision)

A curva de precisão-sensibilidade pode ser visualizada na Fig. 5. Observa-se que a sensibilidade tende a aumentar conforme a precisão abaixa. Contudo, a precisão varia com falsos positivos e verdadeiros positivos, criando um padrão de zigue-zague.

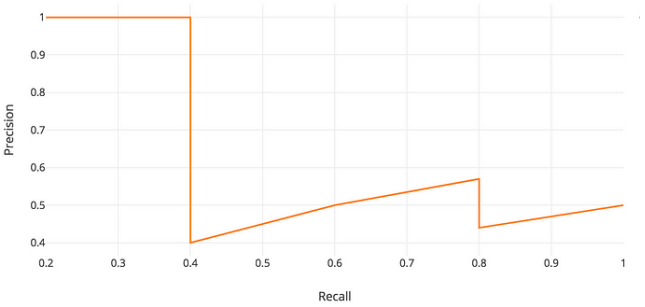

Fig. 5. Curva de precisão-sensibilidade.

Ao encontrar a área sob a curva de precisão-sensibilidade é obtida a precisão média. Como a precisão e a sensibilidade variam entre 0 e 1, a precisão média também estará nesse intervalor de valores [12]. Dessa forma, tem-se (5), em que a função *p(r)* indica a precisão na sensibilidade *r*.

$$AP = \int_0^1 p(r)dr \quad (5)$$

Já a métrica mAP define-se pela média da precisão média entre todas as classes e/ou dentre os limites de IoU. Contudo, as métricas podem ter o mesmo significado, se assim for previamente definido [12]. Um exemplo em que isso acontece é no desafio de avaliação COCO (*Common Objects in Context*), em que é documentada a não diferenciação entre as métricas AP e mAP [14].

Por fim, a métrica mAP pode ser representada com o(s) limite(s) IoU relacionado(s). Por exemplo, mAP50 indica a mAP com IoU de 0,5. Já a mAP50-95 indica a mAP dentre os IoU de 0,5 até 0,95.

C. Protocolo RTSP (Real Time Streaming Protocol)

O protocolo RTSP permite o estabelecer e controlar um ou vários fluxos sincronizados no tempo de mídia contínua, como áudio e vídeo, atuando como um controle remoto de rede para servidores multimídia. Dessa forma, não existe a noção de conexão RTSP, mas sim uma sessão identificada mantida por um servidor. Ainda, uma sessão RTSP não é vinculada a uma conexão de nível de transporte, como TCP (*Transmission Control Protocol*). A sintaxe e operação do protocolo é similar ao protocolo HTTP/1.1 (*HyperText Transfer Protocol*), permitindo que mecanismos de extensão para HTTP possam ser adicionados ao RTSP [15].

As diferenças entre o protocolo RTSP e HTTP são: a introdução de novos métodos e um identificador para o protocolo, a necessidade do servidor de manter um estado padrão, envio de requisições pelo servidor e pelo cliente RTSP, entre outros. Ademais, o protocolo permite uma hospedagem virtual mais fácil, visto que uma máquina pode hospedar vários documentos com um mesmo endereço IP. Por fim, é possível especificar um usuário e uma senha para ter acesso ao *streaming* de dados [15].

III. METODOLOGIA

A. *Preparação do Banco de Imagens*

Após pesquisas na internet na busca de imagens, foi acumulado um total de 3482 imagens, em sua maior parte contendo fogo. Essas imagens vêm da junção de outros bancos de imagens disponíveis em plataformas como Kaggle – plataforma e comunidade para ciências de dados.

O *upload* dessas imagens foi feito em uma ferramenta online chamada Roboflow, que tem como objetivo facilitar tarefas de visão computacional. Suporta tanto modelos de detecção de objetos quanto de classificação [16]. Da mesma forma, também permite a anotação de imagens em formatos como JPG, PNG e BMP e a exportação do *dataset* (banco de imagens) para o formato desejado [16]. Os formatos de exportação incluem os necessários para várias versões dos algoritmos YOLO [16].

Sendo assim, após o *upload* deu-se início ao processo de anotação das imagens. Nesse processo as imagens são passadas uma a uma para que sejam adicionadas manualmente as caixas delimitadoras contendo as ocorrências de fogo (Fig. 6). Por fim, com o *dataset* finalizado é feita a separação das imagens nos subconjuntos de treino, validação e teste e sua exportação para o formato do PyTorch referente ao algoritmo YOLOv5.

Fig. 6. Exemplo de anotação feita em uma imagem com a ferramenta Roboflow

O *dataset* final foi composto por 1620 imagens – o valor deve-se ao mínimo de 1500 imagens recomendado pela documentação do YOLOv5 para o treinamento –, sendo 1280 para treinamento (78%), 185 (11%) para validação e 185 (11%) para teste. Vale ressaltar que aproximadamente 20% (a quantidade mínima recomendada é entre 0 e 10% do valor total de imagens [17]) do *dataset* são imagens de fundo. Essas imagens não possuem objetos de interesse e são adicionadas para reduzir a quantidade de falsos positivos [17]. Ainda, todas as imagens foram redimensionadas para a resolução de 416x416 *pixels*, a fim de tornar o processo de treinamento mais rápido. A escolha dessa resolução foi arbitrária, mas levando em consideração que o treinamento é mais rápido e menos custoso para a máquina com imagens com dimensões menores.

O processo de separar e anotar as imagens foi realizado apenas pelo autor do artigo e o tempo de duração foi de aproximadamente 1 mês e 15 dias. Ao todo, foram feitas 3353 anotações manualmente, resultando em uma média de 2 anotações por imagem.

B. *Treinamento do Modelo*

O treinamento foi realizado em uma máquina local, equipada com uma placa de vídeo NVIDIA GeForce RTX 2060 com 6 GB de memória, um processador Intel(R) Core(TM) i7-10750H com 2,6GHz (12 CPUs) e 16GB de memória RAM. Sendo assim, o primeiro passo foi clonar o repositório no Github do YOLOv5 e instalar as dependências do projeto. Ainda, foi necessário instalar o conjunto de ferramentas CUDA (*Compute Unified Device Architecture*) para ser possível utilizar a placa de vídeo em todas as etapas. Dessa forma, com o ambiente pronto inicia-se o treinamento do modelo. Vale destacar que a biblioteca utilizada foi o PyTorch, na versão 1.13.1.

Primeiramente, é imprescindível notar que os pesos iniciais originam de um modelo treinado por um usuário do Github para detecção de fogo para fins didáticos [11]. A mAP50 original no conjunto de validação é de 0,469 e o valor máximo de F1 foi de 0,53 com uma confiança de 0,329. Ainda, é importante lembrar que nenhuma camada foi congelada para que o modelo obtivesse uma melhor generalização durante a fase de aprendizado.

Primeiramente, alguns parâmetros precisam ser definidos para realizar a tarefa sem exceder o limite de memória. O tamanho da imagem de entrada foi ajustado para ser a mesma dimensão das imagens exportadas anteriormente (416x416 *pixels*), a quantidade de épocas foi definida para 300 (mínimo recomendado [17]) e o número de amostras propagadas para a rede (*batch size*) foi ajustado para 24. Este valor foi obtido empiricamente, visto que os valores acima de 24 excederam o limite de memória. O tempo de treinamento com esses parâmetros usando a placa de vídeo foi de aproximadamente uma hora e meia.

Um segundo treinamento foi realizado com o mesmo banco de imagens, porém as imagens foram exportadas para a dimensão 640x640 *pixels*. Dessa maneira, o tamanho da imagem de entrada foi ajustado para a nova dimensão. Os outros parâmetros foram repetidos. Por fim, o tempo de treinamento foi de aproximadamente duas horas e meia.

A documentação do YOLOv5 indica que alterar os valores iniciais dos hiperparâmetros pode aprimorar o resultado do treinamento. Contudo, foram utilizados os valores padrão para essa etapa, recomendados quando em dúvida dos valores iniciais. Esses valores são otimizados para o treinamento no *dataset* COCO com YOLOv5, fazendo uso de pouca augmentação [18]. A Fig. 7 mostra todos os valores utilizados.

```yaml
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.05
cls: 0.5
cls_pw: 1.0
obj: 1.0
obj_pw: 1.0
iou_t: 0.2
anchor_t: 4.0
fl_gamma: 0.0
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
```

Fig. 7. Hiperparâmetros utilizados no treinamento do modelo.

C. *Desenvolvimento do Sistema de Gerenciamento*

O sistema de gerenciamento é constituído por duas partes. O servidor e a aplicação *web*. Como o algoritmo YOLOv5 já possui uma implementação em Python, a mesma linguagem foi escolhida para o desenvolvimento do servidor. Dessa forma, pôde-se aproveitar a implementação original para realizar a leitura e a preparação do modelo. Ainda, o *framework* FastAPI foi utilizado na construção da API (*Application Programming Interface*). Por outro lado, a aplicação *web* foi desenvolvida com a linguagem de programação TypeScript e a biblioteca *React*. Na Fig. 8 é apresentada a arquitetura geral do sistema. A aplicação *web* conversa com o servidor através das interfaces REST e WebSocket.

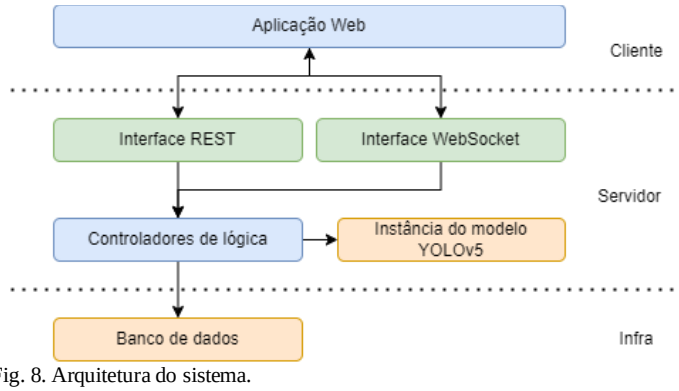

Fig. 8. Arquitetura do sistema.

1) *Servidor*

Os recursos implementados no servidor são: cadastro de usuários; autenticação; autorização; gerenciamento de câmeras IP; *streaming* das câmeras cadastradas. Ações como cadastrar um novo usuário e realizar o *login* não exigem um usuário autenticado. Ainda, após autenticar-se é possível fazer a leitura dos recursos cadastrados ou registrar uma câmera IP nova, por exemplo. Todos os recursos são armazenados em um banco de dados. Nessa implementação foi utilizado o banco de dados PostgreSQL. Ademais, para facilitar o controle das tabelas do banco foi aplicada a ferramenta Alembic (versão 1.10.2) – ferramenta leve de migração de banco de dados – em conjunto com o ORM (*Object Relational Mapper*) SQLAlchemy (versão 2.0.5.post1).

Ao receber uma requisição válida, o controlador associado a rota irá realizar os acessos necessários ao banco de dados e retornar a resposta apropriada para o contexto. A autorização é definida a nível de rota. Se a ação a ser realizada requer autenticação, então apenas usuários autenticados podem ter acesso a rota. Dessa forma, o acesso aos recursos pode ser controlado a fim de obter um nível de segurança maior na aplicação.

O protocolo RTSP foi adotado para realizar a conexão com as câmeras IP. Essas câmeras atuam como servidor multimídia em tempo real e fornecem as imagens para o cliente que fez a solicitação. Dessa forma, o servidor se torna um intermediário entre a aplicação *web* e a câmera IP, atuando como um intermediário. Então, é possível processar as imagens obtidas através da sessão RTSP com o modelo de *Machine Learning* obtido anteriormente. Por fim, a aplicação *web* tem acesso às respostas em tempo real.

2) *Aplicação Web*

A implementação dessa aplicação foi realizada com a biblioteca *React*. Essa biblioteca possui um vasto suporte na comunidade e tornou-se muito popular na área de *front-end* (termo relacionado ao desenvolvimento de interfaces gráficas) nos últimos anos. Ainda, existem diversas outras bibliotecas que permitem o desenvolvimento de aplicações sem a necessidade de reinventar a roda. Também foram utilizadas bibliotecas como *yup* (versão 1.0.2) para criar esquemas de validação dos formulários, *fscreen* (versão 1.2.0) para entrar e sair da tela cheia e até mesmo *styled-system* (versão 5.1.5) para ter uma melhor organização e controle dos estilos dos componentes e páginas.

A aplicação *web* fica restrita nas rotas de *login* e cadastro de usuário até que o *login* seja realizado. Uma vez que o usuário seja autenticado ele será redirecionado para a tela principal do sistema de gerenciamento. Na tela principal as câmeras IP serão listadas e cada conexão será iniciada automaticamente.

Em um primeiro momento um novo usuário não terá câmeras cadastradas. Para cadastrar uma nova câmera é possível utilizar a ação de adicionar uma nova câmera IP – a opção localiza-se na barra de ações no topo da aplicação *web*. O cadastro de uma câmera requer um nome e a URL que pode ser acessada. Dessa maneira, após o cadastro a lista de câmeras pode ser atualizada e a conexão estabelecida. A edição e deleção da câmera também são possíveis através de ícones dentro da área de *streaming*.

3) *Integração da Aplicação Web com o Servidor*

As câmeras IP cadastradas são exibidas ao entrar na tela principal da aplicação *web*. Cada câmera IP é gerenciada por um componente separado. Ao renderizar esse componente, uma

conexão *WebSocket* (WS) será iniciada com o servidor, gerando um *token* do lado do servidor (etapa 1). Esse *token* é controlado no servidor, sendo adicionado em uma camada de *cache* em tempo de execução com informações sobre a conexão *WebSocket* e com o identificador da câmera (etapas 2 e 3). Ao receber o *token*, a próxima etapa é iniciada (etapa 4).

A aplicação salva o *token* obtido da primeira mensagem do servidor e o utiliza para realizar uma requisição HTTP, iniciando o streaming da câmera IP (etapa 5). O servidor irá validar o *token* recebido e a existência da câmera associada. Depois da validação a conexão com a câmera IP é iniciada – no servidor é iniciada a sessão RTSP (etapa 6). Dessa maneira, o servidor recebe cada imagem que a câmera dispõe e processa com a instância do modelo YOLOv5 (etapas 7 e 8). Assim, a imagem com as possíveis caixas delimitadoras é enviada (etapas 9 e 10). Na sequência, a conexão *WebSocket* é utilizada para enviar uma nova mensagem dizendo se alguma ocorrência de fogo foi detectada (etapa 11).

Por fim, a aplicação *web* recebe a imagem processada e a mensagem *WebSocket* simultaneamente, possibilitando alertar o usuário sobre a detecção de fogo quando for positiva. Todo o processo pode ser visualizado na Fig. 9. A ordem das etapas no diagrama segue a mesma apresentada.

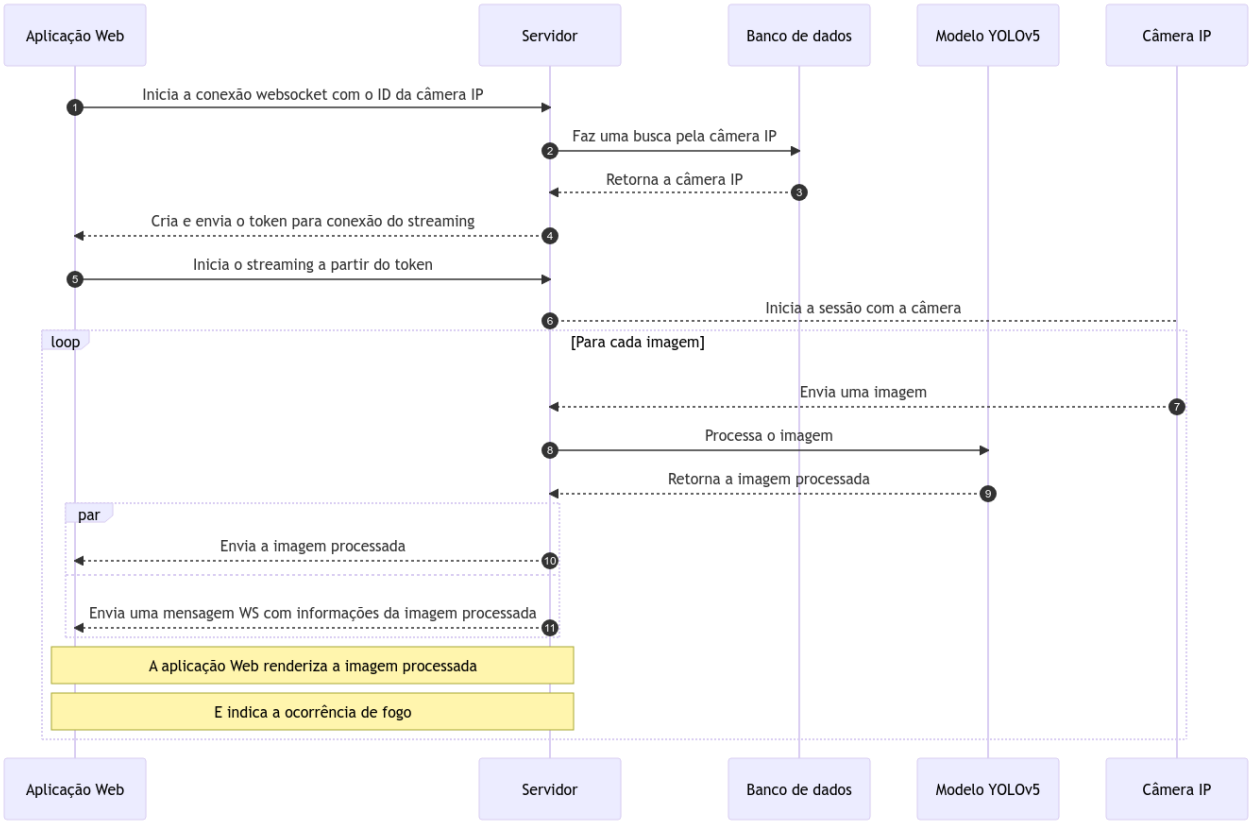

Fig. 9. Diagrama da integração entre a aplicação web e o servidor ao conectar-se com uma câmera cadastrada.

4) Simulação de Câmeras IP

Para testar o sistema de gerenciamento com múltiplas câmeras IP foi utilizada uma ferramenta para reproduzir vídeos e disponibilizar o streaming através do protocolo RTSP (*Real Time Streaming Protocol*). A criação desse ambiente foi possível com o *Docker Compose*, uma ferramenta que permite definir e executar múltiplos *containers* – um container é um processo executado de forma isolada do sistema operacional. Ainda, a criação desse *container* pode ser feita a partir de uma imagem base. Essa imagem é um *container* previamente criado e configurado disponibilizada no *Docker Hub*, uma plataforma de hospedagem para imagens de *containers*.

Permitir que um vídeo seja acessado via protocolo RTSP é possível com o *framework* FFmpeg. Esse *framework* possibilita várias manipulações com arquivos e a mais importante no contexto desse artigo é o *streaming*. Dessa maneira, a imagem base utilizada (linuxserver/ffmpeg:version-4.4-cli) foi escolhida por já ter esse *framework* configurado e pronto para ser usado via linha de comando. Por fim, foram utilizados vídeos diferentes para criar algumas simulações de câmeras IP, permitindo o cadastro no sistema de gerenciamento. Na Fig. 10 é possível observar as câmeras cadastradas e a ocorrência de fogo detectada na última câmera.

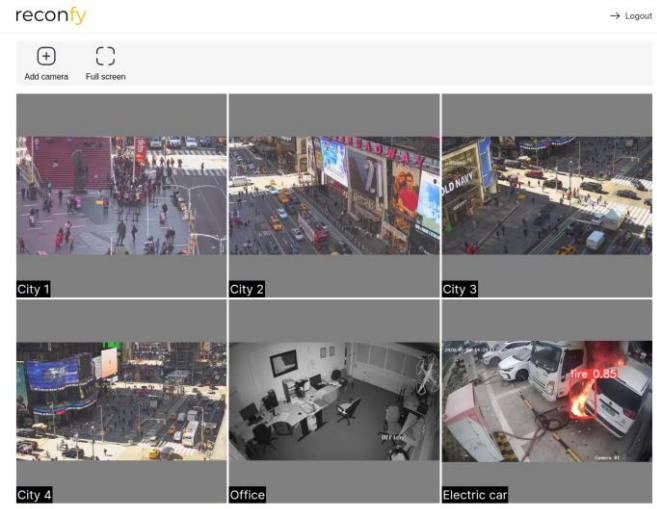

Fig. 10. Aplicação web com algumas câmeras cadastradas.

D. *Escalabilidade do Servidor*

Para estudar como o servidor se comporta com múltiplas requisições simultâneas foi utilizada a ferramenta k6. Essa ferramenta possui código aberto e permite realizar testes de carga com inúmeros parâmetros disponíveis. Dessa forma, os parâmetros definidos foram a quantidade de usuários virtuais e o total de iterações, ambos com valor de 10. A escolha do valor deve-se a configuração padrão do uvicorn – implementação web ASGI (*Asynchronous Server Gateway Interface*) para Python – para executar o servidor. Sendo assim, o servidor é executado em um único processo, tornando as requisições sequenciais. Em outras palavras, o servidor torna-se muito lento com requisições simultâneas feitas pelo k6, pois lida com elas sequencialmente.

Antes de executar o teste de carga, é necessário que o servidor e as simulações das câmeras IP estejam iniciados e pelo menos uma câmera deve estar cadastrada. Assim sendo, a estrutura do teste de carga implementada realiza duas operações principais. Primeiramente é necessário conectar-se via *WebSocket* à rota para obter o *token*. Na sequência, o *token* é utilizado para conectar à rota de *streaming*. As verificações de sucesso envolvem validar se o *status* retornado pela conexão *WebSocket* foi 101, indicando a troca de protocolo, e validar se a resposta da rota de *streaming* retornou algum dado. Ademais, a métrica utilizada para avaliar o servidor foi a *http_req_waiting*, pois indica o tempo esperado para obter uma resposta do servidor remoto, também chamado de latência do servidor [19]. Por fim, os valores analisados foram os valores percentis P(90) e P(95), representando as latências de 90% e 95% das requisições, respectivamente [20].

IV. Resultados

Como abordado anteriormente, a métrica F1 avalia a qualidade geral do modelo, relacionando a precisão e a sensibilidade. Na Fig. 11 é observada a relação de F1 com a confiança do modelo. A confiança que maximiza o valor de F1 é de 0,329.

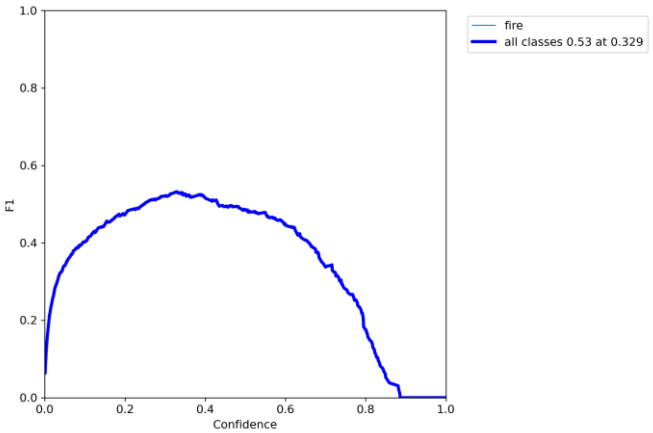

Fig. 11. Relação entre a métrica F1 e a confiança do modelo no conjunto de validação do modelo base [11].

O modelo base possui uma mAP50 de 0,469. A falta de habilidade do modelo é notável pela Fig. 12, em que a curva de precisão-sensibilidade permanece praticamente centrada.

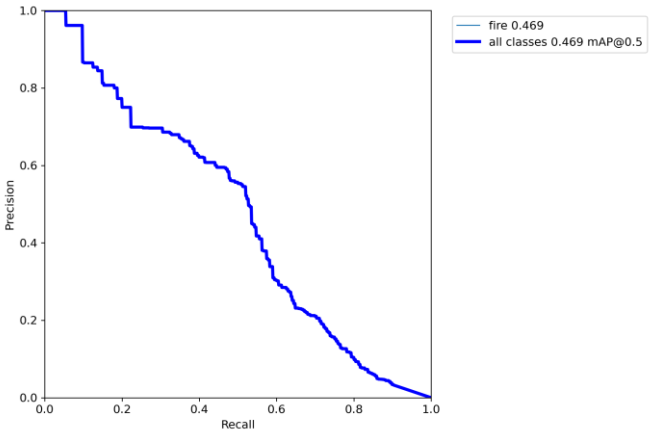

Fig. 12. Relação entre precisão e sensibilidade no conjunto de validação do modelo base.

O primeiro modelo treinado obteve uma mAP50 de 0,577, uma mAP50-95 de 0,303, precisão de 0,663 e sensibilidade de 0,552. Por outro lado, o segundo modelo obteve uma mAP50 de 0,552, mAP50-95 de 0,312, precisão de 0,614 e sensibilidade de 0,557. Em resumo, o modelo escolhido para ser utilizado na aplicação foi o primeiro, pois conta com uma mAP50 superior ao segundo modelo. Ainda, a mAP50 obtida apresenta um aumento de 23% da mAP50 do modelo base.

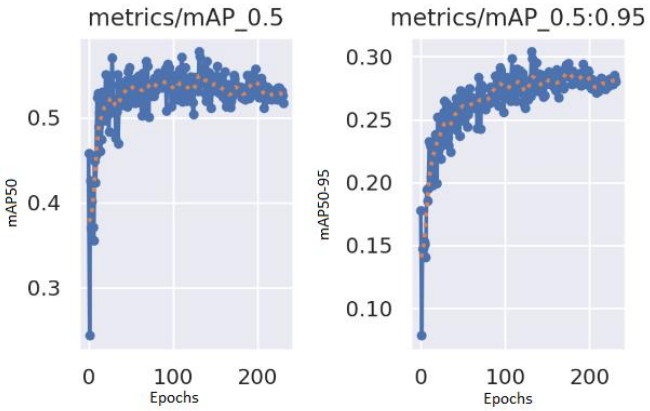

Fig. 13. Evolução das métricas mAP50 e mAP50-95 do modelo escolhido.

Na Fig. 13 vê-se a evolução das métricas mAP50 e mAP50-95 ao decorrer das épocas. O valor de épocas escolhido foi 300, porém nota-se que o gráfico se encerra antes, mais especificamente na época 233. Isso deve-se a parada automática no treinamento ao não ser observada nenhuma melhoria no modelo nas últimas 100 épocas. Observa-se no gráfico que os valores da mAP50-95 são muito baixos, pois aumenta o limite IoU e requer caixas delimitadoras mais próximas das originais. O objeto em questão é fogo e o fogo tem muitas variações em sua forma e suas tonalidades contrastando com as cores de fundo. Dessa maneira, para detecção de fogo não é muito conveniente utilizar um limite IoU muito alto, então o limite definido – arbitrariamente – no servidor foi de 0,6.

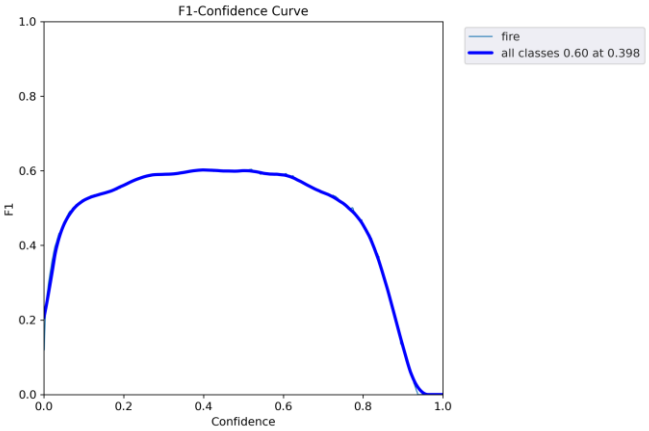

Fig. 14. Relação entre a métrica F1 e a confiança do modelo no conjunto de validação do modelo obtido.

Na Fig. 14 pode-se observar que a confiança que maximiza o valor de F1 é de 0,398, otimizando os valores de precisão e sensibilidade. Com uma breve comparação com a curva do modelo base, é possível notar que a qualidade geral do modelo melhorou. Retomando, é desejável ter uma confiança maior sempre que possível. Pelo gráfico, pode-se notar que o valor de F1 só começa a decair por volta da confiança de 0,65, então é possível utilizar um valor abaixo próximo. Desse modo, o valor de confiança definido no servidor foi de 0,6, levando em consideração o decaimento da qualidade acima de 0,65.

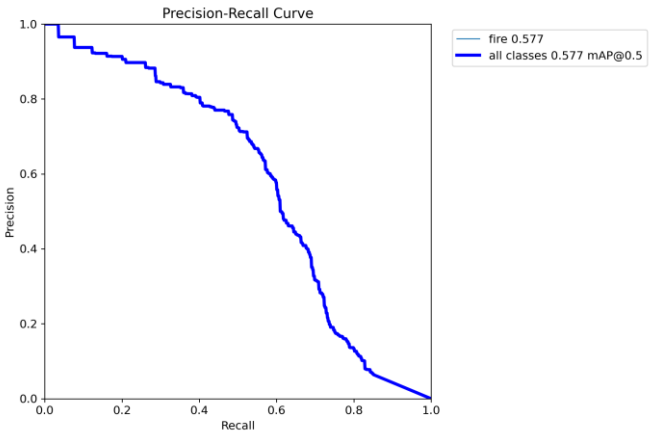

Fig. 15. Relação entre precisão e sensibilidade no conjunto de validação do modelo obtido.

A partir da Fig. 15 é possível analisar a relação entre precisão e sensibilidade. Uma sensibilidade baixa indica uma precisão maior, enquanto em sensibilidades mais altas a precisão começa a decair. Com a mAP50 de 0,577, tem-se uma precisão de 0,663 e sensibilidade de 0,552.

Todas as métricas apresentadas até o momento estão relacionadas ao conjunto de validação utilizado. Na Fig. 16 são apresentadas algumas imagens desse conjunto com as caixas delimitadoras originais e na Fig. 17 as mesmas imagens são exibidas com as caixas delimitadoras detectadas pelo modelo.

Fig. 16. Algumas imagens do conjunto de validação com as caixas delimitadoras originais.

Fig. 17. Algumas imagens do conjunto de validação com as caixas delimitadoras detectadas.

Na Fig. 17 as caixas delimitadoras detectadas são plotadas com a confiança que o modelo tem de que o objeto se trata de uma ocorrência de fogo. Observa-se que a ocorrência de fogo varia de tamanho e na segunda imagem da primeira linha há uma detecção de baixa confiança (0,3). Dessa forma, utilizar a confiança de 0,6 faria essa ocorrência não ser reportada como fogo.

O desempenho do modelo no conjunto de teste foi consideravelmente pior. A mAP50 obtida foi de 0,413 e a mAP50-95 foi de 0,19.

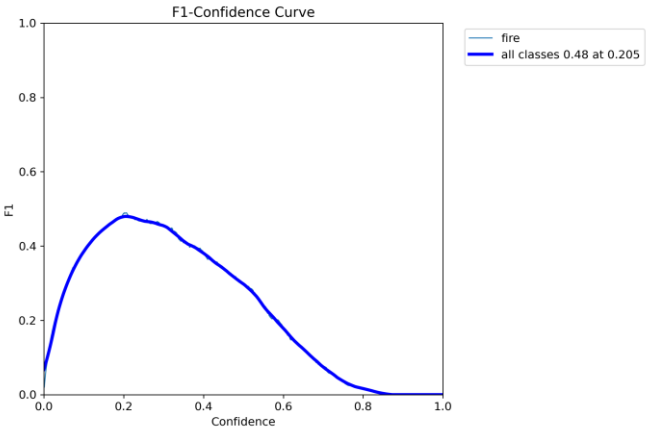

Fig. 18. Relação entre a métrica F1 e a confiança do modelo no conjunto de teste.

Na Fig. 18 é notável que a qualidade geral do modelo está baixa. O valor máximo de F1 é 0,48 com uma confiança de 0,205. Esse valor baixo para F1 indica baixa precisão e baixa sensibilidade. Em outras palavras, o modelo gera falsos positivos e falsos negativos com frequência indesejada.

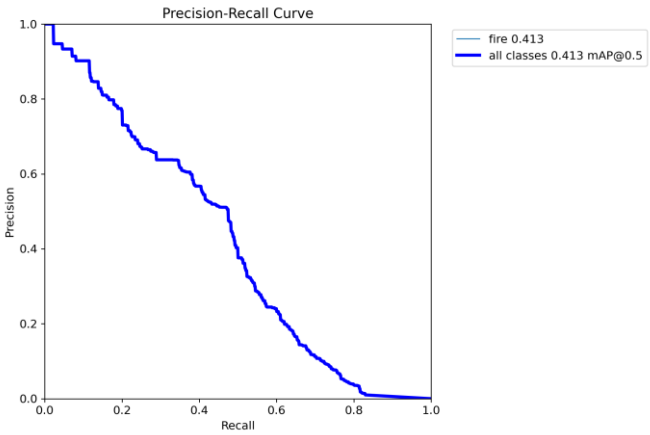

Fig. 19. Relação entre precisão e sensibilidade no conjunto de teste.

Pela Fig. 19 observa-se que a curva se mantém centralizada, diferentemente do que foi visto na curva do conjunto de validação. Neste caso, é correto dizer que o modelo tem pouca habilidade, devido à baixa precisão (0,514) e sensibilidade (0,448). A falta de habilidade pode ser associada a qualidade e a quantidade de imagens que foram utilizadas no treinamento do modelo. Ainda, devido às limitações de recursos computacionais não foi possível realizar treinamentos variando a dimensão da imagem de entrada e o número de amostras propagadas para a rede, ambos os parâmetros são passíveis de ajudar na fase de treinamento e aprimorar o modelo.

Por fim, o limite de confiança definido no servidor foi de 0,6. No conjunto de teste esse valor possui uma pontuação F1 baixa e maior sensibilidade do que precisão. Sendo assim, na prática esse modelo gera mais falsos positivos do que falsos negativos, mas ambos estão presentes. Da mesma forma, reduzir o limite de confiança para 0,205 piora os resultados práticos, pois a quantidade de falsos positivos aumenta consideravelmente.

Com o propósito de validar a escalabilidade do servidor foi realizado um teste com a ferramenta k6. Primeiramente, define-se a quantidade de usuários virtuais e o total de iterações para 10. Em seguida, o teste pode ser iniciado. O teste durou cerca de 1 minuto e a métrica *http_req_waiting* indicou uma P(90) de 58,64 segundos e P(95) de 59,31 segundos. O tempo de latência alto era esperado, pois o servidor retorna apenas uma resposta por vez, enquanto as requisições são feitas simultaneamente. Para que o servidor se torne escalável, seria necessário alterar a configuração do uvicorn para utilizar múltiplos processos para executar o servidor. Dessa forma, com múltiplas instâncias a latência P(90) e P(95) tenderiam a reduzir, pois o servidor passaria a retornar respostas paralelas.

V. Conclusões

Conforme abordado anteriormente, as ocorrências de incêndio são muito danosas e a ação rápida do corpo de bombeiros é crucial para que a propagação do fogo seja diminuída e os danos minimizados, evitando vítimas e outros possíveis problemas. Dessa forma, os protocolos de segurança estão em constante desenvolvimento e evoluem para que os incêndios sejam prevenidos ou para que sejam extinguidos o quanto antes. O presente artigo visou a aplicação de um modelo de *Machine Learning* com o algoritmo YOLOv5 para o desenvolvimento de um sistema de gerenciamento capaz de

detectar fogo em tempo real, permitindo responder às ocorrências de incêndio com maior velocidade.

A base de imagens utilizada foi composta por 1620 imagens. Dessas imagens, 1280 foram utilizadas para o treinamento do modelo, 185 para validação e outras 185 para teste. Após o treinamento, o modelo obteve uma mAP50 de 0,577 no conjunto de validação. Pela Fig. 14 nota-se que a qualidade geral do modelo começa a decair com uma confiança acima de 0,65. Contudo, no conjunto de teste a performance do modelo não foi boa, pois a mAP50 foi de 0,413 e a qualidade geral do modelo começa a decair com uma confiança acima de 0,2. Dessa forma, o valor utilizado na aplicação foi de 0,6, levando em consideração o decaimento da qualidade geral no gráfico plotado. Ainda, o valor limite de IoU utilizado foi de 0,6. Este levou em consideração que valores altos iriam requerer caixas delimitadoras mais próximas das originais, sendo uma tarefa dificultosa quando objeto a ser detectado é fogo.

O sistema de gerenciamento desenvolvido foi composto por uma aplicação *web* e um servidor. A aplicação *web* foi estruturada para utilizar todos os recursos disponibilizados pelo servidor. A integração para conexão de uma câmera IP deu-se por dois processos. O primeiro é abrir a conexão *WebSocket* e o segundo é conectar na rota que faz o *streaming* das imagens processadas pelo modelo. Ademais, para realizar os testes na aplicação foi utilizada uma ferramenta para simular câmeras IP. A partir disso, foi possível cadastrar as câmeras IP das simulações para validar o comportamento da aplicação.

O sistema de gerenciamento foi desenvolvido a fins de produzir o projeto de conclusão de curso (PFC), portanto o código é aberto e está hospedado no Github do autor, disponível em https://github.com/devjvao/reconfy. Dessa forma, existem diversas melhorias que podem ser aplicadas. Por exemplo, é possível adicionar a configuração para controlar a confiabilidade das predições e o limite IoU a ser utilizado, permitindo ao usuário definir se quer mais detecções com falsos positivos ou menos detecções, porém com mais falsos negativos. Ainda, uma funcionalidade interessante seria reordenar a lista de câmeras para que as que possuem detecções de incêndio sejam realocadas para as primeiras posições, facilitando a visualização para tomadas de decisão. Outro ponto é a implementação de um sistema de alertas por mensagens SMS, e-mails e dispositivos sonoros, visando tomadas de decisão mais rápidas.

Por fim, conclui-se que o sistema desenvolvido pode ser utilizado para monitoramento de ocorrências de incêndio via câmeras IP. Fazer uso de um modelo de *Machine Learning* para automatizar esse processo de detecção de incêndios é uma forma de aprimorar a segurança no âmbito geral e foi possível por conta do algoritmo YOLOv5, pois o processamento é mais rápido com a estratégia de estágio único.

Contudo, o sistema carece de um modelo com maior precisão e sensibilidade. Para tal, é possível aprimorar a qualidade das imagens, aumentar a quantidade de imagens utilizadas para realizar o treinamento, aumentar a resolução das imagens, aumentar o número de amostras propagadas para a rede e encontrar os valores ótimos para os hiperparâmetros na etapa de treinamento. Dessa maneira, com os devidos ajustes e uma máquina com mais recursos para realizar o treinamento seria possível obter um modelo mais preciso e sensível. É válido ressaltar a necessidade de se utilizar o algoritmo YOLOv5 para que o modelo seja compatível com o servidor. Ainda, para escalar o servidor é necessário alterar a configuração do uvicorn para que execute múltiplas instâncias, tornando o tempo de resposta para as requisições mais rápido e menos penoso para um único processo.

O monitoramento de câmeras IP precisa ser feito de forma mais segura. Sendo assim, é desejável que o usuário e a senha para abrir a sessão RTSP da câmera IP também sejam cadastrados. Então, a senha pode ser codificada com algum algoritmo antes de ser salva no banco de dados e o acesso ao seu conteúdo só seria permitido através do *token* gerado ao iniciar a conexão com um usuário autenticado.

VI. Referências

[1] L. Sofia Batalha de Moura, “Drone de combate aos incêndios florestais,” Março 2019.

[2] L. Gustavo Oliveira de Freitas, “PROTOTIPAGEM DE SISTEMA DE ALARME DE INCÊNDIO BASEADO EM,” Junho 2021.

[3] K. Facelli, A. Carolina Lorena, J. Gama, T. Agostinho de Almeida e A. Carlos Ponce de Leon Ferreira de Carvalho, “Inteligência Artifical: Uma Abordagem de Aprendizado de Máquina,” 2011, pp. 1-7;122-135.

[4] T. Michael Mitchell, “Machine Learning,” McGraw Hill, 1997, pp. 40-41.

[5] Y. Zhang, Z. Guo, J. Wu, Y. Tian, H. Tang e X. Guo, “Real-Time Vehicle Detection Based on Improved YOLO v5,” *Sustainability*, vol. 14, nº 19, p. 12274, Setembro 2022.

[6] J. Hosang, R. Benenson e B. Schiele, “Learning non-maximum suppression,” p. 1, 2017.

[7] T. Diwan, J. V. Tembhurne e G. Anirudh, “Object detection using YOLO: challenges, architectural successors, datasets and applications.,” *Multimedia Tools and Applications*, 27 Janeiro 2021.

[8] B. Kumar Pradhan, “A Guide to the YOLO Family of Computer Vision Models,” Data Phoenix, 8 Junho 2023. [Online]. Available: https://dataphoenix.info/a-guide-to-the-yolo-family-of-computer-vision-models/. [Acesso em 23 Julho 2023].

[9] C. Imane, “YOLO v5 model architecture,” OpenGenus, [Online]. Available: https://iq.opengenus.org/yolov5/. [Acesso em 23 Julho 2023].

[10] C.-F. Wang, “The Vanishing Gradient Problem,” Towards Data Science, 8 Janeiro 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. [Acesso em 7 Agosto 2023].

[11] “yolov5-fire-detection,” 3 Março 2022. [Online]. Available: https://github.com/spacewalk01/Yolov5-Fire-Detection.

[12] J. Hui, “mAP (mean Average Precision) for Object Detection,” 6 Março 2018. [Online]. Available: https://jonathan-hui.medium.com/map-mean-average-

precision-for-object-detection-45c121a31173. [Acesso em 31 Julho 2023].

[13] R. dos Santos Leal, “Métricas Comuns em Machine Learning: como analisar a qualidade de chat bots inteligentes — métricas (3 de 4),” 22 Maio 2017. [Online]. Available: https://medium.com/as-m%C3%A1quinas-que-pensam/m%C3%A9tricas-comuns-em-machine-learning-como-analisar-a-qualidade-de-chat-bots-inteligentes-m%C3%A9tricas-1ba580d7cc96. [Acesso em 31 Julho 2023].

[14] “Detection Evaluation,” COCO Common Objects in Context, [Online]. Available: https://cocodataset.org/#detection-eval. [Acesso em 2023 Julho 31].

[15] H. Schulzrinne, R. Lanphier e A. Rao, “Real Time Streaming Protocol (RTSP),” Abril 1998. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2326#page-5. [Acesso em 07 Agosto 2023].

[16] M. Faizan, “Roboflow,” Ref Buffer, 15 Abril 2022. [Online]. Available: https://medium.com/red-buffer/roboflow-d4e8c4b52515. [Acesso em 2023 Julho 2023].

[17] G. Jocher e S. Waxmann, “Tips for Best Training Results,” Ultralytics, 21 Abril 2023. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/. [Acesso em 30 Julho 2023].

[18] G. Jocher e S. Waxmann, “Hyperparameter evolution,” Ultralytics, 21 Abril 2023. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/hyperparameter_evolution/. [Acesso em 07 Agosto 2023].

[19] “Built-in metrics,” Grafana Labs, [Online]. Available: https://k6.io/docs/using-k6/metrics/reference/. [Acesso em 2023 Agosto 05].

[20] C. Nguyen Duc, “Mastering Latency Metrics: Understanding P90, P95, and P99,” 8 Junho 2023.

João Vitor Almeida de Oliveira é graduando em Engenharia de Computação na Universidade Federal de Goiás (2023). Possui experiência no desenvolvimento de soluções tecnológicas na área de *frontend* e *backend*. Atualmente é desenvolvedor de *software* na Croct (São Paulo).