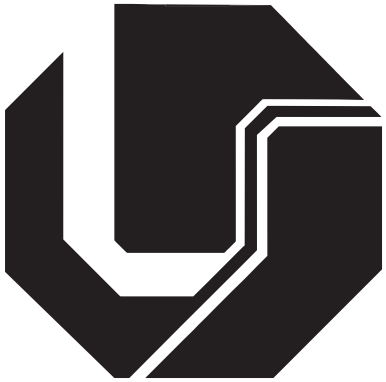
UNIVERSIDADE FEDERAL DE UBERLANDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



Inserção Automática de Componentes em Ambientes Virtuais de Treinamento para Substações de Energia utilizando Inteligência Artificial

Leandro Sena Zuza

Uberlândia

2024

Leandro Sena Zuza

Inserção Automática de Componentes em Ambientes Virtuais de Treinamento para Substações de Energia utilizando Inteligência Artificial

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como parte dos requisitosexigidos para obtenção do Título de Mestre em Ciências.

Julho X, 2024.

Membros da Banca:

Prof. Alexandre Cardoso, Dr.
Orientador - UFU

Prof. Daniel S. Caetano, Dr.
Coorientador - UFU

Agradecimentos

[INSERIR NOVOS AGRADECIMENTOS]

"Então considerei que as botas apertadas são uma das maiores venturas da Terra,
porque, fazendo doer os pés, dão azo ao prazer de as descalçar."
(Machado de Assis, em Memórias Póstumas de Brás Cubas)

Abstract

The precise identification of equipment in images plays a crucial role in various operations related to power substations, facilitating not only maintenance but also the monitoring of these facilities. In this work, we present results on the efficiency of YOLOv8 in detecting equipment present in power substations from images obtained by Unmanned Aerial Vehicles (UAVs). We employ different optimization techniques to enhance detection efficiency, aiming to achieve more accurate and faster results. Additionally, this study aims to go beyond mere training with captured photos, seeking to identify the best-trained model to create a script capable of selecting, from a database of virtual reality models, the elements necessary for assembling a virtual power substation. Thus, we aim not only to improve the maintenance and monitoring processes of power substations in physical reality but also to streamline and enhance the generation of Virtual Training Environments for procedures related to these substations. With this advancement, we hope to not only optimize the use of detection technology in power substations but also to significantly contribute to the creation of realistic and efficient virtual environments for training in procedures related to the operation and maintenance of these facilities.

Keywords

Keywords - Power Substation; UAV; YOLOv8; Optimization; Virtual Training Environments

Resumo

A identificação precisa de equipamentos em imagens desempenha um papel crucial em várias operações relacionadas às subestações de energia, facilitando não apenas a manutenção, mas também o monitoramento dessas instalações. Neste trabalho, apresentamos resultados da eficiência da YOLOv8 na detecção de equipamentos presentes em subestações de energia, a partir de imagens obtidas por Veículos Aéreos Não Tripulados (VANTs). Utilizamos diferentes técnicas de otimização para aprimorar a eficiência na detecção, visando alcançar resultados mais precisos e rápidos. Além disso, este estudo visa ir além do mero treinamento com as fotos capturadas, buscando identificar o melhor modelo treinado para criar um script capaz de selecionar, a partir de uma base de modelos de realidade virtual, os elementos necessários para a montagem de uma subestação de energia virtual. Dessa forma, almejamos não apenas melhorar os processos de manutenção e monitoramento das subestações de energia na realidade física, mas também agilizar e aprimorar a geração de Ambientes Virtuais de Treinamento para procedimentos relacionados a essas subestações. Com este avanço, esperamos não só otimizar o uso de tecnologia de detecção em subestações de energia, mas também contribuir significativamente para a criação de ambientes virtuais realistas e eficientes para treinamento em procedimentos relacionados à operação e manutenção dessas instalações.

Palavras Chave

Subestação de Energia; VANTs; YOLOv8; Otimização; Ambientes Virtuais de Treinamento

Publicações

As publicações relacionadas à pesquisa e ao trabalho realizado são listadas a seguir:

1. Zuza, L. S., Cardoso, A., Lamounier, E., & Caetano, D. (2024). Análise dos Parâmetros da YOLOv8 na eficiência da identificação de reatores de núcleo de ar a partir de imagens de subestações de energia. In: *CISTI'2024 - 19ª Conferência Ibérica de Sistemas e Tecnologias de Informação*, 25-28 de junho de 2024, Salamanca, Espanha.

Sumário

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

VANT	Veículo Aéreo Não Tripulado
RV	Realidade Virtual
NA	Neurônio Artificial
RNC	Rede Neural Convolucional
YOLO	You Look Only Once
GD	Gradiente Descendente
SGD	Stochastic Gradient Descent
ADAM	Adaptive Moment Estimation

1 Fundamentação Teórica

1.1 Subestações de Energia

Atualmente, o sistema elétrico do Brasil é formado por um conjunto de usinas, subestações, linhas de transmissão e demais equipamentos, que viabilizam a produção, transmissão e distribuição de energia elétrica. A energia é gerada em usinas que variam conforme os recursos utilizados, como hidroelétricas, termoelétricas, eólicas, nucleares, entre outras, e é então transportada por linhas de transmissão até subestações de energia de empresas distribuidoras. Nestas subestações, a tensão é ajustada para os níveis adequados de consumo. Esse processo elétrico é composto por três etapas principais: geração, transmissão e distribuição (Figura **??**). O sistema elétrico é composto por diversos elementos, sendo os mais destacados os geradores, responsáveis pela transformação de energia mecânica em elétrica, e os sistemas de transmissão e distribuição, que conduzem a energia até os consumidores. As subestações têm um papel fundamental ao ajustar a tensão na transmissão e distribuição, assegurando a confiabilidade e a qualidade do serviço elétrico fornecido aos clientes (**??**).

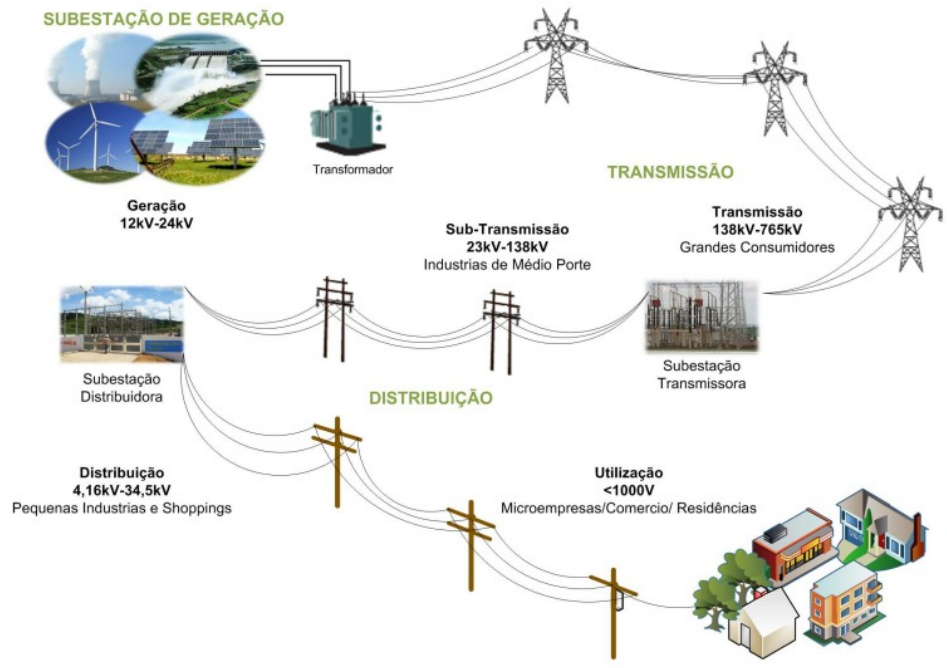


Figura 1 – Representação dos componentes do sistema de geração e transmissão de energia (**??**)

.

De modo geral, portanto, a energia é, obtida por geradores e transmitida para

subestações primárias. Há aí a primeira aferição de qualidade dos níveis de transmissão. Após isso, a energia transformada é enviada para subestações de distribuição, alterada para níveis mais baixos para serem transmitidas para setores industriais e urbanos (**??**).

É importante destacar que a energia produzida no Brasil é oriunda majoritariamente de fontes renováveis, em especial da geração hidroelétrica, correspondendo a 69% de toda malha produtora. Ela é considerada uma energia limpa, de baixo custo e possui reduzida emissão de gases de efeito estufa. (**??**).

Analisando mais detalhadamente as subestações, podemos afirmar que elas são encarregadas de regular a tensão tanto na transmissão quanto na distribuição de energia. Elas operam de forma autônoma, contando com sistemas supervisórios para o controle e supervisão remotos. Dentro das subestações, funções como comutação, proteção e controle também são desempenhadas. Um dos problemas que as subestações resolvem é a interrupção de curtos-circuitos, utilizando para isso disjuntores. Existem diversos tipos de subestações: as de transmissão, que conectam linhas de transmissão com diferentes voltagens; as de distribuição, que ligam linhas de transmissão a linhas de distribuição e regulam a tensão; e as subestações coletoras, que são utilizadas na geração de energia eólica para conectar a geração às linhas de transmissão. Portanto, as subestações são fundamentais para garantir a confiabilidade e a qualidade do fornecimento de energia (**??**).

Figura 2 – Reatores de Núcleo de ar em uma subestação de energia (**??**)

.

Dentre os diversos equipamentos que compõem a subestação de energia, um em particular será objeto de estudo neste trabalho: os reatores de núcleo de ar (Figura **??**).

Contudo, o interesse reside em seu formato geométrico, que será tratado posteriomente. Nota-se que possui um corpo cilíndrico. Em seu topo, pode possuir ou não uma proteção circular contras as intempéries. Estes dispositivos são essenciais para a regulação da reatividade e controle de correntes indutivas em sistemas de transmissão e distribuição de energia. Diferentemente dos reatores de núcleo de ferro, os reatores de núcleo de ar oferecem vantagens como uma resposta mais linear e a capacidade de operar sem saturação magnética, o que os torna ideais para aplicações onde é necessário um comportamento previsível e estável (**??**).

1.2 Realidade Virtual

RV pode ser definida como um ambiente gerado a partir de um sistema computacional em que o usuário não apenas se sente dentro do contexto artificial, como também possibilita ao usuário a consciência de que pode navegar e interagir neste ambiente virtual. Trata-se, por isso, de uma interessante interface entre um ser humano e um computador. Nela, é possível que haja navegabilidade, interações e, principalmente, imersão em um ambiente sintético, gerado computacionalmente por meio de canais multisensoriais. Ou seja, podemos concluir que para que haja RV é necessário que existam esses três conceitos: interação, imersão e navegação (**??**).

Pode-se classificar a RV de duas formas diferentes: Imersiva e Não-imersiva. Na primeira, cria-se uma RV que isole o usuário do mundo real. Seus sentidos são bloqueados para recepção dos estímulos do seu entorno, para que sejam direcionados àqueles oriundos do mundo fictício. Para tanto, uma ampla variedade de equipamentos, como fones de ouvidos, luvas de dados, óculos de RV, entre outros, são empregados para os resultados esperados. Já a RV Não-Imersiva não se isola do mundo real. Ou seja, o usuário tem consciência de que está em um ambiente artificial. De modo similar, uma ampla variedade de equipamentos, convencionais e não-convencionais, é empregada para gerar essa interação, incluindo dispositivos do cotidiano, como mouses, monitores de computador (**??**).

O surgimento da RV data de 1963, em que foi apresentado uma aplicação de manipulação de objetos tridimensionais em um computador, denominada Sketchpad (**??**). A aplicação conseguia reproduzir interatividade por meio de uma caneta óptica, que era utilizada para seleção de objetos projetados em uma tela. Neste trabalho, surgiram alguns dos principais termos da RV, como representação visual, dispositivos especiais, e interações em tempo real. Em 1968, o mesmo autor do trabalho anterior, Sutherland, publicou outro trabalho marcante para a história: A Head-Mounted Three Dimensional Display (**??**). Nele, foi introduzido o conceito de imersividade para uma RV; no caso, um capacete capaz de projetar fotos diretamente aos olhos do usuário, assim como rastrear o movimento da cabeça, afim de que este movimento fosse responsivo no ambiente virtual. Seguido a isso,

uma série de outros equipamentos foram desenvolvidos a fim de sofisticar as soluções do ramo e propor novos modelos. A tendência sempre prevaleceu de surgir ferramentas mais simples e acessíveis ao usuário final (**??**).

Além das aplicações lúdicas, a RV atua de maneira séria e efetiva em diversas áreas técnicas. Como ferramenta para treinamento de operadores em ambientes de risco e de difícil simulação, torna-se uma alternativa viável para capacitação. O trabalho de (**??**) demonstra bem essa ideia ao investigar técnicas de RV para que uma pessoa comum, sem treinamento anterior, possa, de maneira segura à uma subestação em funcionamento e à própria pessoa e todos ao seu redor, interagir com um transformador de energia, executando todos procedimentos que teria em um ambiente real, contudo de maneira virtual. Este treinamento já seria uma base interessante para que um profissional pudesse agilizar seu treinamento no mundo real ou mesmo absorvê-los. Em trabalhos de alta periculosidade tudo é crítico; então, qualquer intervenção que possa atenuar os riscos inerentes à natureza do trabalho, traz grandes avanços ao processo de profissionalização de técnicos e pessoas interessadas.

1.3 Redes Neurais Artificiais

A RNA trata-se de um conjunto de técnicas que buscam simular o funcionamento do cérebro humano, a partir de algoritmos computacionais, a fim de resolver problemas específicos. Sua eficiência está relacionada com a quantidade de interações que as unidades de processamento que o compõem realizam entre si. Comparam-se as RNA à mente humana devido a sua capacidade de aprendizado, podendo generalizar funções a partir de alguns exemplos informados, e delas prever o comportamento de valores para os quais não foi fornecido a resposta esperada. A base do funcionamento da RNA está no conceito do neurônio artificial (NA), que se traduz como uma pequena unidade de processamento, capaz de receber um sinal simples, e a partir dele gerar uma resposta. De acordo com (**??**), a primeira noção de NA surge em 1943, no trabalho Warren McCulloch e Walter Pitts, no artigo: “A Logical Calculus of the Ideas Immanent in Nervous Activity”.

Matematicamente, o NA recebe um valor de entrada, realizando com ele o produto desse valor a um outro denominado peso. O resultado é comparado com um diferenciador: se for maior, será dada a saída verdadeira; se menor, falso. Na Figura **??**, esse processo é demonstrado de maneira esquemática. A operação descrita, chamada também de threshold logic, i.e., lógica limiar, em tradução livre, mimetiza o funcionamento de um componente eletrônico chamado transistor, base do funcionamento dos processadores computacionais modernos, em que a passagem de corrente elétrica por ele é interrompida ou permitida com base no sinal de entrada (**??**).

Aprofundando-se na álgebra relacionada do NA, pode-se visualizar o esquema do

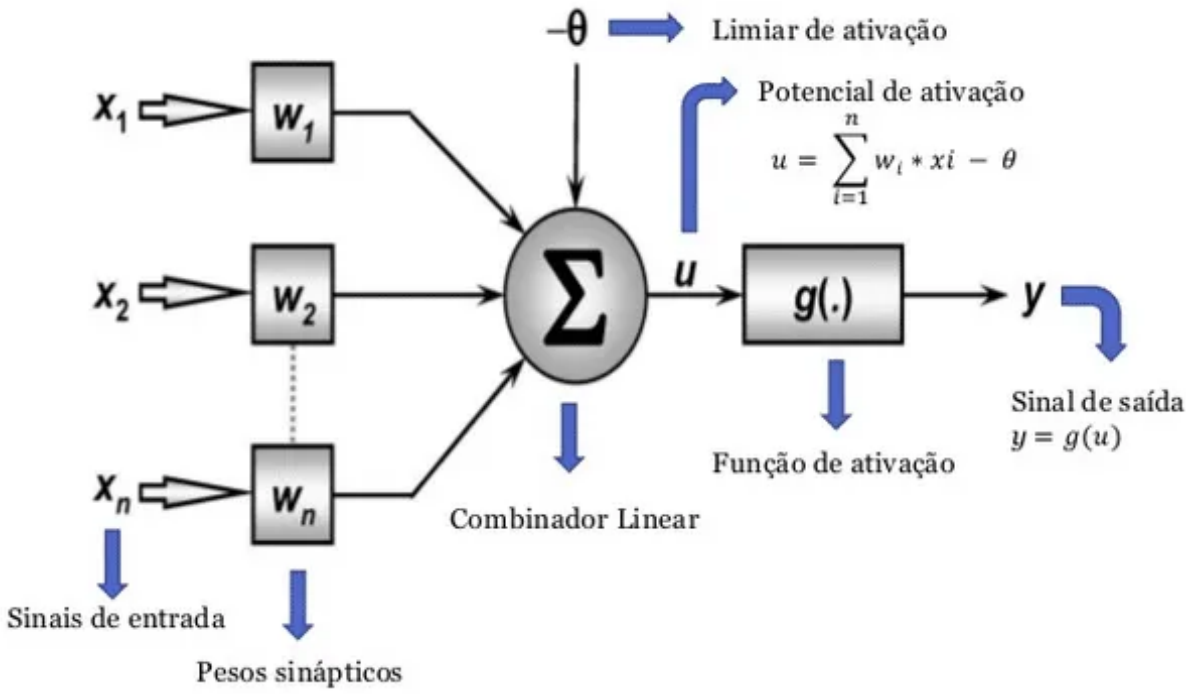


Figura 3 – Representação de um NA (**??**)

.

seu funcionamento na Figura **??**, e também acompanhar a definição de cada envolvido nesta operação (**??**):

- **Sinais de entrada**: Constituem-se os sinais, ou valores, externos ao modelo, muitas vezes submetidos a algum tratamento prévio, responsáveis por alimentar a rede.
- **Pesos sinápticos**: Também chamados apenas de pesos, ponderam os sinais de cada entrada da rede.
- **Combinador linear**: Operação aritmética envolvendo os sinais de entrada a fim de gerar um potencial de ativação.
- **Bias**: O bias é um valor adicional que é somado à combinação linear das entradas antes de passar pela função de ativação, visando ajustá-la para que os dados melhor se adaptem à rede.
- **Limiar de ativação**: Também denominado threshold, determina o nível adequado em que o resultado obtido pelo combinador linear pode acionar a ativação.
- **Potencial de ativação**: É o resultado decorrente da discrepância entre o valor gerado pelo combinador linear e o limiar de ativação. Se esse resultado for positivo, indicando $u \geq 0$, o neurônio gera um potencial de excitação; caso contrário, o potencial será inibitório.

- **Função de ativação**: Sua função é restringir a saída de um neurônio dentro de um determinado intervalo de valores.
- **Sinal de saída**: Resultado final da saída, que pode ser utilizado como entrada para outros neurônios interligados sequencialmente.

Em suma, a RNA, enquanto um conjunto de NA, pode ser entendido como um processador robusto, distribuído de maneira paralela, com pequenas unidades de processamento. Sua semelhança ao cérebro humano, deve-se, portanto, à capacidade de aprendizado e aos sinais sinápticos, existentes entre os neurônios, responsáveis pelo processo de armazenamento de conhecimento (**??**).

De acordo com (**??**), em 1958, após à concepção do NA, a primeira RNA a se notabilizar e a continuar sendo utilizada até o presente momento trata-se da Perceptron (**??**). Este algoritmo possuía a capacidade de alterar os pesos dos neurônios, conforme o avanço do processamento da rede, de modo a resolver problemas da classificação linear. Seguido a ela, em 1960, houve a concepção da rede Adaline (**??**). Diferentemente da perceptron, esta rede já se mostrava capaz de produzir resultados que iam além dos valores binários, sendo capaz de gerar respostas de valores presentes em todo o conjunto dos números reais. O cálculo da rede Adaline era baseado na regra Delta, que era capaz de aproximar os valores dos pesos gerados para aqueles valores com menor erro possível. Ambas as redes, contudo, apresentavam uma limitação quanto à modificação dos pesos em todas suas multicamadas. Para esse cenário, foi proposto o conceito de backpropagation, ou, em tradução livre, retropropagação do erro (**??**). A ideia da retropropagação se inicia com o feedforward, em que todos os dados de entrada de uma RNA são transmitidos do início à camada de saída, sem nenhuma alteração de peso. Desta forma, calcula-se o erro total dessa primeira passagem, comparando o valor resultante com o esperado. Essa métrica de erro torna-se o guia para o ajuste dos pesos entre as conexões de cada neurônio. Com ela, é feito o caminho de volta, e calculado um gradiente de erro, que será o fator decisivo para a atualização dos pesos durante todo o treinamento.

A topologia básica de uma rede neural é dividida em três níveis de camada: a camada de entrada, a camada oculta e a camada de saída, conforme mostrado na Figura **??**. A camada de entrada é, naturalmente, aquela que recebe os dados de entrada. Nela, cada nó representa uma característica ou atributo dos dados que estão sendo alimentados na rede neural. Por exemplo, em uma aplicação de reconhecimento de imagens, cada nó na camada de entrada pode representar um pixel da imagem. A camada oculta, que não se restringe a apenas uma, podendo ser várias, representa um processo intermediário entre a camada de entrada e a camada de saída. Cada neurônio em uma camada oculta recebe entradas das camadas anteriores, realiza algum tipo de transformação não linear dessas entradas e passa o resultado para a próxima camada. A presença de múltiplas camadas

ocultas permite que a rede aprenda representações complexas e abstratas dos dados. Por bim, a camada de saída representa a camada final, e ela é responsável por gerar as saídas desejadas. A estrutura da camada de saída depende do tipo de problema que está sendo resolvido. Por exemplo, em um problema de classificação, cada nó na camada de saída pode representar uma classe diferente, e a saída pode ser interpretada como a probabilidade de pertencer a cada classe (**??**).

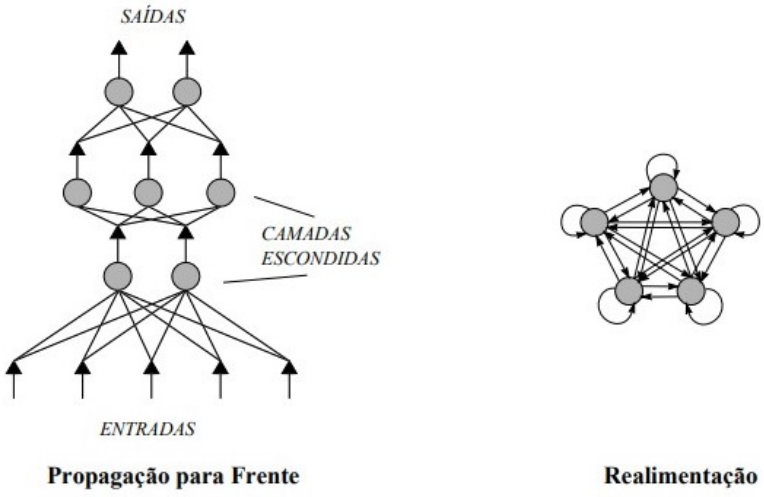


Figura 4 – Topologia básica de uma rede neural (**??**)

.

Essencialmente, as redes neurais aprendem iterativamente ajustando os pesos de suas conexões através do processo de treinamento, onde são apresentados a um conjunto de dados de entrada e as correspondentes saídas desejadas. Com o tempo, a rede neural é capaz de aprender a mapear efetivamente os padrões nos dados de entrada para as saídas desejadas, tornando-se assim capaz de realizar tarefas como reconhecimento de padrões, classificação, regressão, entre outros (**??**).

Contudo, para problemas envolvendo aprendizado por meio de imagens, a forma de lidar com as informações é diferente. De fato, imagens são dados, organizados em formato de matrizes de duas dimensões ou três dimensões (se for considerada a camada de cores), em que cada unidade de informação se chama pixel, conforme se nota no exemplo da Figura **??**. Para processar esses dados, um tipo de RNA destaca-se: a Rede Neural Convolucional (RNC). Comparando ambas, nota-se que a RNA é composta por camadas intrinsecamente conectadas, em que cada neurônio de uma camada se conecta a todos os neurônios da camada seguinte. Isso faz com que ela seja adequada para dados vetorizados e para problemas de classificação e regressão sem uma estrutura espacial ou temporal específica. Por sua vez, a RNC recorre a dois tipos de camadas (Figura **??**): a convolucional, cujo objetivo é extrair características locais, e a de pooling, responsável por manter a estrutura espacial essencial (**??**).

A camada de convolução, portanto, é responsável pela feature extractor, i.e., pela

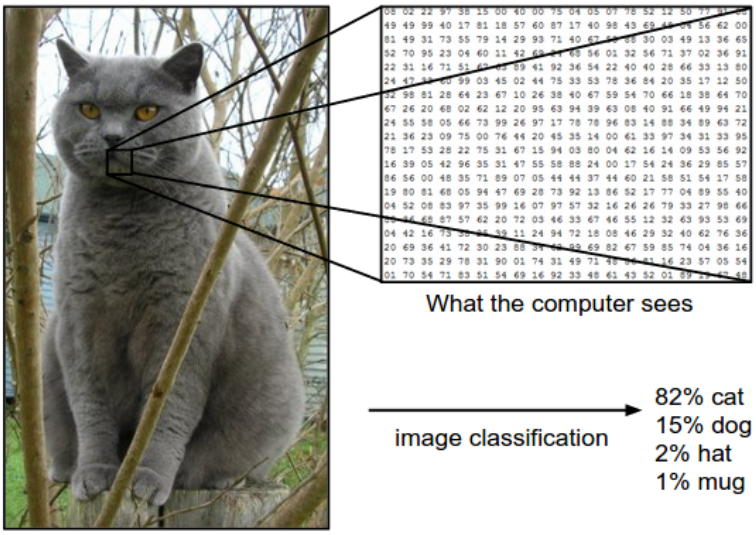


Figura 5 – Exemplificação de como uma imagem nada mais é que uma matriz, e como a filtragem busca padrões pré-estabelecidos dentro da imagem (**??**)

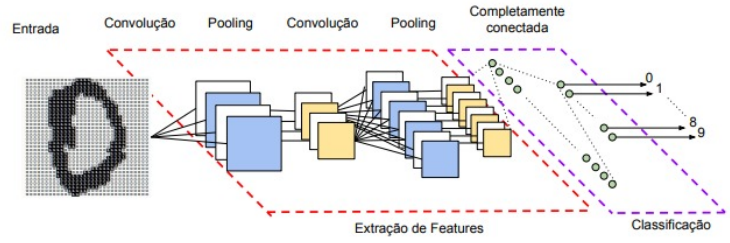


Figura 6 – Esquema de funcionamento de uma RNC (**??**)

.

extração de características de interesse em uma imagem. Basicamente, são aplicados filtros, também chamados de kernel, à imagem a fim de buscar padrões. São comumente empregados filtros de detecção de bordas (edge detection), desfoque (blur), nitidez (sharpen). Conforme na Figura **??**, cada elemento do filtro é multiplicado pelo elemento de mesma posição na região em que o filtro está sendo aplicado naquele instante. Ao final dessas operações matriciais, adiciona-se os resultados dessas multiplicações para ter um único valor como saída, que será o pixel correspondente na imagem filtrada (**??**) .

A camada de pooling é aplicada após cada camada convolucional, com o objetivo de reduzir as dimensões das imagens enquanto mantém a profundidade do volume. Esse processo promove a invariância da Rede Neural Convolucional (RNC) às transformações geométricas, permitindo à rede reconhecer objetos na imagem independentemente de sua posição. Além disso, o uso da camada de pooling contribui para diminuir significativamente o custo computacional da rede. Existem diferentes tipos de pooling, como o max pooling, que seleciona o valor máximo em uma região específica da imagem, e o average pooling, que calcula a média dos valores nesta região. O max pooling é o mais comum, pois

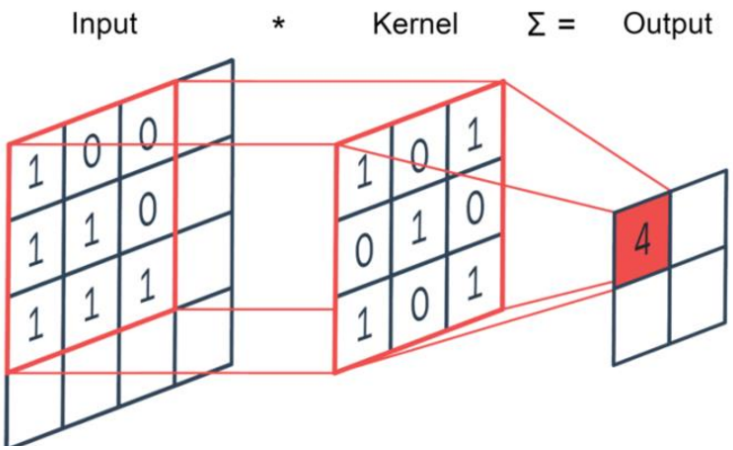


Figura 7 – Operação de filtragem na camada convolucional. Nota-se que o filtro é uma multiplicação de matrizes (**??**)

preserva as características mais salientes das imagens, enquanto reduz o ruído. A camada de pooling, ao reduzir o tamanho espacial dos mapas de características, também ajuda a evitar um problema comum no aprendizado de máquina, em que um modelo aprende não apenas os padrões relevantes dos dados de treinamento, mas também o ruído e as particularidades específicas desses dados. Chama-se este problema de overfitting. Evitando-o, a rede consegue construir um treinamento mais generalizável e que possa gerar resultados relevantes para dados diferentes do modelo treinado (**??**).

Por último, a camada totalmente conectada emprega uma função de ativação na sua camada de saída para realizar a classificação. Ser "totalmente conectada"significa que todos os neurônios da camada anterior estão ligados a todos os neurônios da camada subsequente. Ela recebe as saídas da camada de convolução e da camada de pooling e, em seguida, utiliza essas informações para determinar a classe à qual a imagem de entrada pertence (**??**).

As RNC representam por si só uma revolução no campo da visão computacional, tornando-se a base para uma variedade de aplicações, desde reconhecimento de imagens até análise de vídeos. Entre as arquiteturas mais notáveis estão a LeNet, AlexNet, VGGNet, GoogLeNet e ResNet, cada uma contribuindo com avanços significativos em profundidade, eficiência e precisão. Além dessas, uma destaca-se em especial: a YOLO (“You Only Look Once”), uma abordagem inovadora para treinamento e detecção de objetos (**??**).

1.3.1 You Look Only Once

YOLO, uma Rede Neural Convolutiva (RNC), é conhecida por sua eficácia e precisão na detecção de objetos em imagens e vídeos. Desenvolvida por Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi em 2015, a arquitetura do YOLO foi apresentada em sua primeira versão no artigo "You Only Look Once: Unified, Real-Time Object Detection"(**??**).

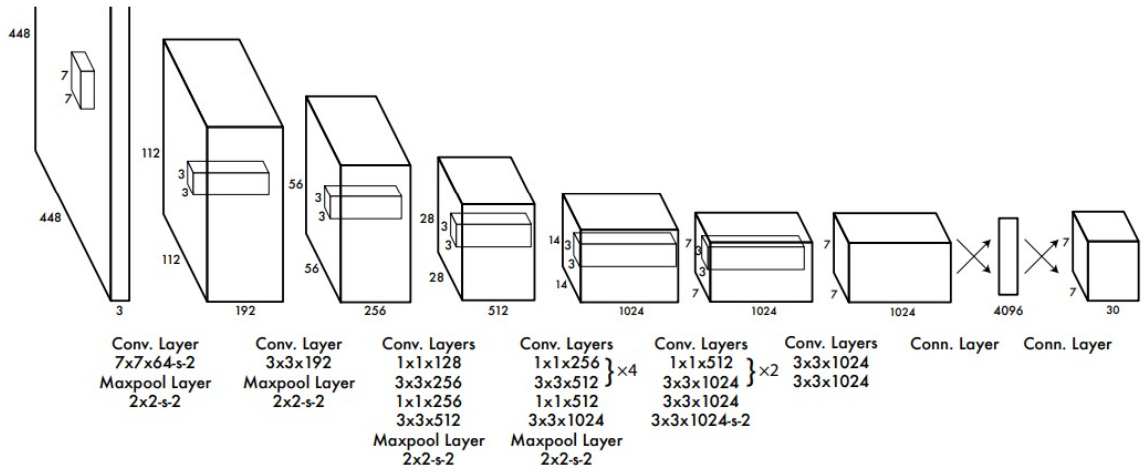


Figura 8 – Disposição das 24 camadas convolucionais e das 2 camadas totalmente conectadas (**??**)

A arquitetura do YOLO é singular: ao contrário de outras RNCs que fazem múltiplas passagens pela imagem em busca de objetos, o YOLO analisa a imagem inteira de uma vez, justificando seu nome, que em tradução livre significa “você olha apenas uma vez”. Essa abordagem permite ao YOLO aplicar uma única rede neural à imagem completa. Durante o processamento, a imagem é dividida em regiões menores, e a rede prevê caixas delimitadoras, as probabilidades de existência de objetos nessas caixas, e a classe provável de cada objeto. Sua classificação, enquanto rede neural, vai além da RNC, pois ela utiliza uma arquitetura conhecida como Darknet, um tipo de rede neural profunda (variação da RNC) e sua implementação original foi desenvolvida em C, embora agora esteja disponível em várias outras linguagens de programação, graças ao apoio da comunidade e de empresas. Quando utilizada para treinamento sua estrutura básica, estabelecida em sua primeira versão, segue, evidentemente, o padrão de uma RNC. As camadas convolucionais iniciais da rede extraem características da imagem, enquanto as camadas totalmente conectadas preveem as probabilidades de saída e as coordenadas (Figura **??**). Em sua primeira versão, a rede possuia 24 camadas convolucionais seguidas por 2 camadas totalmente conectadas (**??**).

Embora existam várias versões do YOLO, há um funcionamento geral da arquitetura. O primeiro passo do YOLO é dividir a imagem em uma grade de células S por S. Nas versões iniciais, essa grade era de 13x13, totalizando 169 células, enquanto nas versões mais recentes é de 19x19. Cada célula é responsável por prever/detectar 5 caixas delimitadoras, pois pode haver múltiplos objetos em uma única célula. Portanto, na versão de exemplo do YOLO, há um total de 845 caixas delimitadoras (13x13x5). Na Figura **??**, esse processo é representado. Notam-se diversas caixas, aquelas com bordas mais grossas indicam que a probabilidade de a demarcação estar representando o objeto que se propõe é alta. (**??**).

A partir da YOLOv3, foram introduzidos conceitos para definir algumas partes

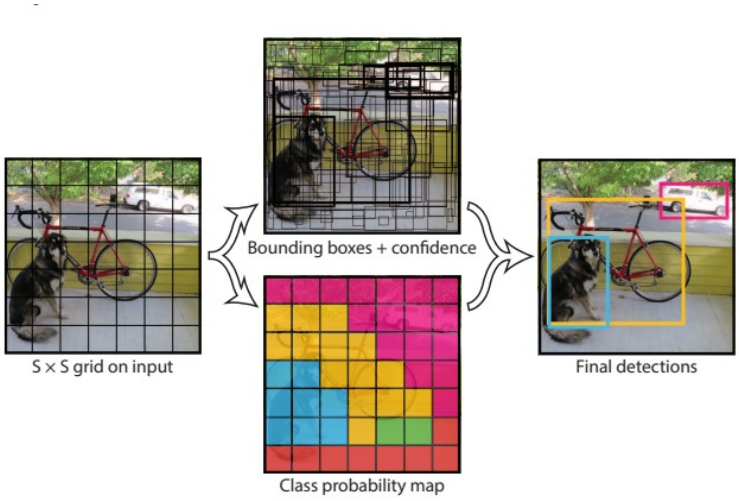


Figura 9 – Processo de predição da Yolo (**??**)

de sua arquitetura. Primeiramente, define-se a camada de backbone, ou espinha dorsal. Basicamente, é nesta etapa que ocorre a extração de características significativas para o processo de classificação. Ela captura características hierárquicas em diferentes escalas, com características de nível inferior (como bordas e texturas) nas camadas iniciais e características de nível superior (como partes de objetos e informações semânticas) nas camadas mais profundas. O neck, ou pescoço, conecta a camada backbone à head, ou cabeça, agregando e refinando as características extraídas, muitas vezes focando em melhorar a informação espacial e semântica em diferentes escalas. A head processa as características fornecidas pelo neck, gerando previsões para cada candidato a objeto. Resumindo, a partir da YOLOv3, a arquitetura foi dividida em três partes: backbone para extração de características; neck para agregar e refinar características; e head para fazer previsões. A espinha dorsal usa uma CNN para capturar características hierárquicas, o pescoço melhora a informação espacial e semântica, e a head gera as previsões finais, que são refinadas por pós-processamento para eliminar sobreposições (**??**).

Atualmente, a oitava versão da YOLO, ou, YOLOv8 , é considerada estado da arte em se tratando de análise e treinamento de imagens (**??**). YOLOv8 traz uma série de avanços significativos em relação às versões anteriores, destacando-se pela melhoria no desempenho e precisão das detecções. A arquitetura foi otimizada para equilibrar velocidade e precisão, utilizando backbones modernos como CSPDarknet para uma extração de características mais eficiente. O processo de treinamento foi acelerado e aprimorado com melhores métodos de regularização e otimização, enquanto a facilidade de uso e integração foi aprimorada, atendendo às demandas tanto de dispositivos de alta performance quanto de dispositivos móveis com menor capacidade de processamento (**??**).

1.3.2 Batch

Um dos aspectos cruciais do funcionamento da YOLO é o conceito de "batch"(em tradução livre, "lote") durante o treinamento da rede neural. Ao agrupar várias imagens em lotes para processamento simultâneo, a YOLO aproveita a capacidade de processamento paralelo das GPUs, acelerando significativamente o treinamento. Durante a propagação direta, cada imagem no lote é processada pela rede neural para gerar previsões de detecção de objetos. Em seguida, a perda é calculada em relação às anotações verdadeiras, e os pesos da rede são atualizados para minimizar essa perda, usando algoritmos de otimização como o gradiente descendente. Esse processo é repetido para vários lotes de imagens até que a rede convirja para uma solução adequada. Assim, o uso eficiente de lotes na YOLO não apenas acelera o treinamento, mas também contribui para a robustez e eficácia dos modelos de detecção de objetos resultantes (**??**).

1.3.3 Otimizadores

O objetivo dos algoritmos de aprendizado de máquina supervisionados é otimizar a redução de uma função objetivo, geralmente chamada de função de custo J. Reduzir essa função é crucial, pois a rede precisa aprender os pesos que melhor representam o relacionamento entre os dados, formando um modelo preditivo para fazer previsões em novos conjuntos de dados. O algoritmo de otimização mais simples para encontrar esses pesos é o Gradiente Descendente (GD), onde se realiza pequenos passos em direção ao mínimo global da função de custo. No entanto, o GD enfrenta dificuldades com conjuntos de dados muito grandes devido à sua abordagem em lote, tornando o treinamento computacionalmente caro. Uma alternativa popular é o Gradiente Descendente Estocástico (SGD, sigla para a expressão em inglês “Stochastic Gradient Descent”), que aplica atualizações de peso com base em amostras aleatórias de dados de treinamento, resultando em convergência mais rápida. O SGD possui três parâmetros importantes que podem influenciar no treinamento da rede neural: a taxa de aprendizagem, que determina quão rápido o otimizador tentará treinar a rede neural, com uma taxa muito alta podendo levar a falhas no treinamento e uma taxa muito baixa resultando em treinamento lento; o momentum, que indica quanto a direção da mudança de peso anterior deve influenciar na etapa atual, acelerando o treinamento ao permitir que o otimizador persista em direções com histórico de melhoria; e o decay, utilizado para diminuir a taxa de aprendizagem conforme os erros diminuem durante o treinamento, evitando oscilações excessivas e permitindo uma convergência mais suave do modelo (**??**).

Além do SGD, outro algoritmo de otimização recorrente é o Adam (“Adaptive Moment Estimation”, em tradução livre, “Estimação Adaptativa de Momento”). O otimizador Adam é um algoritmo popular de otimização de gradientes utilizado em treinamento de redes neurais e outras tarefas de aprendizado de máquina. Ele calcula e mantém esti-

mativas adaptativas dos momentos do gradiente, incluindo o primeiro momento (média) e o segundo momento (variância). Essas estimativas são utilizadas para calcular uma correção de gradiente, chamada de momento Adam, que ajusta adaptativamente a taxa de aprendizagem para cada parâmetro. Além disso, o Adam inclui termos de regularização L2 para evitar que os pesos cresçam excessivamente durante o treinamento. Essa abordagem combinada de momentos adaptativos e ajuste de taxa de aprendizagem torna o Adam eficiente, fácil de implementar e eficaz em uma variedade de problemas de otimização (**??**).

Há também uma variação do Adam, que seria o AdamW. Esta variante do otimizador Adam que inclui regularização de peso ponderada diretamente no processo de atualização dos parâmetros, ao contrário do Adam tradicional, onde a regularização é aplicada separadamente após a atualização. Isso ajuda a corrigir problemas de "decaimento de pesos"em modelos de aprendizado profundo, resultando em melhor desempenho e capacidade de generalização (**??**).

1.3.4 Precisão e Recall

A fim de avaliar o desempenho de um treinamento na arquitetura YOLO, é preciso entender os resultados fornecidos pelo modelo. De acordo com (**??**), basicamente a YOLO utiliza a métrica chamada Average Precision (AP) (“Precisão Média”, em tradução livre). Ela se baseia no conceito de IoU (“Intersection over the Union”, Intersecção sobre a União, em tradução livre), que calcula uma razão entre a interseção da detecção feita pelo algoritmo com relação à marcação (bounding box) realizada em cima da área dividida pela união dessas duas áreas (Figura **??**). Essa razão poderá ser comparada com um valor pré-estabelecido, o thresholds, que será referido por L. A partir desse valor, é possível que no processo de detecção retorne três diferentes resultados na pesquisa pela classe desejada. São eles: Verdadeiro Positivo (VP), Falso Positivo (FP) e Falso Negativo (FN). VP trata-se dos resultados considerados corretos que a rede neural retorna, que seriam todos resultados que IoU $>$ L. FP já seriam os resultados em que IoU $<$ L, que são tidos como incorretos.FN, por sua vez, trata dos resultados totalmente fora do esperado.

$$\text{Recall} = \frac{VP}{VP + FN} \tag{1}$$

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{2}$$

Com esses valores, pode-se calcular os resultados da saída que o treinamento da rede YOLO fornece. Em (1), tem-se o cálculo da Recall, que calcula a capacidade da rede de detectar todos os objetos relevantes em uma imagem. Já a Precisão (2), refere-se à capacidade da rede de encontrar apenas resultados relevantes.

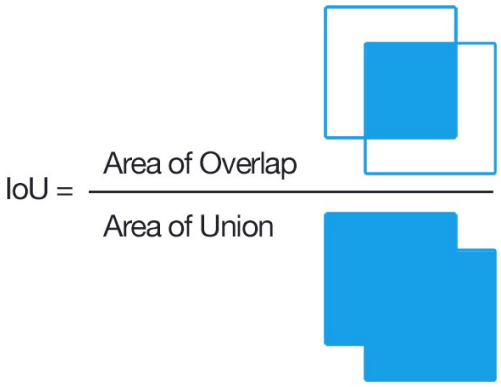


Figura 10 – Cálculo de IoU. (**??**)

A arquitetura YOLO disponibiliza um dataset, ou seja, um banco de imagens e weights, comum em todas as versões, chamado de COCO ("Common Objects in Context", que em tradução livre seria "Classes Comuns de Objetos") com classes pré-treinadas e imagens para realização de treinamentos. A partir dele, verificou-se por meio de testes a eficiência das quatro últimas versões da YOLO, a fim de identificar se nas mais recentes houve melhoras significativas em termos de performance e precisão. Na Figura **??**, é apresentado o comparativo das versões. Nota-se que a v8, para um menor número de parâmetros que as demais, apresentou uma mAP50-95, maior que nas versões v5, v6 e v7. Além disso, com relação a velocidade de processamento, a v8 também se sobressai, com maior rapidez no processamento, ao processar maior quantidade de imagens para um mesmo intervalo de tempo (**??**).

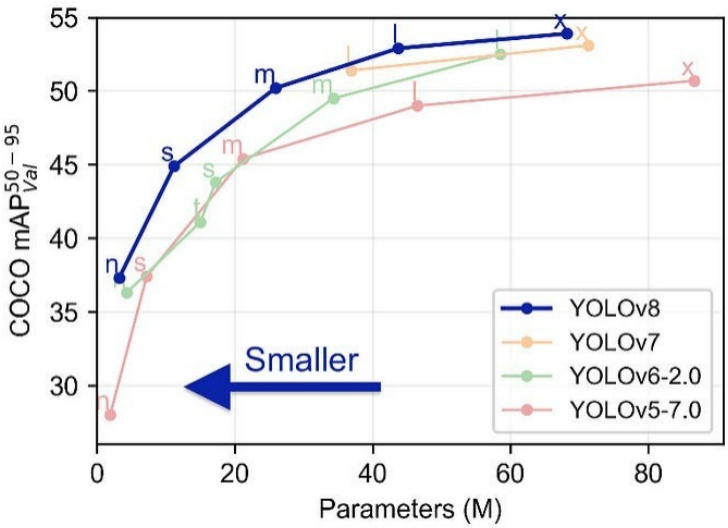


Figura 11 – Aumento de desempenho da precisão média, versão 5 para 8. (**??**)

1.4 Considerações finais

Neste capítulo, foi apresentado o arcabouço teórico necessário para o entendimento da proposta dessa dissertação. A apresentação das redes neurais, e em específico, o modo que a arquitetura da YOLOv8 sobrepõe-se em termos de eficiência em relação às demais arquiteturas abordadas em outros trabalhos científicos, demonstra o direcionamento assertivo deste trabalho. Além disso, a apresentação da Realidade Virtual como uma disciplina inovadora e muito útil para servir à diversos propósitos dentro da indústria e ciência, corroboram para o entendimento da proposta desta dissertação.

Referências

LIMA, M. E. A.; OLIVEIRA, R. C. Precarização e acidentes de trabalho: os riscos da terceirização no setor elétrico. *Revista Brasileira de Saúde Ocupacional*, SciELO Brasil, v. 46, p. e6, 2021. Nenhuma citação no texto.

OĞCU, G.; DEMIREL, O. F.; ZAIM, S. Forecasting electricity consumption with neural networks and support vector regression. *Procedia-Social and Behavioral Sciences*, Elsevier, v. 58, p. 1576–1585, 2012. Nenhuma citação no texto.

PALMEIRA, E. G. Q. et al. O uncanny valley das mãos virtuais em aplicações de realidade virtual imersiva: uma revisão sistemática da literatura. *Revista Ibérica de Sistemas e Tecnologias de Informação*, Associação Ibérica de Sistemas e Tecnologias de Informacao, n. E31, p. 497–512, 2020. Nenhuma citação no texto.

RANDOLPH, J. Electric power substations engineering [book reviews]. *IEEE Power and Energy Magazine*, IEEE, v. 11, n. 3, p. 103–105, 2013. Nenhuma citação no texto.

ZHOU, K.; FU, C.; YANG, S. Big data driven smart energy management: From big data to big insights. *Renewable and sustainable energy reviews*, Elsevier, v. 56, p. 215–225, 2016. Nenhuma citação no texto.

A Apêndice A

A.1 api-rv.py

```
from ultralytics import YOLO
from flask import Flask, request, jsonify
import os

app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    image_path = data['image_path']
    model = YOLO("best.pt")
    results = model.predict(source=image_path)

    response = []
    for result in results:
        for box in result.boxes:
            if model.names[box.cls.item()] == 'reator':
                response.append({
                    'class_name': model.names[box.cls.item()],
                    'bbox': box.xyxy.tolist()
                })

    return jsonify(response)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

B Apêndice B

B.1 ModelLoaderMenu.cs

```
using UnityEditor;
using UnityEngine;
using System.Collections.Generic;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

public class ModelLoaderMenu : EditorWindow
{
    private string imagePath = "";
    private Texture2D loadedImage;

    [MenuItem("Automa  o␣Ambiente/Processar␣Imagem")]
    public static void ShowWindow()
    {
        ModelLoaderMenu window = GetWindow<ModelLoaderMenu>("Processar␣
            Imagem");
        window.minSize = new Vector2(300, 300);
        window.maxSize = new Vector2(300, 300);
    }

    void OnGUI()
    {
        GUILayout.Label("Carregar␣e␣Processar␣Imagem", EditorStyles.
            boldLabel);

        if (GUILayout.Button("Selecionar␣Imagem"))
        {
            imagePath = EditorUtility.OpenFilePanel("Selecione␣uma␣
                imagem", "", "png,jpg,jpeg");
            Debug.Log("imagePath:␣" + imagePath);
        }

        imagePath = EditorGUILayout.TextField("Caminho␣da␣Imagem",
            imagePath);

        if (GUILayout.Button("Processar"))
        {
            ProcessImage();
        }
```

```
    }

    async void ProcessImage()
    {
        if (string.IsNullOrEmpty(imagePath))
        {
            Debug.LogError("Nenhuma␣imagem␣carregada␣para␣processar.");
            return;
        }

        Vector3 basePosition = Vector3.zero;
        Vector3 spacing = new Vector3(10, 0, 0);

        List<DetectionResult> detectionResults = await
           SendImageForPrediction(imagePath);

        if (detectionResults != null && detectionResults.Count >= 4)
        {
            string modelPath = "Assets/Model/FILTER_F10.fbx";
            GameObject model = AssetDatabase.LoadAssetAtPath<GameObject
               >(modelPath);

            if (model != null)
            {
                for (int i = 0; i < detectionResults.Count; i++)
                {
                    GameObject modelInstance = (GameObject)PrefabUtility
                       .InstantiatePrefab(model);
                    if (modelInstance != null)
                    {
                        modelInstance.transform.position = basePosition
                           + spacing * i;
                        modelInstance.transform.rotation = Quaternion.
                           Euler(-90, 0, 0);
                        Debug.Log("Reator␣inserido␣na␣posi   o:␣" +
                           modelInstance.transform.position);
                    }
                    else
                    {
                        Debug.LogError("Erro␣ao␣instanciar␣o␣modelo␣
                           virtual.");
                    }
                }
            }
            else
            {
```

```
                Debug.LogError("Erro ao carregar o modelo virtual.
                    Verifique o caminho.");
            }
        }
    }

    async Task<List<DetectionResult>> SendImageForPrediction(string
        imagePath)
    {
        string url = "http://192.168.12.13:5000/predict";
        var httpClient = new HttpClient();
        var content = new StringContent($"{{\"image_path\":\"{imagePath
            }\"}}", Encoding.UTF8, "application/json");

        HttpResponseMessage response = await httpClient.PostAsync(url,
            content);

        if (response.IsSuccessStatusCode)
        {
            string jsonResponse = await response.Content.
                ReadAsStringAsync();
            DetectionResultsWrapper wrapper = JsonUtility.FromJson<
                DetectionResultsWrapper>($"{{\"results\":{jsonResponse}}}
                ");

            return wrapper.results;
        }
        else
        {
            Debug.LogError("Erro ao chamar o endpoint Flask: " +
                response.ReasonPhrase);
            return null;
        }
    }
}

[System.Serializable]
public class DetectionResult
{
    public string ClassName;
    public List<float> BoundingBox;
}

[System.Serializable]
public class DetectionResultsWrapper
{
    public List<DetectionResult> results;
```

```
}
```