CENTRO UNIVERSITÁRIO FEI

BRUNO MOREIRAS DO NASCIMENTO

EDUARDO BORGES BRASIL

FILIPE MORAIS MARANGONI

JEAN CARLOS DE MATOS SANTANA

RECONHECIMENTO DO USO DE EPI POR MEIO DE VISÃO COMPUTACIONAL UTILIZANDO UMA CNN

São Bernardo do Campo

2020

BRUNO MOREIRAS DO NASCIMENTO
EDUARDO BORGES BRASIL
FILIPE MORAIS MARANGONI
JEAN CARLOS DE MATOS SANTANA

RECONHECIMENTO DO USO DE EPI POR MEIO DE VISÃO COMPUTACIONAL UTILIZANDO UMA CNN

Trabalho de conclusão de curso, apresentada ao Centro Universitário da FEI. Orientado pela Professora Doutora Arianne Soares do Nascimento Pereira.

São Bernardo do Campo

2020

Reconhecimento do uso de EPI por meio de visão computacional utilizando uma CNN / Bruno Moreiras do Nascimento...[et al.]. São Bernardo do Campo, 2020.
151 f. : il.

Trabalho de Conclusão de Curso - Centro Universitário FEI.
Orientadora: Prof.ª Dra. Arianne Soares do Nascimento Pereira.

1. Identificação. 2. EPI. 3. CNN. 4. Machine Learning. 5. software. I. Moreiras do Nascimento, Bruno. II. Borges Brasil, Eduardo. III. Morais Marangoni, Filipe. IV. Matos Santana, Jean Carlos de. V. Soares do Nascimento Pereira, Arianne, orient. VI. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

BRUNO MOREIRAS DO NASCIMENTO
EDUARDO BORGES BRASIL
FILIPE MORAIS MARANGONI
JEAN CARLOS DE MATOS SANTANA

RECONHECIMENTO DO USO DE EPI POR MEIO DE VISÃO COMPUTACIONAL UTILIZANDO UMA CNN

Trabalho de Conclusão de Curso, apresentado ao Centro Universitário FEI, como parte dos requisitos necessários para obtenção do título de Bacharel em Engenharia Elétrica.

Comissão julgadora

_____

Arianne Soares do Nascimento Pereira.

_____

Marcelo Gonzaga de Oliveira Parada

_____

Rudolf Theoderich Buhler

São Bernardo do Campo
2020

AGRADECIMENTOS

Agradecemos a nossa professora orientadora Arianne Soares pelo apoio ao nosso projeto.

A todos os nossos professores do curso de Engenharia Elétrica do Centro Universitário FEI pelos conhecimentos passados.

Aos nossos pais e familiares, que sempre estiveram ao nosso lado durante toda a trajetória de formação.

Agradecemos aos servidores da seção de segurança do trabalho do Arsenal de Guerra de São Paulo que ajudaram com a formação do banco de dados.

Por fim, agradecemos a Deus, por nos ajudar a ultrapassar todos os obstáculos ao longo do curso.

RESUMO

Atualmente a utilização de Machine Learning vem evoluindo diariamente, trazendo um avanço tecnológico na análise de dados antes nunca vista. Podemos dizer com muita tranquilidade que estamos em um crescimento exponencial de conhecimento, utilizando ferramentas computacionais para facilitar e melhorar funções antes exercidas pelos seres humanos. Encontramos, diariamente, tecnologias de ponta como o tracking de face utilizado pelo Snapchat e Instagram na aplicação de filtros nas imagens com rostos e a categorização, além de reconhecimento facial no Facebook e Google Fotos, onde é possível distinguir quantas pessoas estão na fotos, assim como quem são essas pessoas, utilizando seus próprios usuários como treinadores de seu algoritmo de reconhecimento. Com esta pequena generalização, este projeto possui a intenção de demonstrar a análise de vídeo em real-time para a automatização de um serviço manual, antes executado por um ser humano, este papel poderá ser levado ao ambiente digital, onde podemos usar métricas e algoritmos de forma a acelerar e manter a segurança de certas instalações físicas. Utilizamos a junção de software e hardware para análise e processamento destas informações para demostrar um sistema de segurança no reconhecimento da utilização de Equipamentos de Proteção Individual (EPIs). O vídeo da face da pessoa analisada foi enviado de um computador, podendo ser, por exemplo, seu computador com webcam ou um Raspberry Pi com módulo de câmera, para um servidor que executou a nossa Convolutional Neural Network (CNN), capaz de identificar e categorizar os EPIs previamente cadastrados e treinados, após a correta identificação foi enviado um sinal de saída, que em nossa plataforma WEB foi sinalizada por um ícone verde, no caso de EPI identificado e vermelho quando não identificado, essa saída poderia ser também uma liberação de acesso para o indivíduo a determinada área. Este sistema pode ser utilizado em diversos locais, levando segurança e um viés integro para o acesso restrito, onde são necessários tais tipos de equipamentos de proteção, impedindo assim problemas de segurança e processos trabalhistas, pois com esta ferramenta seria possível a comprovação da utilização dos equipamentos conforme as normas impostas.

Keywords: Identificação. EPI. CNN. Machine Learning. software.

ABSTRACT

Currently, the use of Machine Learning has been evolving daily, bringing a technological advance in data analysis never seen before. We can say very calmly that we are in an exponential growth of knowledge, using computational tools to facilitate and improve functions previously performed by human beings. We find, on a daily basis, cutting edge technologies such as face tracking used by Snapchat and Instagram when applying filters to images with faces and categorization, in addition to facial recognition on Facebook and Google Photos, where it is possible to distinguish how many people are in the photos, as well like who these people are, using their own users as trainers of their recognition algorithm. With this small generalization, this project intends to demonstrate real-time video analysis for the automation of a manual service, previously performed by a human being, this role can be taken to the digital environment, where we can use metrics and algorithms in order to speed up and maintain the security of certain physical installations. We use the combination of software and hardware for analysis and processing of this information to demonstrate a security system in the recognition of the use of Personal Protective Equipment (PPE).The video of the analyzed person's face was sent from a computer, which could be, for example, his computer with a webcam or a Raspberry Pi with camera module, to a server that ran our Convolutional Neural Network (CNN), capable of identifying and categorize PPE previously registered and trained, after correct identification an exit signal was sent, which on our WEB platform was signaled by a green icon, in the case of identified PPE and red when not identified, this exit could also be a release of access for the individual to a certain area.This system can be used in several places, taking security and an integral bias for restricted access, where such types of protective equipment are needed, thus preventing safety problems and labor processes, since with this tool it would be possible to prove the use of equipment according to the imposed standards.

Keywords: Identification. PPE. CNN. Machine Learning. software.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AWS	Amazon Web Services
Caffe	Convolutional Architecture for Fast Feature Embedding
CNN	Convolutional Neural Network
COCO	Common Object in Context
EPI	Equipamento de Proteção Individual
FN	False Negative
FP	False Positive
HTTP	Hypertext Transfer Protocol
ICE	Interactive Connectivity Establishment
IoU	Intersect Over Union
NAT	Network Address Translator
PPE	Personal Protective Equipment
ROI	Region of Interest
RPN	Region Proposal Network
SDP	Session Description Protocol
SSD	Single Shot Detector
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
WebRTC	Web Real-Time Communication
YOLO	You only look once

SUMÁRIO

1 INTRODUÇÃO

A visão computacional, como o próprio nome sugere, é um campo focado no estudo e automação de tarefas de percepção visual por computadores. Apesar da especificidade dessa subárea de inteligência artificial, o volume de problemas derivados dessa abordagem é bastante extenso.

Mas como é possível, para uma máquina ou computador, identificar elementos em uma fotografia constituída por incontáveis componentes? (ZHAO; CHANG; ITTI, 2016) Para isso temos diversos cálculos matemáticos envolvidos e, normalmente, são realizados diversos treinamentos das redes criadas, onde são apresentadas diversas imagens do que queremos reconhecer e dizemos se o objeto está ou não naquela foto, assim como uma criança aprende a diferenciar esses objetos a máquina também aprende padrões e começa a estimar se o objeto está ou não numa nova imagem.(SHIN et al., 2016)

Na atualidade, dentro deste cenário de evolução tecnológica, temos vários modelos de redes neurais com resultados fantásticos e muito precisos.(REN et al., 2015)

Diante de falhas em processos de fiscalização quanto ao uso de EPI, um fator que permanece em evidência é a importância de uma fiscalização onipresente, para assim garantir segurança aos envolvidos nos diversos processos

A partir destas considerações, visa-se responder a seguinte pergunta: Como aplicar tal tecnologia na resolução dessa falha?

Como foi dito, existem diversos ramos da inteligência artificial, mas a rede neural que mais se destacou na resolução de problemas de visão computacional foi a rede neural convolucional, com excelentes resultados foram obtidos, precisão e velocidade que, como veremos ao longo deste trabalho, permitirão a implementação do nosso projeto.

1.1 MOTIVAÇÃO

Todas as atividades profissionais onde possa existir algum tipo de risco físico para o trabalhador devem ser cumpridas com o auxílio de EPIs, que incluem botas, capacetes, cintos de segurança, luvas, máscaras, óculos, protetores auriculares e outros itens de proteção. Esses acessórios são indispensáveis, por exemplo, em fábricas, processos industriais e químicos em geral, hospitais e construção civil.

O empregador deve orientar os trabalhadores sobre as normas de segurança no trabalho, exigir e fiscalizar o uso do EPI. A recusa do empregado em utilizar o equipamento, não exime a culpa do empregador quanto aos danos causados ao trabalhador em eventual acidente. (RODRIGUES; SANTOS, 2020)

A obrigatoriedade do uso do EPI encontra amparo legal na Constituição Federal de 1988, Consolidação das Leis do Trabalho – CLT e NR – 06. Segundo o artigo 7º, inciso XXII, da Constituição Federal de 1988, dispõe que:

> "Art. 7º São direitos dos trabalhadores urbanos e rurais, além de outros que visem à melhoria de sua condição social: XXII – redução dos riscos inerentes ao trabalho, por meio de normas de saúde, higiene e segurança."

Este artigo impõe como dever do empregador reduzir os riscos inerentes ao trabalho e nesse sentido, está o fornecimento de EPIs e a garantia de utilização por parte do empregado, mediante fiscalização do empregador. No artigo 166 da CLT temos:

> "Art. 166 – A empresa é obrigada a fornecer aos empregados, gratuitamente, equipamento de proteção individual adequado ao risco e em perfeito estado de conservação e funcionamento, sempre que as medidas de ordem geral não ofereçam completa proteção contra os riscos de acidentes e danos à saúde dos empregados".

A CLT traz de forma clara, é obrigação exclusiva da empresa o fornecimento do EPI, e o mesmo deve ser feito de forma gratuita, e que seja adequado ao risco que sua atividade ofereça.

A NR – 06 que é o instrumento legal que define e regula o uso e aprovação dos EPIS, estabelece as obrigações do empregador e empregado, conforme o trecho selecionado a seguir:

> "6.6 Responsabilidades do empregador.
> 6.6.1 Cabe ao empregador quanto ao EPI:
>
> a) adquirir o adequado ao risco de cada atividade;
> b) exigir seu uso;
> c) fornecer ao trabalhador somente o aprovado pelo órgão nacional competente em matéria de segurança e saúde no trabalho;
> d) orientar e treinar o trabalhador sobre o uso adequado, guarda e conservação;
> e) substituir imediatamente, quando danificado ou extraviado;
> f) responsabilizar-se pela higienização e manutenção periódica; e,
> g) comunicar ao MTE qualquer irregularidade observada.
> h) registrar o seu fornecimento ao trabalhador, podendo ser adotados livros, fichas ou sistema eletrônico."

Dentre as responsabilidades do empregador, é importante frisar que é encargo dele exigir o uso, orientar e treinar o trabalhador. De nada adianta apenas entregar o EPI e não haver o aspecto educacional envolvido nessa relação. Outro ponto interessante é a responsabilidade apresentada no item h, registrar o fornecimento do EPI ao trabalhador, podendo ser por meio eletrônico. Temos também na NR – 06 as responsabilidades do trabalhador:

> "6.7 Responsabilidades do trabalhador.
> 6.7.1 Cabe ao empregado quanto ao EPI:
>
> a) usar, utilizando-o apenas para a finalidade a que se destina;
> b) responsabilizar-se pela guarda e conservação;
> c) comunicar ao empregador qualquer alteração que o torne impróprio para uso; e,
> d) cumprir as determinações do empregador sobre o uso adequado."

Porém, apesar de toda uma regulamentação que deixa clara ambas as responsabilidades, estima-se que temos, só no Brasil, um trabalhador sofrendo acidente a cada 48 segundos e um morrendo a cada 4 horas. São dados do Ministério Público do Trabalho (MPT) e mostram que em cinco anos mais de 14 mil trabalhadores morreram no exercício da profissão. O número pode ser maior, já que só um em cada sete casos são notificados. (ROCHA, 2020)

Como resultado temos muitos processos trabalhistas sobre esse tema, e somado aos outros, o Brasil detém o posto de campeão mundial, em relação ao número de processos trabalhistas, com estimativas entre 50% e 65% dos processos (dados de 2001 a 2016) de todo o mundo, com somente 2,8% da população mundial. (MARCHESAN, 2020)

E é aqui que entra nosso projeto, temos uma legislação e regulamentação que estabelece as responsabilidades, principalmente de fiscalização e temos nas empresas os técnicos de segurança do trabalho e membros da Comissão Interna de Prevenção de Acidentes - CIPA, então, por que ainda há tantos acidentes, mortes e processos trabalhistas no Brasil? Quando pensamos sobre essa situação e observamos o funcionamento numa empresa, claramente o gargalo está na fiscalização.

Outro fato que nos motivou a fazer tal projeto foi um caso real que ocorre na empresa onde um dos integrantes do grupo trabalha, para que seja liberado o acesso a determinada área é necessário que alguem que está dentro da sala, trabalhando e com EPI, venha até a porta, verifique se quem deseja entrar está com o equipamento correto, e só assim é liberado o acesso.

Respondendo à pergunta anterior, quando pensamos sobre essa situação e observamos o funcionamento numa empresa, o problema pode estar na fiscalização.

É impossível o responsável pela fiscalização estar presente em todos os lugares ao mesmo tempo, além do fato que uma fiscalização mais rígida seria mais burocrática gastando mais tempo para verificar os itens necessários. Se isso pudesse ser automatizado garantiria mais velocidade na fiscalização, mais segurança para os trabalhadores, mais segurança jurídica aos empregados e menos tempo e dinheiro gastos em processos trabalhistas.

1.2 JUSTIFICATIVA

A nossa ideia tem como objetivo solucionar essas falhas, com a criação de uma inteligência artificial que, por meio de câmeras, monitora as entradas de áreas onde o uso de EPIs é obrigatório, liberando ou não a entrada do trabalhador, caso esteja com os equipamentos corretos.

É um projeto que requer várias fases de implementação, por isso vamos começar com EPIs mais simples e em menor quantidade, por exemplo, reconhecendo o uso ou não de capacete, e então com os resultados obtidos vamos aprimorando a rede neural dessa inteligência a fim de agregar mais funções ao resultado final, priorizando garantir resultados precisos.

O funcionamento seria a partir de um treinamento de rede neural, tópico que será, assim como outros tópicos teóricos, elucidado mais adiante, criação de um modelo treinado, implementação em um servidor, e a criação e um cliente web. Com isso teríamos as imagens obtidas pela câmera sendo interpretadas pela rede neural hospedada em um local remoto, e com o resultado, emitindo sinais de saída para diferentes funções de volta ao cliente, como por exemplo,

abrir uma porta automática para liberar o trabalha- dor para determinado setor em que tal EPI identificado é de uso obrigatório.

Soma-se a esses motivos o atual momento em que vivemos, uma epidemia de COVID-19 (Coronavirus Disease 2019), uma doença infeciosa causada pelo coronavírus da síndrome respiratória aguda grave 2 (SARS-CoV-2), que nos impõe várias restrições como indivíduos e sociedade, entre elas o distanciamento social. Com isso vemos um crescimento de meios remotos de interação, seja de relações comerciais, interpessoais e de trabalho. Esse fato só corrobora com nossa motivação ao promover um sistema remoto que traz mais segurança na verificação de EPIs.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Como exposto acima a falha de fiscalização quanto ao uso de EPI é um problema crônico que custa, só no Brasil, a integridade física e a vida de milhares de trabalhadores todos os anos, além de diversos custos envolvidos aos empregadores. Nosso principal objetivo é mitigar esses ocorridos ao realizar uma fiscalização automática e inteligente.

1.3.2 Objetivos Específicos

Primeiramente precisamos identificar como construir tal sistema automático e inteligente para reconhecer determinados objetos numa imagem ou vídeo, para isso é necessário conhecer as principais formas de detecção de objetos usados através de inteligência artificial, aprendizado de máquina e redes neurais, como veremos no desenvolvimento teórico.

Selecionado o tipo de modelo a ser estudado e implementado, no caso, as redes neurais convolucionais (CNN), devemos conhecer suas particularidades e arquiteturas.

Vamos comparar as arquiteturas disponíveis e verificar a que melhor se encaixa em nossas necessidades e limitações de memória e hardware.

Devemos também estudar e selecionar as melhores bibliotecas e softwares utilizados na visão computacional.

Normalmente uma rede neural para visão computacional deve ser treinada, através de uma seleção de imagens com o objeto a ser identificado, logo devemos reunir um banco de imagens para realizar tal treinamento.

Outro ponto importante é estudar e analisar as melhores soluções WEB disponíveis para aplicação do nosso servidor de processamento.

Temos também que comparar os hardwares para escolher qual será o utilizado no projeto, escolhendo entre algumas placas que são comumente utilizadas no mercado, com custos e performances diferentes.

Para os testes vamos padronizar as imagens, executar o modelo treinado no hardware, receber as imagens de uma câmera, processar as imagens recebidas e realizar os testes de validação e medir a precisão da rede através de diferentes métricas.

A aplicação final reunirá o hardware que recebe a imagens da câmera, envia essa informação para o servidor, o servidor processará as imagens recebidas e enviará de volta as informações necessárias para a identificação do EPI.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 APRENDIZAGEM DE MÁQUINA

O aprendizado de máquina (Machine Learning) é o estudo de algoritmos de computador, que constroem um modelo matemático baseado em dados de amostra, conhecidos como dados de treinamento, para tomar decisões ou realizar previsões sem ser objetivamente programado para isso (MITCHELL, 2020). Introduziremos nas próximas seções as subdivisões deste campo de estudo.

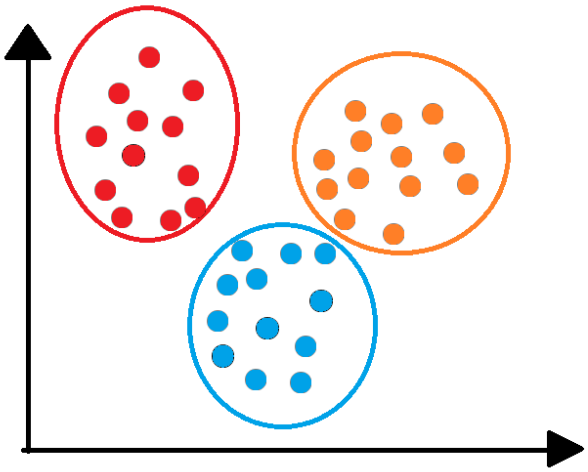
2.1.1 Aprendizagem não supervisionada

A aprendizagem não supervisionada é quando os dados de treino não são pré-classificados, neste caso apenas são fornecidos os dados e o algoritmo consegue identificar similaridades entre grupos de dados e agrupar eles em classes (MISHRA, 2017).

Por exemplo, pode-se a partir de um conjunto de dados de tamanhos medidas de muitas pessoas criar um algoritmo que consiga encontrar similaridades e classificar esses tamanhos em alguns grupos que podem ser utilizados para definir tamanhos padrões (P, M, G).

O estudo desse tipo de algoritmo não será aprofundado pois foge do escopo desse projeto.

Figura 1 – Aprendizagem não supervisionada.



Fonte: Autores

2.1.2 Aprendizagem supervisionada

Neste tipo de modelo os dados de treino recebem rótulos indicando que tipo de dados eles representam e normalmente são divididos em dados de treino e dados de teste.

Este tipo de aprendizagem é divido em dois grandes campos a regressão e classificação. A regressão consiste em a partir de um histórico temporal de dados conseguir prever dados futuros enquanto a classificação é o agrupamento de dados em algumas classes (WILSON, 2019).

Por exemplo, em uma pandemia um modelo que consegue prever números futuros de contaminação é um algoritmo de regressão, enquanto um algoritmo que consiga através de sintomas identificar se uma pessoa está infectada ou não, é um algoritmo de classificação binário, pois existem apenas duas classes (contaminado e não contaminado).

É importante destacar que nem todos os parâmetros de um modelo são treináveis, os que fogem desse padrão são chamados de hiperparâmetros e normalmente devem ser ajustados de forma manual.

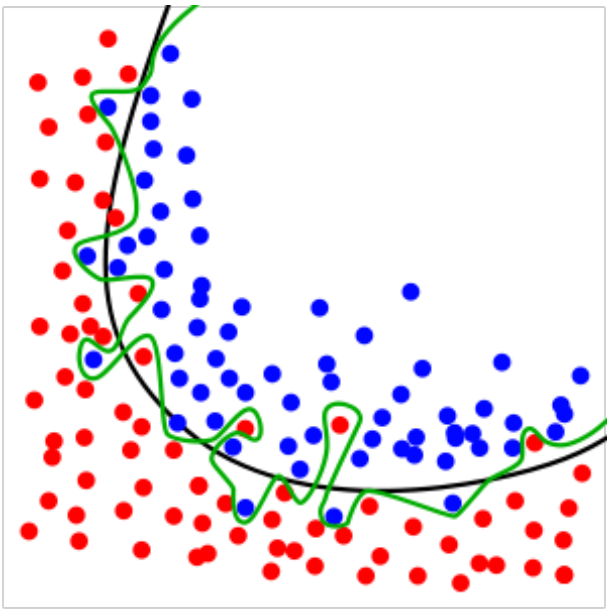
2.1.2.1 Problemas no ajuste de modelos

Ao ajustar um modelo aos dados de treino existem dois principais problemas que podem ocorrer.

a) **Subajuste (Under-fitting)**: É quando o erro é alto tanto nos dados de treino quanto nos dados de teste, então as previsões não são boas o suficiente em ambos os casos. Nesse caso, normalmente mais dados e melhores modelos são suficientes para resolver o problema.

b) **Superajuste (Over-fitting**): Neste caso o modelo funciona perfeitamente nos dados de treino mas quando utilizamos uma amostra nova ele apresenta um erro muito alto. Normalmente o modelo está muito complexo para o problema a ser resolvido então acaba perdendo a capacidade de generalizar para novos exemplos. Para corrigir, é necessário simplificar o modelo ou adicionar mais informações através de um processo chamado regularização.

Figura 2 – Exemplo de Overfitting (linha verde) versus regularização (linha preta).



Fonte: Tetko, Livingstone e Luik, 2020

2.2 CLASSIFICAÇÃO

Um modelo de classificação é um modelo que a partir de uma de entrada atribui um rótulo a esse dado o agrupando em alguma classe.

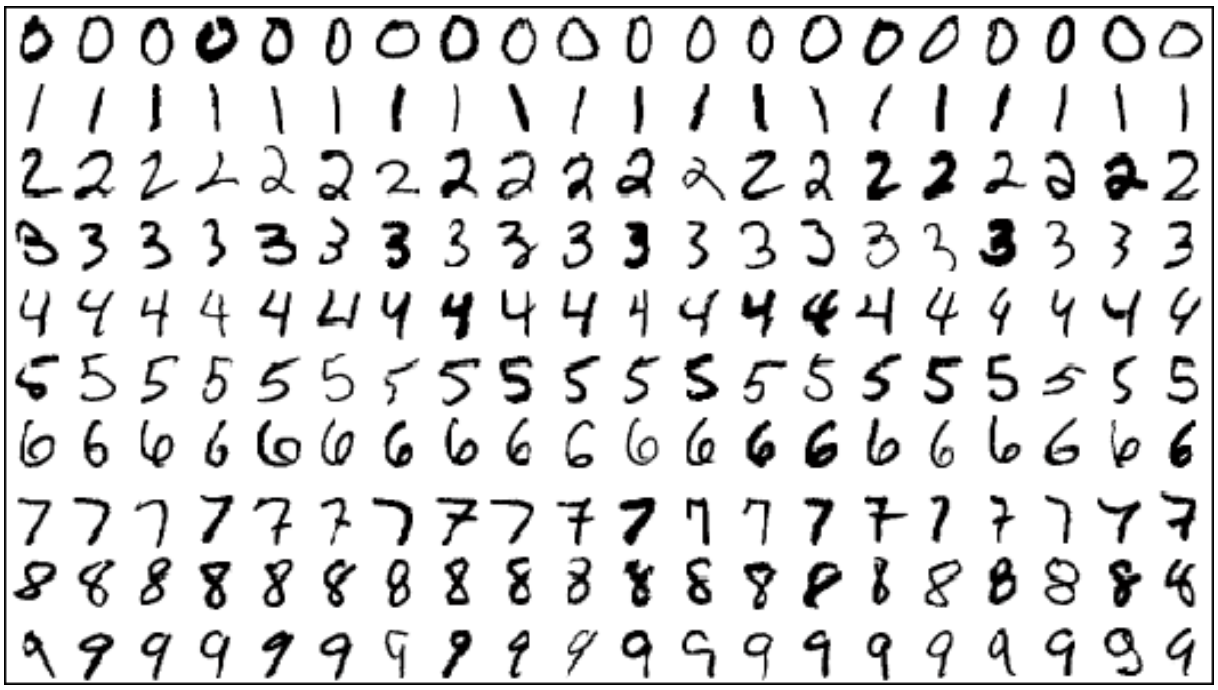
O MNIST é uma base de dados aberta que contém 60 mil imagens escritas a mão e pré-classificadas e é muito utilizada por iniciantes para iniciar os estudos sobre algoritmos de classificação e por profissionais na área para realizar benchmarks e comparar algoritmos.

Um exemplo muito comum é utilizar um algoritmo de classificação para através das imagens escritas a mão do MNIST identificar qual número ela representa, desta forma, classificar em uma das 10 classes de números inteiros de 0 a 9.

2.2.1 Regressão Logística

Apesar do nome, a regressão logística é um método de classificação binária que é amplamente utilizado e abordado por grande parte da literatura como o Bishop (2011). A função logística ou de Sigmoid é definida pela equação (1).

Figura 3 – Base de dados MNIST.

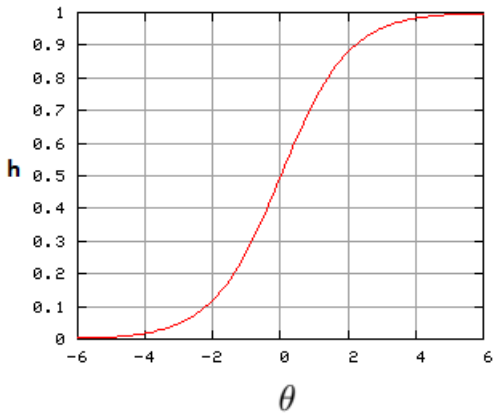


Fonte: LE CUN e Cortes, 2020

$$\sigma(\theta) = \frac{1}{1 + e^{-\theta}} \tag{1}$$

Quando o valor de θ tende a $-\infty$ o valor da função tende a 0 e quando θ tende a ∞ a função tende a 1, no intervalo entre [-5, 5] a função retorna valores entre 0 e 1.

Figura 4 – Gráfico da Função Logística.



Fonte: Han Jun; Morag, 2020

Supondo que queremos utilizar a função logística para classificar uma imagem que contém um número em duas classes "é o número 3"ou "não é o número 3", podemos adotar que

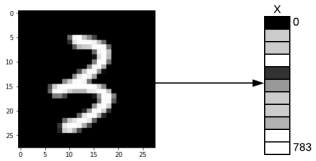
quando a valor da função $\sigma(\theta)$ for maior que um limite de b a imagem será o número 3, caso seja menor que b a imagem não é o número 3, esta convenção depende do tipo de projeto conforme discutido em maiores detalhes na seção 2.5.3. No exemplo da equação (2), onde y representa o resultado da classificação o limite escolhido foi $0{,}5$.

Desta forma, se o valor de $\sigma(\theta)$ for igual a $0{,}8$ o valor de y será igual a 1, portanto a previsão é positiva e o número é igual a 3. Do contrário, caso $\sigma(\theta)$ seja igual a $0{,}3$ o valor de y será igual a 0 e a previsão será negativa, ou seja o número é diferente de 3.

$$y = \begin{cases} 0 & \text{se } h < 0{,}5 \\ 1 & \text{se } h >= 0{,}5 \end{cases} \tag{2}$$

Agora precisamos ajustar a função para um vetor X, com relação ao exemplo, ele é um vetor com todos os pixels da imagem. Na figura 5, como o MNIST é composto de imagens com resolução $28x28$ pixels, o vetor X possui 784 elementos. A equação (4) realiza as previsões, onde o vetor θ é um parâmetro treinável que contém o ganho de cada pixel.

Figura 5 – Vetor com pixels da imagem.



Fonte: Autores

$$h = \sigma(\theta^T.X) \tag{3}$$

Em seguida, reunimos uma grande quantidade de imagens do número três manuscritas e utilizando a função de custo, equação (5), determinamos o erro para cada imagem e por fim obtemos o custo de todas as instâncias de treinamento, calculando a média simples da soma de todos os custos com a equação (4), onde m é o número de instâncias (imagens de treino).

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(X^{(i)}),y^{(i)}) \tag{4}$$

$$Cost_{(\theta)} = \begin{cases} -log(h_\theta) & \text{se } y = 0 \\ -log(1-h_\theta) & \text{se } y = 1 \end{cases} \tag{5}$$

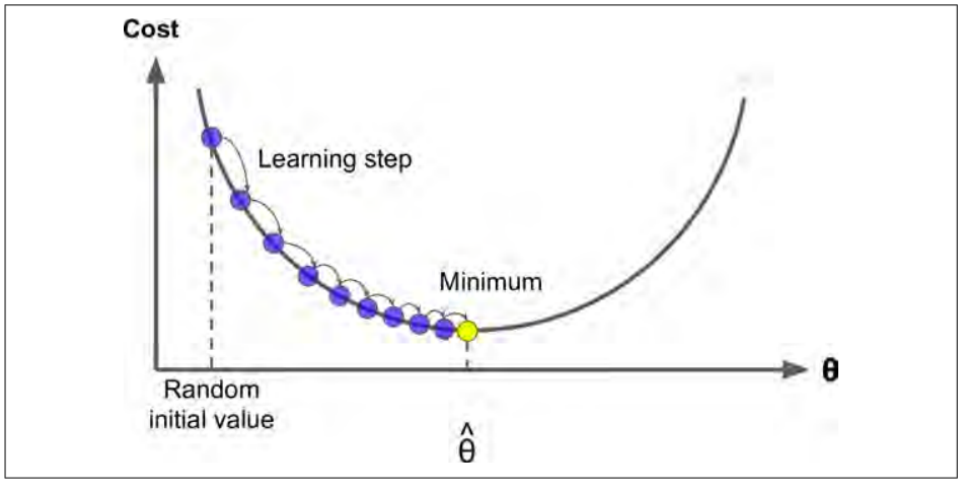
2.2.2 Gradiente Descendente

Após obter a função de custo precisamos encontrar o valor ótimo de θ que a minimiza, para isso é necessário calcular as derivadas, como na equação (4), obtendo a equação (6).

De modo a diminuir o custo de processamento normalmente utiliza-se um método de cálculo numérico, chamado Gradiente Descendente, que consiste em obter as derivadas e dar pequenos passos em direção ao vale da função, como na figura 6 (NIELSEN, 2015). O tamanho desses passos é um hiperparâmetro, que tem esse nome pois não pode ser ajustado através de aprendizagem automática, normalmente adota-se tentativa e erro para encontrar o melhor valor.

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}(h(\theta^T * X^{(i)}) - y^{(i)}) * X_j^{(i)} \tag{6}$$

Figura 6 – Gradiente Descendente.



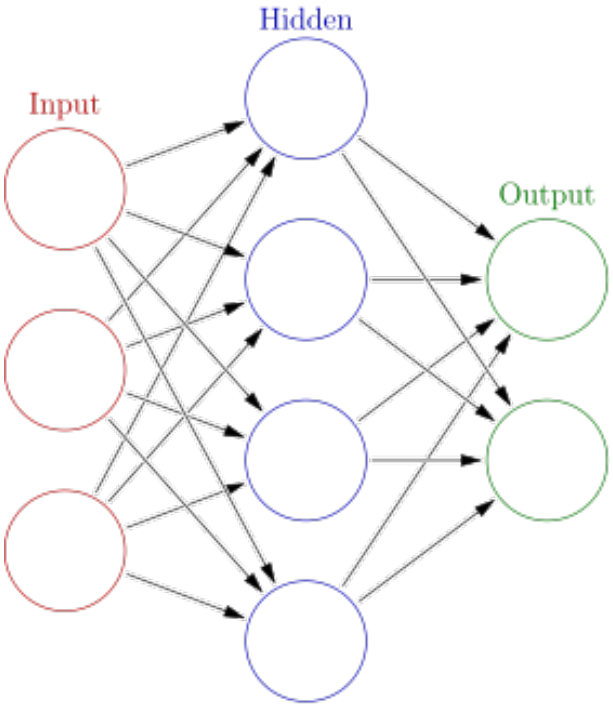
Fonte: Géron, 2017

Com o valor ótimo de θ encontrado, podemos utilizar a equação (3) para classificar qualquer vetor X que representa uma imagem $28x28$.

2.3 REDES NEURAIS

O cérebro humano tem muita facilidade em identificar que as diversas formas de escrever o mesmo número significam a mesma coisa, mas para um programa de computador é muito mais complicado de se realizar esta simples tarefa, pois enquanto o cérebro consegue identificar formas e padrões, tudo o que o computador consegue ver é uma matriz onde cada elemento da matriz corresponde a intensidade de um pixel.

Figura 7 – Um exemplo de rede neural artificial.



Fonte: Ferreira, 2020

As redes neurais são algoritmos que nasceram com o propósito de permitir que um programa de computador também consiga identificar padrões de forma similar ao cérebro humano, elas são sistemas de computação inspirados nos neurônios biológicos, baseados num agrupamento de conexões unitárias chamadas de neurônios artificiais, esses neurônios artificiais são uma função matemática, que recebe uma ou mais entradas e as soma para produzir uma determinada saída. (FERREIRA, 2020; ANTHONY, 2020)

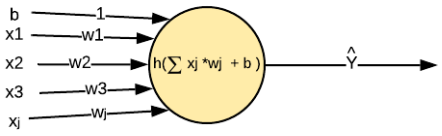
Normalmente uma rede neural é composta de uma camada de entrada, pelo menos uma camada intermediaria chamada de Hidden Layer e uma camada que retorna a inferência final.

2.3.1 Neurônio de Sigmoid

Um neurônio de Sigmoid é um neurônio que tem como função de ativação a função de sigmoid. A saída de um neurônio é o resultado da aplicação da chamada função de ativação na combinação linear dos vetores de entrada. A imagem 8 representa um neurônio que tem três entradas $x1$, $x2$ e $x3$, cada vetor é multiplicado por uma constante $w1$, $w2$ e $w3$ e além disso

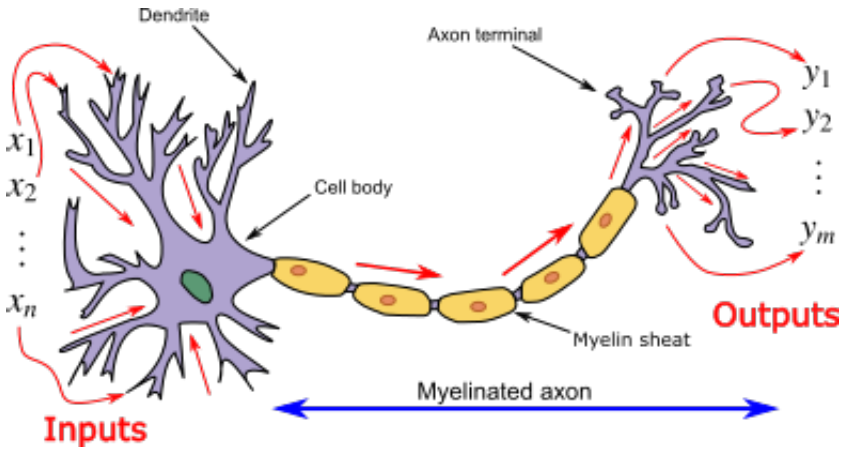
temos uma entrada que chamamos de viés, é representada pela constante b que determina o deslocamento no eixo y da função de ativação.

Figura 8 – Esquema de um neurônio.



Fonte: Autores

Figura 9 – Comparação entre neurônio biológico e artificial.



Fonte: Anthony, 2020

Generalizando para um vetor qualquer com dimensão j, a combinação linear dos vetores de entrada pode ser expressa como $(xj)^t * wj + b$. Desta forma, utilizando a função de sigmoid como função de ativação a saída de um neurônio qualquer pode ser calculada com a equação:

$$Y = h((x_j)^t * w_j + b) = \frac{1}{1 + e^{-(x_j)^t * w_j - b}} \tag{7}$$

2.4 APRENDIZADO PROFUNDO

O aprendizado profundo (deep learning) é um método de aprendizado de máquina baseado em redes neurais artificiais, ele tem esse nome devido à grande quantidade de camadas intermediarias (hidden layers).

Existem diversas arquiteturas de aprendizado profundo, entre elas as redes neurais profundas (deep neural networks), redes de crenças profundas (deep belief networks), as redes neurais recorrentes (recurrent neural networks) e redes neurais convolucionais (convolutional neural networks), utilizadas, principalmente em campos incluindo visão computacional, reconhecimento de voz, processamento de linguagem natural, reconhecimento de áudio, filtragem de rede social, tradução automática, bioinformática, a concepção de medicamentos, análise de imagens médicas, inspeção de materiais e programas de jogos de tabuleiro, onde eles produziram resultados comparáveis e, em alguns casos, superiores ao desempenho de especialistas humanos.

Estas arquiteturas usam várias camadas para extrair recursos da entrada. Por exemplo, no processamento de imagens, as camadas inferiores podem identificar bordas, enquanto as camadas superiores podem identificar os conceitos relevantes para um ser humano, como dígitos, letras ou faces. (SCHMIDHUBER, 2020)

2.4.1 Processamento de Imagem

Uma imagem pode ser representada de diversas formas, a forma mais comum de representar uma imagem digitalmente é através de uma matriz A_{mxn} onde m e n representam respectivamente a quantidade de pixels de largura da imagem e a quantidade de pixels de altura da imagem, o valor de cada elemento da matriz representa a intensidade de cada pixel.

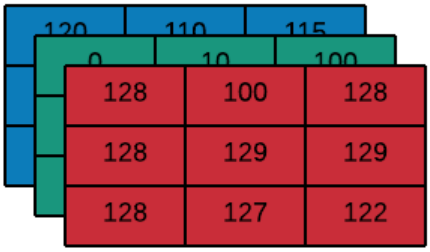
Quando uma imagem colorida é processada pelo computador ela é visualizada como três matrizes chamadas de canais onde cada canal representa uma cor, o padrão mais comum para esse tipo de representação é o padrão RGB, onde temos canais nas cores vermelho, azul e verde.

Para facilitar o processamento, tanto na etapa do treinamento, quanto na etapa de detecção da imagem via hardware, podemos realizar diferentes transformações nas imagens, para reduzir ou aumentar seu tamanho, a fim de padronizar dimensões, ou até deixar as imagens em escala de cinza, para trabalhar com apenas uma matriz de cor.

2.4.1.1 Matriz de convolução

A convolução é um importante tópico no processamento de imagem e no aprendizado de máquina. A matriz de convolução, também pode ser chamada de Kernel ou Máscara em al-

Figura 10 – Representação matricial
de uma imagem 3x3.



Fonte: Autores

gumas literaturas, é muito utilizada para aplicar diversos filtros em imagens, como por exemplo embaçamento de imagem (Blur), afiação de imagem (Sharpening) e detecção de bordas que é a principal aplicação no Machine Learning.

Figura 11 – Aplicação dos filtros Sharpen e Blur.

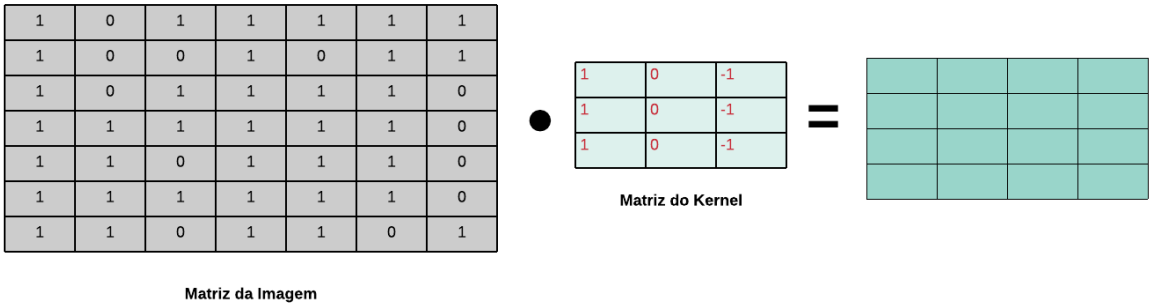
Fonte: Autores

Normalmente adota-se um algoritmo para determinar a convolução, para exemplificar, considere a convolução entre as duas matrizes representadas na figura 12. A matriz maior representa uma imagem, vamos chama-la de matriz A, em seguida temos uma matriz do Kernel que vamos chamar de B e uma matriz que será o resultado da operação que chamaremos de C.

O algoritmo consiste em sobrepor a matriz A e a matriz de *kernel* e em seguida multiplicar os elementos sobrepostos e finalmente somar os resultados, obtendo um elemento da matriz C. Para evitar perder informações na borda da imagem normalmente utilizamos uma margem chamada de *Padding* e também adotamos um número de deslocamento da matriz kernel, esse valor normalmente é chamado de *Stride*.

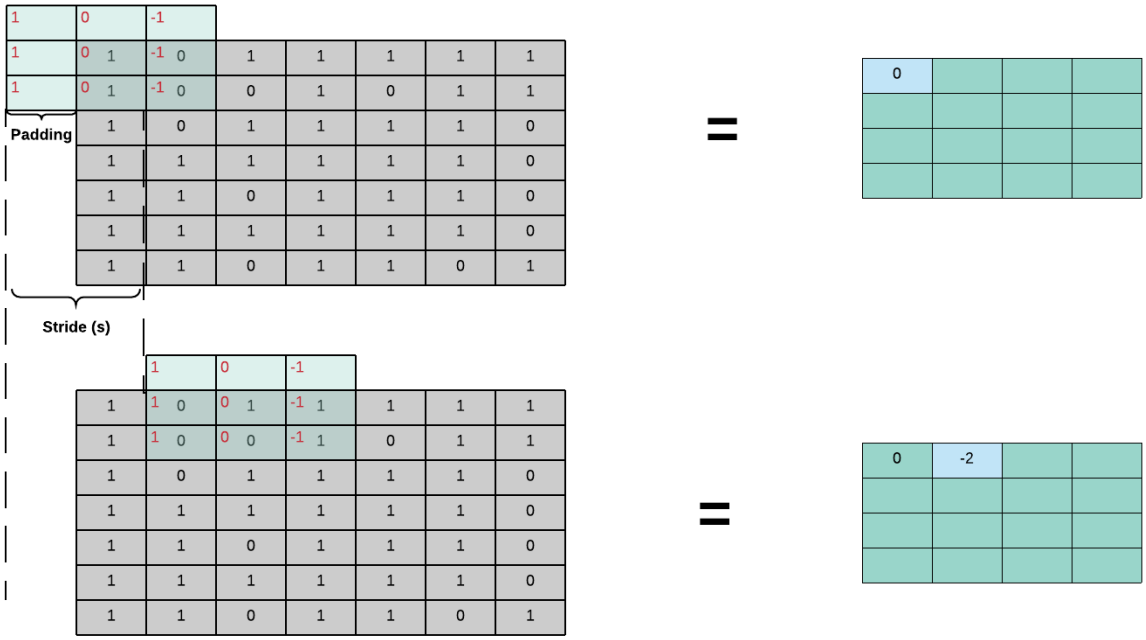
Na figura 13, estamos realizando a convolução de uma matriz A_{7x7} e uma matriz de B_{3x3}, utilizando um *padding*(p) de 1 elemento e um *stride*(s) de 2 elementos, portando podemos utilizar a equação (8) para calcular qual será a dimensão da matriz resultante.

Figura 12 – A Operação de convolução.



Fonte: Autores

Figura 13 – Algoritmo de convolução.



Fonte: Autores

$$dim(C) = [\frac{m + 2 * p - f}{s} + 1]x[\frac{n + 2 * p - f}{s} + 1] \tag{8}$$

Para a equação (8) a dimensão de saída será:

$$dim(C) = [\frac{7 + 2 * 1 - 3}{2} + 1]x[\frac{7 + 2 * 1 - 3}{2} + 1]$$

$$dim(C) = 4x4$$

2.4.1.2 Reconhecimento de Bordas

Uma das principais aplicações da convolução é o reconhecimento de bordas em imagens, para exemplificar, considere a seguinte matriz da tabela 1, ela é um filtro de convolução que pode ser utilizado para encontrar bordas verticais.

Tabela 1 – Matriz de Kernel

	x_1	x_2	x_3
y_1	-1	0	1
y_2	-1	0	1
y_3	-1	0	1

Fonte: Autores

A matriz da tabela 2 é parte de uma imagem que contém uma borda, podemos observar a existência de uma borda através da mudança na intensidade dos pixels de uma coluna para a outra, isto indica a existência de uma *linha vertical*.

Tabela 2 – Um parte de uma imagem que contém uma borda

	x_1	x_2	x_3
y_1	25	25	250
y_2	25	25	250
y_3	25	25	250

Fonte: Autores

A matriz da tabela 3 é uma parte de uma imagem que não possui bordas, podemos observar esta característica através da homogeneidade dos pixels.

Tabela 3 – Parte de uma imagem que não contém uma borda

	x_1	x_2	x_3
y_1	250	250	250
y_2	250	250	250
y_3	250	250	250

Fonte: Autores

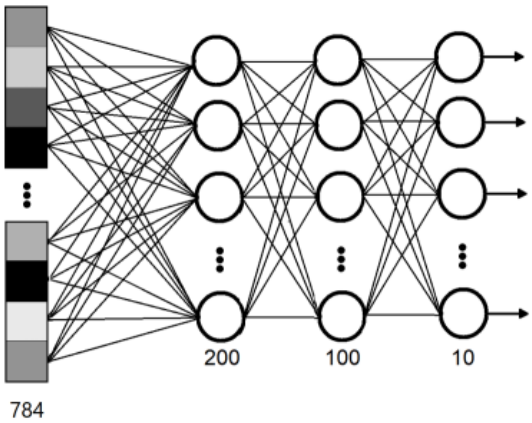
Ao realizarmos a convolução da matriz da tabela 1 com a matriz da tabela 3 o resultado é 0, indicando que não existe nenhuma borda na janela observada. Agora ao realizar a convolução da matriz 1 com a matriz 2 o resultado é 225, o número com maior valor absoluto indica a ocorrência de uma borda vertical na imagem.

2.4.2 Redes Neurais Convolucionais

As redes neurais convolucionais (ConvNet / Convolutional Neural Network / CNN) são uma classe das redes neurais de aprendizado profundo (deep neural networks), que por sua vez fazem parte da família de métodos de aprendizado de máquina (machine learning / ML), baseados em redes neurais artificiais (artificial neural networks). Para melhor compreensão das CNN vamos distinguir os conceitos citados acima. (KRIZHEVSKY; SUTSKEVER; HINTON, 2020)

As CNNs possuem grandes vantagens em relação aos tradicionais perceptrons de múltiplas camadas, principalmente com relação à classificação de imagens, no exemplo da figura 14 é utilizado uma rede de perceptrons para identificar um número do MNIST, onde cada um dos 784 pixels da imagem é mapeado para um neurônio. Devido à grande quantidade de conexões (conexão total) dessas redes elas são propensas a superajustar (overfitting) os dados. As CNNs atenuam esse problema e são chamados de perceptron de multicamadas regularizados, pois adicionam mais informações ao modelo através da detecção de linhas verticais, horizontais, círculos, entre outros, como discutido brevemente na seção 2.4.1.2.

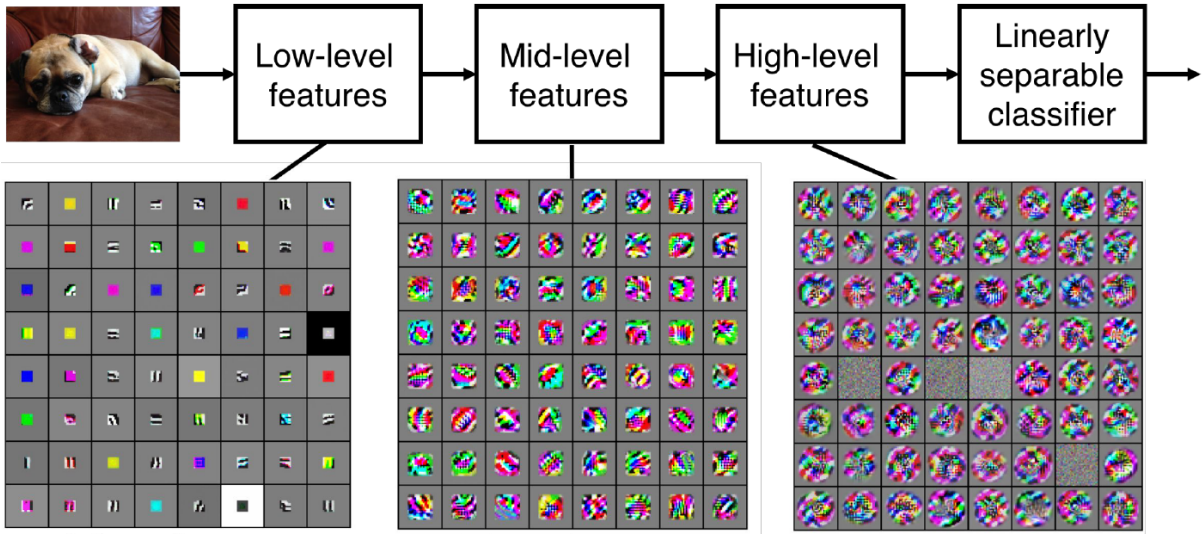
Figura 14 – Perceptron de múltiplas camadas.



Fonte: Tetko, Livingstone e Luik, 2020

Essas redes adotam uma abordagem diferente em relação à regularização: tiram vantagem do padrão hierárquico dos dados e montam padrões mais complexos usando padrões menores e mais simples. Na figura 15 é possível visualizar o que cada kernel da CNN está identificando, a primeira figura detecta vários tipos de linhas, em seguida os outros níveis de kernels detectam padrões ainda maiores e mais complexos da imagem.(KRATSIOS, 2020)

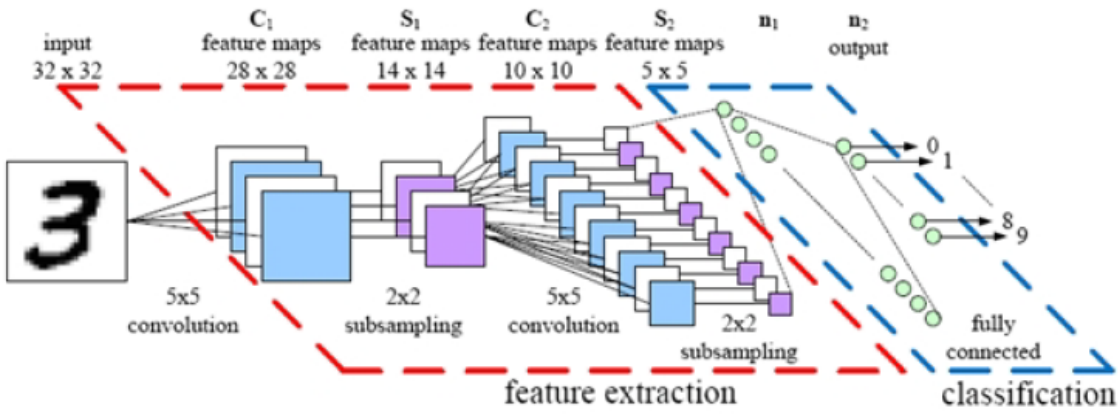
Figura 15 – Exemplo de extração de padrões simples e complexos.



Fonte: Visual Recognition (Spring 2017) - Stanford University School of Engineering, 2020

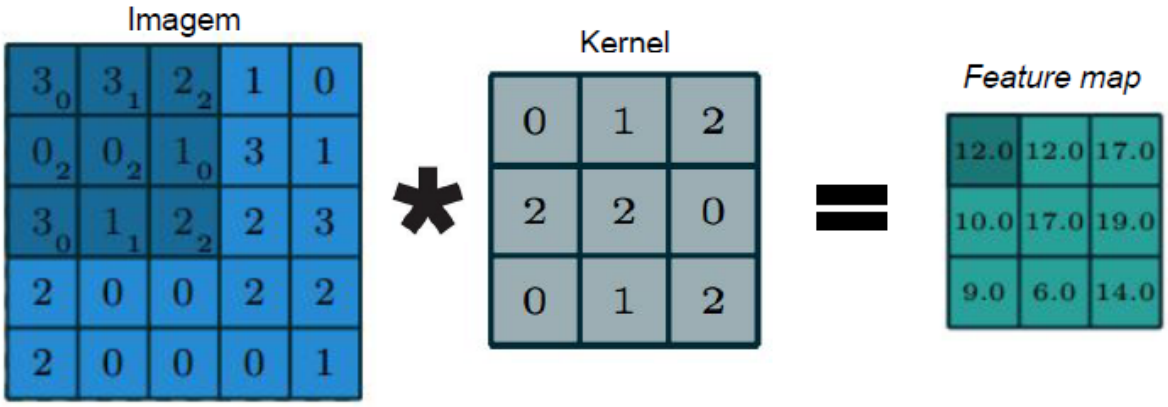
Quanto a sua arquitetura, uma CNN, com foco na visão computacional, consiste em uma camada de entrada, várias camadas ocultas, onde temos os extratores de características e uma de saída, com o classificador que normalmente é composto de perceptrons multicamadas, como podemos observar na figura 16. (HATTORI, 2020; FLORINDO, 2020; VISUAL RECOGNITION (SPRING 2017) - STANFORD UNIVERSITY SCHOOL OF ENGINEERING, 2020)

Figura 16 – Exemplo de arquitetura de rede neural convolucional.



Fonte: Florindo, 2020

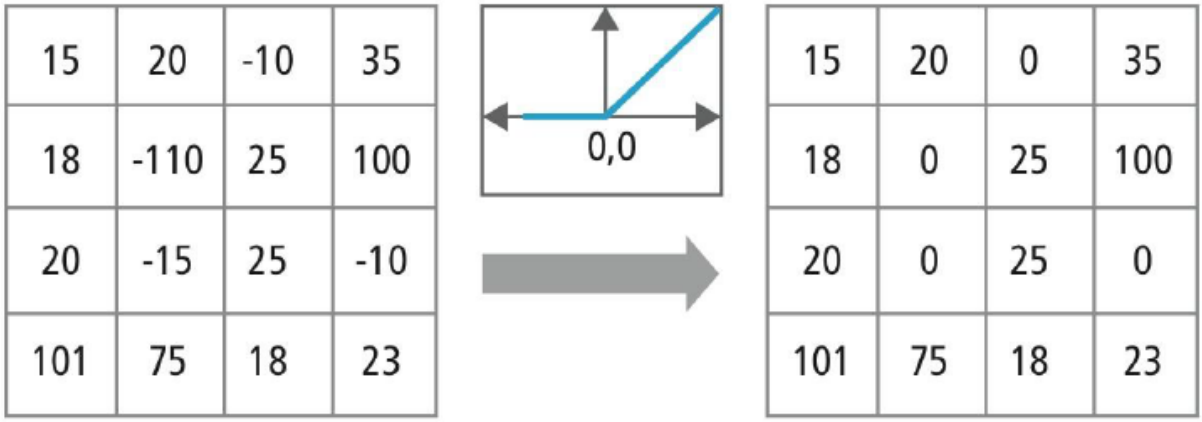
Figura 17 – Exemplo da operação de convolução numa imagem (matriz) 5x5 com kernel 3x3.



Fonte: Florindo, 2020

A extração de características consiste em uma série de camadas convolucionais, que realizam a op eração de convolução, ou seja, deslizar um filtro ou kernel por uma imagem para gerar outra imagem, comumente chamada de mapa de características, mapa de atributos ou feature map, como visto na figura 17. Isso ocorre pela facilidade de se utilizar kernels para identificar bordas e padrões mais complexos. Funciona de forma similar ao cérebro humano, que por exemplo, consegue identificar o número 8 através de padrões. Isso ocorre porque mesmo que a caligrafia não seja perfeita o número 8 sempre conterá dois círculos. (HATTORI, 2020; FLORINDO, 2020) (VISUAL RECOGNITION (SPRING 2017) - STANFORD UNIVERSITY SCHOOL OF ENGINEERING, 2020)

Figura 18 – ReLU aplicada aos feature maps.



Fonte: Visual Recognition (Spring 2017) - Stanford University School of Engineering, 2020

A função de ativação é geralmente uma camada ReLU (Rectified Linear Unit), figura 18, e é subsequentemente seguida por convoluções adicionais, como camadas de pooling. O poo-

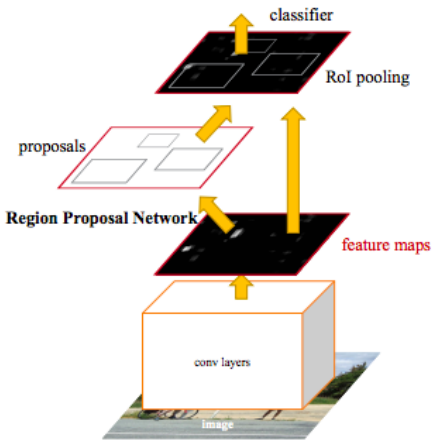
ling tem dois propósitos gerais, reduzir dimensionalidade e adicionar uma pequena invariância à translação, rotação etc. Essa operação também usa um kernel como janela deslizante. (HATTORI, 2020; FLORINDO, 2020; VISUAL RECOGNITION (SPRING 2017) - STANFORD UNIVERSITY SCHOOL OF ENGINEERING, 2020)

2.4.3 Rede de Proposição de Regiões (RPN)

A utilização da Region Proposal Network (RPN) torna o processo de classificação da Rede Neural Convolucional muito mais veloz, inserindo a camada RPN após a última camada de convolução é possível identificar os objetos e classificá-los em tempo real. (SOARES, 2020)

A função do RPN é produzir de forma eficiente propostas de regiões de maior probabilidade de se encontrar objetos, para isso, é utilizada a última camada do feature map, onde a imagem foi tratada com filtros definidos de forma a acelerar a resposta e manter a precisão da criação dos proposals para a classificação destes elementos, conforme treinamento assim, dividindo o feature map de qualquer tamanho como diversas áreas retangulares onde podem existir objetos, como visto na figura 19 . (REN et al., 2015)

Figura 19 – Rede de proposição de região.

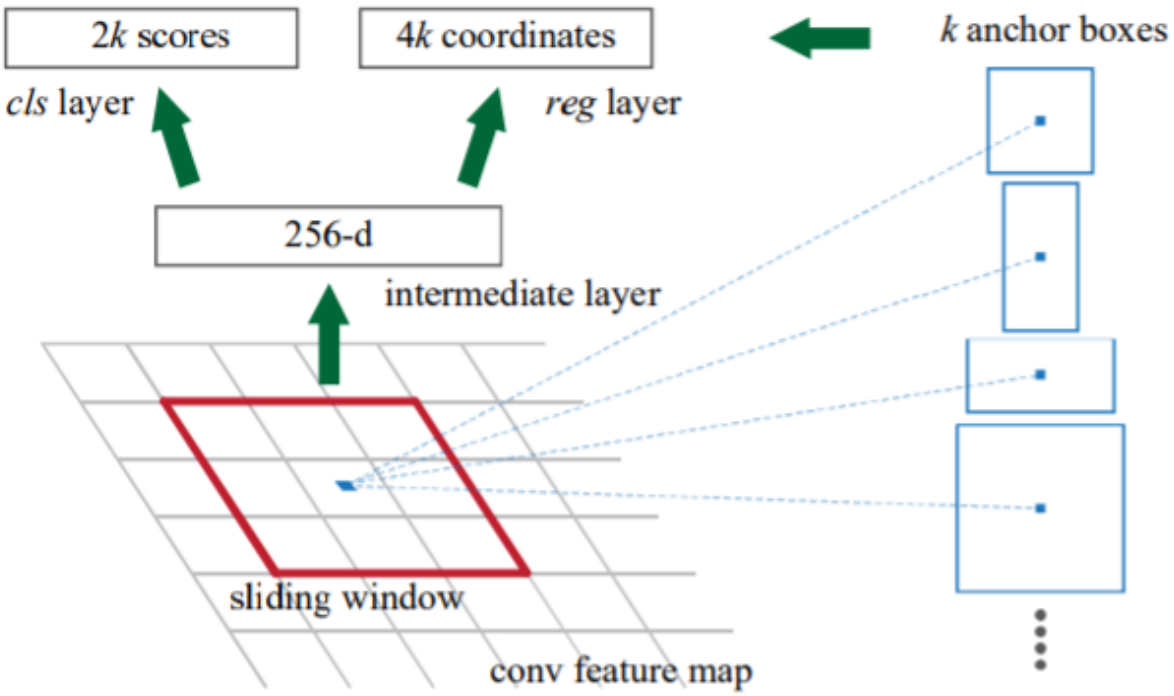


Fonte: Ren et al., 2015

É necessário dizer que a RPN utiliza da BBOX (Bouding-box Regression) para identificar e envolver cada elemento em sua caixa de classificação, assim como o CNN comum. Porém, como as regiões entregues pelo RPN possuem mais probabilidade estatística da existência, o processo é feito cerca de 250 vezes mais rápido.

A fonte de resultados pelo RPN se dá a uma mini-rede (REN et al., 2015) capaz de aceitar uma entrada de nxn de amostras deste feature map, onde cada janela de entrada é mapeada e convertida a uma menor dimensão e normalmente é seguida de outra duas camadas convolucionais camadas de reg e cls (REN et al., 2015), representando as coordenadas e suas probabilidades, respectivamente.

Figura 20 – RPN, Âncoras.



Fonte: Ren et al., 2015

Permitindo, portanto, a visualização da classificação dos elementos em tempo real, levando milissegundos para obter a resposta de classificação, sem perda na sua capacidade de identificação.

2.4.3.1 ÂNCORAS

Para cada localização das janelas de análise (sliding-window) é encontrada múltiplas propostas de regiões, onde o número máximo é denominado como k, portanto, a camada reg (coordenadas) possui o máximo de 4k saídas assim como a camada cls (probabilidade) possui 2k, que estima a probabilidade de um objeto existir ou não em tais coordenadas.

Uma âncora é encontrada no centro da janela de análise e é relacionada com o tamanho e escala desta janela. É sabido portando, que para uma imagem de A x L, existem A×L×k âncoras, sendo k, a quantidade de T×E escolhidas para se analisar (T = Tamanho, E = Escala).

2.4.4 Modelos de CNN para detecção de objetos

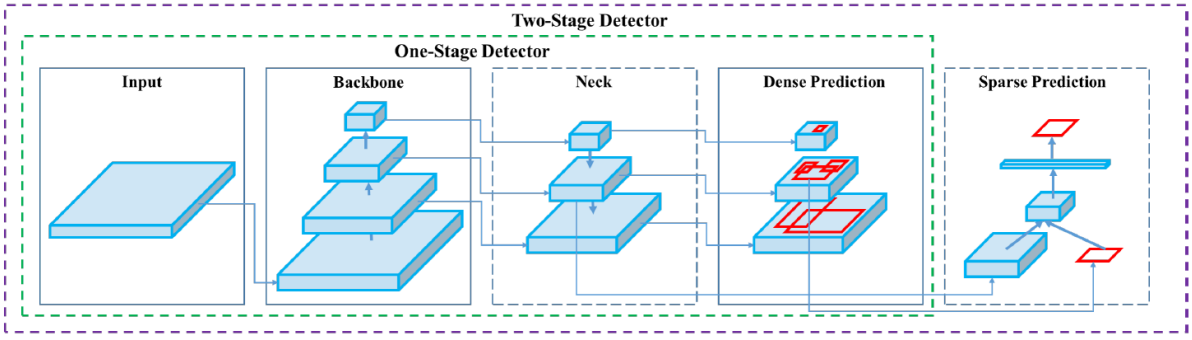
Um detector moderno é composto, normalmente, de duas partes, uma espinha dorsal (backbone) pré-treinada no ImageNet e uma cabeça (head) que é usada para prever classes e caixas delimitadoras (bounding boxes) de objetos. Backbones mais usados são VGG16, ResNet50, MobileNet e MobileNetv2 e DarkNet53. Quanto à head, geralmente é categorizada em dois tipos, detector de objeto de um estágio e detector de objetos de dois estágios. O mais representativo detector de objetos de dois estágios é a série R-CNN, incluindo Fast R-CNN, Faster R-CNN, R-FCN, e Libra R-CNN. Quanto ao detector de objetos de um estágio, os modelos mais representativos são You only look once (YOLO), Single Shot Detector (SSD) e RetinaNet. Detectores de objetos desenvolvidos nos últimos anos geralmente inserem algumas camadas entre o backbone e a head, essas camadas são geralmente usadas para coletar mapas de recursos de diferentes estágios. Nós podemos chamá-lo de pescoço (neck) de um detector de objetos. Geralmente, um pescoço é composto por vários caminhos de baixo para cima e vários de cima para baixo. As redes equipadas com esse mecanismo incluem Rede de pirâmides de recursos (FPN), agregação de caminhos Rede (PAN), BiFPN e NAS-FPN, essa parte da arquitetura não será objeto de estudo do nosso projeto. (BOCHKOVSKIY; WANG; LIAO, 2020)

Em resumo, um detector de objeto comum é composto de muitas partes:

- Entrada: Imagem, Patches, Pirâmide de Imagem;
- Backbones: VGG16, ResNet50, SpineNet, EfficientNet-B0 / B7, CSPResNeXt50, CSPDarknet53 e MobileNet;
- Neck:
    - Blocos adicionais: SPP, ASPP, RFB, SAM;
    - Blocos de agregação de caminho: FPN, PAN, NAS-FPN, Fully-connected FPN, BiFPN;
- Heads:
    - Previsão Densa (uma etapa):
        - RPN, SSD, YOLO, RetinaNet (baseado em âncora);

- Previsão esparsa (duas etapas):
  - Faster R-CNN, R-FCN.

Figura 21 – Arquitetura de Detector de Objetos.



Fonte: Bochkovskiy, Wang e Liao, 2020

2.5 MEDINDO A PERFORMANCE

Para que seja possível avaliar a qualidade de um algoritmo de classificação precisamos desenvolver alguns métodos para medir a performance.

2.5.1 Matriz de confusão

Considere um exemplo em que estamos tentando classificar um número escrito a mão e identificar se este número é o número 5. Como resultado, a predição do algoritmo de classificação vai cair obrigatoriamente em 4 estados diferentes.

a) **True Negative (TN):** Os números não eram iguais a 5 e foram corretamente classificados como não sendo iguais a 5.

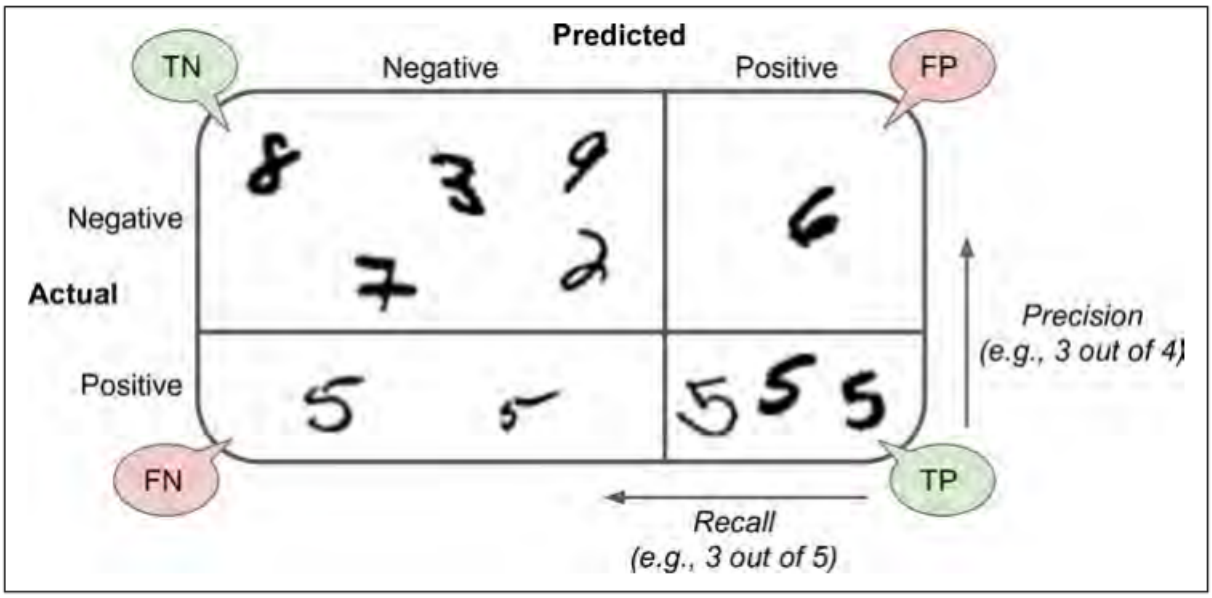
b) **False Positive (FP):** Os números não eram iguais a 5 e foram classificados de forma errada como sendo iguais a 5.

c) **False Negative (FN):** Os números eram iguais a 5 e não foram classificados como sendo iguais a 5.

d) **True Positive (TP):** Os números eram iguais a 5 e foram corretamente classificados como iguais a 5.

Estes resultados normalmente são organizados em uma matriz, chamada de matriz de confusão conforme apresentado na figura 22.

Figura 22 – Matriz de Confusão.



Fonte: Géron, 2017

2.5.2 Acurácia

A acurácia é uma das formas mais comuns e básicas de se avaliar um modelo, ela é definida conforme a equação (9), onde utilizamos o número de acertos tanto negativos como positivos divido pelo número total de classificações. Entretanto, este tipo de métrica pode ser problemática em bases de dados desbalanceadas.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

Considerando um banco de dados de fraudes em cartão de crédito (ULB, 2020), fornecidos por bancos europeus onde temos na tabela 4 as três primeiras transações de 284.807, sendo que apenas 492 transações são fraudes. Este banco de dados é composto pelas seguintes colunas:

a) **Time:** Indica quando ocorreu a transação.

b) **V1-V28:** São dados da transação escondidos para manter a privacidade do usuário.

c) **Ammount:** Valor da transação.

d) **Class:** Classe da transação. Quando igual a 1 indica uma fraude e quando igual a 0 indica uma transação normal.

Tabela 4 – Primeiras três linhas de dados do dataset de análise de fraude de cartão de crédito

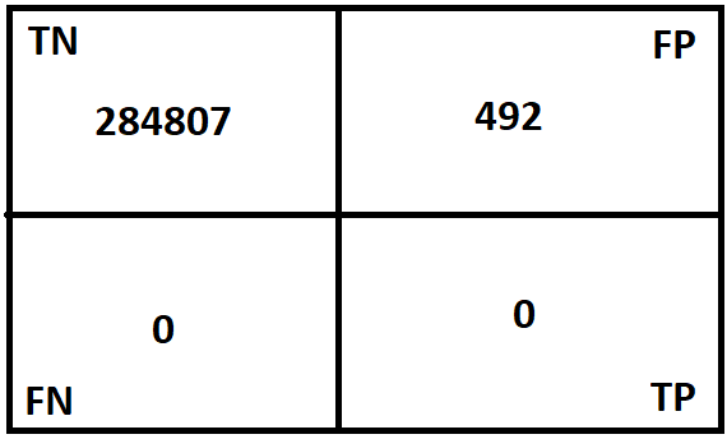
Time	V1	V2	...	V28	Ammount	Class
0.0	-1.359807	-0.072781	...	-0.021053	149.62	0
0.0	1.191857	0.266151	...	0.014724	2.69	0
1.0	-1.358354	-1.340163	...	-0.059752	378.66	0

Fonte: ULB, 2020.

O abismo do desbalanceamento dos dados pode ser visualizada através do gráfico da figura 24. Imagine que criamos um modelo e esse modelo classificou todas as transações chegando à conclusão que não existe nenhuma fraude, o número de casos positivos que ele acertou (TP) será igual a 0, entretanto o número de casos negativos (TN) será igual a todo o resto, ou seja, $TN = 284.807 - 492 = 284.315$.

Calculando a acurácia do modelo obtemos $99{,}83\%$. Ou seja, quase 100% de acurácia mesmo errando todas as predições positivas, pois a quantidade de casos negativos é muito maior.

Figura 23 – Matriz de confusão para o caso de fraudes de cartão de crédito.



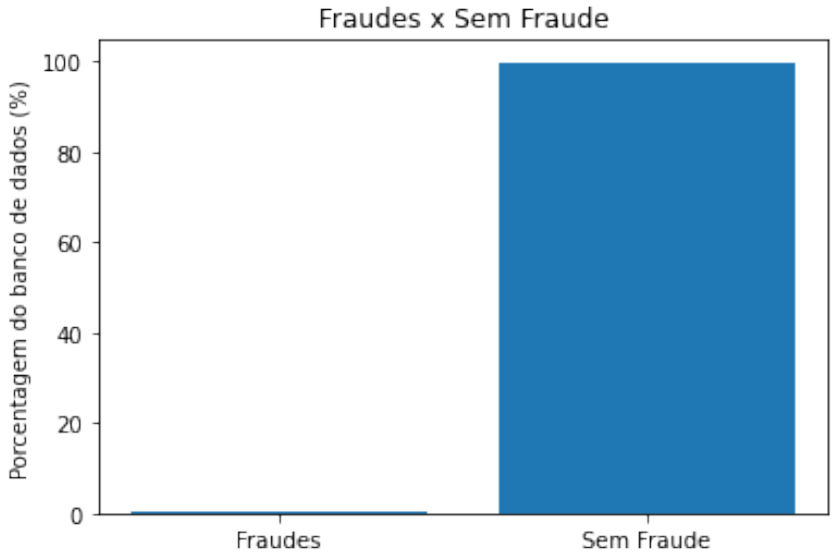
Fonte: Autores

2.5.3 Precisão e revocação

A precisão e a revocação (recall) surgiram para fornecer métricas mais consistentes mesmo com dados desbalanceados.

Uma forma de se definir a qualidade de um modelo é considerando quantas fraudes (casos positivos) que realmente foram identificados corretamente, ou seja, quantos TP tivemos em relação ao número de positivos, como representado matematicamente na equação (10) este

Figura 24 – Porcentagem de dados com Fraudes x Sem Fraude.



Fonte: Autores

tipo de métrica é chamado de **precisão**. Caso calcularmos a precisão para o exemplo da seção 2.5.2 o resultado será igual a 0%, um valor mais adequado que a acurácia de $99{,}83\%$.

$$precision = \frac{TP}{TP + TN} \tag{10}$$

Outra forma de se obter o desempenho do modelo é calculando quantas transações que não eram fraudes foram corretamente identificadas, ou seja, quantos casos negativos TN foram obtidos em relação ao total, como representado na equação (11), neste caso chamamos de **revocação**.

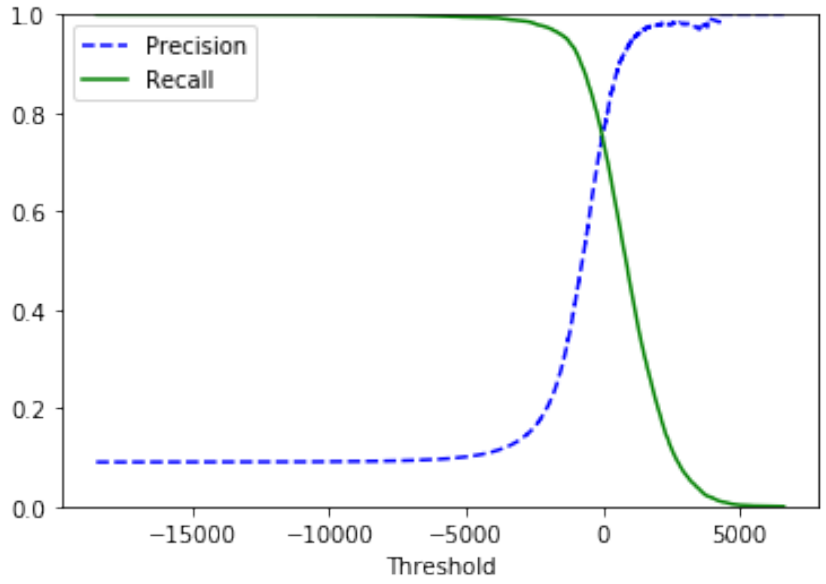
$$recall = \frac{TN}{TP + FN} \tag{11}$$

Em modelos reais é impossível obter 100% de precisão e recall, como podemos observar na imagem 25 conforme a precisão diminui, o recall aumenta e vice-versa. Apenas é possível mudar o limite central onde ambas as curvas se encontram para priorizar um dos índices, podemos fazer isso ajustando a constante para se considerar um acerto, conforme discutido na seção 2.2.1.

A métrica que iremos priorizar depende do objetivo do modelo que está sendo desenvolvido, por exemplo, caso seja criado um modelo que consiga identificar se uma pessoa está infectada com uma doença em uma pandemia é desejável que tenhamos um modelo mais preciso, pois queremos que todos os casos positivos sejam identificados com muita exatidão e

não desejamos ter alta revocação, para evitar que pacientes não infectados sejam enviados ao hospital com infectados e acabem contraindo a doença.

Figura 25 – Precision x Recall.



Fonte: Autores

Uma forma de se sintetizar ambas métricas é através da **F1-Score** conforme pode ser observado na equação (12).

$$F1Score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 * \frac{precision * recall}{precision + recall} \tag{12}$$

2.5.4 Detecção de objetos

Em algoritmos de detecção de objetos normalmente queremos encontrar uma região dentro de uma imagem maior que contenha um elemento de interesse e delimita-la com o auxílio de uma figura geométrica. Nesta seção iremos definir algumas das formar de se avaliar um modelo de detecção de objetos e como utilizar essas métricas para otimizar modelos.

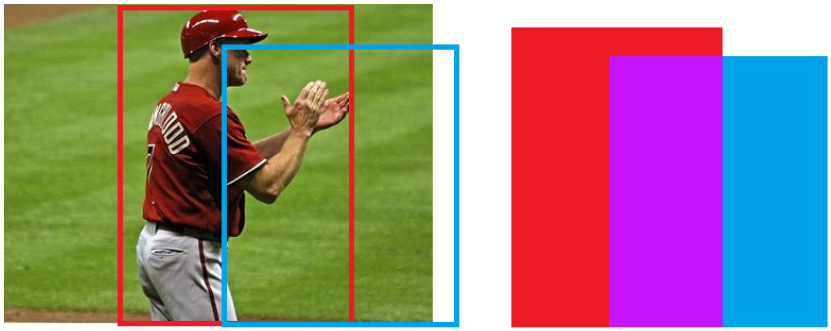
2.5.4.1 Intersect Over Union (IoU)

Na figura 26 estamos tentando detectar a pessoa na imagem, o retângulo vermelho representa a localização exata da pessoa, enquanto o azul é uma predição do algoritmo, nesta

seção definiremos uma forma de se analisar quantitativamente se os retângulos coincidem através do IoU . Se considerarmos a área prevista como A e a área real como B, é possível defini-lo conforme a equação (13).

$$IoU = \frac{A \cap B}{A \cup B} \tag{13}$$

Figura 26 – Exemplo de Iou.



Fonte: Adaptado de Yao et al., 2011

Caso ambas as caixas estejam completamente sobrepostas o numerador e o denominador serão iguais, portando IoU será igual a 1. Porém, caso não estejam sobrepostas o resultado será igual a 0. Na figura 27 podemos visualizar alguns exemplos para diferentes situações.

Figura 27 – Exemplos de cálculo de Iou em diferentes situações.

Fonte: Adrian Rosebrock, 2020

2.5.4.2 Intersecção sobre União Generalizada (GIoU)

O IoU é o método mais utilizado para cálculo de métricas em algoritmos de reconhecimento de imagem, entretanto não é muito eficiente para ser utilizado como função de custo para otimização de modelos, pois caso não haja sobreposição entre as caixas delimitadoras o

resultado dele sempre será zero ($A \cap B = 0$). Desta forma não é possível avaliar se as duas caixas não sobrepostas estão longe ou perto uma das outras.

Para resolver esse problema foi introduzido no artigo (REZATOFIGHI et al., 2019) o GIoU, representado pela equação 14, onde substituindo o resultado da equação 13 obtemos 15.

$$GIoU = \frac{A \cap B}{A \cup B} - \frac{|C \backslash A \cap B|}{|C|} \tag{14}$$

$$GIoU = IoU - \frac{|C \backslash A \cap B|}{|C|} \tag{15}$$

Onde C é a menor região convexa possível que contém A e B com o mesmo tipo de geometria de ambos. No caso da figura (28) C é um retângulo maior contendo A e B pois os dois são retângulos, o IoU para esta figura é igual a 0 pois não existe intersecção entre as imagens.

Logo, a expressão do GIoU ficará igual a equação (16) onde o numerador é igual a área não ocupada por nenhuma das imagens e o denominador é igual a área C que enclausura A e B, desta forma mesmo que não exista sobreposição é possível computar a distância entre os retângulos.

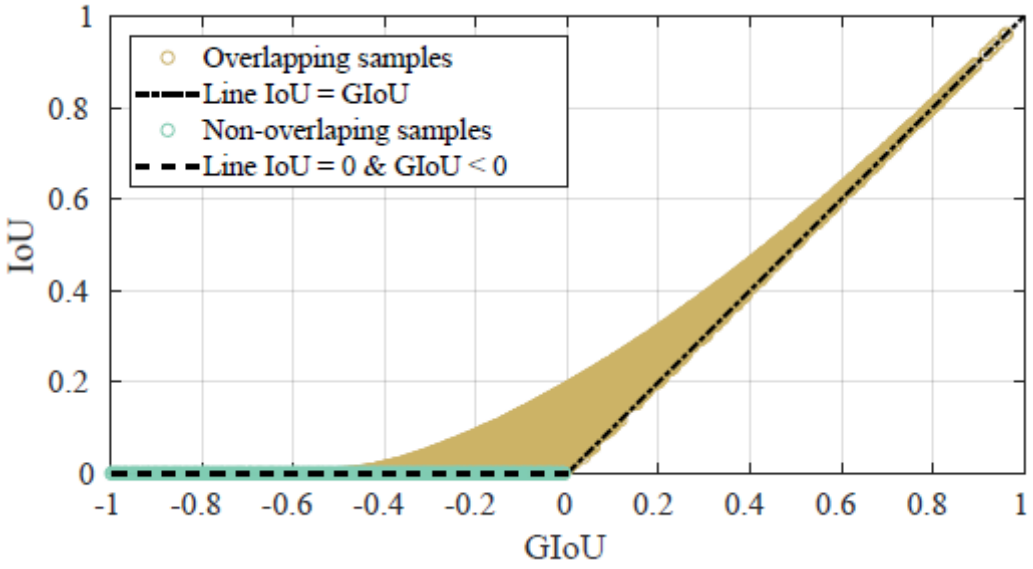
$$GIoU = -\frac{|C \backslash A \cap B|}{|C|} \tag{16}$$

Figura 28 – Exemplo de GIou.

Fonte: Adaptado de Yao et al., 2011

A figura 29 compara as duas métricas, podemos observar que enquanto existe intersecção entre as caixas delimitadoras o GIoU é igual ao IoU. A partir do momento que as caixas não estão mais sobrepostas o IoU é igual a 0 e o GIoU apresenta resultados negativos proporcionais a distâncias das caixas.

Figura 29 – Comparação entre IoU e GIoU.



Fonte: Rezatofighi et al., 2019

2.5.4.3 Mean Average Precision (mAP)

Para calcularmos a precisão média inicialmente precisamos definir um valor de limite de IoU para considerarmos um acerto. Por exemplo, podemos escolher que caso o IoU calculado entre o previsto e o real seja maior que $0{,}5$, consideraremos que o objeto foi detectado corretamente, esta escolha é arbitrária e depende do projeto. Caso seja escolhido um limite maior que $0{,}5$ a correspondência entre o previsto e real precisará ser maior e o modelo será mais preciso, caso um valor menor que $0{,}5$ seja escolhido o modelo priorizará a revocação.

Em seguida, montamos uma tabela como a 5, onde ordenamos as previsões de acordo com o IoU e calculamos a precisão e revocação de cada previsão. Em seguida, representamos na figura 30 o gráfico da precisão por revocação.

A curva B representa o gráfico real, mas geralmente as descontinuidades do gráfico são interpoladas de forma que fique igual ao gráfico C. Em seguida, podemos calcular o mAP conforme definido pela equação (17), onde é basicamente a área sob a curva.

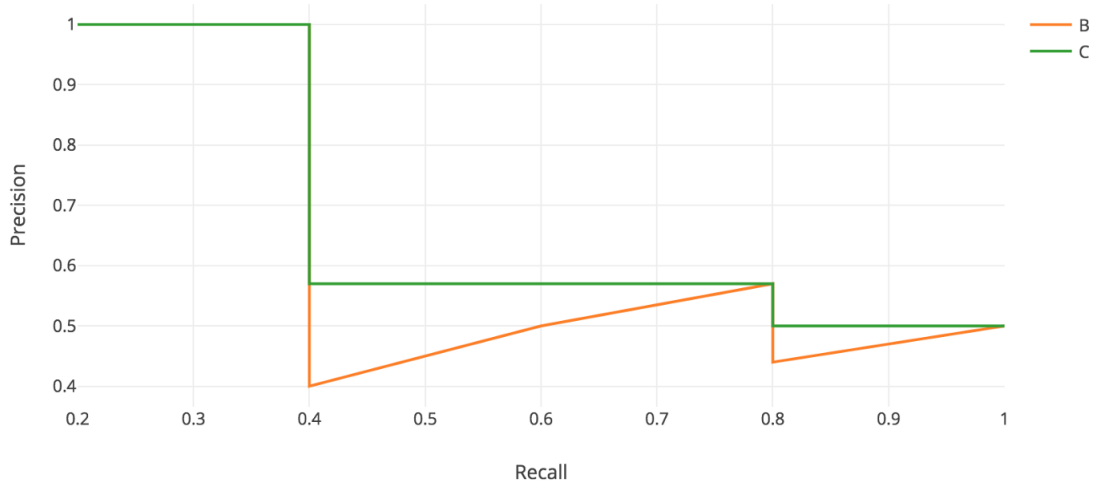
$$AP = \int_0^1 p(r)dr \tag{17}$$

Tabela 5 – Precisão e Recall de classificações

Rank	Correto	Precisão	Recall
1	Sim	1	0,2
2	Sim	1	0,4
3	Não	0,67	0,4
4	Não	0,5	0,4
5	Não	0,4	0,4
6	Sim	0,5	0,6
7	Sim	0,57	0,8

Fonte: HUI, 2020

Figura 30 – Curva de precisão x Revocação.



Fonte: HUI, 2020

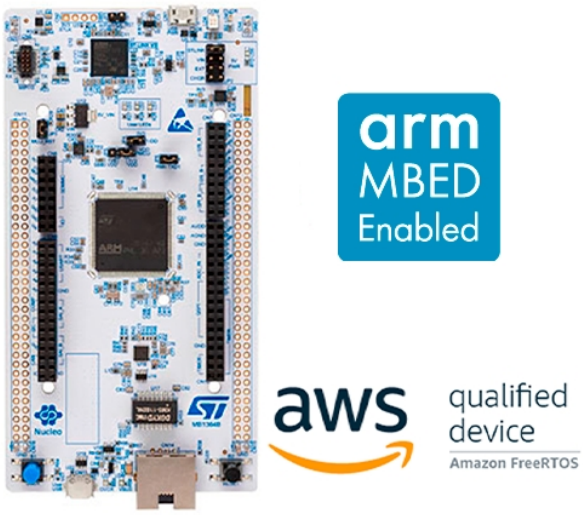
2.6 ESTUDO DE HARDWARE

Inicialmente tínhamos como planejamento utilizar um hardware dedicado para executar as rotinas da rede neural. Com o amadurecimento do projeto e inclusão de novas ideias, decidimos não optar por esse uso do hardware, mas sim utilizá-lo como ponte entre a aplicação e um servidor, que será explicado posteriormente. Como ambas as premissas são aplicáveis, e já havíamos feito tal pesquisa, que pode ajudar a implementações e melhorias futuras, optamos por manter esse estudo. Para escolher qual seria o hardware utilizado no projeto escolhemos algumas placas que são comumente utilizadas no mercado com custos e performances diferentes.

2.6.1 Nucleo-H743ZI

Esta placa na verdade é um kit de desenvolvimento, como mostrado na figura reffig-NucleoH7, que utiliza um microcontrolador STM32H743ZI da ST que recentemente vem investindo bastante recursos em Machine Learning embarcado. Graças ao software MX CUBE fornecido pelo fabricante é possível converter modelos de diversos frameworks como Keras, TensorFlow Lite, Caffe , ConvNetJs para um código C altamente otimizado. As especificações do microcontrolador estão disponíveis na tabela 6.

Figura 31 – Placa NUCLEO-H743ZI.



Fonte: ST, 2020

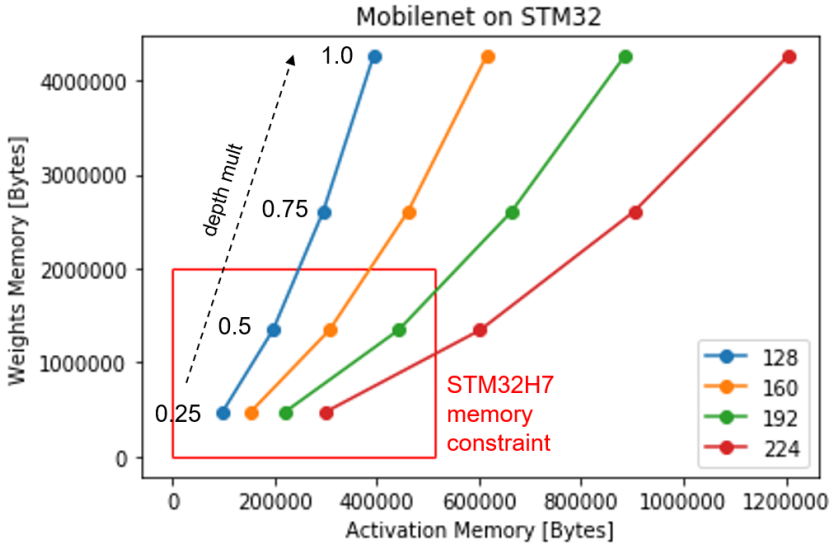
Tabela 6 – Especificações relevantes - STM32H743ZI

Core	32-bit Arm® Cortex®-M7 480 MHz
Flash	2 Mbytes of Flash
RAM	1 Mbytes of RAM
Camera Interface	8- to 14-bit camera interface (up to 80 MHz)

Fonte: ST, 2020

Entretanto, por enquanto apenas pode ser utilizado para inferências mais simples com resoluções baixas como podemos observar na figura 32 que mostra o consumo de memória por resolução de imagem considerando uma arquitetura MobileNet.

Figura 32 – Gráfico de utilização de memória da Mobilenet.



Fonte: RUSCI, 2020

2.6.2 Coral Dev Board

A Coral Dev Board, representada na figura 34, é uma placa desenvolvida para aplicações de inteligência artificial, entre as suas especificações possui um acelerador Google Edge para processamento de tensores, especialmente útil para redes neurais e memória interna suficiente para grandes resoluções de imagem e frames. Entretanto, não é muito acessível financeiramente e precisa ser importada. As especificações mais relevantes estão disponíveis na tabela 7 e uma foto da placa pode ser visualizada na figura 34.

Tabela 7 – Especificações relevantes - Coral Dev Board

CPU	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F)
GPU	Integrated GC7000 Lite Graphics
ML accelerator	Google Edge TPU coprocessor
RAM	1 GB LPDDR4 (option for 2 GB or 4 GB coming soon)
Flash memory	8 GB eMMC
Wireless	Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5GHz) and Bluetooth 4.2

Fonte: CORAL, 2020

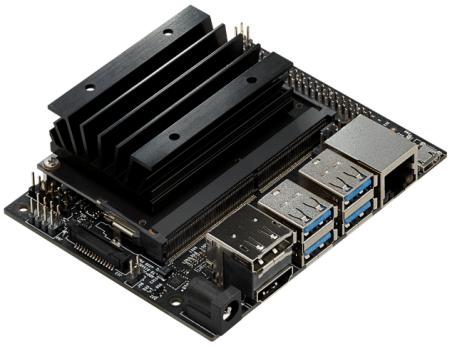
Figura 33 – Placa Coral Dev Board.

Figura 34 – Fonte: CORAL. . . , 2020

2.6.3 Nvidia Jetson Nano

É a placa com hardware embarcado que possui as melhores especificações do mercado, pode ser visualizada na figura 35, criada especificamente rodar múltiplas redes neurais em paralelo e conforme as especificações técnicas da tabela 8 possui interface para duas câmeras permitindo uma resolução de até 4K. O custo desta placa também é bastante alto e não está disponível no Brasil.

Figura 35 – Placa Nvidia Jetson Nano.



Fonte: Developer, 2020

Tabela 8 – Especificações relevantes - Nvidia Jetson Nano

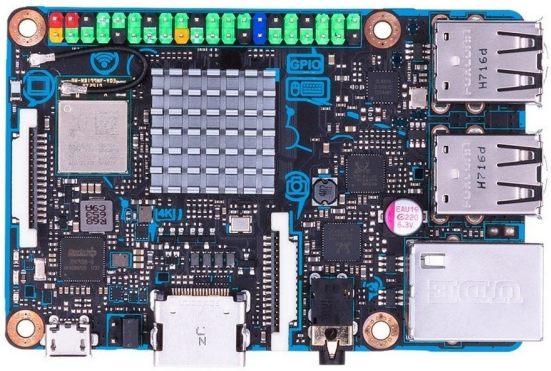
CPU	Quad-core ARM A57 @ 1.43 GHz
GPU	128-core Maxwell
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Video	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes

Fonte: Developer, 2020

2.6.4 Assus Tinker Board R

A Asus Tinker Board R, pode ser visualizada na figura 36 e conforme especificações da tabela 9 possui um hardware bastante eficiente para inteligência artificial contando com um coprocessador para processamento de redes neurais. Entretanto, infelizmente ainda não possui um suporte muito grande de software como em outras plataformas por ser bastante nova no mercado.

Figura 36 – Placa Asus Tinker Board.



Fonte: Asus, 2020

Tabela 9 – Especificações relevantes - Coral Dev Board

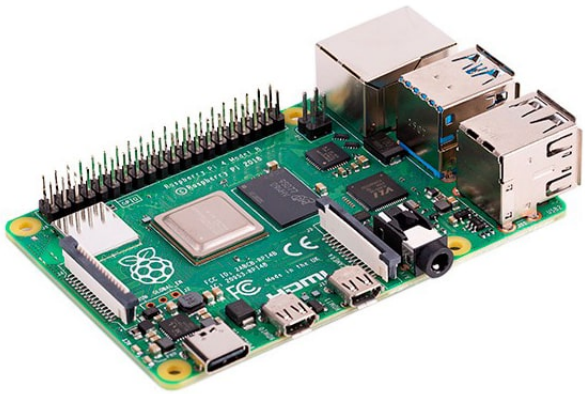
CPU	Hexa core. 2x Cortex-A72 cores up to 1.8 GHz, 4x Cortex-A53 cores @ 1.4 GHz
GPU	800 MHz Mali-T860 MP4 GPU
NPU	3 TOPS of performance
RAM	4 GB dual channel LPDR4 for system, 2 GB LPDDR3 for NPU
Storage	16GB eMMC + removable MicroSD slot (supporting SD 3.0)

Fonte: Asus, 2020

2.6.5 Raspberry Pi 4

A Raspberry Pi já é amplamente conhecida e utilizada no mercado sendo que a maior parte dos softwares de desenvolvimento já foram migrados para a plataforma, e mesmo que não tenha um hardware tão poderoso ou não tenha aceleradores de tensores nativos, possui software e biblioteca bastante otimizados que permitem uma performance aceitável.

Figura 37 – Placa Raspberry Pi 4.



Fonte: RASPBERRY, 2020

Tabela 10 – Especificações relevantes - Raspberry Pi 4

CPU	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz.
GPU	Broadcom VideoCore VI
RAM	4 4GB LPDDR4-2400 SDRAM
STORAGE	Cartão SD Externo

Fonte: RASPBERRY, 2020

2.6.6 Esp32

O Esp32 é um microcontrolador da Expressif, que pode ser visualizado na figura 38 que se tornou bastante popular por integrar conectividade wi-fi e bluetooth a um preço bastante

acessível, contando inclusive com diversas bibliotecas e SDKs que abstraem grande parte da complexidade de protocolos tcp/ip.

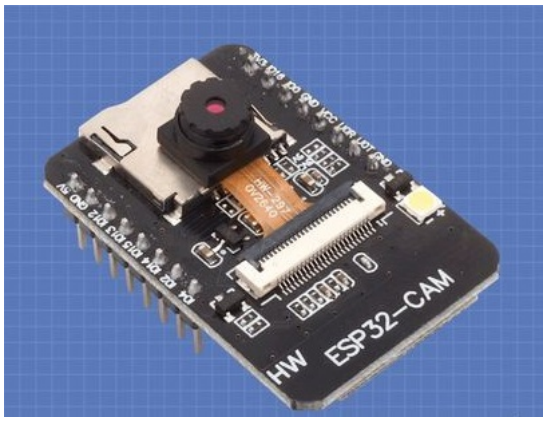
As especificações técnicas da ESP32 são bastante simples, conforme tabela 11 e pode não ser suficiente para processamento de uma Rede Neural Convolucional, entretanto pode ser utilizado em conjunto com um servidor para processamento remoto das informações, podendo ser facilmente integrado devido a grande disponibilidade das bibliotecas.

Tabela 11 – Especificações ESP32

CPU	Clock speed up to 160 MHzz
Bluetooth	4.2 with BLE
Wi-Fi	802.11b/g/n
SRAM	520 KB

Fonte: Workshop, 2020

Figura 38 – ESP32 com módulo de câmera.



Fonte: Workshop, 2020

2.6.7 Câmera OV5647

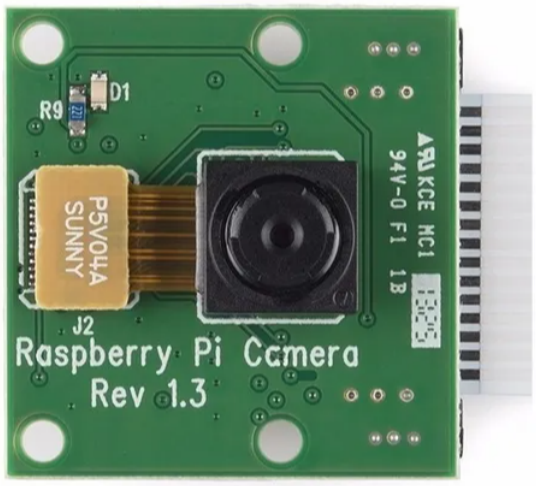
Em decorrência de optarmos pela placa Raspberry Pi 4 para o projeto, escolhemos a câmera da figura 39. Pois ela possui uma boa qualidade de imagem e conexão através de CSI permitindo uma conexão de maior velocidade com a placa escolhida.

Tabela 12 – Especificações relevantes - OV5647

Sensor	1/4” Omnivision OV5647
Pixels	5 Mega Pixels
Tipo de foco	Fixo
Campo de visão	62°

Fonte: Omnivision, 2020

Figura 39 – Câmera OV 5647.



Fonte: Omnivision, 2020

2.6.8 Comparação

A placa Nucleo-H743ZI apresentou um desempenho abaixo do desejável e foi descartada. Enquanto as placas Coral Dev Board e Nvidia Jetson Nano tem desempenho além do necessário, custos bastante elevados, além da baixa disponibilidade para compra no Brasil.

Assim, sobraram duas opções com custos similares, nesse caso elegemos o Raspberry Pi 4 na versão com 4GB de RAM, o melhor hardware para aplicar em nosso projeto, por possuir um suporte maior com relação ao software e ser mais adotada e consolidada pelo mercado mesmo possuindo hardware um pouco inferior.

2.7 ESTUDO DE SOFTWARE

Keras é uma biblioteca open-source (código aberto), de alto nível, para redes neurais e é escrita em Python. Utilizada para implementação de redes neurais, utiliza o Tensorflow como base para funcionar. O Keras acessa o Tensorflow ou Theano para utilizar suas funcionalidades

para o desenvolvimento (LITE, 2020; A.; S., 2017). Benefícios em usar o Keras são diversos por se tratar de uma ferramenta de fácil uso, acessível e modular. Quanto as limitações do Keras temos a dificuldade de trabalhar com redes muito complexas, a necessidade de uma estrutura de back-end (processo interno) (ROSEBROCK, A., 2020; SHAIKH, 2020).

O Tensorflow também é uma biblioteca open-source desenvolvida pelo Google, utilizada em redes neurais e aprendizado de máquina (LITE, 2020; NEURONS, 2020). Benefícios em usar o Tensorflow são devido a sua robusta produção para aprendizado de máquina e de fácil construção de modelos. A principal limitação do Tensorflow é uma biblioteca de baixo nível. Por exemplo, por ser considerado como uma linguagem no nível da máquina, depende muito das especificações do hardware para seu funcionamento eficiente (ROSEBROCK, A., 2020; SHAIKH, 2020).

a) **Arquitetura**:

Keras – Arquitetura simples e fácil

Tensorflow – dá ao Keras sua estrutura

b) **Prototipagem**:

Keras – Modelos Complexos podem ser feitos rapidamente escrevendo códigos no Keras

Tensorflow - Difícil de codificar do zero

c) **Codificação**:

Keras – Diversas funções e modelos complexos podem ser implementados facilmente devido as suas bibliotecas

Tensorflow – Dificuldade alta ao tentar implementar novas funções

d) **Depuração** (debugging):

Keras - Menor necessidade de depuração repetida em redes simples

Tensorflow – Existe um pouco de dificuldade ao tentar depurar um código.

e) **Tempo de treino do modelo**:

Keras – Demora em média 2 horas para cada 4000 passos

Tensorflow – Demora em média 20 minutos para cada 4000 passos

f) **Conjuntos de Dados**:

Keras - Projetado para pequenos conjuntos de dados e execução lenta

Tensorflow - Usado para grande volume de conjuntos de dados e execução mais rápida

g) **Performance**:

Keras – Sua performance é um pouco mais devagar.

Tensorflow – Performance alta e rápida

Após uma análise detalhada de cada uma das ferramentas, foi decidido de comum acordo, que seria utilizado a ferramenta Keras para desenvolvimento do modelo inicial, mesmo com seus contrapontos em relação a desempenho, sua facilidade de uso e sua eficácia na aplicação de nosso trabalho de conclusão de curso. Analisamos também o tempo para aprendizagem da ferramenta, desenvolvimento e execução, seguindo esses fatores verificamos que a optimização do Tensorflow sobrepõe o Keras para sua utilização no raspberry, a principal vantagem foi o tempo de execução que foi possível ver uma melhora de 500% no seu tempo de execução. Teste baseado no processamento de 500 imagens.

2.7.1 Tensor Flow Lite

O Tensor Flow Lite é um framework que permite a otimização de modelos através de quantização das variáveis numéricas de ponto flutuante 32 bits para inteiros 8 bits, permitindo que os modelos sejam executados de forma mais eficiente em hardwares mais limitados.

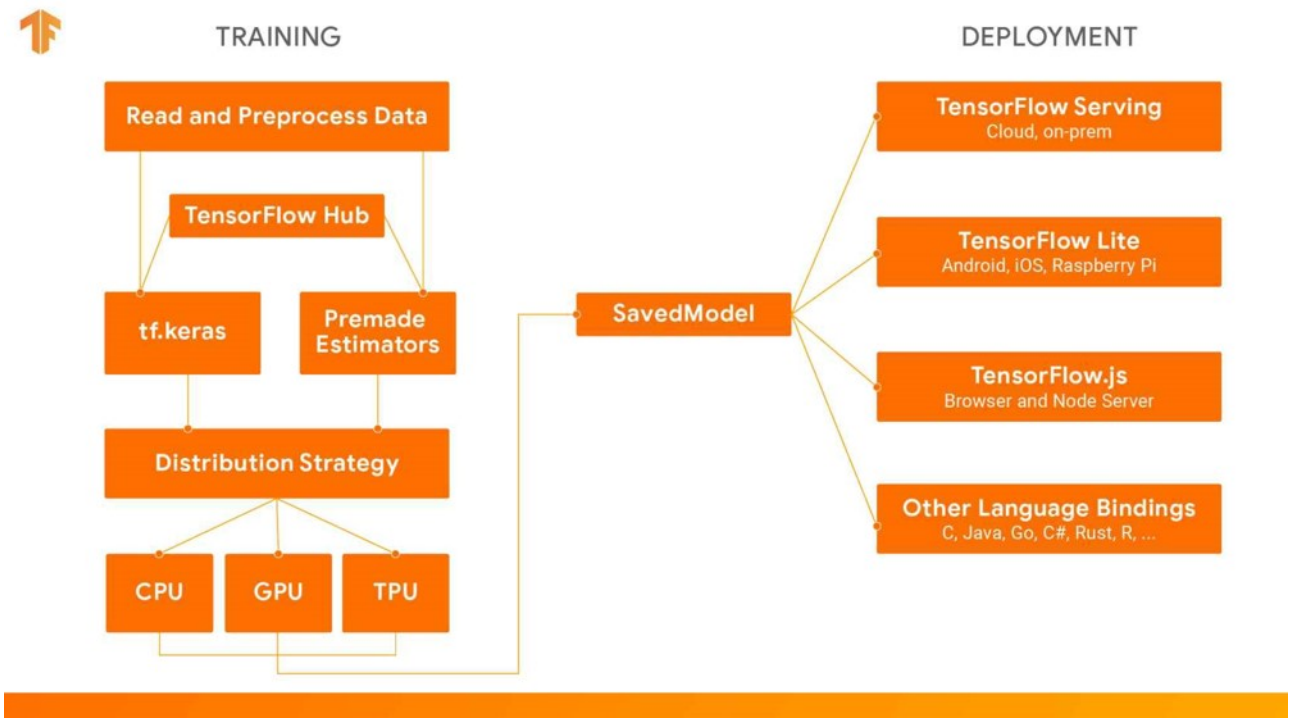
Seu uso seria de extrema importância caso utilizássemos a abordagem descrita no estudo de hardware, de executar as rotinas no próprio Raspbery Pi, por exemplo.

A figura 40 ilustra o funcionando do framework. Um bloco inicial é responsável pela leitura e pré-processamento de dados, esses dados podem ser visualizados em um dashboard, chamado de TensorFlow Hub e podem ser utilizados para treinar modelos existentes ou criados no Keras. Para realizar o treinamento, o ambiente do TensorFlow permite que o processamento seja distribuído em múltiplos computadores com CPUs, GPUs e TPUs. Em seguida, o modelo compilado pode ser executado em um interpretador do TensorFlow disponível em dispositivos mobiles, Raspberry Pi, na nuvem, entre outros.

2.8 ANÁLISE DE ARQUITETURAS

O estudo de visão computacional é bastante antigo, mas nos últimos anos ocorreu um progresso muito grande neste campo devido ao uso de redes neurais convolucionais, permitindo que esses algoritmos funcionem em hardwares cada vez menos robustos, como computadores domésticos, celulares e até mesmo dispositivos eletrônicos embarcados. Neste projeto, o nosso

Figura 40 – Fluxograma de funcionamento do Tensor Flow Lite.



Fonte: (LITE, 2020)

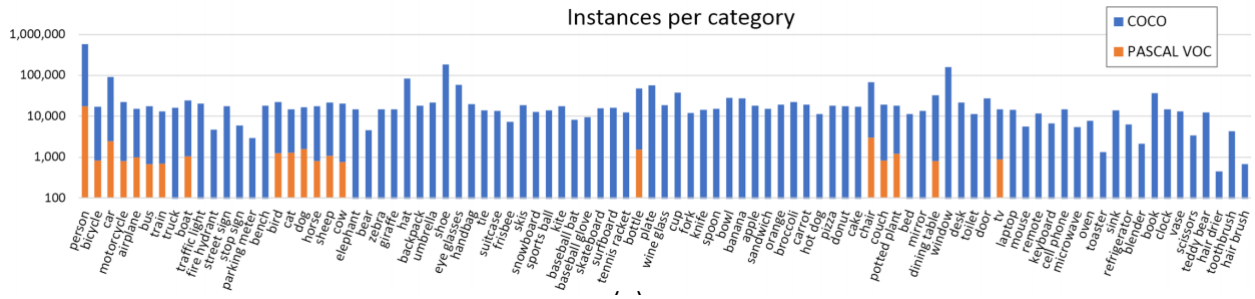
objetivo é executar detecção de objetos em um desses dispositivos, através de uma comunicação com um servidor, portanto devemos escolher arquiteturas que priorizem velocidade de inferência, mas também precisamos de uma boa precisão para funcionar com uma câmera de baixo custo.

2.8.1 Metodologia

O dataset Common Object in Context (COCO) é uma famosa coleção de imagens amplamente utilizada para avaliação de arquiteturas de detecção de objetos, este dataset possui 2.500.000 instâncias de 91 categorias em 328.000 imagens, como podemos observar na figura 41. Estes dados serão a base para comparar as diversas arquiteturas de segmentação e classificação existentes.

Conforme discutimos na 2.5.4.3, o AP é uma métrica muito utilizada para comparação de arquiteturas, na tabela 13 temos algumas variações definidas pelo COCO Dataset para diferentes tamanhos de imagens.

Figura 41 – Dados disponíveis no COCO Dataset.



Fonte: Lin et al., 2014

Tabela 13 – Métricas definidas pelo COCO

AP para diversos tamanhos	
AP_S	AP para objetos pequenos: area < 32^2
AP_M	AP for objetos médios: 32^2 < area < 96^2
AP_L	AP for objetos grandes: area > 96^2

Fonte: COCO, 2020

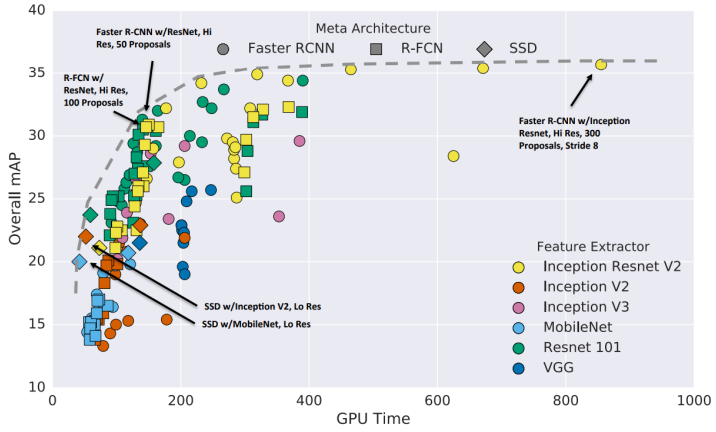
2.8.2 Estudo das arquiteturas

Neste projeto, em decorrência da limitação do hardware, é necessário uma arquitetura que priorize a quantidade de quadros por segundo (FPS), mas também tenha uma boa precisão (AP).

O artigo (FATHI et al., 2017) analisou algumas soluções mais otimizadas para dispositivos com hardware menos robustos temos a comparações de alguns tipos de arquiteturas de segmentação e classificação.

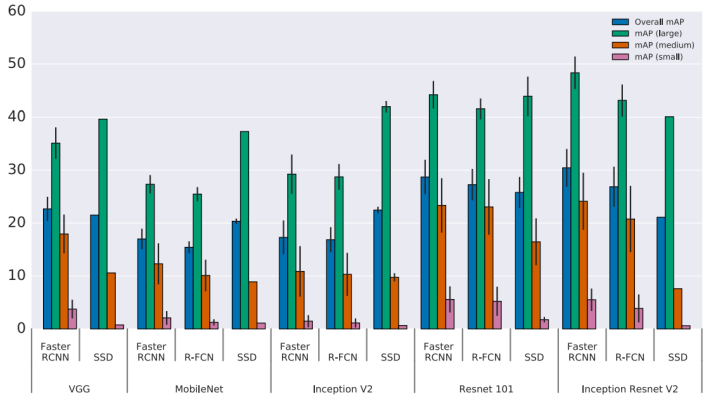
Observando a figura 42 estão representadas algumas arquiteturas de classificação através das cores e a rede segmentação através da figura geométrica, além disso, quanto menor o tempo de GPU e maior o mAP melhor, portanto a arquitetura de classificação que se destaca é o SSD por apresentar os menores tempos de inferência entre os modelos, principalmente em conjunto com a Mobilenet. As arquiteturas Inception V2 e Resnet 101 junto com o SSD também apresentaram resultados próximos em relação a velocidade e uma precisão um pouco melhor. Nas figuras 43 e 44 também podemos verificar em maiores detalhes a comparação entre precisão e velocidade de cada uma das arquiteturas.

Figura 42 – Comparação de precisão das principais arquiteturas.



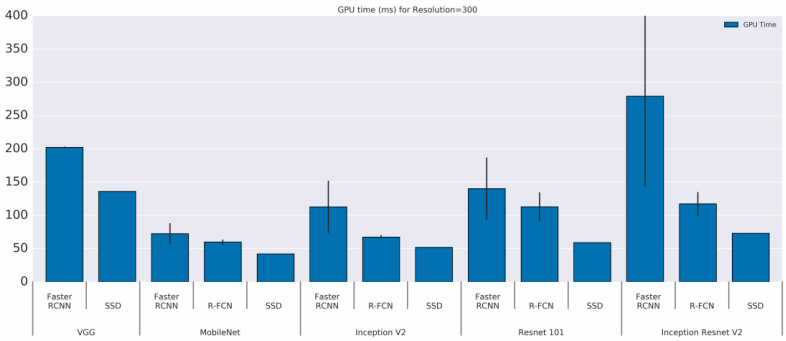
Fonte: Fathi et al., 2017

Figura 43 – Comparação de precisão das principais arquiteturas.



Fonte: Fathi et al., 2017

Figura 44 – Comparação da velocidade das principais arquiteturas.



Fonte: Fathi et al., 2017

2.8.3 MobileNet

A MobileNet foi introduzida no artigo (HOWARD et al., 2017), ela tem esse nome por ser uma arquitetura focada em dispositivos mobiles como smartphone, apesar disso, a arquitetura pode ser utilizada em qualquer dispositivo sendo especialmente útil para hardwares menos robustos como microcontroladores e microcomputadores como o Raspberry PI.

2.8.3.1 Convolução Separável em Profundidade

O principal motivo da performance da MobileNet é graças a *"Depthwise Separable Convolution"* introduzida no artigo (HOWARD et al., 2017). Este tipo de camada substitui uma convolução tradicional por dois dois novos tipos de filtros de convolução, apelidados pelo paper de Depthwise Convolutional Filters e Pointwise Convolution.

Como podemos observar na figura 45 a camada convolução tradicional é composta por N filtros com $D_k x D_k x M$. É possível decompor essa mesma convolução em duas equivalentes, com N filtros de $D_k x D_k x 1$ e M filtros de $1x1xM$. O primeiro filtro é o chamado Depthwise Convolutional Filters e o segundo o Pointwise Convolution, juntos formam a Depthwise Separable Convolution.

Não entraremos em detalhes neste texto, mas é demonstrado matematicamente no artigo 45 como essa separação diminui drasticamente o custo computacional e complexidade da arquitetura.

2.8.3.2 Batch Normalization

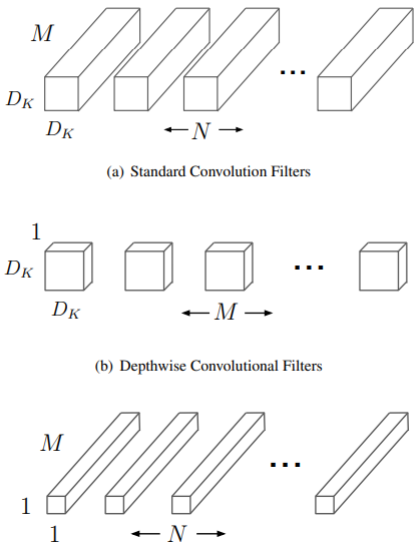
Por último, a normalização em lote é bastante utilizada nas arquiteturas MobileNet e MobileNetV2 para aumentar a estabilidade do sistema. A normalização em lote normalmente é aplicada nas camadas intermediares para que as saídas dos neurônios tenham escala parecida.

Por exemplo, pode acontecer que um neurônio de uma terminada camada possua sua saída entre 0 e 1, enquanto outro possua sua saída entre 0 e 1000.

A normalização em lote traz benefícios como aumento na velocidade de treino e diminui o efeito do sobreajuste.

Conforme introduzido no artigo (IOFFE; SZEGEDY, 2015), a normalização em lote consiste, em selecionar pequenos lotes do dataset ($B = X_1...X_m$) calcular a média, conforme

Figura 45 – Convolução Separável em Profundidade



Fonte: Howard et al., 2017

equação (18), variança deste lote, como na equação 19 e em seguida realizar a normalização, equação 20.

Em seguida, é aplicado um fator de escala γ e um nivel médio β, que são parâmetros treináveis calculando-se a saída da camada, como na equação (21).

$$\mu_\beta = \frac{1}{m} * \sum_{i=1}^{m} x_i \tag{18}$$

$$\sigma_\beta^2 = \frac{1}{m} * \sum_{i=1}^{m} (x_i - \mu\beta)^2 \tag{19}$$

$$\mu_\beta = \frac{x_i - \mu\beta}{\sqrt{\sigma^2\beta + \epsilon}} * \sum_{i=1}^{m} x_i \tag{20}$$

$$y_i = \gamma x_i + \beta \tag{21}$$

2.8.4 MobileNetV2

A MobileNetV2 foi apresentada no artigo (SANDLER et al., 2018) e expande os conceitos apresentados pela MobileNet implementando uma nova camada chamada residual invertida.

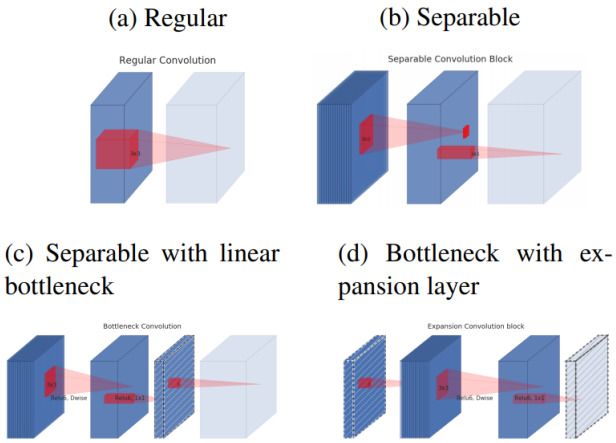
2.8.4.1 Bottlenecks Lineares

A ReLU é uma das funções de ativação mais utilizadas, presente em praticamente todas as arquiteturas modernas, isso é devido a sua não linearidade que permite representar sistemas mais complexos.

Entretanto, conforme provado no artigo (SANDLER et al., 2018), quando linearidades são adicionadas em imagens com baixa dimensionalidade ocorre uma grande perda de informações. Uma estratégia para reduzir está perda de informações consiste expandir uma imagem em uma dimensão maior, aplicando funções de ativação lineares, que evitam a perda de informação e quando o vetor estiver em uma dimensão alta aplicar a função de ativação não linear.

Na figura 46, podemos observar que inicialmente temos uma imagem de baixa dimensão, no caso da MobileNet por exemplo, a entrada da arquitetura é uma imagem com 224x224x3, ou seja a dimensão é 3. Em seguida a dimensão dessa imagem é expandida, onde as transformações não lineares são aplicadas. Por fim, existe uma nova redução de dimensionalidade. É importante ressaltar que as convoluções realizadas são do tipo separável em profundidade .

Figura 46 – Evolução dos blocos de convolução separados.



Fonte: Sandler et al., 2018

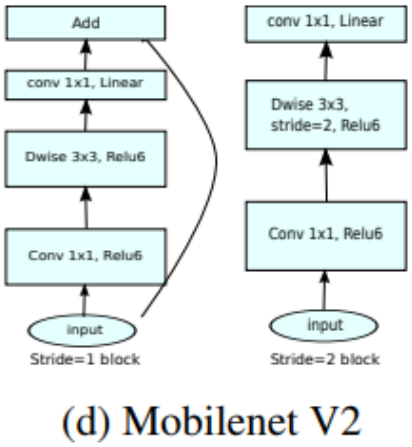
2.8.4.2 Conexão Residual Invertida

Além disso, conforme imagem 47, caso o valor de stride da camada bottleneck seja igual a 1, a entrada é adicionada a saída da camada, essa adição é chamada de resíduo e essa conexão

é a residual invertida, pois a conexão residual tradicional conecta camadas de dimensão alta, enquanto a bottleneck conecta imagens de baixa dimensão conforme comparação da figura 48 .

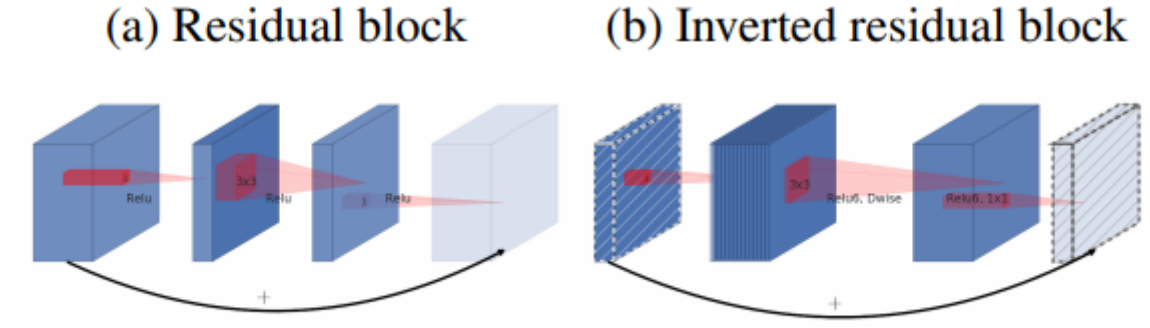
Este tipo de conexão é criado para evitar a perda de informação durante a convolução.

Figura 47 – Residual Invertida nos bottlenecks.



Fonte: Sandler et al., 2018

Figura 48 – Residual x Residual Invertida.



Fonte: Sandler et al., 2018

2.8.4.3 Visão Geral da Arquitetura

A MobileNet é composta das camadas da imagem 49, onde Input são as dimensão do vetor de entrada (largura=altura x canais), t é o fator de expansão, abordado na seção 2.8.4.1, c é o numero de canais do vetor de saída, n é o numero de repetições da camada e o s é o fator de

stride da convolução, explicado na seção 2.4.1.1. É possível visualizar todos esses elementos na implementação da arquitetura realizada pelo grupo no apêndice B.

Figura 49 – Camadas da MobileNetV2 simplificada.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Fonte: Howard et al., 2017

2.9 APLICAÇÃO WEB

Após desenvolvimento inicial do projeto, foi estudada uma forma de disponibilizar a aplicação para os usuários, inicialmente foi considerado distribuir um pacote completo com os modelos treinados, entretanto, para executar o modelo completo é necessário possuir uma série de dependências instaladas no computador e um hardware que consiga executar de maneira apropriada.

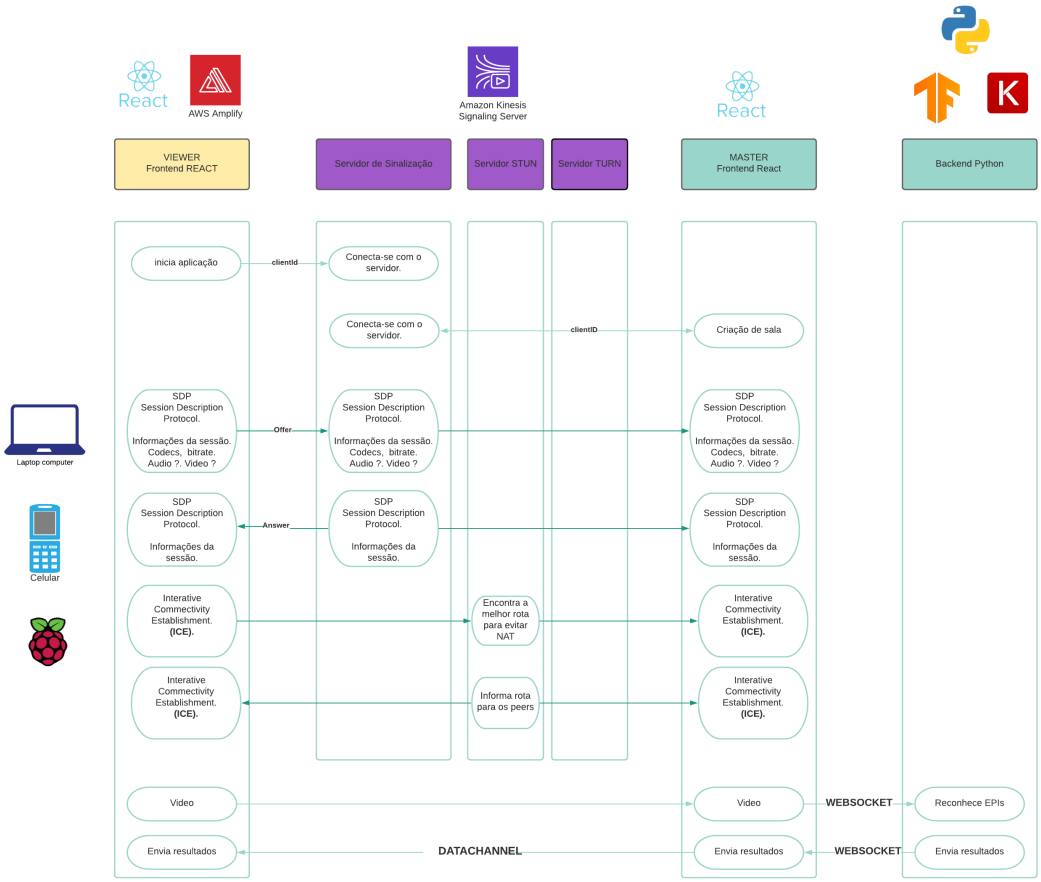
2.9.1 Arquitetura Cloud da Aplicação

Tendo em vista as dificuldades de se distribuir uma aplicação completa, optamos por uma aplicação WEB que processa as informações em um servidor remoto.

Definimos a arquitetura apresentada na figura 50, composta por uma aplicação FrontEnd que será acessada através de uma página web, construída em ReactJs e hospedada no serviço AWS Amplify da Amazon, conforme o protocolo Web Real-Time Communication (WebRTC) o servidor de sinalização que será melhor explorado na seção 2.9.3.1, hospedado no serviço AWS Kinesis permitirá a conexão entre a aplicação do usuário, chamada Viewer, e a do servidor, chamada Master também desenvolvida em React. O WebRTC é composto dos subprotocolos Interactive Connectivity Establishment (ICE), que permite que a conexão seja estabelecidada,

e o Session Description Protocol (SDP) que estabelece os dados que serão trafegados após finalizar a conexão. Em seguida o Master envia os dados para um servidor local, através do protocolo Websockets que executa os modelos de machine learning e retorna uma resposta pelo mesmo protocolo.

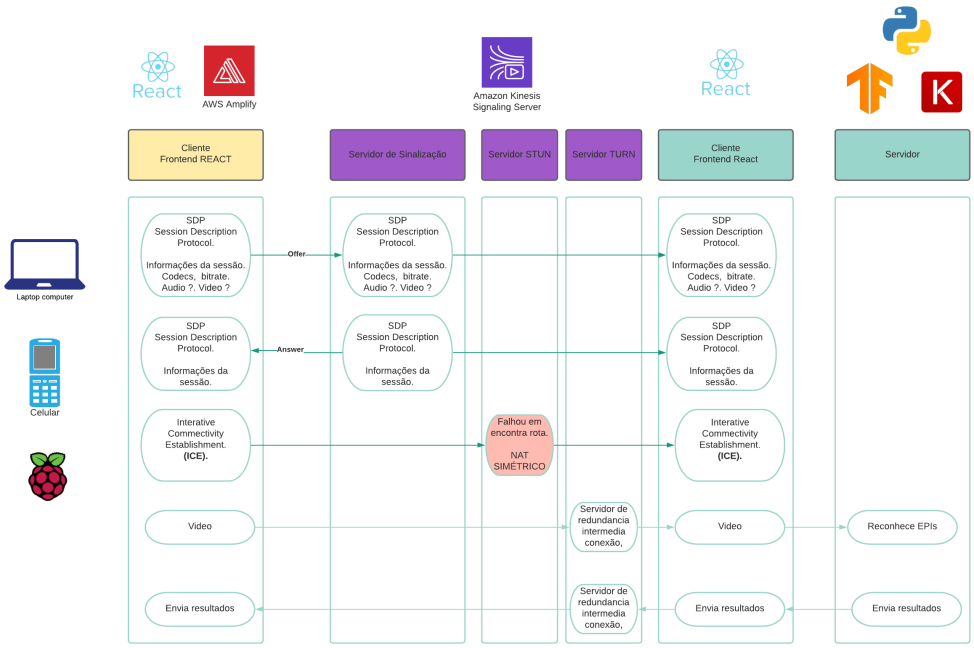
Figura 50 – Arquitetura da aplicação.



Fonte : Autores

Caso o protocolo SDP falhe em encontrar uma rota entre os dois peers, normalmente ocorre com Network Address Translator (NAT) simétrico, o servidor de redundância Traversal Using Relays around NAT (TURN) é acionado intermediando a comunicação e arquitetura da aplicação segue de forma similar a figura 51.

Figura 51 – Arquitetura da aplicação NAT Simétrico.



Fonte : Autores

2.9.2 ReactJs

No desenvolvimento de aplicações web clássico as páginas são dividas em três tipos de arquivos, os arquivos html que guardam a estrutura da páginas, os arquivos css que cuidam da aparência da página e os arquivos javascript que implementam toda a lógica de programação para interatividade da página. Além disso, as aplicações tradicionais utilizavam um servidor (normalmente chamado de backend) que retornava todos os arquivos sempre que alguma seção da aplicação fosse acessada, necessitando que toda aplicação fosse carregada sempre que houvesse alguma interação do usuário.

As aplicações mais modernas são constituídas apenas de um arquivo html que é populado através de dados recebidos pelo backend, esses dados normalmente são recebidos no formato JSON, e então um código javascript é responsável por ler e alterar o html de acordo com os dados recebidos, dessa forma a página não precisa ser recarregada integralmente sempre que um recurso é acessado, este tipo de aplicação é chamada de página estática.

O ReactJs é um framework que facilita o processo de se desenvolver uma aplicação estática, pois ele permite que os arquivos html e javascript sejam escritos em um único arquivo chamado de Jsx, desta forma é possível alterar o html diretamente utilizando o javascript. Por

exemplo, imagine que uma aplicação de cronometro que atualiza a página a cada 1 segundo, utilizando javascript puro, seria necessário acessar o elemento html que exibe o cronometro, através de um ID, a cada segundo e altera-lo manualmente, o React permite que a variável javascript seja adicionada diretamente no html e seja atualizada simultaneamente em ambos.

2.9.3 Protocolos WebRTC

O Web Real-Time Communication (WebRTC) é uma coleção de protocolos, padrões e APIS em javascript que permitem uma conexão direta para envio de vídeo, áudio e dados entre dois computadores (Peers) através da internet. Este protocolo é open source e foi iniciado pelo Google em 2011 e teve a sua primeira versão com transmissão video em 2013. Em seguida passou a ser adotado por grandes serviços como o Google Hangout, Google Meet, Facebook Messenger, Discord, Webbex, entre outros.

2.9.3.1 Servidor de Sinalização

Realizar a conexão entre dois dispositivos através da internet não é uma tarefa simples, pois além de ser necessário localizar o endereço de destino existem barreiras como o NAT que impossibilita que a rede interna de outro dispositivo seja acessada diretamente. O servidor de sinalização é necessário no inicio de toda comunicação através do protocolo para que sejam trocadas mensagens entre os peers antes que a conexão seja estabelecida.

2.9.3.2 Session Description Protocol (SDP)

O WebRTC utiliza o protocolo SDP para descrever os parâmetros de vídeo que serão trafegados após a conexão, como por exemplo, tipo de multimédia (vídeo, áudio e dados), tipo de transporte (TCP ou UDP), codecs que serão utilizados e suas configurações, informações de largura de banda e outros metadados.

O SDP é um protocolo de texto descrito no documento (F.; M., 2006), onde um dos peers gera uma oferta com as suas informações conforme descrito e envia através do servidor de sinalização para o outro Peer, o outro peer registra essas informações e envia uma resposta indicando se aceita e suporta todas as configurações multimédia.

```
        (... snip ...)
m=audio 1 RTP/SAVPF 111 ...
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=candidate:1862263974 1 udp 2113937151 192.168.1.73 60834 typ
host ..
. a=mid:audio
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10
(... snip ...)
```

2.9.3.3 Interactive Connectivity Establishment (ICE)

Estabelecer a conexão entre dois peers não é uma tarefa simples, pois existem inúmeras barreiras como o NAT e Firewalls que impossibilitam uma conexão direta.

O protocolo ICE, descrito no documento (A.; C.; J., 2018), simplifica esse processo através dos agentes ICE que são responsáveis por encontrar duplas de IP e portas, chamadas de candidatas, checar se é possível estabelecer uma conexão através dessas portas e por manter a conexão aberta.

Para localizar as portas o agente ICE utiliza de um servidor acessível através da internet que é chamado de Session Traversal Utilities for NAT (STUN), que implementa um protocolo padrão de métodos de atravessamento de NAT.

Caso o servidor STUN falhe em encontrar um caminho entre os dois peers, pode ser utilizado um servidor de redundância chamado de Traversal Using Relays around NAT (TURN). O servidor TURN funciona como um intermediário recebendo todos os dados de um peer e enviado para o destinatário.

Na aplicação clássica do protocolo ICE, os candidatos são enviados em conjunto com o protocolo SDP, causando um grande atraso na conexão, pois os candidatos de ambos os peers tem que ser encontrados para que se inicie a transmissão. Para resolver esse problema uma extensão chamada de Trickle ICE foi adicionada ao protocolo, que permite que o SDP seja enviado sem os candidatos e assim que os candidatos estejam prontos eles são enviados através do servidor de sinalização.

2.9.3.4 WebSockets

Na web atual a maior parte da informação é transitada através do procolo Hypertext Transfer Protocol (HTTP) que é um protocolo Half-Duplex síncrono onde a conexão é aberta, uma solicitação é enviada para o servidor e quando a resposta é recebida pelo cliente a conexão é fechada.

O Websocket foi definido no documento (I. et al., 2018) para possibilitar a transmissão de informações de forma bidirecional (Full-duplex) em um socket de conexão TCP, este tipo de conexão é iniciada através do protocolo HTTP através de um troca de informações chamada de handshake. Após finalização do handshake a conexão fica aberta e a informação pode ser trocada a vontade entre o cliente e servidor até que um dos pares decida fechar a conexão.

Este tipo de protocolo é muito utilizado em chats, jogos online ou qualquer tipo de aplicação em que o servidor precisa ser capaz de enviar informações para o cliente de forma assíncrona, ou seja, por iniciativa própria sem que o cliente realize uma requisição .

2.9.3.5 DataChannel

Assim que a conexão é estabelecida entre os peers, também é possível criar uma conexão para o envio de dados. Chamado de DataChannel, este tipo de conexão é do tipo fullduplex (Bi-direcional) e assíncrona, se assemelhando bastante com o protocolo Websockets só que peer to peer e toda a informação é criptografada .

2.10 AMAZON WEB SERVICES

A Amazon Web Services (AWS) é maior e mais utilizada plataforma cloud do mundo, com mais de 175 serviços presentes em data centers em todo o mundo. Utilizaremos alguns dos serviços da AWS para construção de nossa infraestrutura cloud.

2.10.1 AWS Amplify

O AWS Amplify é um conjunto de bibliotecas e serviços da Amazon que permite a hospedagem de aplicações estáticas, como as desenvolvidas com ReactJs. O Amplify ainda possibilita a utilização gratuita de um certificado SSL que permite que os dados sejam trafega-

dos de forma segura através da internet, o que também é um requisito exigido pelos navegadores para que a imagem da webcam dos usuários seja acessada.

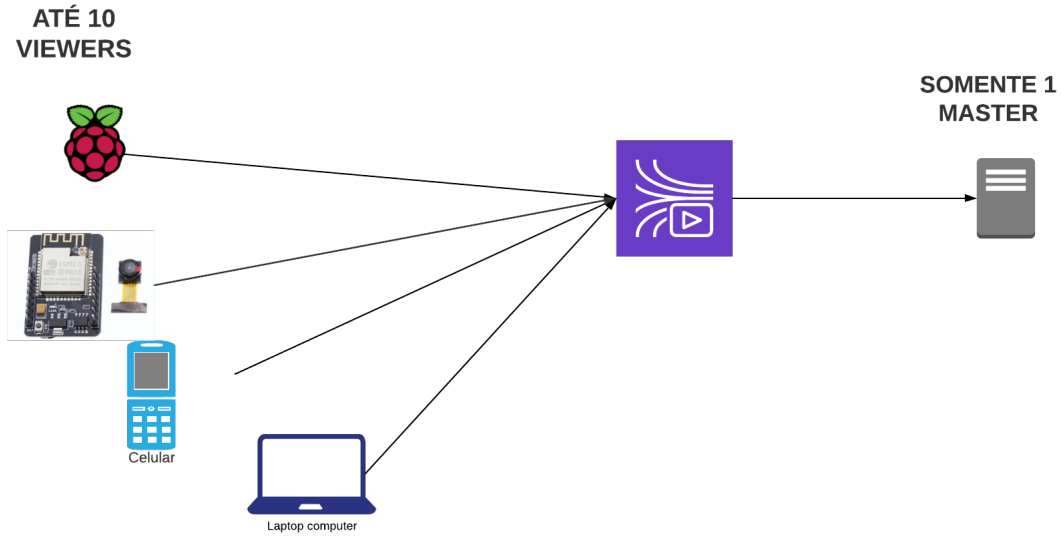
Além da hospedagem da aplicação frontend, o Amplify permite autenticação utilizando um servidor próprio e link de conta através de provedores de redes sociais como Facebook, Google Sign-In, ou Login With Amazon. Desenvolvimento de aplicações mobile IOS/Android com React Native. Serviço de análise de dados de uso, entre outros .

2.10.2 Amazon Kinesis Video Stream

O Amazon Kinesis Video Streams é a implementação do protocolo WebRTC no lado do servidor, na forma de um microserviço da AWS, incluindo servidores de sinalização, TURN e STUN para atravessamento de NAT.

A versão da AWS impõe uma limitação ao protocolo e introduz os conceitos de Viewer e Master. No Kinesis um servidor de sinalização é limitado a 10 peers chamados de Viewers . e 1 peer chamado de Master , onde os viewers não podem se comunicar entre si e apenas podem se comunicar com o Master como na figura 52. .

Figura 52 – Arquitetura da aplicação.



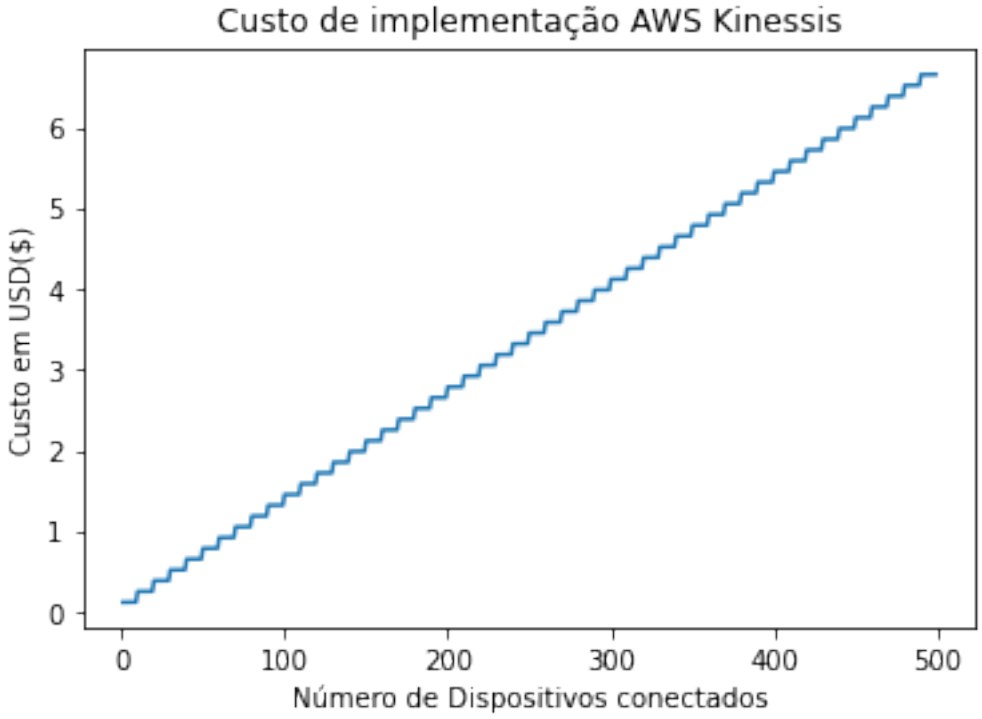
Fonte: Autores

O Kinessis é um serviço serverless, que é uma modalidade de servidor que apenas é iniciado quando existe demanda dos usuários, de forma que o tempo ocioso do mesmo não é

cobrado. Além disso, elimina a necessidade de especificar a capacidade de processamento ou memória para a aplicação, tudo isso é gerenciado pela própria AWS.

Cada servidor de sinalização permite a conexão de até 10 usuários simultâneos, sendo cobrado um custo fixo de USD 0.13 para cada instância e mais USD 6.2 para cada milhão de mensagens trocadas com o servidor de sinalização. Na figura 53, podemos observar a evolução do custo em função do número de usuários, sendo que para 500 usuários temos um custo mensal inferior a USD 7.

Figura 53 – Custo com servidores da aplicação.



Fonte: Autores

3 PROCEDIMENTOS METODOLÓGICOS

Como demonstrado na figura 50 a solução do projeto está dividida em um servidor que possuirá um modelo de CNN treinado. E uma aplicação que servirá para enviar os dados do usuário para serem processados no servidor, que retornará os EPIS utilizados e posição do rosto.

Neste capitulo, descreveremos como foi implantação destas etapas.

3.1 TREINAMENTO DO MODELO DE CNN

Inicialmente propomos utilizar modelos de CNN para detecção de objetos, cujo objetivo era identificar os Equipamentos de Proteção Individual. Ao longo dos estudos e desenvolvimento de códigos iniciais, verificamos que tal abordagem demandaria muitos recursos da máquina, principalmente poder de processamento, de forma que, muitas vezes, não era possível prosseguir com o treino ou executar inferências.

Pensando em outras possíveis abordagens, verificamos que poderíamos utilizar detectores de face, que são, em resumo, detectores de objetos que procuram faces humanas, e a partir da face encontrada, selecionaríamos essa determinada área e nela iríamos identificar quais os EPIs utilizados ou não pela pessoa. Com isso teríamos um modelo para detecção de faces previamente treinado com inúmeros rostos humanos, validado e preciso, além de rápido e otimizado.

A próxima etapa seria escolher tal modelo, e optamos por um modelo de aprendizado profundo Convolutional Architecture for Fast Feature Embedding (Caffe), pré-treinado e fornecido pela OpenCV. Este modelo detecta e localiza faces em uma imagem. O modelo foi criado com estrutura SSD usando arquitetura ResNet-10 como backbone. O modelo foi treinado na estrutura Caffe em um conjunto de dados enorme. A escolha se deu por alguns motivos, destacamos a estrutura SSD, que era nosso foco inicial para a detecção dos EPIs, e por ser um modelo Caffe, que garante qualidade e velocidade, além da facilidade de integração na montagem final com outros modelos, é uma estrutura de aprendizado profundo feita com expressão, velocidade e modularidade em mente. Foi desenvolvido pela Berkeley AI Research (BAIR) e por colaboradores da comunidade. Yangqing Jia criou o projeto durante seu PhD na UC Berkeley (JIA et al., 2014).

Para melhorar o desempenho, treinamos cada EPI em um modelo binário, ou seja, positivo e negativo de determinado equipamento. O intuito foi melhorar a precisão, já que em modelos com diversos possíveis valores de saída, a precisão costuma ser menor, algo que demandaria

mais ajustes e processamento, tanto nos datasets, quanto nos parâmetros de treinamento, além de dificultar a combinação de mais de um EPI.

Definidas as condições acima, vamos montar as estruturas necessárias para o treinamento e teste práticos.

```
+---dataset_earmuff
|   +---earmuff
|   \---without_earmuff
+---dataset_glass
|   +---glass
|   \---without_glass
+---dataset_helmet
|   +---helmet
|   \---without_helmet
+---dataset_mask
|   +---mask
|   \---without_mask
+---examples
+---face_detector
|       deploy.prototxt
|       res10_300x300_ssd_iter_140000.caffemodel
+---modelos_treinados_finalizado
|   +---earmuff
|   +---glass
|   +---helmet
|   \---mask
|   train_earmuff_detector.py
|   train_glass_detector.py
|   train_helmet_detector.py
|   train_mask_detector.py
|   earmuff_detector.model
|   glass_detector.model
|   helmet_detector.model
|   mask_detector.model
|   plot_earmuff.png
|   plot_glass.png
|   plot_helmet.png
|   plot_mask.png
\---output
```

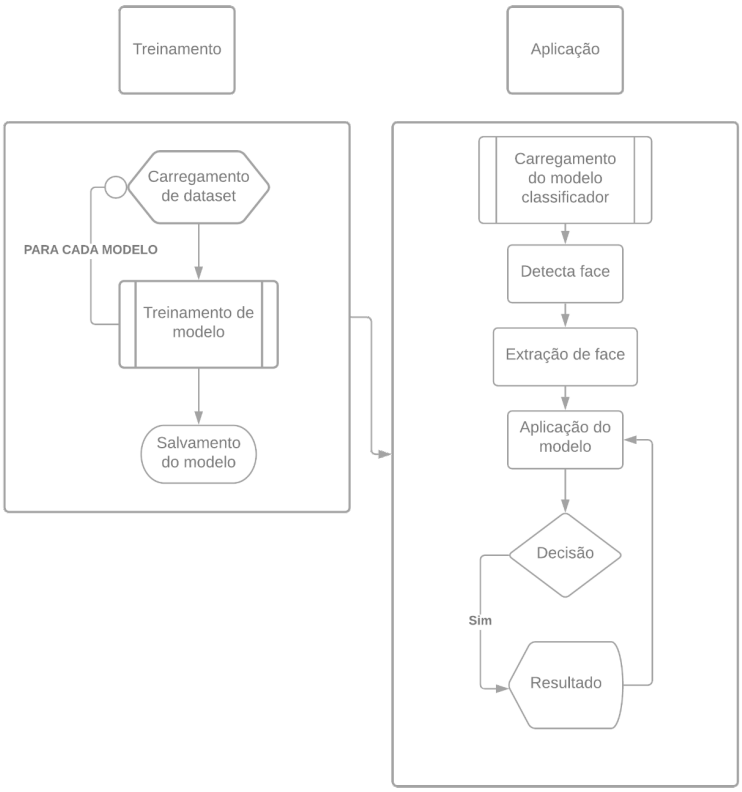
Como discutido anteriormente, vamos utilizar o backbone MobileNetV2 para nossa rede neural, por ser uma arquitetura altamente eficiente que pode ser aplicada a dispositivos embarcados com capacidade computacional limitada (por exemplo, Raspberry Pi).

Realizamos a codificação da MobileNetV2, utilizando a API funcional do Keras, o código completo está disponível no apêndice A, assim como a arquitetura completa no apêndice B .

Com ilustrado na figura 54, vamos separar o detector de EPI em duas fases, treinamento e aplicação do detector. As etapas do treinamento serão, carregar o dataset de cada EPI, treinar o modelo classificador usando Keras/TensorFlow, salvar o modelo classificador.

As etapas de aplicação serão, carregar o modelo classificador, detectar faces em uma imagem ou vídeo, extrair cada face (região de interesse), aplicar o modelo classificador na região extraída, determinar o uso ou não do EPI e mostrar o resultado (output).

Figura 54 – Fluxograma, etapas: treinamento e aplicação.



Fonte: Autores

3.1.1 Preparação do dataset

Para nosso treinamento foi necessário a construção de datasets para cada EPI. Usamos algumas imagens de domínio público da internet, porém a maioria foi obtida por meio do apoio dos servidores da seção de segurança do trabalho do Arsenal de Guerra de São Paulo, que gentilmente nos apoiaram com os Equipamentos e serviram de modelos para a construção dos nossos próprios datasets, conforme autorização do uso de imagem no apêndice H. Um ponto positivo de usar imagens de pessoas que não são integrantes do grupo é que isso garante, nos testes práticos, resultados não enviesados.

Com as imagens obtidas, teríamos de ajustá-las para melhor corresponder a região de interesse (ROI), ou seja, recortá-las para obter somente a face. Esse recorte tem a função de utilizar somente essa ROI para ser treinada, já que na detecção, será também buscada uma ROI .

Considere o exemplo da figura 55 onde rosto detectado foi recortado como o exemplo da figura 56.

Figura 55 – Imagem original, dos servidores com EPIs diversos, obtida por vídeo (frame).

Fonte: Autores

Após o recorte das faces foi necessário organizar os datasets, com imagens positivas e negativas para cada EPI a ser treinado. Verificamos, também, que colocar outros EPIs no

Figura 56 – Imagem recortada, com a região de interesse (ROI).

Fonte: Autores

negativo do dataset, assim como combinar EPIs, sendo o específico do dataset mais outro, no positivo, ajudava a melhor a precisão, podemos ver alguns exemplos no apêndice E.

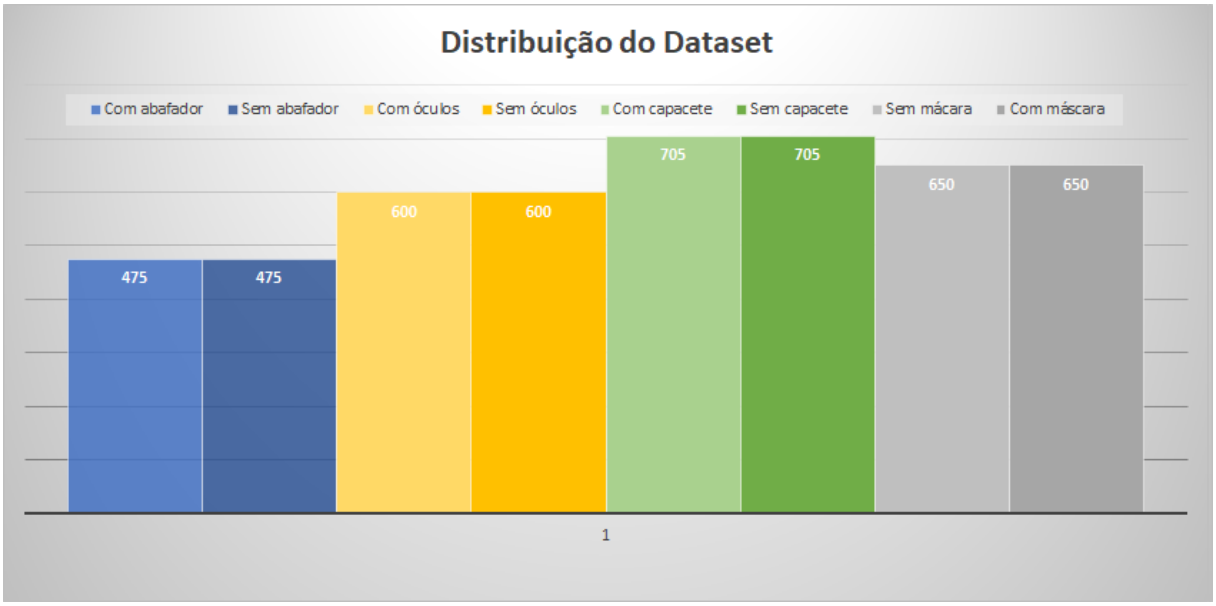
Os datasets ficaram da seguinte forma:

```
+---dataset_earmuff
|   +---earmuff (475 imagens)
|   \---without_earmuff (475 imagens)
+---dataset_glass
|   +---glass (600 imagens)
|   \---without_glass (600 imagens)
+---dataset_helmet
|   +---helmet (705 imagens)
|   \---without_helmet (705 imagens)
+---dataset_mask
|   +---mask (650 imagens)
|   \---without_mask (650 imagens)
```

Como é possível observar na figura 57 nos preocupamos em manter os datasets balanceados em quantidade para evitar qualquer viés, de forma que os datasets da cada EPI possuem a mesma quantidade de exemplos negativos e positivos.

Uma dificuldade em particular, foi a elaboração do dataset dos óculos de proteção, depois de alguns testes, com mais de um tipo de óculos, verificamos que o modelo treinado era impreciso e apontava falsos positivos. Para solucionar escolhemos somente o modelo da figura

Figura 57 – Distribuição de imagens no Dataset.



Fonte: Autores

59 e realizamos o treinamento dessa forma, mantendo os critérios anteriores. Esse modelo, conforme figura abaixo, foi escolhido por ser um dos mais utilizados.

Outro ponto importante foi adicionar imagens com diversos ângulos de rostos, iluminação e características físicas das pessoas, como na imagem 58.

Figura 58 – Imagens com diversos ângulos de rosto.

Fonte : Autores

Figura 59 – Modelo de óculos de proteção escolhido.

Fonte: FICHA. . . , 2020

3.1.2 Treinar modelo com Keras/Tensorflow

a) Data Augmentation

- Abrange uma ampla gama de técnicas usadas para gerar novas amostras de treinamento a partir das originais, aplicando instabilidades e perturbações aleatórias, mas ao mesmo tempo, garantindo que os rótulos de classe dos dados não sejam alterados.
- O objetivo ao aplicar data augmentation é aumentar a generalização no modelo.
- Como a rede está constantemente vendo novas versões ligeiramente modificadas dos dados de entrada, a rede é capaz de aprender recursos mais robustos.
- No momento do teste, não aplicamos o data augmentation e simplesmente avaliamos nossa rede treinada nos dados de teste não modificados, na maioria dos casos, ocorrera um aumento na precisão do teste, talvez às custas de uma ligeira queda na precisão do treinamento.
- Por exemplo, podemos aplicar transformações geométricas simples e aleatórias, como:
    i. Translações

ii. Rotações

iii. Mudanças de escala

iv. Corte

v. Flip horizontal (e em alguns casos, vertical)

b) Carregar o MobileNetV2 classifier (vamos ajustar este modelo com pesos pré-treinados do ImageNet)

- Carregar MobileNetV2 com pesos ImageNet pré-treinados, deixando de fora a camada head de rede;
- Construir uma nova camada head e anexar à base no lugar da antiga;
- Congelar as camadas de base da rede. Os pesos dessas camadas de base não serão atualizados durante o processo de retro propagação, enquanto os pesos da camada de cabeça serão ajustados.

c) Pré-processando da imagem

- As etapas de pré-processamento incluem redimensionamento para 224 × 224 pixels, conversão para formato de matriz e dimensionamento das intensidades de pixel na imagem de entrada para o intervalo [-1, 1] (através da função `preprocess_input`

d) Épocas, hiperparâmetros, e outros detalhes do treino.

- Uma época (epoch) é um termo usado em aprendizado de máquina e indica o número de passagens de todo o conjunto de dados de treinamento que o algoritmo de aprendizado de máquina concluiu. Os conjuntos de dados geralmente são agrupados em lotes (batches), especialmente quando a quantidade de dados é muito grande;
- Ao treinar uma rede neural, a taxa de aprendizado (initial learning rate) costuma ser o hiperparâmetro mais importante a ser ajustado;
- Uma taxa de aprendizado muito pequena e sua rede neural pode não aprender nada;
- Uma taxa de aprendizagem muito grande e você pode ultrapassar as áreas de baixa perda (ou até mesmo overfit desde o início do treinamento);
- Nos nossos treinamentos verificamos que valores de taxa de aprendizado inicial `(INIT_LR)` entre $10^(-3)$ e $10^(-4)$ apresentaram melhores resultados,

optamos então por testar o valor de $INIT_LR = 5.5e-4$ e observamos ser o mais preciso nos treinos;

– Vejamos abaixo os testes com diferentes `INIT_LR` para treinar os modelos de detecção dos óculos de proteção.

Figura 60 – Resultados com taxa de aprendizagem de 1e-2.

Fonte: Autores

Podemos observar, na figura 60 que com uma taxa de aprendizado de $1e^{-2}$ temos, na primeira época baixa precisão de treino e validação e alto erro, e ao decorrer do treinamento, a precisão tem dificuldade de tender a 1, e o erro dificuldade de tender a zero, principalmente o de validação.

Com uma taxa de aprendizado de 1e-3, como na figura 61 vemos uma melhora, nas primeiras épocas a precisão sobe com maior velocidade, principalmente a precisão do treino e o erro também cai mais rápido, porém o de validação fica estagnado num patamar de aproximadamente 0.3.

Vejamos uma taxa de aprendizado de 1e-4, assim como na figura 62 também com um bom resultado, a precisão e erro tem comportamento mais linear, em comparação aos outros valores utilizados, e alcançam bons valores finais.

Com uma taxa de aprendizado de $1e^{-5}$, como mostrado na figura 63 vemos uma piora dos resultados, temos baixa precisão e alto erro, tanto de treino quanto validação. Precisão na casa de 0.8 a 0.9 e erro de 0.3 a 0.4.

Figura 61 – Resultados com taxa de aprendizagem de 1e-3.

Fonte: Autores

Figura 62 – Resultados com taxa de aprendizagem de 1e-4.

Fonte : Autores

Podemos observar que com uma taxa de aprendizado de $5.5e^{-4}$, no gráfico da figura 64, temos um resultado com características próximas a taxas de $1e^{-3}$ e $1e^{-4}$, como esperado, produzindo excelente precisão de treino, tendendo a 1, precisão de validação próxima a 0.9. Verificando os valores de erro temos um dos menores erros de treino, próximo a 0.1 e erro de validação próximo a 0.2.

Figura 63 – Resultados com taxa de aprendizagem de 1e-5.

Fonte: Autores

Figura 64 – Resultados com taxa de aprendizagem de 5.5e-4.

Fonte: Autores

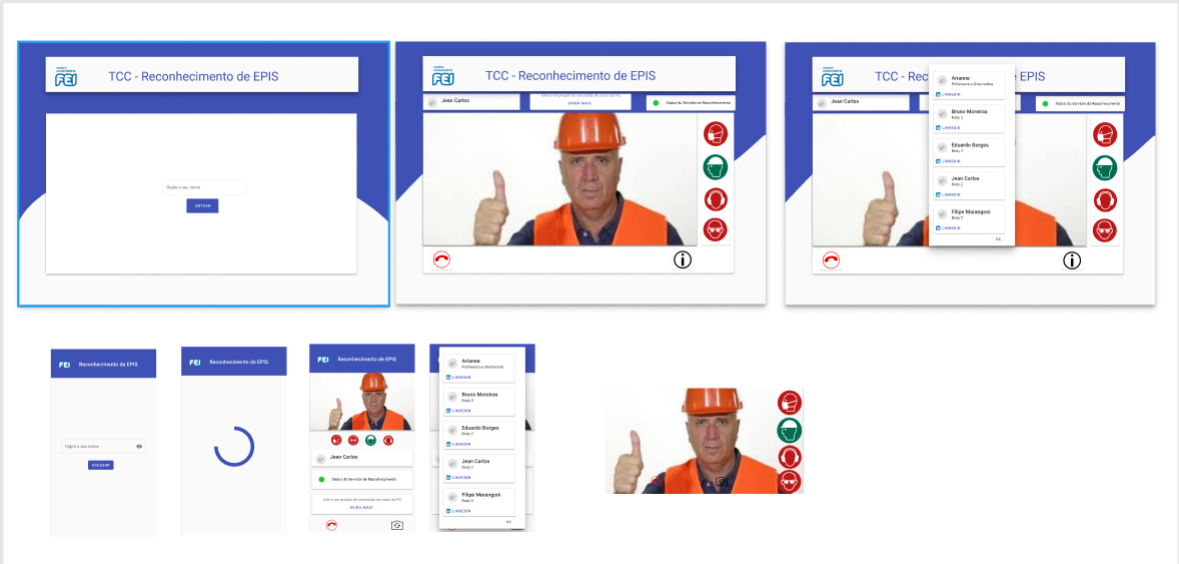
Verificando os resultados obtidos optamos por utilizar essa taxa, de $5.5e^{-4}$, como padrão para os nossos treinamentos.

3.2 APLICAÇÃO REACT

Inicialmente realizamos a prototipação da aplicação com através do FIGMA quer permite criar layouts de referência para posteriormente serem criados em javascript .

O primeiro protótipo da aplicação do usuário, que chamamos de Viewer, está representada na figura 65. O usuário inicialmente deve digitar seu nome e em seguida clicar em entrar para acessar o sistema de reconhecimento, será possível acessar a página através do computador ou de dispositivos móveis pois a página é responsiva, se adaptando ao tamanho da tela do dispositivo .

Figura 65 – Protótipo Figma - Viewer.



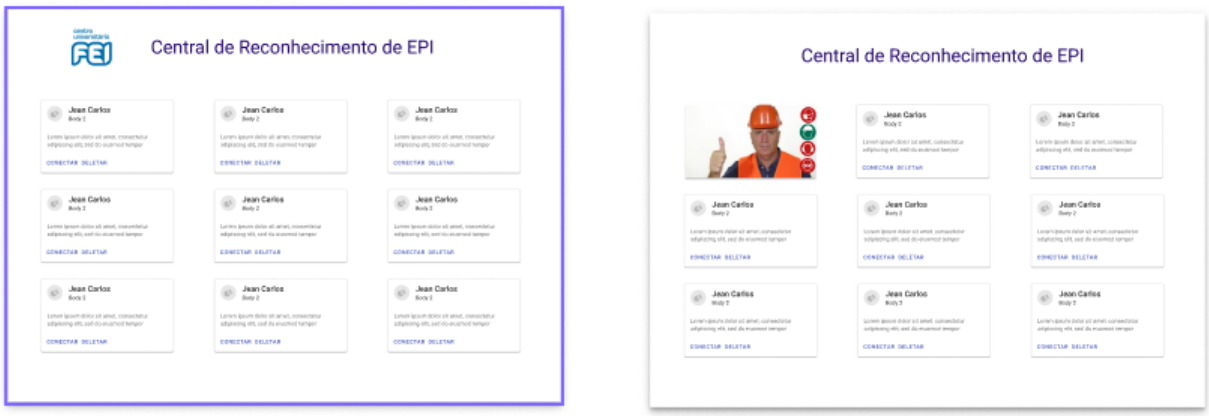
Fonte: Autores e adaptado de Yao et al., 2011

Após o usuário acessar o sistema, sua imagem é enviada para a aplicação Master, da figura 66 que será utilizada pelos administradores do sistema, que poderão visualizar os usuários e gerenciar funções do servidor. Inicialmente iríamos utilizar uma seleção de usuários no servidor, onde o administrador do sistema autorizaria os usuários, mas decidimos remover essa etapa e todos os "streams"de vídeo são enviados automaticamente para o Master .

Então o Master processa as informações e responde com as coordenadas do rosto do usuário e o tipo de EPI que está sendo utilizado.

Em seguida, dividimos a aplicação em componentes e inciamos a implementação desses componentes de forma isolada. A estrutura de pastas e os componentes criados estão representados na figura 67.

Figura 66 – Protótipo Figma - Master.



Fonte: Autores e adaptado de Yao et al., 2011

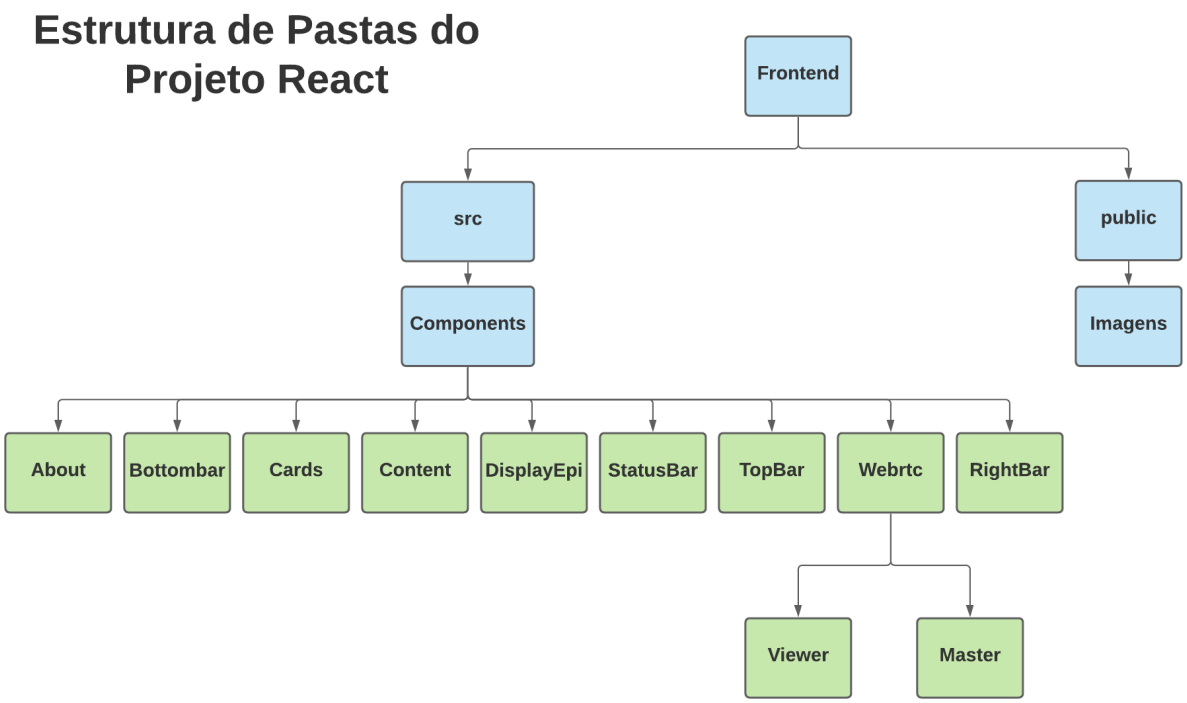
O projeto possuí a pasta "src"onde dentro da mesma esta a pasta "Components"que contém todos os componentes React. Implementados os seguintes componentes:

a) About - Componente que exibe informações sobre o grupo e o projeto;
b) Bottombar - Barra inferior com botões para desligar aplicação;
c) Cards - Cartões que contém dados do grupo;
d) Content - Componente que exibe o vídeo da webcam do usuário;
e) DisplayEPI - Componente na barra da lateral direita que exibe EPIs utilizados, ficando verde ou vermelho;
f) StatusBar - Barra de status que exibe o nome do usuário que acessa o sistema. Inicialmente também exibiria o status do servidor. Mas, a ideia foi abandonada pois não agregaria muito ao projeto e não era possível realizar de forma precisa;
g) TopBar - Barra superior que exibe o titulo da página;
h) WebRTC - Implementa o protocolo WebRTC e possui dois subcomponentes que são o Viewer (Usuario que envia os vídeos) e Master (Servidor que recebe vídeo e processa dados). O código completo do Master está disponível do apêndice F e seus estilos no apêndice G.
i) RightBar - Barra lateral direita.

A componentização é uma característica importante de projetos React, pois permite a reutilização de código dentro do projeto. Dentro de cada pasta de componente existem dois arquivos jsx. Um que define toda a lógica e outro que é utilizado para a definição de estilos e aparência da página.

Além disso, os arquivos na pasta "public"são arquivos que poderão ser visualizados diretamente pelo usuário como imagens e vetores (SVG).

Figura 67 – Estrutura projeto React.



Fonte : Autores

Após desenvolvimento local da aplicação, a mesma foi publicada no serviço AWS Amplify da Amazon. Na versão publicada, algumas modificações foram realizadas com relação ao protótipo do FIGMA, o Layout foi levemente alterado, foi adicionado uma seção com informações sobre o projeto e foram removidas algumas informações desnecessárias como status de servidor e seleção de usuários do lado do master.

4 RESULTADOS

4.1 TREINAMENTO

Durante o desenvolvimento do projeto era esperado que os EPIs capacete e máscara tivessem um treinamento mais fácil e melhor precisão, considerando a facilidade prática do próprio ser humano em identificar esses objetos. Já com o abafador e, principalmente os óculos de proteção, era esperado mais dificuldade e menor precisão. Veremos que os resultados refletem essa projeção inicial.

O código deste treinamento está disponível no no apêndice C.

Abaixo temos o log do treinamento do capacete em que obtivemos excelente resultado.

```
Epoch 1/50
70/70 - 12s 173ms/step - loss: 0.1728 - accuracy: 0.9317 - val_loss: 0.0285
    - val_accuracy: 0.9929
Epoch 2/50
70/70 - 10s 142ms/step - loss: 0.0604 - accuracy: 0.9811 - val_loss: 0.0158
    - val_accuracy: 0.9965
Epoch 3/50
70/70 - 10s 142ms/step - loss: 0.0712 - accuracy: 0.9784 - val_loss: 0.0200
    - val_accuracy: 0.9929
.
.
.
Epoch 48/50
70/70 - 10s 147ms/step - loss: 0.0170 - accuracy: 0.9946 - val_loss: 0.0176
    - val_accuracy: 0.9894
Epoch 49/50
70/70 - 10s 148ms/step - loss: 0.0094 - accuracy: 0.9964 - val_loss: 0.0118
    - val_accuracy: 0.9965
Epoch 50/50
70/70 - 10s 148ms/step - loss: 0.0028 - accuracy: 0.9991 - val_loss: 0.0150
    - val_accuracy: 0.9929
[INFO] evaluating network...
                precision    recall  f1-score   support

        helmet       0.99      0.99      0.99       141
without_helmet       0.99      0.99      0.99       141
```

```
    accuracy                           0.99       282
   macro avg       0.99      0.99      0.99       282
weighted avg       0.99      0.99      0.99       282
[INFO] saving helmet detector model...
```

Durante o treinamento, foram usadas 50 épocas e o código foi dividido 80% do dataset para treino e 20% para teste, com tamanho de 705 imagens, logo:

$705 * 0.8 = 564$ (imagens para treino)

Foi utilizado batch size (BS) de 16,

$\frac{564}{16} = 35(lotes)$

Isso significa que o conjunto de dados será dividido em 35 lotes, cada um com dezesseis amostras. Os pesos do modelo serão atualizados após cada lote de 16 amostras. Isso também significa que uma época envolverá 35 lotes (atualizações no modelo). No nosso caso serão 35 lotes que serão analisadas 2 vezes, uma para treinar e outra para validar, por isso o valor de "70/70".

Com 50 épocas, o modelo será exposto ou passará por todo o conjunto de dados 50 vezes. Isso é um total de 1.750 lotes durante todo o processo de treinamento.

Após finalizado o treinamento, temos o modelo salvo. Também temos como saída o gráfico da figura 68 com o resultado do treino:

Verificamos na figura 68 que os valores de acurácia "sobem", porém não chegam próximos a 1 até pelo menos a época 40, depois se aproximam um pouco mais. Os valores de perda "caem", mas demoram também a tender a zero, principalmente na validação da perda, que fica estagnado em torno de 0.12.

Também podemos analisar os seguintes dados:

```
              precision    recall  f1-score   support
        glass       0.96      0.97      0.96       120
without_glass       0.97      0.96      0.96       120
```

Precisão, recall e f1-score, com valores entre 0.96 e 0.97, bons resultado, vamos analisar melhor com testes práticos.

Figura 68 – Treinamento com configurações finais.

Fonte: Autores

4.1.1 Testes práticos

Em seguida realizamos os testes práticos, podemos observar na figura 69, que em imagens onde a pessoa usa capacete, há o reconhecimento sendo destacado pelo retângulo verde. Já onde não há detecção há o retângulo vermelho.

Figura 69 – Exemplos de detecção de capacete.

Fonte: Adaptado de Productions, s.d.

Na figura 70 temos duas pessoas sem óculos de proteção, apesar de um deles estar com óculos de leitura, a identificação é precisa em não aprovar como óculos de proteção. Enquanto na figura 71 temos uma pessoa com e outra sem óculos.

Figura 70 – Homem com óculos comum.

Fonte : Adaptado de Alan, s.d.

Figura 71 – Exemplos de detecção com modelo de óculos.

Fonte: Adaptado de photography33, 2012

Na figura 72 observamos que a detecção de pessoas com e sem máscara também ocorre de forma adequada. Enquanto, na figura 73 observamos a precisão do modelo com pessoas utilizando e não utilizando abafadores.

Figura 72 – Exemplos de detecção de Máscara.

Fonte: Adaptado de Productions, s.d.

Figura 73 – Exemplos de detecção de abafador de ouvido.

Fonte: Adaptado de photography33, 2012

4.1.2 Dificuldades durante os treinos

Ao realizar os treinamentos iniciais verificamos alguns pontos que estavam causando resultados imprecisos, destacamos:

a) Composição do dataset:

- O dataset precisa ser equilibrado em quantidade, para não enviesar o modelo;
- É necessário diversificação nas imagens, quanto as características físicas das pessoas, para produzir resultados precisos em qualquer fisionomia;
- Diversidade em relação ao ângulo em que os rostos das pessoas aparecem nas fotos, por exemplo, quando tínhamos somente faces de lado nas imagens positivas e não nas negativas, mesmo sem o uso do EPI, o modelo acusava um falso positivo, quando analisava um rosto de lado.

b) Otimização dos hiperparâmetros:

- Fizemos testes com algumas composições diferentes de initial learning rate (INITLR), número de épocas (epochs) e tamanho de lote (batch size – BS);
- Verificamos que a composição usada, INITLT = $5.5 * 10^{-4}$, EPOCHS = 50 e BS = 16, produziu ótimos resultados.

4.2 APLICAÇÃO WEB

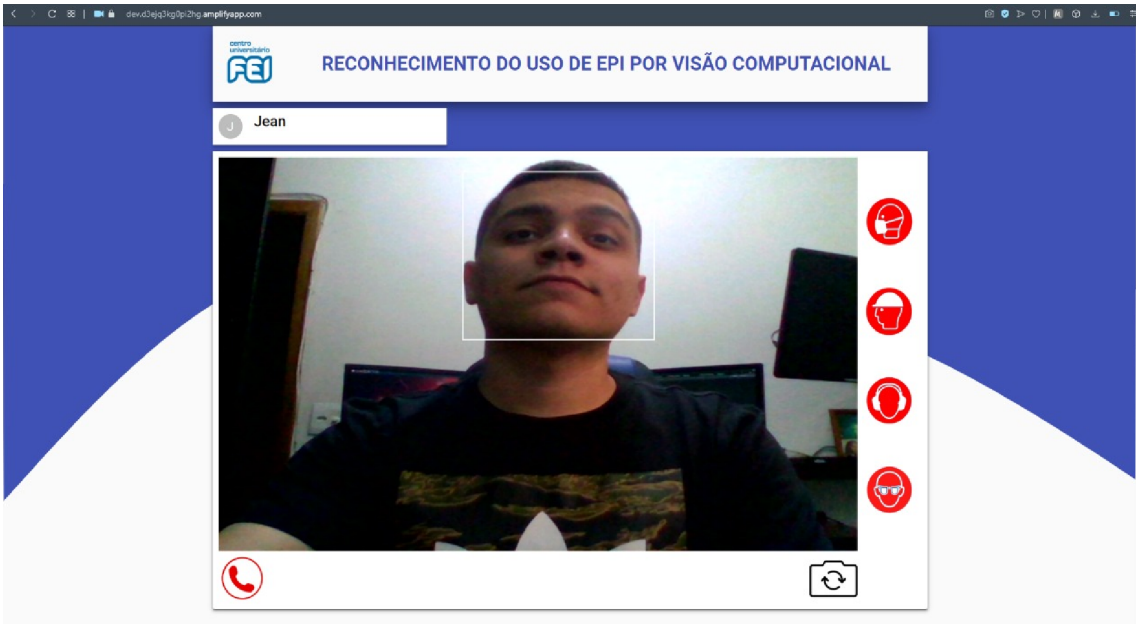
Por fim, desenvolvemos a aplicação e web e tivemos sucesso em realizar a comunicação entre os protocolos. A página ficou conforme a figura 74, onde inicialmente o usuário deve digitar o seu nome e em seguida conseguirá acessar o sistema de reconhecimento.

Figura 74 – Pagina inicial - Desktop.

Fonte: Autores

Ao digitar seu nome, o usuário será encaminhado para o reconhecimento de EPI conforme figura 75. Onde os resultados do servidor de reconhecimento recebidos serão renderizados na forma de um retângulo e o EPI correspondente ficará na cor verde.

Figura 75 – Seção de reconhecimento.



Fonte: Autores

Na parte inferior da página, conforme figura 75, adicionamos uma seção com informações sobre o projeto e os integrantes do grupo. A página está hospedada e pode ser acessada na url "https://dev.d3ejq3kg0pi2hg.amplifyapp.com/".

4.2.1 WebRTC e servidor de sinalização

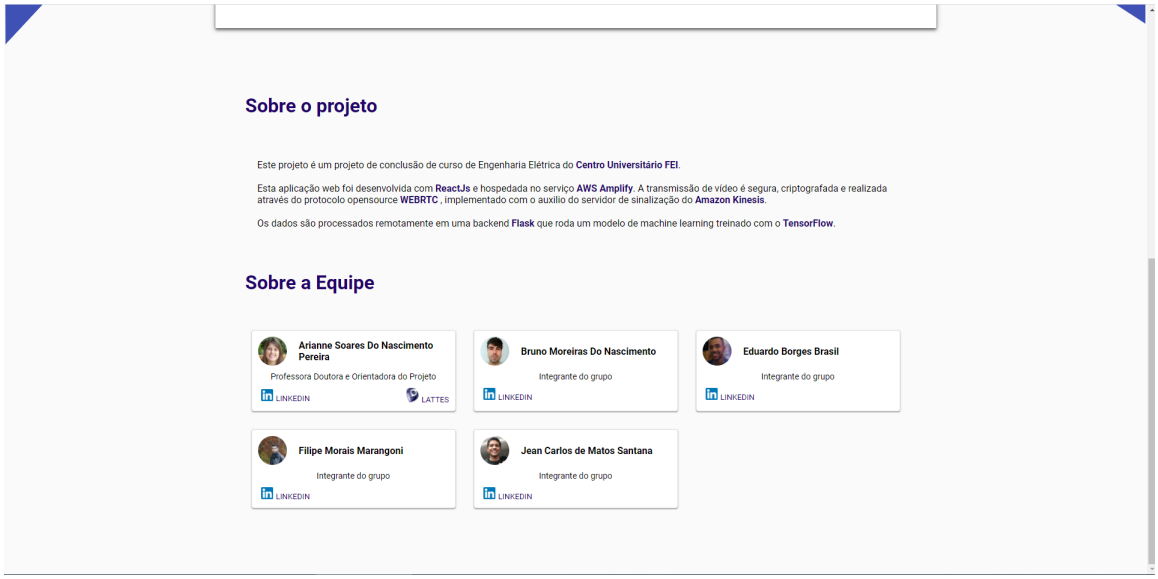
Em seguida implementamos o protocolo WebRTC, os logs das transações realizadas pelo Viewer e Master estão representados nas figuras 77 e 78 .

No geral eles são compostos pelas seguintes etapas:

1 - Conexão com servidor de sinalização - Inicialmente o Viewer e Master relizam a conexão com o servidor de sinalização e requisitam os endpoints dos servidores ICE (TURN e STUN).

2 - Ambos os peers geram suas ofertas SDP e realizam as trocas de ofertas. Ao mesmo tempo que geram os candidatos ICE, que são conjuntos de portas e IP para que a conexão direta seja criada. Até essa etapa todas as trocas de dados são realizados através do servidor de sinalização. É possível observar que a geração de candidatos ICE e envio do SDP não são

Figura 76 – Pagina inicial. Informações Gerais. - Desktop.



Fonte: Autores

síncronas, ou seja, são enviadas em momentos diferentes, característica da utilização do Trickle ICE.

3 - A troca de candidatos ICE é realizada e em seguida a conexão direta entre os peers é executada e o servidor de sinalização não é mais necessário.

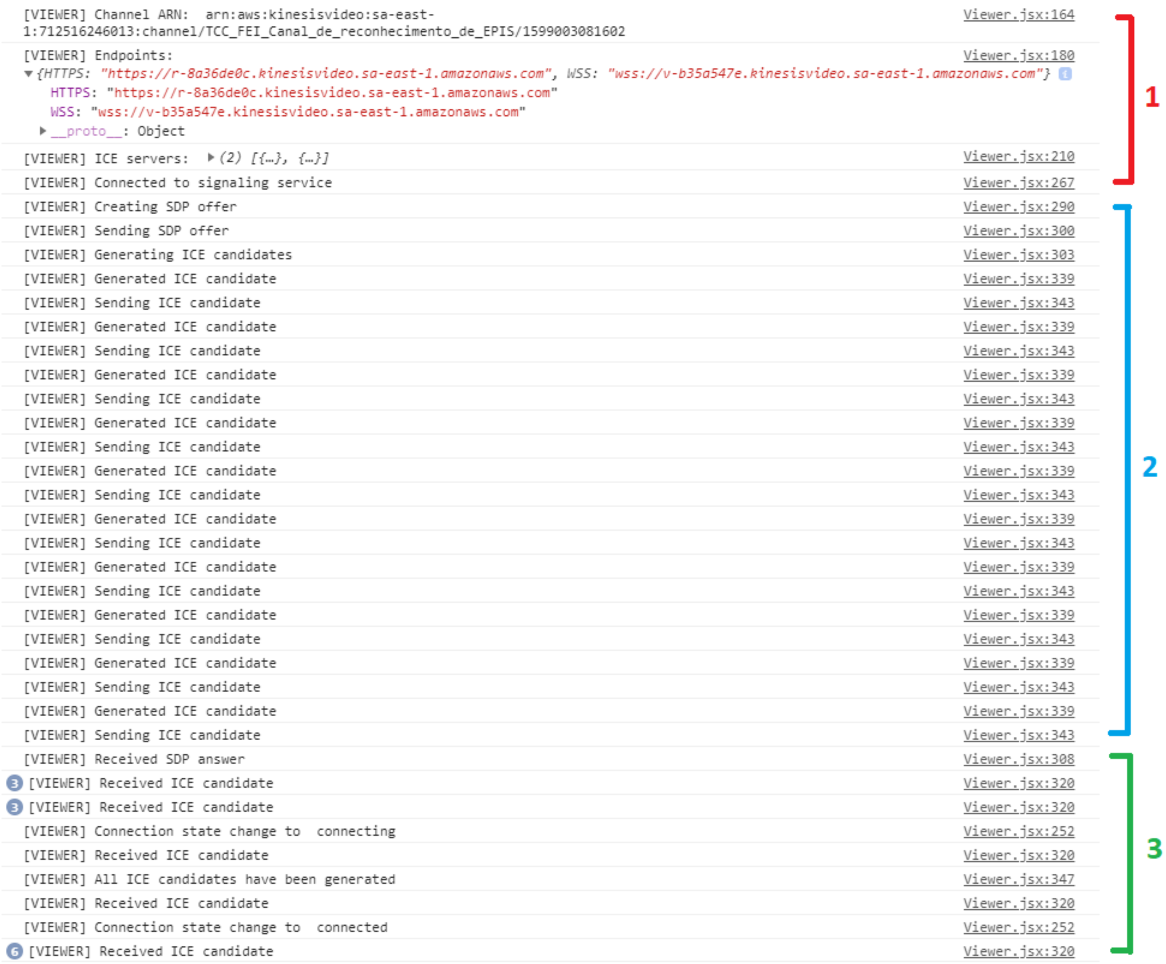
4 - É aberto o canal de dados, que será utilizado pelo Master para comunicar o Viewer sobre EPIs e posições detectadas.

Na figura 78 é necessário destacar o fato de um valor ser utilizado pelo Master para se referir ao Viewer (client). Este valor é o "clientID", ele é gerado aleatoriamente sempre que um Viewer se conecta ao servidor de sinalização e serve para diferenciar caso existam muitos usuários conectados ao mesmo tempo, de forma que o servidor saiba de quem está recebendo os dados e possa responder adequadamente.

Através das figuras 77 e 78 podemos observar que é necessário enviar muitas mensagens através do servidor de sinalização antes que a conexão seja estabelecida. E mesmo assim, no mês de Outubro com quase 15 mil mensagens trocadas o custo do servidor é o representado na figura 79. É cobrado um custo fixo de $USD0{,}13$ mais um custo de $USD6{,}2$ por cada milhão de mensagens de sinalização resultando no mês de Outubro em um custo de $USD0{,}17$.

Desta forma, a solução apresentada possuí custo baixíssimo pois é uma aplicação "Serverless", onde o servidor não executa o tempo todo, apenas é iniciado quando existe alguma

Figura 77 – Log Viewer.



Fonte: Autores

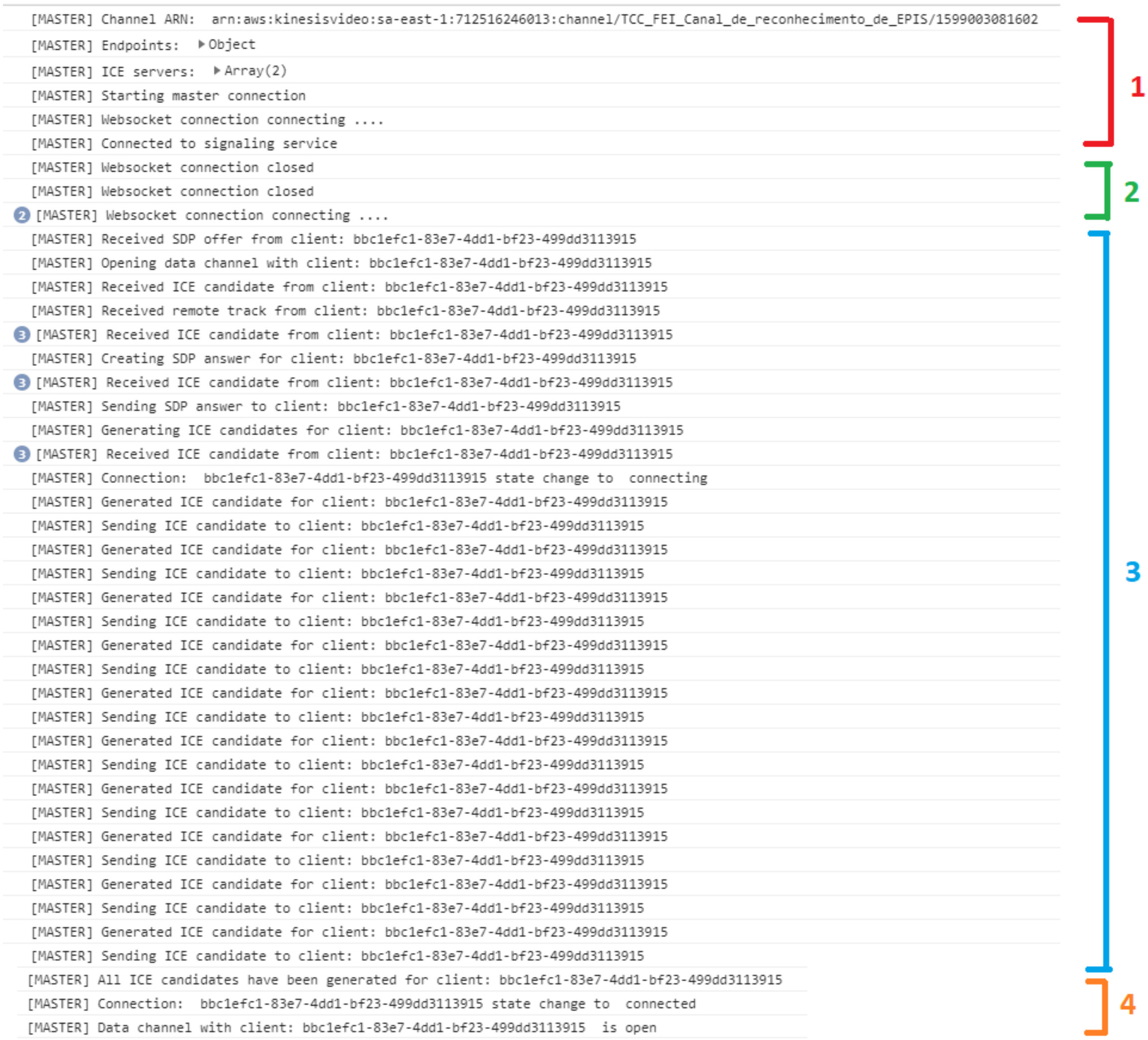
solicitação do usuário e uma vez que a conexão peer to peer é estabelecida o servidor não é mais utilizado.

Quando a conexão peer to peer é estabelecida, a troca de dados acontece conforme o diagrama da figura 80, onde o Viewer envia a imagem da Webcam de forma criptografa para o Master através do protocolo WebRTC.

Então, como é possível observar na figura 81, o Master codifica os frames da imagem em base64, que é um formato padrão para tráfego de arquivos binários em forma de texto na web, e envia junto com o clientID para o servidor backend, utilizando o protocolo WebSocket e que possui o modelo de Machine Learning.

O arquivo no formato json possui o clientID que identifica o Viewer, as coordenadas do rosto humano detectado e os epis utilizados. No exemplo da figura 82, a imagem recebida pelo servidor possuía uma pessoa utilizando uma máscara e um abafador.

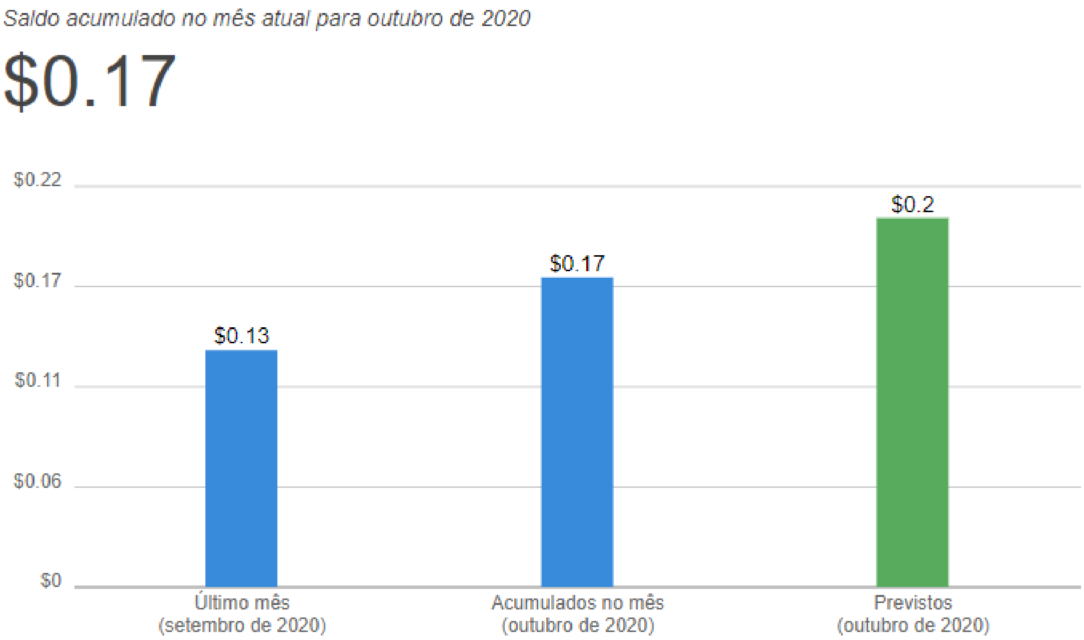
Figura 78 – Log Master.



Fonte: Autores

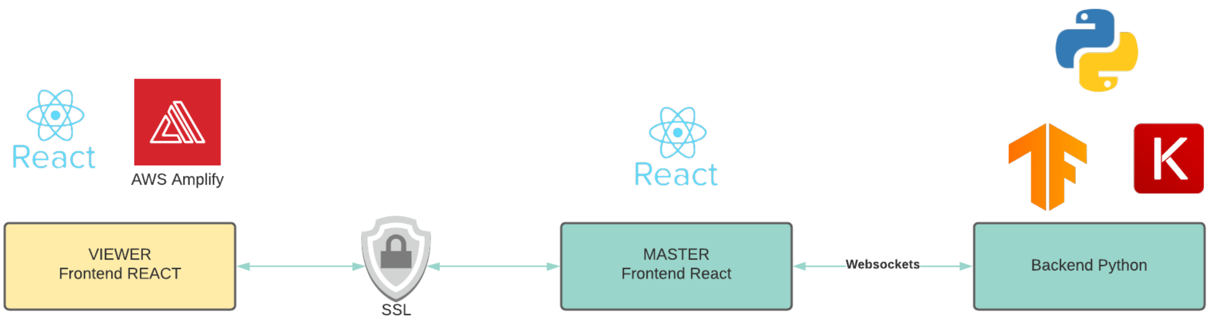
Em seguida, o Master que recebeu a resposta da backend, utiliza o clientID para enviar os dados para Viewer correto, através do Datachannel do protocolo WebRTC, que então exibe os resultados para o usuário.

Figura 79 – Custo do Mês de Outubro Serviços AWS.



Fonte : AWS Billing

Figura 80 – Diagrama de funcionamento da aplicação.



Fonte: Autores

Figura 81 – Dados enviados da aplicação Master para o Backend. A imagem está codificada em base64.


[
{
"clientId":"efbb85d7-5cd1-4503-8ffd-01284f799e51",

"image":"
/2wBDAAMCAgICAgMCAgIDAwMDBAYEBAQEBAgGBgUGCQgKCgkICQkKDA8MCgsOCwkJDRENDg8QEBEQCgw
SExIQEw8QEBD/2wBDAQMDAwQDBAgEBAgQCwkLEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBA
QEBAQEBAQEBAQEBAQEBAQEBD/wAARCAC+AYYDASIAAhEBAxEB
/8QAHgAAAQQDAQEBAAAAAAAAAAAAAAQFBgcCAwgBCQr
/xABYEAABAwMCAwUFBAUFCggPAAABAgMEAAURBiEHEjETIkFRYQgUMnGRFUKBoRYjUrHBFzNigpIJJEN
mcqLC0eHiNZWlsrPS4/AYJScRldjZGV1lK01w9P/xAAbAQABBQEBAAAAAAAAAAAAAAAQIDBAUGB
//EAEQRAAEDAgQCBQkFBAkFAAAAAAEAAgMEEQUSITETQQYYUIlHRBxVSYXGRobGyFjKBotJCs9PhIyUzV
GKCo8HwQ1Nyc4P/2gAMAwEAAhEDEQA

Fonte: Autores

Figura 82 – Resposta da BackEnd para o Master.


```
{
  "clientId": "efbb85d7-5cd1-4503-8ffd-01284f799e51",
  "bbox": [
    {
      "startX": 150,
      "endX": 238,
      "startY": 94,
      "endY": 187,
      "label": {
        "hasHelmet": 0,
        "hasMask": 1,
        "hasEarmuff": 1,
        "hasGlass": 0,
        "label": ""
      }
    }
  ]
}
```


Fonte: Autores

4.2.2 Estimativas de custo e operação

A solução que desenvolvemos poderia ser aplicada na prática de diversas formas, dependendo, principalmente, da demanda necessária para a execução de determinada operação.

Podemos exemplificar com 2 cenários possíveis.

Primeiro cenário: Uma obra civil de pequeno porte, onde há um ou dois pontos de entrada a áreas cujo uso de EPI seja obrigatório.

Nesse caso podemos utilizar um hardware dedicado para processar a rede neural, como comentado na seção 2.6. Com isso teríamos que otimizar os modelos utilizando o Tensor Flow Lite e poderíamos utilizar, por exemplo, o Raspberry Pi para executar as rotinas. Assim teríamos de custo somente a aquisição do hardware e câmera, sem que seja necessária uma infraestrutura de servidor para conexão com a aplicação WEB.

Segundo cenário: Uma empresa de grande porte que possui diversos laboratórios de acesso controlado e uso obrigatório de EPI.

Assim sendo, nossa solução seria baseada fortemente na aplicação WEB. Quanto aos custos, teríamos que adquirir hardware (de processamento menor que o Raspberry Pi, já que o processamento não ocorrera no hardware, reduzindo custos) e câmeras para transmitir a informação para um servidor central (servidor que poderá estar na empresa ou em qualquer outro lugar, com alto poder de processamento), que realizará a rede neural das imagens capturadas sempre que necessário. Dessa forma temos economia na aquisição dos hardwares e custo baixo na utilização do servidor, como comentado na seção anterior.

5 CONCLUSÕES

Como visto no desenvolvimento do presente projeto, o principal fator de motivação é a busca por uma solução otimizada que proporciona uma análise precisa do uso de equipamentos de proteção individual por trabalhadores que tenham que acessar áreas onde seu uso é obrigatório, para garantir, acima de tudo, sua segurança e integridade física.

Realizamos diversos estudos, analisando fundamentos teóricos, softwares e hardwares que poderíamos utilizar para definir os melhores procedimentos para alcançar nossos objetivos.

Desta forma, decidimos utilizar o detector de faces caffemodel, modelo com estrutura Single Shot Detector (SSD) usando arquitetura semelhante a ResNet-10 como backbone, com objetivo de obter o rosto da pessoa analisada (região de interesse) para que nosso modelo realizasse a correta identificação do EPI. Além disso, escolhemos o framework Keras pela sua facilidade de implementação e uma arquitetura convolucional chamada MobileNetV2.

O momento atual em que elaboramos este projeto, onde estamos atravessando a pandemia de COVID-19, também nos trouxe oportunidades de melhoria e a possibilidade de implementação de outras tecnologias ao nosso projeto, tendo em vista que não haveria a apresentação presencial deste trabalho. Com isso realizamos estudos de aplicações WEB para permitir que qualquer dispositivo e pessoa possa executar a nossa solução, sendo necessário somente estabelecer conexão com o servidor que realizará o processamento.

Quanto aos custos do projeto, não tivemos gastos com materiais, tendo em vista a solução web adotada, no entanto, se tivéssemos a implementação física, utilizaríamos o Raspberry Pi 4 e a câmera, além de eventuais gastos com a montagem de uma maquete, e custo de $USD0{,}13$ no mês e $USD0{,}17$ no mês de outubro com o servidor de sinalização. Já pensando no uso em ambientes fabris ou construção civil, por exemplo, sabe-se que o custo de EPIs representa baixo percentual em relação ao custo global de uma obra, portanto, a inclusão de câmeras e um sistema inteligente, que garante a segurança dos trabalhadores e do empregador, não teria grande impacto nos custos diretos e poderia até reduzir os custos indiretos, como disputas judiciais.

5.1 PROPOSTAS DE CONTINUIDADE DO PROJETO

Nosso projeto apresentou bons resultados e proporcionou a integração de diversas tecnologias e diferentes técnicas para alcançar nossos objetivos. As contribuições obtidas neste trabalho podem ser exploradas em pesquisas futuras, como exemplo:

a) Implementação de reconhecimento de EPIs no corpo todo, como coletes, luvas e botas;
b) Interligação de hardwares simples, como Raspberry Pi, para o envio da imagem ao vivo ao servidor via rede, permitindo a modularização do projeto por completo e instalação em locais com extensões diversas;
c) Criação de interface para seleção de quais EPIs pré-treinados o usuário final gostaria de utilizar, facilitando a independência da solução;
d) Aplicação de reconhecimento facial como mais uma medida de segurança na liberação de acesso, assim como a utilização dos EPIs;
e) Aferição de temperatura corporal, visando evitar liberar acesso de pessoas possivelmente doentes e, com isso, impedir a disseminação de doenças.

APÊNDICE A – CÓDIGO COMPLETO MOBILENETV2

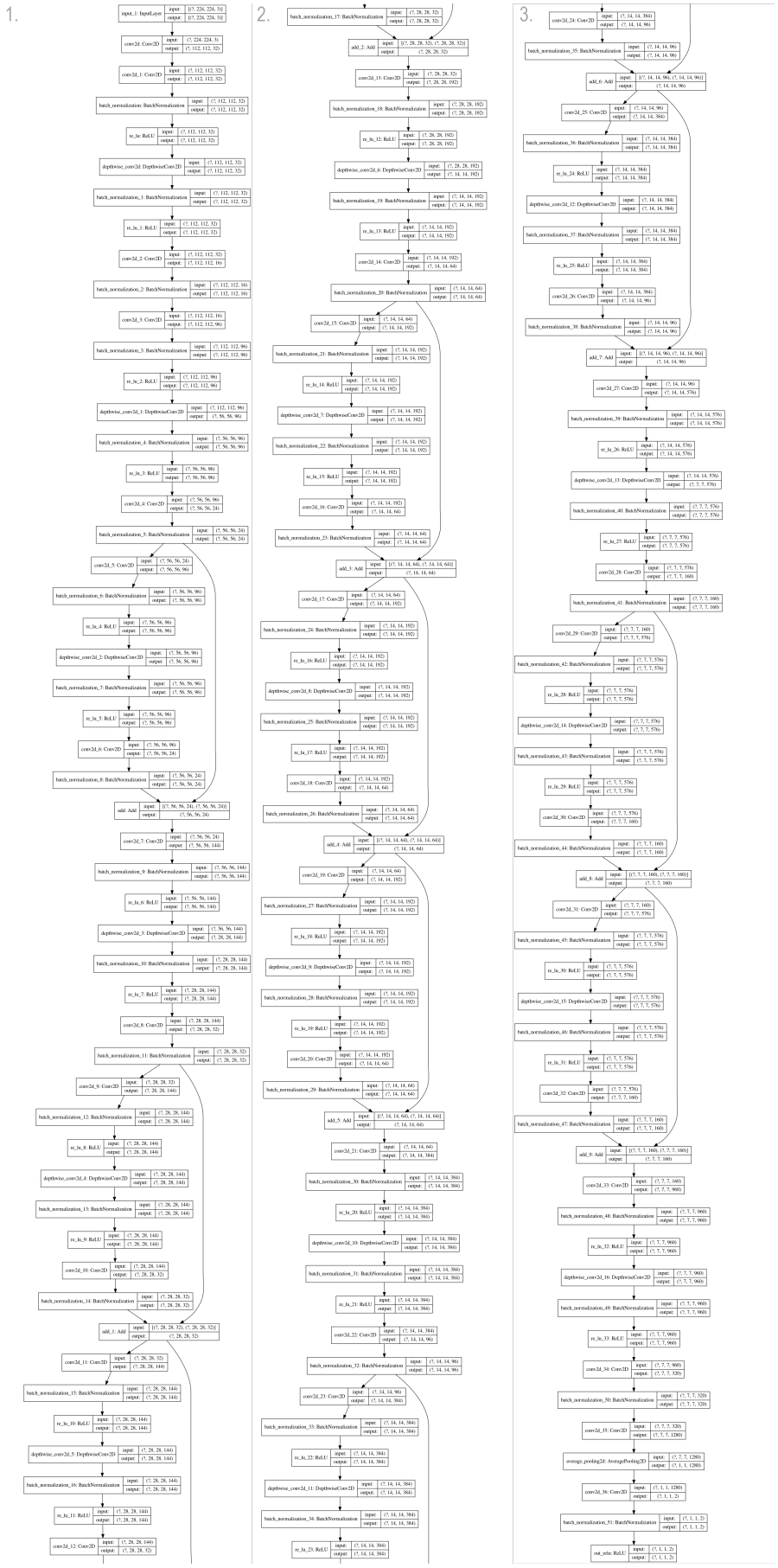
```
from tensorflow.python.keras import layers
from tensorflow.keras.utils import plot_model
from tensorflow.python.keras.applications import imagenet_utils
from tensorflow.keras.layers import Layer, Conv2D, DepthwiseConv2D,
    BatchNormalization, ReLU, Add, add, Concatenate
from tensorflow.nn import relu6
from tensorflow.keras import Sequential, Input, layers

import tensorflow.keras as keras

def MobileNetV2(input_shape = (224,224,3)):
    """
        Implementação do body modelo Mobile Net utilizando o Keras
    """
    inputImg = Input(shape=input_shape)
    model = layers.Conv2D(filters=32, kernel_size=2, strides=2)(inputImg)

    def bottleneck(model, input_shape, output_shape, filters, strides,
        expansion_factor=6):
        """
            Implementação layers intermediarias Bottlenecks.
        """
        input_channels = int(input_shape[2])
        InputModel = model

        def expansion(model):
            model = layers.Conv2D(filters=int(input_channels*
                expansion_factor), kernel_size=1, use_bias=False)(model)
            model = layers.BatchNormalization()(model)
            model = layers.ReLU(6.)(model)
            return model

        def filtering(model):
            model = layers.DepthwiseConv2D(kernel_size=3, strides=strides,
                padding='same', use_bias=False)(model)
            model = layers.BatchNormalization()(model)
            model = layers.ReLU(6.)(model)
            return model
```

```
35
        def contraction(model):
37          model = layers.Conv2D(filters=filters, kernel_size=1, use_bias=
                False)(model)
            model = layers.BatchNormalization()(model)
39          return model

41      model = expansion(model)
        model = filtering(model)
43      model = contraction(model)

45      try:
            if strides==1 and InputModel.shape[1:] == model.shape[1:]:
47              model = add([model, InputModel])
        except:
49          pass

51      return model

53  #declaração de camadas MobileNet Conforme Especificação
    bottleneckLayers = [
55      {"inputShape": (112, 112, 32), "expansionFactor":1, "outputChannels
            ":16,"repeats":1,"stride":1},
        {"inputShape": (112, 112, 16), "expansionFactor":6, "outputChannels
            ":24,"repeats":2,"stride":2},
57      {"inputShape": (56, 56, 24), "expansionFactor":6, "outputChannels"
            :32,"repeats":3,"stride":2},
        {"inputShape": (28, 28, 32), "expansionFactor":6, "outputChannels"
            :64,"repeats":4,"stride":2},
59      {"inputShape": (14, 14, 64), "expansionFactor":6, "outputChannels"
            :96,"repeats":3,"stride":1},
        {"inputShape": (14, 14, 96), "expansionFactor":6, "outputChannels"
            :160,"repeats":3,"stride":2},
61      {"inputShape": (7, 7, 160), "expansionFactor":6, "outputChannels"
            :320,"repeats":1,"stride":1},
    ]
63

    output_shape = (112,112,32)
65
```

```
    for layer in bottleneckLayers:
67          stride = layer['stride']
            for repeat in range(layer["repeats"]):
69              model = bottleneck(model, layer["inputShape"], output_shape,
                    layer['outputChannels'], stride, layer['expansionFactor'])
                if stride != 1:
71                  stride = stride - 1

73          output_shape = layer['outputChannels']

75      model = layers.Conv2D(filters=1280, kernel_size=1, use_bias=False,
            strides=1)(model)
        model = layers.AvgPool2D(pool_size=7)(model)
77      model = layers.Conv2D(kernel_size=1, filters=2)(model)
        model = layers.BatchNormalization()(model)
79      model= layers.ReLU(6., name='out_relu')(model)
        model = keras.Model(inputs=inputImg, outputs=model, name="MobileNetV2")
81
        return model
83
model = MobileNetV2()
85 model.summary()

87 plot_model(
        model,
89      to_file="model.png",
        show_shapes=True,
91      show_layer_names=True,
        rankdir="TB",
93      expand_nested=False,
        dpi=96,
95 )
```

APÊNDICE B – ARQUITETURA COMPLETA IMPLEMENTADA MOBILENETV2

Figura 83 – Arquitetura MobileNetV2



Fonte: Autores

APÊNDICE C – TREINAMENTO DO CAPACETE

```
# USAGE
# python train_helmet_detector.py --dataset dataset_helmet
# importar os pacotes necessários
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical

from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths

import matplotlib.pyplot as plt
import numpy as np
import argparse
import os

# constrói a análise de argumentos (argparse)
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot_helmet.png",
help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str,
default="helmet_detector.model",
help="path to output helmet detector model")
args = vars(ap.parse_args())
```

```
# inicializa a taxa inicial de aprendizado (initial learning rate), número de
# épocas (epochs) do treino, e tamanho de pacote (batch size)
INIT_LR = 5.5e-4
EPOCHS = 50
BS = 16

# capta a lista de imagens no seu diretório de dataset, e inicializa
# a lista de imagens e classes das imagens
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []


# loop sobre o caminho das imagens
for imagePath in imagePaths:

# extrai o tipo de classe do nome do arquivo
label = imagePath.split(os.path.sep)[-2]

# carrega a imagem (224x224) e realiza o pré-processamento
image = load_img(imagePath, target_size=(224, 224))
image = img_to_array(image)
image = preprocess_input(image)

# atualiza os dados e a lista de rótulos
data.append(image)
labels.append(label)

# converte os dados e rótulos em vetores NumPy
data = np.array(data, dtype="float32")
labels = np.array(labels)

# codifica os rótulos de classe
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
```

```
77 # particionar os dados em divisães de treinamento e teste usando 80% dos
   # dados para treinamento e os 20% restantes para teste (trainX, testX,
      trainY, testY) = train_test_split(data, labels,
79 test_size=0.20, stratify=labels, random_state=42)

81 # realiza data augmentation
   aug = ImageDataGenerator(
83 rotation_range=20,
   zoom_range=0.15,
85 width_shift_range=0.2,
   height_shift_range=0.2,
87 shear_range=0.15,
   horizontal_flip=True,
89 fill_mode="nearest")

91 # carrega a rede MobileNetV2, assegura que a camada head FC
   # está de fora da rede
93 baseModel = MobileNetV2(weights="imagenet", include_top=False,
   input_tensor=Input(shape=(224, 224, 3)))
95
   # constrói a head do modelo que será colocada no topo
97 # do modelo base
   headModel = baseModel.output
99 headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
   headModel = Flatten(name="flatten")(headModel)
101 headModel = Dense(128, activation="relu")(headModel)
   headModel = Dropout(0.5)(headModel)
103 headModel = Dense(2, activation="softmax")(headModel)

105 # coloca o modelo de head FC no topo do modelo base (isso será
   # o modelo a ser treinado)
107 model = Model(inputs=baseModel.input, outputs=headModel)

109 # realiza o loop por todas as camadas do modelo e congela as camadas
   # base da rede. Os pesos da camada head serão ajustados
111 for layer in baseModel.layers:
   layer.trainable = False
113 # compila o modelo
   print("[INFO] compiling model...")
```

```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
metrics=["accuracy"])

# treina a camada head da rede
print("[INFO] training head...")
H = model.fit(
aug.flow(trainX, trainY, batch_size=BS),
steps_per_epoch=len(trainX) // BS,
validation_data=(testX, testY),
validation_steps=len(testX) // BS,
epochs=EPOCHS)

# realiza previsões no conjunto de teste
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# para cada imagem dentro da área de teste é preciso achar o índice
# dos rótulos correspondentes a maior previsão provável
predIdxs = np.argmax(predIdxs, axis=1)

# mostra a classificação formatada
print(classification_report(testY.argmax(axis=1), predIdxs,
target_names=lb.classes_))

# serializa e salva o modelo
print("[INFO] saving helmet detector model...")
model.save(args["model"], save_format="h5")

# plota os resultados do treino
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
```

```
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

APÊNDICE D – CÓDIGO SERVIDOR DE RECONHECIMENTO

```
#tensor flow dependecies
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
import numpy as np
import argparse
import cv2
import base64
import re

from flask import Flask, request, Response
import json
import asyncio
import websockets

from models.models import load_models, detectFace, detectEpi

PORT = 8035

def post_image():
    """
        Receive a image for classification

    """

    r = request
    nparr = np.frombuffer(r.data, np.uint8)
    image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    resp = epi_detector(image)

    return Response(response=resp, status=200, mimetype="application/json")

def after_request(response):
  response.headers.add('Access-Control-Allow-Origin', '*')
  response.headers.add('Access-Control-Allow-Headers', 'Content-Type,
      Authorization')
  response.headers.add('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE
      ,OPTIONS')
  response.headers.add('Access-Control-Allow-Credentials', 'true')
```

```
    return response

ClientBuffer = {}

async def start(websocket, path):
    while 1:
        data = json.loads(await websocket.recv())

        for client in data:
            image = client['image']
            clientId = client['clientId']
            bbox = post_imageb64(image, clientId)

            resp = {
                "clientId" : clientId,
                "bbox" : bbox
            }

            print(resp)

            try:
                await websocket.send(json.dumps(resp))
            except websockets.exceptions.ConnectionClosedOK:
                pass


def post_imageb64(rdata, clientId):
    """
        Receive a image for classification

    """

    patern = 'data:([a-zA-Z/]+);base64'

    datamatch = re.match(patern, rdata)

    if datamatch.group(1) == 'image/jpeg':

        data = re.split("base64", rdata)[-1]
```

```
76          nparr = np.frombuffer( base64.b64decode(data), np.uint8)


78

            try:
80              image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
            except :
82              return "fail"

84          resp = epi_detector(image)


86

            return resp#Response(response=resp, status=200, mimetype="
                application/json")
88

        return "fail" #Response(response={"error": "Imagem não é váçl"}, status
            =200, mimetype="application/json")
90

    def epi_detector(image):
92      """
            Return bbox and labels classes
94

            Keyword arguments:
96          image -- Open Cv Image Object

98          Returns list of bbox as example:
            [
100             {
                    "startX": 15,
102                 "endX": 146,
                    "startY": 43,
104                 "endY": 175,
                    "label": 1
106             },
                {
108                 "startX": 62,
                    "endX": 192,
110                 "startY": 94,
                    "endY": 183,
112                 "label": 0
```

```
            }
        ]
    """

    CONFIDENCE = 0.5
    model = models
    detections_list = []

    (confidence, startX, startY, endX, endY) = detectFace(model, image)

    if confidence > CONFIDENCE:
        # extract the face ROI, convert it from BGR to RGB channel
        # ordering, resize it to 224x224, and preprocess it
        face = image[startY:endY, startX:endX]

        try:
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
        except:
            return json.dumps([])

        face = cv2.resize(face, (224, 224))
        face = img_to_array(face)
        face = preprocess_input(face)
        face = np.expand_dims(face, axis=0)

        label = detectEpi(model, face)

        detection = {
            "startX" : startX,
            "endX" : endX,
            "startY" : startY,
            "endY" : endY,
            "label" : label
        }

        detections_list.append(detection)

    return detections_list
```

```
152 models = load_models()
    start_server = websockets.serve(start, "localhost", PORT)
154 asyncio.get_event_loop().run_until_complete(start_server)
    asyncio.get_event_loop().run_forever()
```

APÊNDICE E – EXEMPLOS NEGATIVOS E POSITIVOS DO DATASET

Figura 84 – Exemplo positivo e negativo de óculos, respectivamente. Fonte : Autores

Fonte : Autores

Figura 85 – Exemplo positivo e negativo de capacete.

Fonte: Autores

Figura 86 – Exemplo positivo e negativo de máscara, respectivamente.

Fonte: Autores

Figura 87 – Exemplo positivo e negativo de abafador, respectivamente.

Fonte: Adaptado de Handyman, 0023 e Pixabay, 0023

APÊNDICE F – COMPONENTE REACT MASTER WEBRTC

```
import AWS from 'aws-sdk'
import axios from 'axios'
import React, {useRef, useEffect, useState} from 'react'
import Grid from '@material-ui/core/Grid';
import {useEpi, useEpiUpdate} from '../../../contexts/EpiContext'
import Dynamo from '../../../api/Dynamo/Dynamo'
import {Container, Item} from './style.jsx'
const {SignalingClient} = require('amazon-kinesis-video-streams-webrtc');
const KVSWebRTC =  require('amazon-kinesis-video-streams-webrtc');


const videoConstraints = {
    width: 640,
    height: 480,
    facingMode: "user"
  };

const master = {
    signalingClient: null,
    peerConnectionByClientId: {},
    dataChannelByClientId: {},
    boxByClientId: {},
    localStream: null,
    remoteStreams: [],
    peerConnectionStatsInterval: null,
    isDataChannelOpen : false
};

const {natTraversalDisabled, forceTURN, region,
    accessKeyId, secretAccessKey, sessionToken, endpoint,
    Xmax, Ymax, channelARN, openDataChannel,
    onRemoteDataMessage, onStatsReport, useTrickleICE,
    masterSendVideo, masterSendAudio, widescreen,
    viewerSendAudio, viewerSendVideo, masterSizeX, masterSizeY,
    showBoxOnMaster} =  require('../config.json');


export function Master(props)
```

```
{
    var startX = 0
    var endX = 0
    var startY = 0
    var endY = 0
    var boxlabel = 0
    var localStream
    var imageData
    var Role = KVSWebRTC.Role.MASTER;
    var boxData = []

    var hasVideo = false
    const containerRef = useRef(null)
    const EpiUpdate = useEpiUpdate()
    const remoteClientId = props.clientId
    const clientId = props.clientId
    const [videoStreams, setVideoStreams] = useState({mArray:[]})

    function addVideoStream(stream, clientId)
    {

        var streamsArray = videoStreams.mArray

        const clientToAdd = {
            clientId,
            stream
        }

        if(!streamsArray.includes(clientToAdd))
        {

            streamsArray.push(clientToAdd)
            setVideoStreams({mArray: streamsArray})
            console.log(containerRef.current.children[streamsArray.length -
                1].children[1])
            containerRef.current.children[streamsArray.length - 1].children
                [1].srcObject = stream
            console.log(stream)
        }
```

```
    }

77

    useEffect(() =>
    {
        var socket;
        master.iswsConnected = false;
83      master.clientId = remoteClientId
        async function init()
85      {
            function drawCanvas()
87          {
                function drawRec(ctx, boxData)
89              {
                    function getColor(label)
91                  {
                        const colorMap = [
93                          {"label" : "0" , "color" : 'red'},
                            {"label" : "1" , "color" : "yellow"},
95                          {"label" : "2" , "color" : "blue"},
                            {"label" : "3" , "color" : "white"},
97                      ]
                        var color;
99                      try
                        {
101                         color = colorMap[label].color
                        }
103                     catch(e)
                        {
105                         color = colorMap[0].color
                        }
107                     return  color
                    }
109

                  for (var i in boxData)
111               {
                    if(typeof boxData !== "undefined" )
113                 {
                        if(showBoxOnMaster)
```

```
115                         {
                                ctx.beginPath();
117                             ctx.moveTo(boxData[i].startX, boxData[i].endY);
                                ctx.lineTo(boxData[i].endX, boxData[i].endY);
119                             ctx.lineTo(boxData[i].endX, boxData[i].startY);
                                ctx.lineTo(boxData[i].startX, boxData[i].startY
                                    );
121                             ctx.lineTo(boxData[i].startX, boxData[i].endY);
                                ctx.strokeStyle = getColor(boxData[i].label);
123                             ctx.lineWidth = 2
                                ctx.stroke()
125                             ctx.closePath();
                            }
127                         EpiUpdate(boxData[0].label);
                        }
129                   }

131               }

133               try{
                      for (var i in videoStreams.mArray)
135                   {
                          var canvas = containerRef.current.children[i].
                              children[0]
137                       var video = containerRef.current.children[i].
                              children[1]

139                       if(canvas !== null && hasVideo)
                          {
141                           canvas.width = masterSizeX;
                              canvas.height = masterSizeY;
143                           const ctx = canvas.getContext("2d")
                              ctx.drawImage(video, 0, 0, masterSizeX,
                                  masterSizeY);
145                           drawRec(ctx, master.boxByClientId[videoStreams.
                                  mArray[i].clientId])
                          }
147
```

```
149                     }

151                 }
                    catch(e)
153                 {
                        console.log(e)
155                 }

157             }

159             const kinesisVideoClient = new AWS.KinesisVideo({
                    region,
161                 accessKeyId,
                    secretAccessKey,
163                 sessionToken,
                    endpoint,
165                 correctClockSkew: true,
                });
167
                console.log('[MASTER] Channel ARN: ', channelARN);
169
                // Get signaling channel endpoints
171             const getSignalingChannelEndpointResponse = await
                    kinesisVideoClient
                    .getSignalingChannelEndpoint({
173                     ChannelARN: channelARN,
                        SingleMasterChannelEndpointConfiguration: {
175                         Protocols: ['WSS', 'HTTPS'],
                            Role: KVSWebRTC.Role.MASTER,
177                     },
                    })
179                 .promise();

181             const endpointsByProtocol = getSignalingChannelEndpointResponse
                    .ResourceEndpointList.reduce((endpoints, endpoint) => {
                    endpoints[endpoint.Protocol] = endpoint.ResourceEndpoint;
183                 return endpoints;
                }, {});
185
```

```
            console.log('[MASTER] Endpoints: ', endpointsByProtocol);
187
            // Create Signaling Client
189         master.signalingClient = new KVSWebRTC.SignalingClient({
                channelARN,
191             channelEndpoint: endpointsByProtocol.WSS,
                role: KVSWebRTC.Role.MASTER,
193             region,
                credentials: {
195                 accessKeyId,
                    secretAccessKey,
197                 sessionToken,
                },
199             systemClockOffset: kinesisVideoClient.config.
                    systemClockOffset,
            });
201
            // Get ICE server configuration
203         const kinesisVideoSignalingChannelsClient = new AWS.
                KinesisVideoSignalingChannels({
                region,
205             accessKeyId,
                secretAccessKey,
207             sessionToken,
                endpoint: endpointsByProtocol.HTTPS,
209             correctClockSkew: true,
            });
211         const getIceServerConfigResponse = await
                kinesisVideoSignalingChannelsClient
                .getIceServerConfig({
213                 ChannelARN: channelARN,
                })
215             .promise();
            const iceServers = [];
217
            if (!natTraversalDisabled && !forceTURN) {
219             iceServers.push({ urls: `stun:stun.kinesisvideo.${region}.
                    amazonaws.com:443` });
            }
```

```
221
            if (!natTraversalDisabled) {
223             getIceServerConfigResponse.IceServerList.forEach(iceServer
                    =>
                    iceServers.push({
225                     urls: iceServer.Uris,
                        username: iceServer.Username,
227                     credential: iceServer.Password,
                    }),
229             );
            }
231         console.log('[MASTER] ICE servers: ', iceServers);

233         const configuration = {
                iceServers,
235             iceTransportPolicy: forceTURN ? 'relay' : 'all',
            };
237
            const resolution = widescreen ? { width: { ideal: 1280 },
                height: { ideal: 720 } } : { width: { ideal: 640 }, height:
                { ideal: 480 } };
239         const constraints = {
                video: masterSendVideo ? resolution : false,
241             audio: masterSendAudio,
            };
243


245         if (masterSendVideo || masterSendAudio) {
                try {
247                 master.localStream = await navigator.mediaDevices.
                        getUserMedia(constraints);
                } catch (e) {
249                 console.error('[MASTER] Could not find webcam');
                    alert("Não foi possível detectar Webcam")
251             }

253         }


255
```

```
            master.signalingClient.on('open', async () => {
257             console.log('[MASTER] Connected to signaling service');
            });
259
            master.signalingClient.on('sdpOffer', async (offer,
                remoteClientId) => {
261             console.log('[MASTER] Received SDP offer from client: ' +
                    remoteClientId);

263             // Create a new peer connection using the offer from the
                    given client
                const peerConnection = new RTCPeerConnection(configuration)
                    ;
265             master.peerConnectionByClientId[remoteClientId] =
                    peerConnection;

267             if (openDataChannel) {
                    console.log("remoteClientId = ", remoteClientId);
269                 master.dataChannelByClientId[remoteClientId] =
                        peerConnection.createDataChannel('kvsDataChannel');
                    console.log('[MASTER] Opening data channel with client:
                         ' + remoteClientId);
271
                    peerConnection.ondatachannel = event => {
273                     console.log('[MASTER] Data channel with client: ' +
                            remoteClientId, ' is open');
                        master.isDataChannelOpen = true
275                     master.clientId = remoteClientId
                        //event.channel.onmessage = onRemoteDataMessage;
277                 };

279             }

281             if (!master.peerConnectionStatsInterval) {
                    master.peerConnectionStatsInterval = setInterval(() =>
                        peerConnection.getStats().then(onStatsReport), 1000)
                        ;
283             }
```

```
285                 peerConnection.addEventListener('icecandidate', ({
                        candidate }) => {
                    if (candidate) {
287                     console.log('[MASTER] Generated ICE candidate for
                            client: ' + remoteClientId);

289                     if (useTrickleICE) {
                            console.log('[MASTER] Sending ICE candidate to
                                client: ' + remoteClientId);
291                         master.signalingClient.sendIceCandidate(
                                candidate, remoteClientId);
                        }
293                 } else {
                        console.log('[MASTER] All ICE candidates have been
                            generated for client: ' + remoteClientId);
295
                        // When trickle ICE is disabled, send the answer
                            now that all the ICE candidates have ben
                            generated.
297                     if (!useTrickleICE) {
                            console.log('[MASTER] Sending SDP answer to
                                client: ' + remoteClientId);
299                         master.signalingClient.sendSdpAnswer(
                                peerConnection.localDescription,
                                remoteClientId);
                        }
301                 }
                });
303

305             peerConnection.onconnectionstatechange = event => {
                    console.log('[MASTER] Connection: ',remoteClientId, '
                        state change to ', peerConnection.connectionState )
307
                    switch(peerConnection.connectionState)
309                 {
                        case 'disconnected':
311                     case 'failed':
                        case 'closed':
```

```
313                         master.isDataChannelOpen = false
                            videoStreams.mArray.map(
315                             (client, index) =>{
                                    if(client.clientId === remoteClientId)
317                                 {
                                        var videostreams = videoStreams
319                                     videostreams.mArray.splice(index,
                                            1)
                                        console.log("[MASTER] Connection
                                            with peer lost removing index=",
                                             index)
321                                     console.log("[MASTER] Array = ",
                                            videostreams.mArray )
                                        setVideoStreams(videostreams)
323                                     if(index === 0)
                                        {
325                                         hasVideo = false
                                        }
327                                 }

329                             }
                            )
331                     break;
                }
333         }

335         // As remote tracks are received, add them to the remote
                view
            peerConnection.addEventListener('track', event => {
337             console.log('[MASTER] Received remote track from client
                    : ' + remoteClientId);
                //videoRef.current.srcObject = event.streams[0]
339             addVideoStream(event.streams[0], remoteClientId)
                hasVideo = true
341

            });
343


345         if (master.localStream) {
```

```
                master.localStream.getTracks().forEach(track => {
                    peerConnection.addTrack(track, master.localStream);
                    console.log(track)});
            }
            await peerConnection.setRemoteDescription(offer);

            console.log('[MASTER] Creating SDP answer for client: ' +
                remoteClientId);
            await peerConnection.setLocalDescription(
                await peerConnection.createAnswer({
                    offerToReceiveAudio: viewerSendAudio,
                    offerToReceiveVideo: viewerSendVideo,
                }),
            );

            if (useTrickleICE) {
                console.log('[MASTER] Sending SDP answer to client: ' +
                    remoteClientId);
                master.signalingClient.sendSdpAnswer(peerConnection.
                    localDescription, remoteClientId);
            }
            console.log('[MASTER] Generating ICE candidates for client:
                ' + remoteClientId);
        });

        master.signalingClient.on('iceCandidate', async (candidate,
            remoteClientId) => {
            console.log('[MASTER] Received ICE candidate from client: '
                + remoteClientId);

            const peerConnection = master.peerConnectionByClientId[
                remoteClientId];
            peerConnection.addIceCandidate(candidate);

        });
```

```
377             master.signalingClient.on('close', async () => {
                    console.log('[MASTER] Disconnected from signaling channel')
                        ;
379                 await Dynamo.del(clientId)
                    //window.location.reload()
381             });

383             master.signalingClient.on('error', async () => {
                    console.error('[MASTER] Signaling client error');
385                 await Dynamo.del(clientId)
                    //window.location.reload()
387
                });
389
                console.log('[MASTER] Starting master connection');
391             master.signalingClient.open();


393
                setInterval(() => {
395
                    async function getUser()
397                 {
                        var dataArray = []
399                     for (var i in videoStreams.mArray)
                        {
401                         var canvas = containerRef.current.children[i].
                                children[0]

403                         const type  = "image/jpeg"
                            imageData = canvas.toDataURL(type)
405
                            const data = {
407                             "image" : imageData,
                                clientId : videoStreams.mArray[i].clientId
409                         }

411                         dataArray.push(data)
                        }
```

```
413
                    if(master.iswsConnected)
415                 {
                        socket.send(JSON.stringify(dataArray));
417                 }

419             }
                if(hasVideo)
421             {
                    getUser()
423             }

425         },500);


427
            setInterval(() => {
429           drawCanvas()
            },1000/30);
431
            function wsconnect()
433         {
                console.log("[MASTER] Websocket connection connecting ...."
                    )
435             socket = new WebSocket('ws://localhost:8035');
            }
437
            wsconnect()
439


441         socket.addEventListener('open', function (event) {
                console.log("[MASTER] Websocket connection opened")
443             master.iswsConnected = true;
                alert("Conectado ao servidor de reconhecimento")
445         });


447
            socket.addEventListener('message', function (event) {
449             const data = JSON.parse(event.data);
                master.boxByClientId[data.clientId] = data.bbox
```

```
451
                if(master.isDataChannelOpen)
453             {
                    master.dataChannelByClientId[data.clientId].send(JSON.
                        stringify(data.bbox))
455             }
            });
457
            socket.addEventListener('close', event =>
459         {
                setTimeout(wsconnect, 3000);
461             master.iswsConnected = false;
                alert("Conexão com servidor de reconhecimento perdida")
463             console.log("[MASTER] Websocket connection closed")
            })
465
            socket.addEventListener('error', event =>
467         {
                setTimeout(wsconnect, 3000);
469             master.iswsConnected = false;
                alert("Conexão com servidor de reconhecimento perdida")
471             console.log("[MASTER] Websocket connection closed")
            })
473
        }
475     init()
    }, [])
477
    return (
479 <>
        <Container ref={containerRef}>
481             {videoStreams.mArray.map( (stream, i)=>(
                    <Item>
483                     <canvas />
                        <video
485                             style={{width: '0%',
                                heigth: '0%',
487                             minHeight: '0',
                                maxHeight: '100px',
```

```
                                    position: 'relative' }}
                                    autoPlay playsInline />
                        </Item>

        ))}
         </Container>

    </>
    );

}

function stopMaster() {
    console.log('[MASTER] Stopping master connection');
    if (master.signalingClient) {
        master.signalingClient.close();
        master.signalingClient = null;
    }

    Object.keys(master.peerConnectionByClientId).forEach(clientId => {
        master.peerConnectionByClientId[clientId].close();
    });
    master.peerConnectionByClientId = [];

    if (master.localStream) {
        master.localStream.getTracks().forEach(track => track.stop());
        master.localStream = null;
    }

    master.remoteStreams.forEach(remoteStream => remoteStream.getTracks().
        forEach(track => track.stop()));
    master.remoteStreams = [];

    if (master.peerConnectionStatsInterval) {
        clearInterval(master.peerConnectionStatsInterval);
        master.peerConnectionStatsInterval = null;
    }
    /*
    if (master.localView) {
```

```
527             master.localView.srcObject = null;
        }
529     */
        if (master.remoteView) {
531             master.remoteView.srcObject = null;
        }
533
        if (master.dataChannelByClientId) {
535             master.dataChannelByClientId = {};
        }
537 }

539 function sendMasterMessage(message) {
        Object.keys(master.dataChannelByClientId).forEach(clientId => {
541             try {
                    master.dataChannelByClientId[clientId].send(message);
543             } catch (e) {
                    console.error('[MASTER] Send DataChannel: ', e.toString());
545             }
        });
547 }
```

APÊNDICE G – ESTILOS COMPONENTE MASTER

```
import styled from 'styled-components'

export const Container = styled.div`
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 190px 190px 190px;
    padding : 10px;
    box-shadow: 0px 1px 1px rgba(0, 0, 0, 0.14), 0px 2px 1px rgba(0, 0, 0,
        0.12), 0px 1px 3px rgba(0, 0, 0, 0.2);
    border-radius: 4px;
    height: 600px;
`
export const Item = styled.div`

`
```

APÊNDICE H – TERMOS DE USO DE IMAGEM

AUTORIZAÇÃO DE USO DE IMAGEM

Eu, LEONARDO HENRIQUE GALVÃO, portador da Identidade nº [redacted], inscrito no CPF sob nº [redacted], residente à Rua [redacted], nº [redacted], na cidade de [redacted], AUTORIZO o uso de minha imagem em fotos ou filme, sem finalidade comercial, para ser utilizada no trabalho de conclusão de curso "RECONHECIMENTO DO USO DE EPI POR MEIO DE VISÃO COMPUTACIONAL UTILIZANDO UMA CNN".

A presente autorização é concedida a título gratuito, abrangendo o uso da imagem acima mencionada em todo território nacional e no exterior, em todas as suas modalidades. Por esta ser a expressão da minha vontade declaro que autorizo o uso acima descrito sem que nada haja a ser reclamado a título de direitos conexos à minha imagem ou a qualquer outro.

Barueri, São Paulo, 07 de DEZEMBRO de 2020.

Assinatura

AUTORIZAÇÃO DE USO DE IMAGEM

Eu, MICHEL ERIC RABELO DA SILVA, portador da Identidade nº [redacted], inscrito no CPF sob nº [redacted] residente à Rua [redacted], nº [redacted], na cidade de [redacted] AUTORIZO o uso de minha imagem em fotos ou filme, sem finalidade comercial, para ser utilizada no trabalho de conclusão de curso "RECONHECIMENTO DO USO DE EPI POR MEIO DE VISÃO COMPUTACIONAL UTILIZANDO UMA CNN".

A presente autorização é concedida a título gratuito, abrangendo o uso da imagem acima mencionada em todo território nacional e no exterior, em todas as suas modalidades. Por esta ser a expressão da minha vontade declaro que autorizo o uso acima descrito sem que nada haja a ser reclamado a título de direitos conexos à minha imagem ou a qualquer outro.

Barueri, São Paulo, 07 de DEZEMBRO de 2020.

Assinatura

AUTORIZAÇÃO DE USO DE IMAGEM

Eu, Caio Jagussenski da Silva, portador da Identidade nº [redacted], inscrito no CPF sob nº [redacted], residente à Rua [redacted], nº [redacted], na cidade de [redacted], AUTORIZO o uso de minha imagem em fotos ou filme, sem finalidade comercial, para ser utilizada no trabalho de conclusão de curso "RECONHECIMENTO DO USO DE EPI POR MEIO DE VISÃO COMPUTACIONAL UTILIZANDO UMA CNN".

A presente autorização é concedida a título gratuito, abrangendo o uso da imagem acima mencionada em todo território nacional e no exterior, em todas as suas modalidades. Por esta ser a expressão da minha vontade declaro que autorizo o uso acima descrito sem que nada haja a ser reclamado a título de direitos conexos à minha imagem ou a qualquer outro.

Barueri, São Paulo, 07 de Dezembro de 2020.

Caio J.

Assinatura

AUTORIZAÇÃO DE USO DE IMAGEM

Eu, [illegible], portador da Identidade nº [illegible], inscrito no CPF sob nº [illegible], residente à Rua [illegible], nº [illegible], na cidade de [illegible], AUTORIZO o uso de minha imagem em fotos ou filme, sem finalidade comercial, para ser utilizada no trabalho de conclusão de curso "RECONHECIMENTO DO USO DE EPI POR MEIO DE VISÃO COMPUTACIONAL UTILIZANDO UMA CNN".

A presente autorização é concedida a título gratuito, abrangendo o uso da imagem acima mencionada em todo território nacional e no exterior, em todas as suas modalidades. Por esta ser a expressão da minha vontade declaro que autorizo o uso acima descrito sem que nada haja a ser reclamado a título de direitos conexos à minha imagem ou a qualquer outro.

Barueri, São Paulo, 07 de Dezembro de 2020.

Assinatura

REFERÊNCIAS

A., Gulli; S., Pal. **PDeep Learning with Keras. Birmingham: Packt Publishing Ltd.** 2017. Disponível em: <http://ww2.inf.ufg.br/~anderson/deeplearning/Deep%20Learning%20-%20Redes%20Neurais%20Profundas%20Region%20Based%20CNNs%20R-CNN.pdf>. Acesso em: 11 mai. 2020.

A., Keranen; C., Holmberg; J., Rosenberg. **Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal**. [S.l.: s.n.], 2018. Disponível em: <https://tools.ietf.org/html/rfc8445>.

ALAN, Levine. He looks series but is quick to smile. In:

ANTHONY, Martin. **Discrete Mathematics of Neural Networks: Selected Topics**. Disponível em: <https://books.google.com.br/books?id=qOy4yLBqhFcC&pg=PA3&redir_esc=y#v=onepage&q&f=false>. Acesso em: 24 abr. 2020.

ASUS. **Tinker Board**. Disponível em: <https://www.asus.com/br/Single-Board-Computer/Tinker-Board/>. Acesso em: 11 mai. 2020.

BISHOP, Christopher M. **Pattern Recognition and Machine Learning**. [S.l.]: Springer, 2011.

BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. **YOLOv4: Optimal Speed and Accuracy of Object Detection**. [S.l.: s.n.], 2020. arXiv: 2004.10934 [cs.CV].

COCO, Dataset. **Detection Evaluation**. Disponível em: <http://cocodataset.org/#detection-eval>. Acesso em: 15 mai. 2020.

CORAL. **Dev Board**. Disponível em: <https://coral.ai/products/dev-board/>. Acesso em: 9 mai. 2020.

CORAL DevBoard. Disponível em: <https://www.cnx-software.com/>. Acesso em: 21 abr. 2020.

DEVELOPER, Nvidia. **Jetson Nano Developer Kit**. Disponível em: <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>. Acesso em: 11 mai. 2020.

F., Andreasen; M., Baugher. **Session Description Protocol (SDP) Security Descriptions for Media Streams**. [S.l.: s.n.], 2006. Disponível em: <https://tools.ietf.org/html/rfc4568>.

FATHI, Jonathan et al. Speed/accuracy trade-offs for modern convolutional object detectors. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, IEEE, p. 10, 2017. DOI: 10.1109/CVPR.2017.351.

FERREIRA, C. **Designing Neural Networks Using Gene Expression Programming**. Disponível em:

<https://www.gene-expression-programming.com/webpapers/Ferreira-ASCT2006.pdf>. Acesso em: 24 abr. 2020.

FICHA técnica Poli-Ferr. Disponível em: <https://poli-ferr.com.br/wp-content/uploads/2019/11/Ficha_produto-_rio_alterada.pdf>. Acesso em: 16 mai. 2020.

FLORINDO, João B. **Redes Neurais Convolucionais**. Disponível em: <https://www.ime.unicamp.br/~jbflorindo/Teaching/2018/MT530/T10.pdf>. Acesso em: 24 abr. 2020.

GÉRON, Aurélien. **Hands-On Machine Learning with Scikit-Learn and TensorFlow**. [S.l.]: OREILLY, 2017.

HAN JUN; MORAG, Claudio. **The influence of the sigmoid function parameters on the speed of backpropagation learning**. Disponível em: <https://archive.org/details/fromnaturaltoart1995inte/page/195/mode/2up>. Acesso em: 9 mai. 2020.

HANDYMAN, Family. How to Choose the Best Hearing Protection. In:

HATTORI, L. T. **Redes Neurais Convolucionais**. Disponível em: <http://silverio.net.br/heitor/disciplinas/eeica/aulas/aula3-CNN.pdf>. Acesso em: 24 abr. 2020.

HOWARD, Andrew G. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. **CoRR**, abs/1704.04861, 2017. arXiv: 1704.04861. Disponível em: <http://arxiv.org/abs/1704.04861>.

HUI, Jonathan. **mAP (mean Average Precision) for Object Detection**. Disponível em: <https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173>. Acesso em: 21 abr. 2020.

I., Fette et al. **The WebSocket Protocol**. [S.l.: s.n.], 2018. Disponível em: <https://tools.ietf.org/html/rfc6455>.

IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. **CoRR**, abs/1502.03167, 2015. arXiv: 1502.03167. Disponível em: <http://arxiv.org/abs/1502.03167>.

JIA, Yangqing et al. Caffe: Convolutional Architecture for Fast Feature Embedding. **arXiv preprint arXiv:1408.5093**, 2014.

KRATSIOS, Anastasis. **Deep Arbitrage-Free Learning in a Generalized HJM Framework via Arbitrage-Regularization Data**. Disponível em: <https://www.mdpi.com/2227-9091/8/2/40>. Acesso em: 4 mai. 2020.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. **Imagenet classification with deep convolutional neural networks**. Disponível em: <https:

//proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>. Acesso em: 24 abr. 2020.

LE CUN, Yann; CORTES, Corinna. **THE MNIST DATABASE of handwritten digits**. Disponível em: <http://yann.lecun.com/exdb/mnist/>. Acesso em: 21 abr. 2020.

LIN, Tsung-Yi et al. Microsoft COCO: Common Objects in Context. **CoRR**, abs/1405.0312, 2014. arXiv: 1405.0312. Disponível em: <http://arxiv.org/abs/1405.0312>.

LITE, TensorFlow. **Performance benchmarks**. Disponível em: <http://ww2.inf.ufg.br/~anderson/deeplearning/Deep%20Learning%20-%20Redes%20Neurais%20Profundas%20Region%20Based%20CNNs%20R-CNN.pdf>. Acesso em: 11 mai. 2020.

MARCHESAN, Ricardo. **Brasil é campeão de ações trabalhistas no mundo? Dados são inconclusivos**. Disponível em: <https://noticias.uol.com.br/confere/ultimas-noticias/2017/06/27/brasil-e-campeao-de-acoes-trabalhistas-no-mundo-dados-sao-inconclusivos.htm>. Acesso em: 16 mai. 2020.

MISHRA, Sanatan. **Unsupervised Learning and Data Clustering**. Disponível em: <https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a>. Acesso em: 19 mai. 2017.

MITCHELL, Tom. **Machine Learning**. Disponível em: <http://www.cs.cmu.edu/~tom/mlbook.html>. Acesso em: 24 abr. 2020.

NEURONS, Imploding. **TensorFlow or Keras? Which one should I learn?** Disponível em: <https://medium.com/implodinggradients/tensorflow-or-keras-which-one-should-i-learn-5dd7fa3f9ca0>. Acesso em: 11 mai. 2020.

NIELSEN, Michael A. **Neural Networks and Deep Learning**. [S.l.]: Determination Press, 2015. Disponível em: <http://neuralnetworksanddeeplearning.com/>. Acesso em: 21 abr. 2020.

OMNIVISION. **OV5647 Datasheet**. Disponível em: <https://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/ov5647_full.pdf>. Acesso em: 11 mai. 2020.

PHOTOGRAPHY33. Segurança no local de trabalho. In:

PIXABAY. Woman in Gray Vest With Yellow Hard Hat Inside Room. In:

PRODUCTIONS, Syda. Happiness and people concept - smiling man. In:

RASPBERRY. **Raspberry Pi 4**. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>. Acesso em: 11 mai. 2020.

REN, Shaoqing et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. **CoRR**, abs/1506.01497, 2015. arXiv: 1506.01497. Disponível em: <http://arxiv.org/abs/1506.01497>.

REZATOFIGHI, Seyed Hamid et al. Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression. **CoRR**, abs/1902.09630, 2019. arXiv: 1902.09630. Disponível em: <http://arxiv.org/abs/1902.09630>.

ROCHA, Rosely. **No Brasil, a cada 48 segundos um trabalhador sofre acidente e um morre a cada 4h**. Disponível em: <https://spbancarios.com.br/08/2018/no-brasil-cada-48-segundos-um-trabalhador-sofre-acidente-e-um-morre-cada-4h>. Acesso em: 16 mai. 2020.

RODRIGUES, Karla Bertolasce Frauches; SANTOS, Nadson Gutemberg Gomes dos. **As consequências legais pelo não uso do equipamento de proteção individual no ambiente de trabalho: uma breve análise a luz do ordenamento jurídico brasileiro.** Disponível em: <https://ambitojuridico.com.br/edicoes/revista-167/as-consequencias-legais-pelo-nao-uso-do-equipamento-de-protecao-individual-no-ambiente-de-trabalho-uma-breve-analise-a-luz-do-ordenamento-juridico-brasileiro>. Acesso em: 16 mai. 2020.

ROSEBROCK, A. **Keras vs. TensorFlow – Which one is better and which one should I learn?** Disponível em: <https://www.pyimagesearch.com/2018/10/08/keras-vs-tensorflow-which-one-is-better-and-which-one-should-i-learn/>. Acesso em: 11 mai. 2020.

ROSEBROCK, Adrian. **Intersection over Union (IoU) for object detection**. Disponível em: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. Acesso em: 23 abr. 2020.

RUSCI, Manuele. **Running Mobilenet on STM32 MCUs at the edge**. Disponível em: <https://towardsdatascience.com/running-mobilenet-on-stm32-mcus-at-the-edge-e217db934f83>. Acesso em: 10 mai. 2020.

SANDLER, Mark et al. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. **CoRR**, abs/1801.04381, 2018. arXiv: 1801.04381. Disponível em: <http://arxiv.org/abs/1801.04381>.

SCHMIDHUBER, J. **Deep Learning in Neural Networks: An Overview**. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0893608014002135?via%3Dihub>. Acesso em: 24 abr. 2020.

SHAIKH, F. **Deep Learning Guide: Introduction to Implementing Neural Networks using TensorFlow in Python**. Disponível em: <https://www.analyticsvidhya.com/blog/2016/10/an-introduction-to-implementing-neural-networks-using-tensorflow/>. Acesso em: 11 mai. 2020.

SHIN, Hoo-Chang et al. Deep Convolutional Neural Networks for Computer-Aided Detection: CNNArchitectures, Dataset Characteristics and Transfer Learning. **CoRR**, abs/1602.03409, 2016. arXiv: 1602.03409. Disponível em: <http://arxiv.org/abs/1602.03409>.

SOARES, Anderson. **Performance benchmarks**. Disponível em: <http://ww2.inf.ufg.br/~anderson/deeplearning/Deep%20Learning%20-%20Redes%20Neurais%20Profundas%20Region%20Based%20CNNs%20R-CNN.pdf>. Acesso em: 15 mai. 2020.

ST. **STM32H743ZI**. Disponível em: <https://www.st.com/en/microcontrollers-microprocessors/stm32h743zi.html>. Acesso em: 9 mai. 2020.

TETKO, I. V.; LIVINGSTONE, D. J.; LUIK, A. I. **Neural network studies. 1. Comparison of Overfitting and Overtraining**. Disponível em: <http://www.vcclab.org/articles/jcics-overtraining.pdf>. Acesso em: 4 mai. 2020.

ULB, Machine Learning Group. **Credit Card Fraud Detection**. Disponível em: <https://www.kaggle.com/mlg-ulb/creditcardfraud>. Acesso em: 21 abr. 2020.

VISUAL RECOGNITION (SPRING 2017) - STANFORD UNIVERSITY SCHOOL OF ENGINEERING, Convolutional Neural Networks for. **Redes Neurais Convolucionais**. Disponível em: <http://cs231n.stanford.edu/>. Acesso em: 24 abr. 2020.

WILSON, Aidan. **A Brief Introduction to Supervised Learning**. Disponível em: <https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590>. Acesso em: 29 set. 2019.

WORKSHOP, DroneBot. **Getting started with the ESP32-CAM**. Disponível em: <https://dronebotworkshop.com/esp32-cam-intro/>. Acesso em: 8 nov. 2020.

YAO, Bangpeng et al. Human action recognition by learning bases of action attributes and parts. In: p. 1331–1338. DOI: 10.1109/ICCV.2011.6126386.

ZHAO, Jiaping; CHANG, Chin-Kai; ITTI, Laurent. **Learning to Recognize Objects by Retaining other Factors of Variation**. abs/1607.05851. [S.l.: s.n.], 2016. arXiv: 1607.05851. Disponível em: <http://arxiv.org/abs/1607.05851>.

ÍNDICE