UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

VICTOR FREDERICO BEUST DA SILVA

Estudo de impacto de técnicas de IA e comunicação para aplicações edge

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Luigi Carro

Porto Alegre
2023

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Beust da Silva, Victor Frederico

Estudo de impacto de técnicas de IA e comunicação para aplicações edge / Victor Frederico Beust da Silva. – Porto Alegre: PPGC da UFRGS, 2023.

57 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2023. Orientador: Luigi Carro.

1. Yolo. 2. SaaS. 3. Mobile. 4. IoT. I. Carro, Luigi. II. Título.

“Opportunity does not knock, it presents itself
when you beat down the door.”
— KYLE CHANDLER

AGRADECIMENTOS

Agradeço primeiramente a minha família, ao meu orientador Luigi Carro que enxergou o papel fundamental da visão computacional como serviço antes dela se tornar pervavisa; aos colegas do SERPRO Carlos Rodrigo Fonseca Lima que como gestor apoiou a realização desse trabalho e principalmente ao Ricardo Dalmaso e Alexandre Sousa, Pardal, que foram incansáveis na transformação das idéias aqui apresentadas em um serviço disponível à nação brasileira.

Aos amigos Bruno Sousa e Luiz Fernando Damasceno Ribeiro.

RESUMO

O seguinte trabalho apresenta uma avaliação do impacto da tecnologia 5G nas aplicações de reconhecimento de imagens em tempo real tomando por caso concreto o algoritmo You Only Look Once (YOLO) (REDMON; FARHADI, 2016a). É construída uma aplicação em nuvem de forma a verificar os impactos da tecnologia 5G no provimento de visão computacional.

Tendo sido realizado no contexto da pandemia, o trabalho buscou aplicar o algoritmo YOLO na solução do problema real de reconhecimento e extração de dados documentais no contexto de atendimento remoto. Essa necessidade vem de encontro aos novos modelos de cadastro biométrico e documental de vários órgãos governamentais como Tribunal Superior Eleitoral, Auxílio Brasil, Portal Gov.Br entre outros.

Para isso foi construída uma aplicação Python capaz de rodar em conteineres da plataforma Cuda executados em *Graphical Processing Units* NVIDIA. A aplicação tem como entrada a imagem de um documento e como saída o conjunto de informações nele contido. O projeto prevê grande escalabilidade capaz de atender as demandas de uma solução digital para a totalidade da sociedade brasileira.

Neste trabalho focou-se na criação de um serviço de reconhecimento e validação de documentos através de visão computacional, tomando como técnica fundamental o algoritmo You Only Look Once (YOLO). Tal escolha foi baseada na versatilidade e alta performance apresentada pelo algoritmo, a qual se mostrou ideal para a classe de problema encontrado no atendimento dessa demanda. O resultado final deste estudo foi o viabilizador da solução *Cognitive Document Validation*(CDV) do sistema Datavalid disponível em: https://www.loja.serpro.gov.br/datavalid.

Esse cenário foi motivado por demandas excepcionais como a pandemia de Covid-19 e a migração das bases de dados documentais dos registros físicos para a núvem. Surge nesse contexto a necessidade da verificação dos dados documentais em grande velocidade para atender a população, essa verificação precisa ser oferecida de maneira transparente para uma série de dispotivos que acessam aplicativos como o auxílio brasil e sistemas de embarque de companias aéreas. Tem-se como resultado final a elaboração de uma arquitetura de serviçp implementada como um sistema publicamente disponível no barramento de aplicações do Serviço Federal de Processamento de Dados (SERPRO).

Palavras-chave: Yolo. SaaS. Mobile. IoT.

ABSTRACT

The following work presents an evaluation of the alternatives for object recognition in videos through the YOLO (You Only Look Once) algorithm (REDMON; FARHADI, 2016a). Cloud computing, mobile and local processing approaches are explored on different platforms.

Being made in the pandemic context this works aims to apply the YOLO algorithm into a real world case study to recognize and extract documental data in the remote office scenario. This needs comex towards the new models of biometric registration and documental validation of many governmental offices as the Tribunal Superior Eleitoral, Auxílio Brasil, Portal Gov.Br among others.

Starting from a implementation of the Darknet Neural Network running the YOLO algorithm until transform it into and API avaliable on the cloud.

Keywords: Yolo, SaaS, Mobile,IoT.

LISTA DE ABREVIATURAS E SIGLAS

YOLO	You Only Look Once
RCNN	Regions with Convolutional Neural Network
SaaS	Software as a Service
IoU	Intersection over Union
CNN	Convolutional Neural Networks
DNN	Deep Neural Networks
ReLU	Rectified Linear Unit
REST	Representational State Transfer
JSON	JavaScript Object Notation

LISTA DE FIGURAS

Figura 1.1 User Point Function 13

Figura 2.1 Operação de Convolução 18
Figura 2.2 Operação Stride 19
Figura 2.3 Filtro Convolucional 19
Figura 2.4 Operação de Maxpooling 20
Figura 2.5 Ilustração Conceito Algoritmo YOLO 22
Figura 2.6 Ilustração Funcionamento Algoritmo YOLO 22
Figura 2.7 Intersection Over Union (IOU) 23
Figura 2.8 Quadrantes Identificados pelo Algoritmo 24
Figura 2.9 Rede Neural YOLO 24

Figura 4.1 4G to 5G paradigm evolution 31

Figura 5.1 Campos identificados no documento 40
Figura 5.2 Funcionamento Validador 41
Figura 5.3 Topologia da Solução 42
Figura 5.4 Especificação TeslaV100 42
Figura 5.5 Algoritmo HoG em funcionamento 43
Figura 5.6 Resultado execução 43

Figura 6.1 Performance Rappid Fuzz 46
Figura 6.2 Funcionamento Request 47
Figura 6.3 Desvio Padrão campo Data de Nascimento 47

LISTA DE TABELAS

Tabela 6.1 Performance de processamento 46

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Contexto e Motivação 11
1.2 Software-as-a-Service 12
1.3 Multi-Access Edge Computing(MEC) 12
1.4 Off-loading Computacional 13
1.5 Edge Computing 13
1.6 Estudo de Caso 14
1.7 Definição do Problema 15
1.8 Processamento como SaaS 16
1.9 Organização do Trabalho 17
2 O ALGORITMO YOU ONLY LOOK ONCE (YOLO) 18
2.1 Redes Neurais Convolucionais 18
2.2 O Algoritmo You Only Look Once (YOLO) 20
Conceito de Funcionamento 21
Exemplo de funcionamento 21
O framework Darknet 24
Módulo Deep Neural Networks no OpenCV 25
3 ESTADO DA ARTE 27
3.1 Computação de borda e Offloading 27
3.2 Reconhecimento de imagens como serviço 28
4 PANORAMA APLICAÇÕES 5G 30
4.1 Panorama Aplicações 5G 30
Expansão da Tecnologia 5G 30
4.2 Tecnologia 5G e suas implicações 30
Aplicações 12 fatores *12 Factor Apps* 32
4.3 Contêineres e Nuvens sob demanda 35
5 ESTUDO DE CASO ESCOLHIDO 38
5.1 Contexto do Estudo de Caso 38
5.2 Preparação dos dados 38
5.3 A ferramenta Visual Checker 40
6 ANÁLISE DE RESULTADOS 45
6.1 Avaliação da performance 45
6.2 Avanços da Tecnologia 48
7 CONCLUSÃO E TRABALHOS FUTUROS 49
7.1 Conclusão 49
7.2 Desafios Encontrados 49
REFERÊNCIAS 51

1 INTRODUÇÃO

Aplicações de visão computacional têm tomado crescente centralidade no mercado de software como serviço (RESEARCH, 2022). A combinação do custo elevado da plataforma de execução com a complexidade de construção dos serviços de visão computacional, aliados à maior conectividade proporcionada pela tecnologia 5G permitiu o surgimento de uma nova classe de aplicações, que são desenvolvidas numa arquitetura de Software como serviço capaz de trazer o alto poder de computação das *Graphical Processing Units* (Placas de Processamento Gráficos) necessário para visão computacional para dispositivos e sistemas de menor poder computacional.

A adoção dessa metodologia tem como objetivos: simplificar o provimento de visão computacional como serviço por parte de aplicações, sobretudo em dispositivos restringidos em termos de poder computacional e/ou bateria; viabilizar economicamente a construção dessa classe de soluções aumentando a taxa de retorno da aplicação através do provimento de serviços como produtos de prateleira, com baixa customização.

1.1 Contexto e Motivação

Com o advento da pandemia de COVID-19 iniciada em 2019 (BONI et al., 2020), ocorre a aceleração de uma demanda já existente em escala nacional, no processo de extração e validação de dados documentais de forma automática. Essa funcionalidade é parte fundamental de processos como o acesso ao Auxílio Brasil (BRASIL,) e aderência às novas regulações de viagem em aeroportos (DIGITAL, 2022) entre outros.

Esse contexto fomentou o esforço de desenvolver uma solução de baixo custo e rápida prototipação, capaz de oferecer um conjunto de serviços e acelerar os processos de identificação e validação de dados através da checagem em bases digitais, também conhecido como *on-boarding* (AIRLINES, 2023). Nesse processo a extração de dados de documentos em formatos físicos para sua aferição ganhou centralidade. Usando visão computacional numa abordagem de serviços desenvolvemos uma API (*Adanced Programmers interface*) para executar essa tarefa de extração e validação de dados documentais.

Para fins de paradigma, somente o Auxilio Brasil atende a 20 milhões de pessoas, as quais estão geograficamente espalhadas pelo Brasil não tendo necessariamente perto de si uma agência governamental. O processo de cadastramento automático dessa população a qual acessa a internet prioritariamente, em termos de 70%, por celulares (MO-

BILETIME, 2021) é um grande desafio para um *Software-as-a-Service* para validação documental.

1.2 Software-as-a-Service

O uso de *Software-as-a-Service* (SaaS) oferexe aos clientes acesso aum software na forma de serviço um disponível na internet. O provedor SaaS executa as operações de desenvolvimento e manutenções do *software* multi-clientes. Os usuários pagam pelo componentes de software alugados e taxas de uso de serviços, que geralmente são mensais, trimestrais ou anuais (BUXMANN; HESS; LEHMANN, 2008).

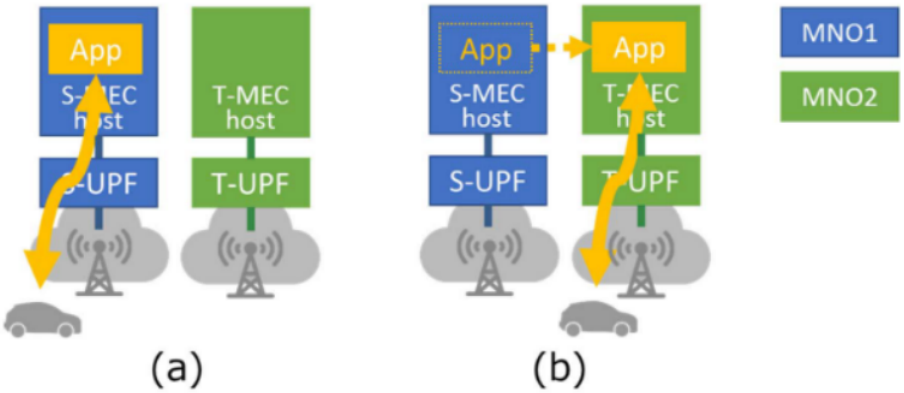
No caso específico de visão computacional como serviço o alto custo das placas de vídeo onde esses programas são executados é amortizado pelo provimento de uma solução comum para múltiplos clientes. A chegada da tecnologia 5G levou os tempos de resposta a níveis compatíveis com o processo de verificação dos dados de documentos em processos de on-boarding (LIYANAGE et al., 2021).

1.3 Multi-Access Edge Computing(MEC)

Multi-Access Edge Computing surge como um possibilitador chave para a disponibilização de aplicações *Internet of Things*(Internet das Coisas) IoT de tempo real (LIYANAGE et al., 2021). MEC trata-se de uma plataforma em nuvem de pequena escala que prove recursos computacionais e serviços de armazenamento na fronteira da rede 5G em proximidade com os dispositivos dos usuários (SUFYAN; BANERJEE, 2020). Tal tecnologia permite a criação de arquiteturas de *off-loading* computacional onde o processamento é transferido dos dispositivos de usuário, limitados em processamento e energia, para os recursos mais robustos da MEC.

De maneira mais específica, MEC provê mecanismos de migração das tarefas entre diferentes User Point Functions (pontos de acesso de usuário) conforme demonstrado na figura 1.1 (REPORT, 2020). Através do MEC é possível que tarefas sejam migradas entre diferentes pontos de acesso de forma transparente à aplicação do usuário, fornecendo uma experiência de cobertura contínua do fluxo da aplicação. Isso é um habilitador fundamental do processo de visão computacional visto que interrupções são críticas no processo de aquisição e processamento das imagens em tempo real.

Figura 1.1: User Point Function



(REPORT, 2020)

1.4 Off-loading Computacional

Off-loading Computacional é a técnica na qual um dispositivo móvel entrega parte ou a totalidade de uma tarefa para um ambiente de nuvem com recursos adequados. As principais razões para a utilização de tal técnica são a conservação de energia e a insuficiência de poder computacional do dispositivo original. Várias abordagens já foram constituídas para a resolução do problema de *off-loading* à citar: *Cyber foraging* permite *off-loading*, aumentando os recursos de dispositivos móveis, enquanto também provê maior eficiência energética(BALAN et al., 2002). CloneCloud(CHUN et al., 2011) efetua o particionamento automático de código no nível de *threads*.O particionamento no nível de componentes foi descrito por Cuckoo (KEMP et al., 2012).

Nos últimos anos, com a advento do *Multi-Access Edge Computing*(MEC), operadores moveram seus serviços dos centros de dados tradicionais para mais perto dos dispositivos de usuários. Nesse contexto os serviços são instalados na borda da rede 5G. Os recursos do MEC podem ser dinamicamente modificados e otimizados baseados no tipo de serviço e aplicação que estarão sendo ali executados. Isso cria um conjunto de novos modelos de negócio para provedores de serviço e usuários finais.

1.5 Edge Computing

A presença de dispositivos inteligentes consolidou-se em nossa sociedade, entre estes podemos citar: *smartphones*, automóveis,*smart watches* e *dash cams*. São capazes de monitorar as atividades do usuário, compartilhando e processando informação com

o auxílio de outros aparelhos. A execução desse monitoramente costuma valer-se da utilização de CNNs (*Convolutional Neural Networks*) (AKTER et al., 2021).

Contudo, CNNs apresentam grandes demandas energéticas para sua execução. Assim, acabam por ter atuação limitada no contexto de tais dispositivos. Uma alternativa cada vez mais explorada é *Edge Computing*.

Infraestruturas localizadas a um *hop* de distância dos dispositivos finais oferecem o posicionamento ideal para a arquitetura de *offload* em baixa latência capaz de suportar aplicações como realidade aumentada, segurança pública, carros autônomos e serviços de saúde (SATYANARAYANAN, 2017).

Tomando como base imagens de containers da plataforma Nvidia disponibilizamos uma API capaz de realizar o processamento de visão e segmentação computacional invocada por dispositivos de captura de imagem. É nesse contexto que contruímos nossa aplicação utilizando uma nuvem privada disponibilizada sob a plataforma Kubernetes(SHAH; DUBARIA, 2019).

1.6 Estudo de Caso

A expansão do 5G permite a construção de plataformas de *offloading* onde a computação é movida para dispositivos de grande performance e seus resultados devolvidos ao usuário final em dispositivos mais simples. Tal cenário é auxiliado pelas arquiteturas *Representational State Transfer* REST (FIELDING, 2000), ou seja, aquela que possui a seguintes características: arquitetura cliente-servidor, ausência de estado na aplicação, utilização de *caches*, codificação em camadas, código sob demanda e um conjunto uniforme de interfaces.

Definimos assim a divisão de tarefas entre cliente servidor em um cenário onde a aplicação trabalha apenas com um conjunto de dados local, não precisando guardar o estado ou valores de outras computações. Combinando o poder de processamento da nuvem, com contêineres provendo o encapsulamento necessário para disponibilizar um novo conjunto de aplicações. Nele, aproveitamos a idéia do tempo de transmissão de dados ser ínfimo perto ao processamento dos mesmos.

Nossa aplicação de escolha é um serviço de classificação de documentos no modelo SaaS (*Software as a Service*) (Software como Serviço). Os documentos são capturados por foto com celulares e suas informações são extraídas, validadas e retornadas como um serviço JavaScript Object Notation (JSON) (JSON.ORG, 2015). Esse procedimento

é feito em tempo real provendo celeridade ao processo de identificação de pessoas junto a sistemas como processos de *on-boarding* ou obtenção de benefícios em repartições públicas.

JSON é um formato aberto para troca de dados que utiliza texto legível para armazenar e transimir objetos compostos por pares chave valor (JSON. . . , 2017). Em nosso projeto o dispositivo do usuário envia a foto a qual é identificada e retornada junto a um JSON para o usuário, note-se que a interface do serviço pode ser a mesma para uma vasta gama de dispositivos como webcams, tablets ou celulares. Assim o custo desse sistema é amortizado entre os diversos usuários de uma API comum entre uma gama de aplicações.

Utilizamos o algoritmo You Only Look Once (Yolo) (REDMON; FARHADI, 2016a), para identificar objetos em imagens através da utilização de redes neurais. Esse algoritmo apresenta grande performance na detecção de objetos em tempo real com velocidades superiores a 60 *frames* por segundo. Contudo CNNs possuem altos requisitos para viabilizar uma performance adequada. CNNs e DNNs (*Deep Neural Networks*)(Redes Neurais Profundas) utilizam uma grande capacidade de processamento e armazenamento o que dificulta sua migração para as dispositivo da nuvem(CHANDAKKAR et al., 2017). Dessa forma os avanços na tecnologia 5G suscitam a possibilidade de efetuar a migração de parte do esforço computacional dos dispositivos móveis para a nuvem(CHANDAKKAR et al., 2017).

1.7 Definição do Problema

O reconhecimento de imagens é um campo de estudo em expansão. Com o advento da tecnologia 5G desenvolvedores são capazes de mover os esforços e custos computacionais para a nuvem. Embora essa tendência tenha entregue resultados robustos nos últimos anos, os custos com energia nos centros de dados começaram a impactar sua adoção. Estudos mostram que o consumo energético dos aumentou 62% de 2005 até 2013, e espera-se que aumente 220% até 2020(ALSBATIN; OZ; ULUSOY, 2017).

Concomitantemente, as técnicas baseadas em Redes Neurais Convolucionais como YOLO(REDMON; FARHADI, 2016b) requerem ao menos uma ordem de magnitude a mais de poder computacional do que encontra-se disponível em dispositivos móveis(ZHU et al., 2018). Desta forma, projetamos uma arquitetura visando diminuir os custos computacionais tanto no *front-end* quanto *back-end* da aplicação. No front-end busca-se a simplificação da lógica de processamento de visão computacional, no back-end foca-se

no reúso da infraestrutura entre as chamadas da API. Nossa metodologia é baseada na premissa em efetuar o *offloading* computacional para obtenção de ganhos energéticos nos dispositivos e na *edge*.

Exploramos as tradicionais restrições de performance dos dispositivos através da utilização de uma nuvem dinâmica. Os ganhos energéticos dão-se em três frentes: a computação pesada é movida para o *Edge*, a conteinerização nos permite a criação de instâncias de execução sob demanda em máquinas *stateless*, sem estado, as quais não gardam nenhuma informação entre as diferentes execuções a cada chamada, economizando memória e processamento. Uma instancia *stateless* significa que suas execuções podem ser reutilizadas sem a necessidade de salvar ou carregar valores em memória. Os parâmetros de execução são enviados junto a requisição de processamento(FOWLER; LEWIS, 2014).

Expandimos tais idéias de forma a explorar o consumo energético de uma aplicação de visão computacional no contexto de dispositivos móveis.

Detecção de objetos é um campo relacionado com visão computacional e processamento de imagem que lida com a busca por instâncias de objetos de uma determinada classe (como humanos, edifícios ou carros), em imagens e vídeos digitais. Possíveis domínios de aplicações para a detecção de objetos incluem identificação de rostos, detecção de pedestres, segmentação de documentos e sistemas de segurança (DASIOPOULOU et al., 2005).

Nesse contexto, a chegada da tecnologia 5G, capaz de promover o trânsito de dados em altíssima velocidade, na ordem de 10 gigabites por segundo, ou seja, dez vezes mais rápida que o atual 4G, cria o cenário ideal para a emergência de uma nova família de aplicações em nuvem. Com a grande banda de dados disponiveis pode-se realizar a ponte entre os dispositivos móveis de usuários, limitados por suas baterias, e o poder computacional da nuvem.

1.8 Processamento como SaaS

O desenvolviemento de soluções de visão computacional como serviço é uma possibilidade recente no cenário da computação. Abaixo listamos alguns dos principais pontos dessa evolução: 2017 - Apple disponibiliza o IphoneX com reconhecimento facial (COMPUTER. . . , 2020).

2018 - O agente de Inteligência Artificial da Alibaba performa melhor que huma-

nos num teste de leitura e compreensão realizado pela Universidade de Standford (COMPUTER. . . , 2020).

2018 - Amazon vende seu primeiro sistema de reconhecimento de faces em tempo real Rekognition para departamentos de polícia (COMPUTER. . . , 2020).

2018 - A polícia alemã apresenta um aplicatívo móvel para a verificação de documentos em tempo real que pode ser usado em celulares(HTTPS. . . , 2018)

2019 - O governo Indiano anuncia um plano de oferecer aos policiais a capacidade de reconhecimento facial através de um aplicativo móvel (COMPUTER. . . , 2020).

Esses projetos foram inspiração para a solução proposta ao longo deste trabalho.

1.9 Organização do Trabalho

Esta dissertação está organizada da seguinte forma: No Capítulo 2 é feita uma revisão do estado da arte. O Capítulo 3 busca explorar a utilização do algoritmo Yolo(REDMON; FARHADI, 2016a) em aplicações reais de detecção de objetos. No capítulo 4 avaliamos a performance das técnicas combinadas na construção do sistema. O Capítulo 5 sugere uma arquitetura responsável pela realização do reconhecimento de imagens como um serviço.

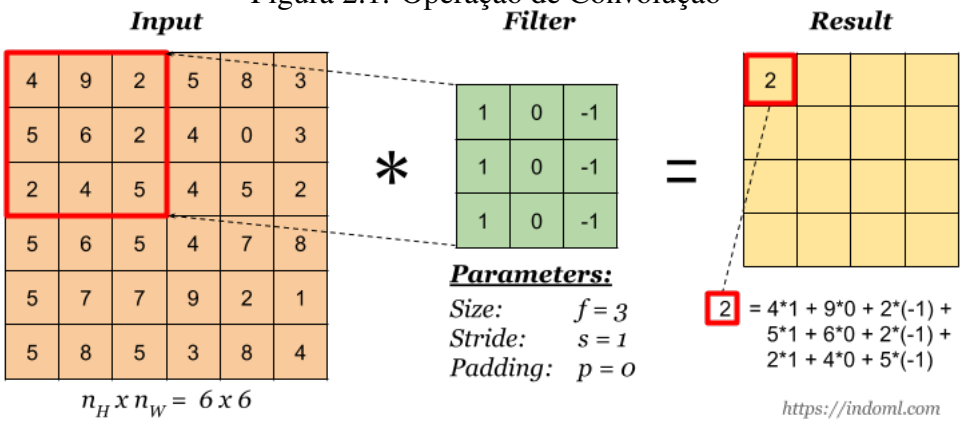
2 O ALGORITMO YOU ONLY LOOK ONCE (YOLO)

2.1 Redes Neurais Convolucionais

Redes Neurais Convolucionais foram propostas para resolver problemas de visão pela primeira vez na identificação de dígitos (LECUN et al., 1995). Desde então, CNNs tornaram-se técnica importante no campo de visão computacional em tarefas como reconhecimento de faces e objetos. Entre as características presentes nas CNNs, que proporcionam o sucesso de sua aplicação, estão: CNNs podem tratar um conjunto maior de dados no treinamento devido a distribuição dos pesos nas camadas convolucionais, os modelos são mais facilmente ajustados para evitar *over-fitting* (BOUREAU; PONCE; LECUN, 2010), a criação e disponibilização de grandes *datasets* como a Imagenet (DENG et al., 2009).

Matematicamente, uma convolução é uma operação linear que a partir de duas funções, gera uma terceira (normalmente chamada de *feature map*). No contexto de imagens, podemos entender esse processo como um filtro/*kernel* que transforma uma imagem de entrada (SMITH, 1997). Um *kernel* é uma matriz utilizada para uma operação de multiplicação entre matrizes. Esta operação é aplicada diversas vezes em diferentes regiões da imagem original.

Figura 2.1: Operação de Convolução

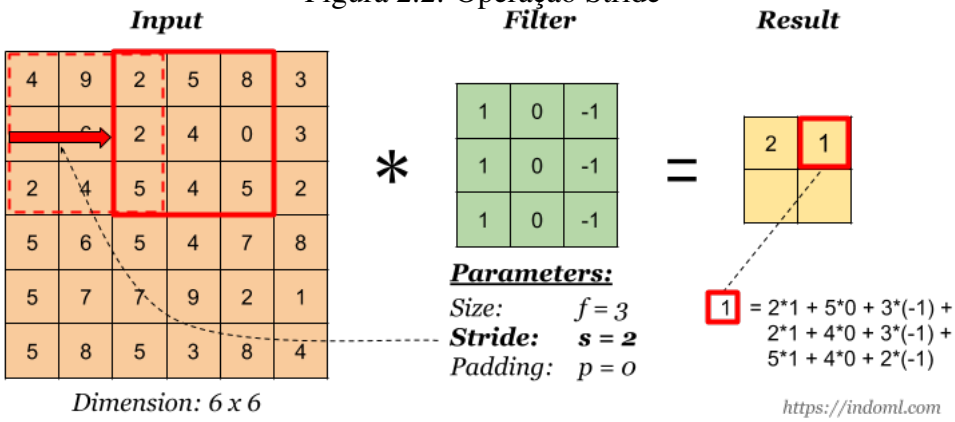


(BELAJAR. . . , 2020)

Um exemplo de operação de convolução pode ser observado na Figura 2.1. A aplicação do filtro (matriz) de convolução sobre a matriz base gera uma matriz resultado com o valor 2 correspondente ao produto entre a matriz e o filtro. A cada aplicação, a região é alterada por um parâmetro conhecido como *stride*. A aplicação da operação de *stride* pode ser visualizada na Figura 2.2.

Uma rede neural convolucional efetua a computação de inferências e extração de características sobre um determinado conjunto de dados através da aplicação de um conjunto de filtros convolucionais. Esses filtros são seguidos por filtros treinados para efetuar a classificação em si, no caso filtros totalmente conectados, responsáveis pela geração do vetor de classificação. Tal fluxo pode ser observado na Figura 2.3 a imagem original recebe a aplicação do filtro.

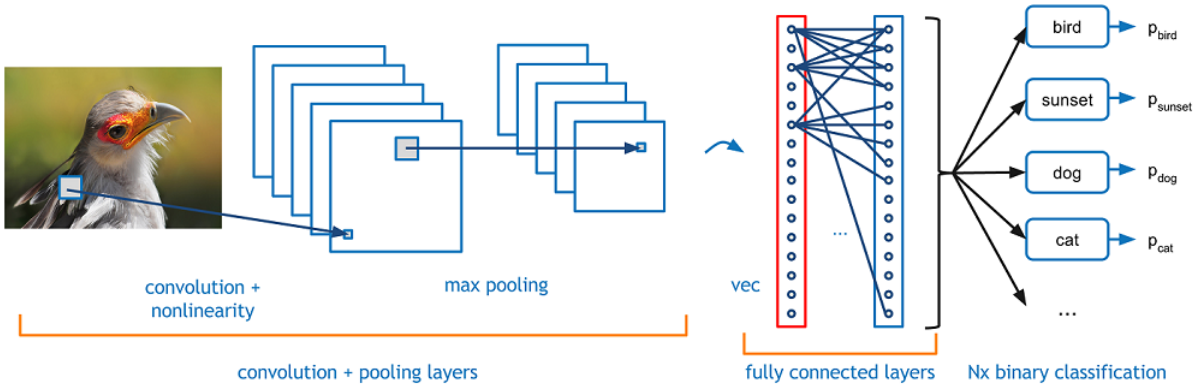
Figura 2.2: Operação Stride



(BELAJAR. . . , 2020)

Nela a operação de convolução sobre a imagem da ave gera uma nova matriz através da operação de *maxpooling* . Maxpooling é um processo de discretização por amostragem cujo objetivo é reduzir o espaço de representação de uma imagem de entrada, mantendo suas principais características na sua imagem transformada. Tal procedimento tem como resultado um menor custo computacional através da diminuição de parâmetros considerados para a operação.

Figura 2.3: Filtro Convolucional
(Adit Deshpande , 2019)

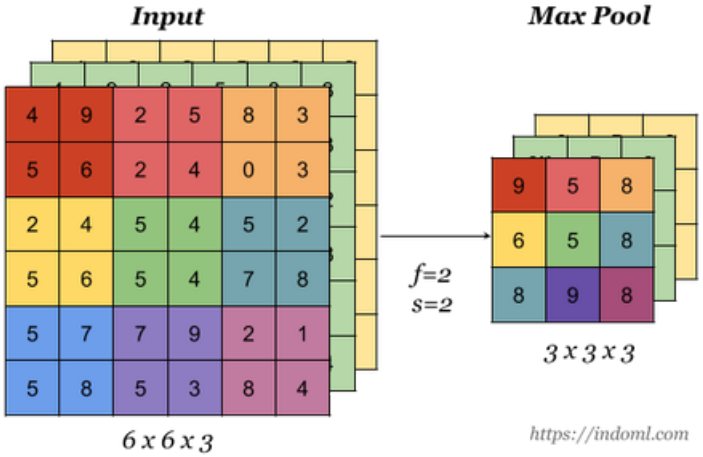


(BELAJAR. . . , 2020)

A figura 2.4 exemplifica a execução da operação de *maxpooling*. A aplicação de um *kernel* gera uma nova matriz cujos maiores valores são selecionados para compor a

matriz que será entrada do vetor de classificação. O vetor de classificação é aquele cujos valores foram extraídos de uma imagem original já classificada e servem como base do processo de inferência da rede neural.

Figura 2.4: Operação de Maxpooling



2.2 O Algoritmo You Only Look Once (YOLO)

O Algoritmo *You Only Look Once* (Redmon et al., 2016), criado por Joseph Redmond, trata-se de uma FCNN (*Fully Convolutional Neural Network*) processando uma imagem (n x n) e produzindo uma predição (m x m). Ou seja, o resultado da deteção é um recorte maior do que a imagem original. Isso difere dos trabalhos anteriores como fast RCNN (GIRSHICK, 2015), que efetuam a detecção em várias regiões candidatas analizando múltiplas predições para várias regiões diferentes da imagem.

Dessa forma é efetuada uma troca entre a precisão da localização do objeto e a velocidade do processo de inferência. Os quadrantes de detecção dos objetos são gerados levemente deslocados, sem que estes estejam em seu centro. Isso torna o algoritmo um dos mais rápidos detectores hoje disponíveis, tendo capacidade de efetuar detecção em vídeos em tempo real.

Adicionalmente, o algoritmo trabalha sob uma representação generalizada dos objetos. A detecção não se da por meio de um matching total entre os objetos comparados com a classe de detecção mas sim pela validação de características dos mesmos extraídas pela rede neural. Ou seja, quando comparado a outros métodos como DPM e R-CNN sua capacidade de detecção mostra-se superior. Dada sua capacidade de generalização o algoritmo é menos propenso a falhas quando aplicado a novos domínios ou dados inesperados (Redmon et al., 2016).

A combinação dessas características permite sua utilização no reconhecimento de imagens em fluxos de vídeo em tempo real. Isso é possível visto que com o hardware atual o algoritmo é capaz de atingir uma performance acima de 25 *frames-per-second*(frames por segundo). Combinado com a tecnologia 5G essa capacidade é habilitadora de uma nova maneira de construir aplicações.

Conceito de Funcionamento

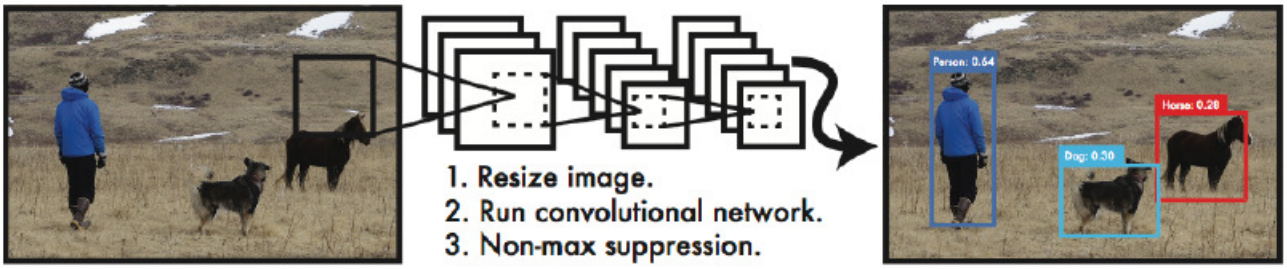
O algoritmo YOLO (You Only Look Once) (REDMON; FARHADI, 2016a) consolidou-se como um padrão para o reconhecimento de objetos em vídeos. As suas possíveis aplicações estendem-se do reconhecimento imagens para pessoas deficientes visuais através de dispositivos móveis como celulares ou tablets até a análise de grandes massas de dados na nuvem, através de tecnologias de virtualização como Docker (DOCKER, 2017) passando por dispositivos IoT(Internet of Things) como Raspbery Pi Raspbery Pi (JAIN; VAIBHAV; GOYAL, 2014).

O carater inovador da YOLO deriva de abordar do problema de reconhecimento de padrões de imagem através de regressão linear aplicada a *bounding boxes* e uma associação a classes de probabilidade e não como o resultado da análise de classificadores clássicos. Uma única rede neural executa a predição das *bounding boxes* e suas classes de probabilidade associadas. Segundo (REDMON; FARHADI, 2016a) o algoritmo YOLO tem como característica apresentar mais erros de localização mas diminui massivamente os falsos positivos onde os objetos não estão efetivamente presentes. Superando todos os outros métodos de detecção como DPM *Deformable Part Models* (FELZENSZWALB et al., 2010) e RCNN *(Regions with Convolutional Neural Network)* (GIRSHICK et al., 2013).

Exemplo de funcionamento

O algoritmo propõe o tratamento da detecção de imagens através de um único problema de regressão. Isso significa que o conjunto de dados é avaliado uma única vez pela rede de neurônios. Cada imagem é associada as coordenadas de uma *bounding box* e uma classe de probabilidades, a seguir as *bounding boxes* são processadas pela rede neural e seu conteúdo recebe um *score*.

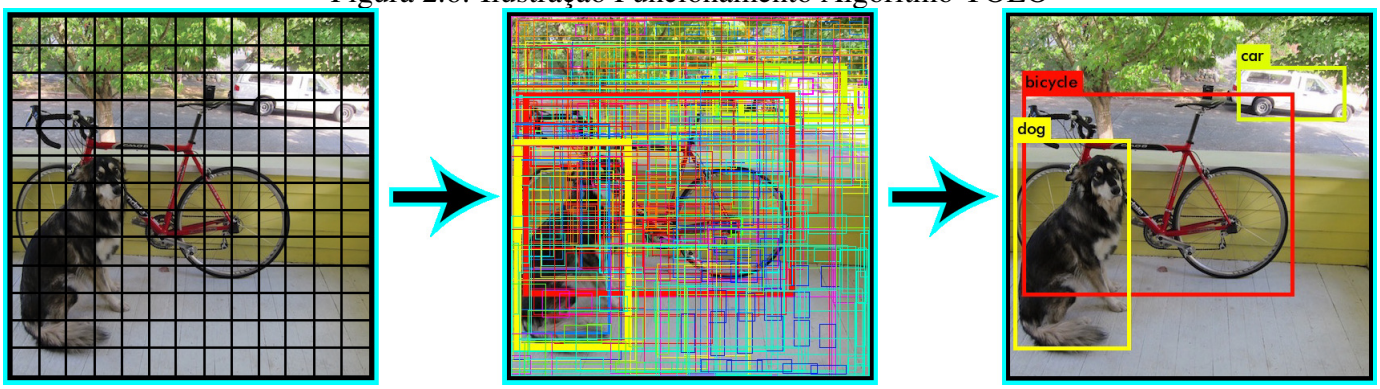
Figura 2.5: Ilustração Conceito Algoritmo YOLO



Fonte: (REDMOND,)

Na figura 2.6 vemos a representação passo a passo do algoritmo em uma foto com 3 objetos presentes nos modelos de detecção. No primeiro quadro é feita a divisão da imagem em 169 *bounding boxes*. Após isso a rede neural encontra uma série de quadrantes candidatos a detecção positiva. Por fim um filtro de limiar compara as possíveis detecções com os modelos de objetos conhecidos.

Figura 2.6: Ilustração Funcionamento Algoritmo YOLO



Fonte: (REDMOND,)

O algoritmo YOLO prediz uma probabilidade para cada quadrante da imagem particionada, independente do número total de partições. Em tempo de execução é calculada uma função de confiança da detecção:

$$Pr(Classe_i|Objeto) * Pr(Objeto) * IOU^{\text{truth}}_{\text{pred}} = Pr(Classe_i) * \text{IOU}^{\text{truth}}_{\text{pred}}$$

Que define a probabilidade de um objeto pertencer a classe i e estar dentro de um IOU.

IOU (*Intersection over Union*) também chamado de *Jaccard Index* (JACCARD, 1901) é uma métrica estatística usada para comparar a similaridade e diversidade de *da-*

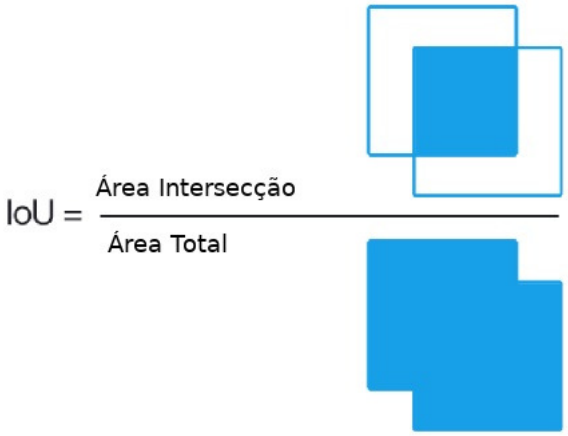
tasets finitos. Matematicamente definido por:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

representando o tamanho da intersecção dividida pela união de dois conjuntos de amostras. Onde se A e B forem ambos vazios definimos $J(A,B) = 1$ e $0 \leq J(A|B) \leq 1$

Seu Funcionamento pode ser entendido observando a figura 2.7

Figura 2.7: Intersection Over Union (IOU)



IoU

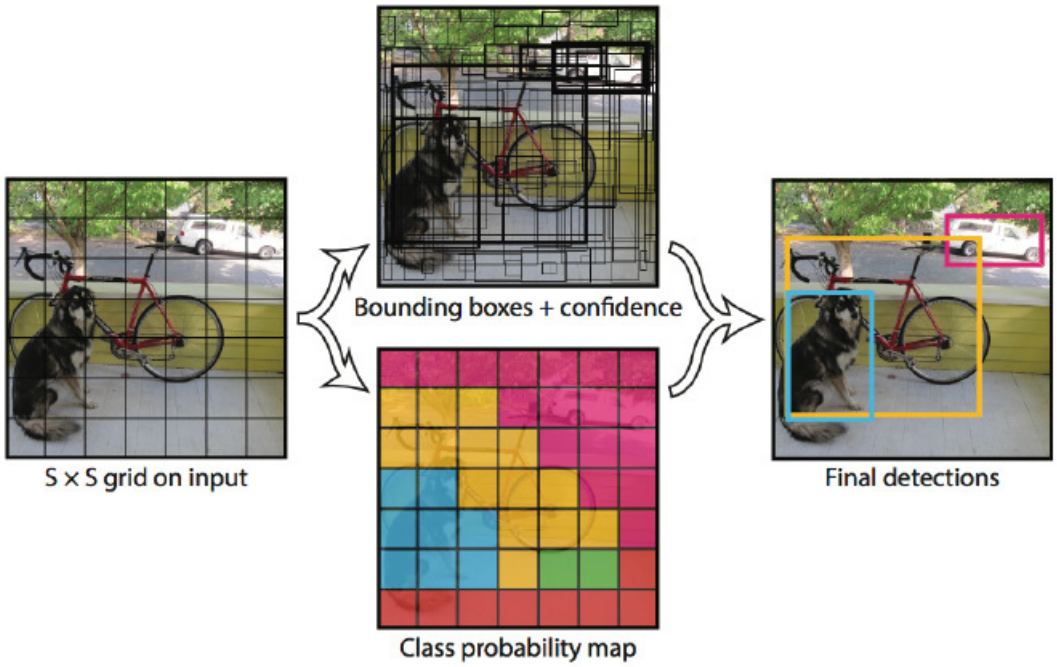
Ou seja a intersecção da área em comum sob a área total.

Conforme pode ser visto em 2.8 os quadrantes azul, laranja e rosa representam respectivamente o batimento dos modelos dos objetos cão, bicicleta e carro com seus respectivos modelos e classes de confiança.

O processamento é efetuado dentro de uma rede convolucional apresentada na figura 2.9. Sua composição é baseada na rede Googlenet (SZEGEDY et al., 2015).

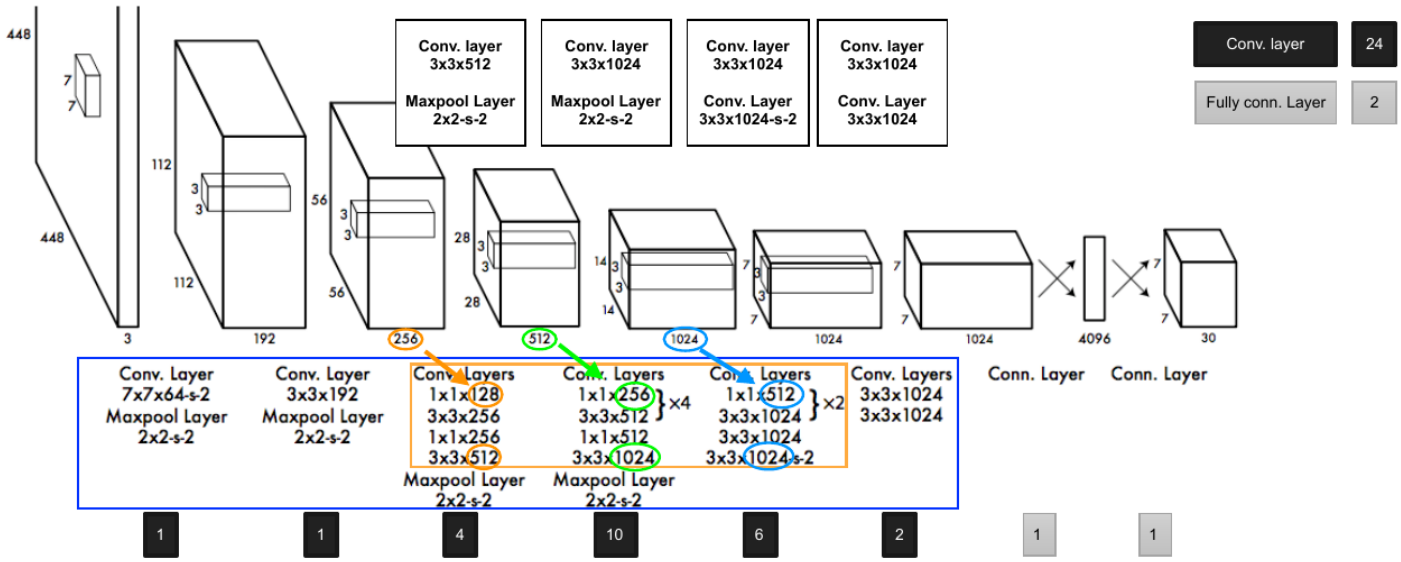
No total temos 24 camadas convolucionais seguidas de duas camadas totalmente conectadas. As camadas são divididas quanto a sua funcionalidade da seguinte maneira; As primeiras 20 camadas convolucionais são formadas por uma camada de média do *pooling* e uma camada totalmente conectada pré-treinada com o *dataset* ImageNet 1000-class. O pré-treinamento para classificação é feito no dataset com uma resolução de 224 x 224. As camadas são compostas de camadas 1 x 1 de redução e camadas 3 x 3 de convolução. As últimas 4 camadas convolucionais são seguidas por 2 camadas totalmente conectadas e são adicionadas para treinar a rede para a detecção de objetos. Como a detecção de objetos requer uma definição maior treinamento é feito com um dataset augmentado para 448 x 448. A camada final prediz as probabilidades de classes e as *bounding boxes*.

Figura 2.8: Quadrantes Identificados pelo Algoritmo



Fonte: (REDMOND,)

Figura 2.9: Rede Neural YOLO



Fonte: (SZEGEDY et al., 2015)

O framework Darknet

Darknet é um *framework* de código aberto para a execução de redes neurais escrito em C e CUDA. Sua implementação suporta modelos computacionais em CPU e GPU (REDMON, 2013–2016). A rede tem como parâmetros um arquivo que descreve a rede

e a disposição das camadas ReLU, convolucionais e de maxpooling, um arquivo com o nome dos objetos do vetor de classificação e um conjunto de pesos.

Essa arquitetura simples e configurável simplifica o processo de treinamento e customização da rede. Uma vez compilado o programa permite a execução do processo de inferência e treinamento através de um conjunto de parâmetros. Durante a realização da dissertação a execução do modo de inferência da rede tornou-se possível através de chamadas nativas da linguagem python.

Isso facilita a aplicação da técnica em aplicações reais voltadas para a arquitetura de *backend* em web-services. Validando as premissas levantadas na concepção desse trabalho.

Módulo Deep Neural Networks no OpenCV

OpenCV (Open Computer Vision) é uma biblioteca lançada originalmente em 1999 (KAEHLER; BRADSKI, 2016) pela INTEL com o objetivo de prover suporte para bibliotecas e aplicações de visão computacional em tempo real. Desde 2010 a plataforma suporta mecanismos de aceleração por *hardware* através da plataforma CUDA. Em 2018 os primeiros módulos suportando a execução do código nativo da rede YOLO foram adicionados ao projeto por Alexey Bochkovskiy (BOCHKOVSKIY, 2017).

A evolução da inclusão deste módulo é peça fundamental da transição da implementação do algoritmo YOLO de um sistema monolítico para uma nuvem de serviços. A inclusão deste módulo permitiu que a execução da rede se de toda dentro do ambiente da aplicação, possibilitando descartar a necessidade de uma chamada ao binário nativo. Do ponto de vista da criação de um serviço isso possui uma série de desdobramentos à citar:

- o conjunto de bibliotecas que precisa ser incluído na aplicação passa a não necessitar do *backend* originalmente escrito em C da rede Darknet. Do ponto de vista do provimento de serviços como contêineres a diminuição do tamanho total da aplicação e seu respectivo consumo de memória tem impacto direto nos gastos energéticos e de infraestrutura.
- o processo de construção do serviço passa a ocorrer todo em tempo de execução não necessitando executar um processo de compilação do binário original. Em infraestruturas em nuvem tal possibilidade simplifica os processos de teste e *deploy* da aplicação.

- a não necessidade de geração de código em tempo de projeto faz com que a plataforma final de execução possa ser abstraída.

Durante o processo inicial da construção desse trabalho esse módulo encontrava-se em testes e fora do ramo estável da biblioteca OpenCV. Sua migração para o ramo estável dá-se por volta do segundo semestre de 2020 (OPENCV, 2020), nesse momento foi possível a reescrita de parte da aplicação tornando-a aderente ao modelo 12 fatores. Além do ganho quanto a simplificação do tratamento das dependência há uma diminuição de cerca de 10% do tempo de execução pela remoção da necessidade da execução de um binário fora do fluxo da aplicação.

3 ESTADO DA ARTE

3.1 Computação de borda e Offloading

A computação de borda é uma nova arquitetura de rede e padrões abertos que convergem as tecnologias de rede para processamento e armazenamento em alta velocidade (W. et al., 2017). Em uma arquitetura de nuvem, a maior parte do processamento ocorre em um contexto remoto, expondo os dados e requisições à latência que pode vir a impactar a experiência de usuário(Fan et al., 2018). Já no processamento em borda, com a proximidade dos pontos de acesso do usário não há necessidade de retornar dados à nuvem, assim reduzem-se os tempos de espera e os custos de rede envolvidos no processamento dos dados(LV; CHEN; WANG, 2018).

Conforme a computação move as tarefas de maior processamento para os servidores de borda mais próximos, é reduzida a congestão de rede, diminuído o atraso, provida uma melhor utilização da banda de rede e eficiência do processamento dos dados (W. et al., 2017). O problema de um balanço entre energia e performance em ambientes de uma nuvem móvel já foi discutido (KOSTA et al., 2012). Os resultados desse trabalho aliados à velocidade do 5G sugerem os ganhos ao mover essas aplicações para a *edge*.

A computação em borda se refere a tecnologias que permitem que o processamento seja executado junto à origem dos dados (Sahni et al., 2017). Com o advento da comunicação em 5G carros, telefones celulares entre outros dispositivos serão capazes de tomar vantagem do poder de processamento da nuvem, enquanto provêm responsividade em tempo real para seus usuários. Sendo a maioria dos dispositivos IoT *(*Internet of Things), como celulares e sensores, limitados energeticamente e a transmissão de dados ter tornado-se mais eficiente em termos de energia a ideia de mover (*offloading*) o processamento se torna interessante (Fan et al., 2018).

Essa metodologia combina uma grande banda de rede com conceitos de *big data* e nuvem, para possibilitar a implantação de aplicações de terceiros na fronteira da rede. Já foi anteriormente provado que o processamento de redes neurais convolucionais é ordens de magnitude mais custoso energeticamente que a transmissão dos dados a serem analisados (W. et al., 2017). Ainda assim o custo energético de tais aplicações se mostra inviável para o os atuais dispositivos móveis. O *offloading* é uma tecnologia chave para possibilitar tais tarefas computacionais. Ele pode ser definido como a prática de enviar trechos de computação intensiva das aplicações para um servidor remoto (AKHERFI;

GERNDT; HARROUD, 2018). Conforme a rede 5G se expandir a parcela de computacão em *offloading* atingirá níveis sem precedentes (SHAH; DUBARIA, 2019).

A quantidade de tráfego em *offloading* na tecnologia 4G era de 57% no final de 2017, e deve atingir 59% em 2022. O percentual em 3G será 40% em 2022, e o total em 2G será 30% (SHAH; DUBARIA, 2019). Conforme 5G é introduzido, onde espera-se planos de dados com velocidades e limites de tráfego generosos, as novas aplicações utilizarão o 5G acelerando a tendência de *offloading* já presente no 4G.

O percentual de *offloading* em 5G está estimado para atingir 71% em 2021(BARNETT et al., 2015), assim uma nova metodologia de executar essas computações remotas é necessária. Concomitantemente, o modelo tradicional de servidores estaticamente alocados perde em termos de performance para a abordagem *edge* especialmente devido a contenção de rede. A combinação desses fatores aliada a ascensão de aplicações de visão computacional em tempo real sugere a necessidade de novas arquiteturas de serviço.

3.2 Reconhecimento de imagens como serviço

É sabido que algoritmos para o processamento de imagem são capazes de trabalhar com um conjunto limitado de documentos como cheques bancários utilizando metodologias multi-camada (GERDES; OTTERBACH; KAMMÜLLER, 1995). Contudo, há uma falta de padronização dos métodos de detecção visando múltiplas classes de objetos em documentos scaneados. Isso se origina da vasta variabilidade na forma de ocorrência de tais objetos (FORCZMAńSKI et al., 2020).

O nosso estudo de caso tem sido objeto de estudo na literatura por mais de três décadas, geralmente sendo chamado de segmentação de páginas e classificação de zonas (OKUN; DOERMANN; PIETIKAINEN, 1999). Abordagens mais recentes começam a tratar esse problema como uma tarefa de Visão Computacional com a utilização de redes neurais convolucionais (FORCZMAńSKI et al., 2020). Nós expandimos esse conceito transformando essa classe de aplicações em um serviço compatível com o cenário de computação de borda. Paralelamente, estudos anteriores já estabeleceram a performance das Unidades de processamento gráfico *Graphical Processing Units (GPUs)* como sendo até duas ordens de magnitude superior a suas respectivas implementações em CPU (Strigl; Kofler; Podlipnig, 2010)(Uetz; Behnke, 2009). É sabido também que a complexidade computacional das camadas de convolução advém de três fatores principais: primeiramente a operação de convolução, o tamanho reduzido dos núcleos da operação de ReLU

*(*Rectified Linear Unit) (**??**) e, por fim, dos acessos não seriais a memória. Nas operações convolucionais, o número de operações por entrada cresce quadraticamente conforme aumenta-se o tamanho do núcleo. Tomando um exemplo prático, um *kernel* 7x7 gera o dobro de operações por pixel de entrada do que um *kernel* 5x5 (ZHANG et al., 2019).

Tais características dificultam a utilização de Redes Neurais em dispositivos móveis, uma alternativa possível é a migração do esforço computacional para uma estrutura em nuvem. As tecnologias que sustentam essa abordagem como Docker e Kubernetes (SHAH; DUBARIA, 2019) são relativamente novas tendo seu lançamento em 2013. Sendo assim a combinação destas técnicas está apenas no início de sua exploração sendo a definição de sua arquitetura um problema em aberto.

A popularização dessas tecnologias tornou possível prover esse processamento em instâncias alocadas na nuvem. Tais instâncias, chamadas contêineres, ampliam a utilização dos recursos computacionais por dois aspectos: primeiramente, como Máquinas Virtuais, múltiplos contêineres podem ser abrigados em um único servidor aumentando o uso de recursos; adicionalmente, contêineres são muito mais eficientes no acesso a memória e dispositivos de entrada e saída, ao mesmo tempo que ainda provém bom isolamento entre diferentes aplicações (CHAE; LEE; LEE, 2019). Assim os custos computacionais podem ser diluídos em um ambiente de processamento compartilhado permitindo que os dispositivos de borda funcionem como clientes que acionam a criação da nuvem.

Os benefícios do particionamento de aplicações DNN *Deep Neural Networks* foi investigado por Kang (KANG et al., 2017) nos contextos de tecnologia 3G,4G e wifi. Os resultados mostram que, para alguns modelos de DNN, o particionamento pode trazer benefícios quanto à latência e consumo de energia, enquanto outros são impactado pela latência da transmissão de dados gerados nas camadas intermediárias da rede. Exploramos essa idéia para prover um conjunto de APIs capazes de atender requisições de dispositivos móveis.

4 PANORAMA APLICAÇÕES 5G

4.1 Panorama Aplicações 5G

Expansão da Tecnologia 5G

A tecnologia 5G, em processo de implementação, expande a capacidade de comunicação heterogênea. Possibilitando a integração entre ativos de rede e elementos de processamento com serviços disponíveis na nuvem, através de técnicas de virtualização e implementação. Nesta nova nuvem orientada a serviços as arquiteturas de *edge-cloud* irão prover o acesso uma maior performance também chamada de *Quality of Service* (QoS), enquanto a virtualização proverá o suporte a arquiteturas multi-serviço e multi-cliente, promovendo assim um provisionamento efetivo de recursos de computação e rede (TALEB et al., 2017).

Podemos tomar por exemplo a criação do *Multi-access Edge Computing group (MEC ISG)* com o objetivo de padronizar e acelerar o avanço da computação em *edge-clouds* nas redes mobile através do lançamento do *MEC Industry Specification Group* em 2014. Com o objetivo de criar um ambiente multi provedores de plataformas em nuvem, no nível da Rede de Acesso a Rádio, acessível a aplicações e provedores de serviço num esforço de superar os desafios da computação em rede distribuída nos requisitos de latência e provimento de altas velocidades.

Esse objetivo é alcançado através da delegação de tarefas computacionalmente custosas para a fronteira da nuvem. Efetuando sua computação próxima aos usuários, desta forma, a rede diminui os efeitos de congestionamento diminuindo a necessidade de modernização, concomitantemente, efetua a computação mais significante desonerando os equipamentos de usuários, muitas vezes limitados por questões energéticas (HU et al., 2015).

4.2 Tecnologia 5G e suas implicações

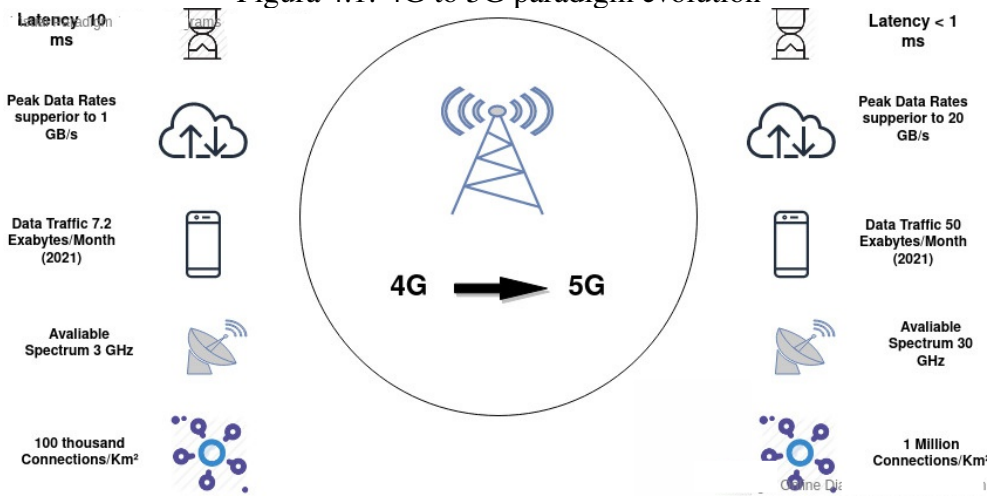
É chamada de 5G a infraestrutura de rede com baixa latência ($<$10ms), alta velocidade ($>$1Gbps), grande capacidade (aumento de mil vezes) e conectividade ampliada (mais de cem bilhões de conexões). Essa conjuntura possibilitará novos patamares de digitalização com a Internet das Coisas, carros, dispositivos de uso pessoal, to-

dos comunicando-se com um grande volume de dos no nível de aplicação (SARFARAZ; HäMMäINEN, 2017). Fluxos de vídeo gerados em dispositivos móveis vão gerar mais de três quartos do volume de tráfego até 2021 (CISCO. . . , 2017).

A arquitetura 5G dá acesso a uma tecnologia chave para a execução de *offloading*, chamada *Multi-acess Edge Computing*(MEC), que se refere ao conceito de delegar capacidades de computação em nuvem e um serviço de operação de dados na fronteira da rede celular. A idéia básica no conceito de MEC é que uma vez que as computações sejam trazidas para mais próximo dos clientes a latência média e o congestionamento de rede são diminuídos. Espera-se que MEC seja implementada nas estações rádio base da rede celular.

A tecnologia MEC aplicada em computação de borda permite que aplicações sejam executadas em um ambiente virtualizado na forma de uma instância de software, e a plataforma de infraestrutura a ser alocada na fronteira da rede, próxima aos clientes (LV; CHEN; WANG, 2018). A futura arquitetura da rede 5G promoverá computação em *edge*, borda em português. O 5G irá possibilitar uma arquitetura orientada a serviços, onde os módulos de protocolos podem ser invocados de forma flexível conforme os requisitos do negócio, provendo um padrão tecnológico para a construção da rede *Edge* (W. et al., 2017).

Figura 4.1: 4G to 5G paradigm evolution



Fonte:(QORVO, 2020)

A mudança de paradigma trazida pela tecnologia 5G pode ser melhor vista na figura 4.1 adaptada de (QORVO, 2020). Conforme a latência diminuiu mais de dez vezes de 10 ms para menos de 1 ms. As taxas máximas de transmissão cresceram mais de 20 vezes de 1 Gigabyte por segundo para mais de 20 Gigabytes por segundo enquanto o tráfego de dados cresceu de 7.2 para 50 Exabytes por mês.

Concomitantemente, o poder de processamento médio dos aparelhos de celular não chegou a dobrar e suas autonomias de bateria tiveram uma melhora inferior a 50%. Por exemplo,o aparelho Samsung Galaxy S6 (ANALYSIS; MANUAL, 2012) possui uma bateria de 2550mAh e um processador de 1.7Ghz, enquanto o três gerações mais moderno Galaxy S10 tem uma bateria de 3100mAh e um processador de 2.4 GHz (SAMSUNG, 2019).

Observa-se assim que a latência vem caindo em ordens de magnitude superiores ao crescimento do poder de processamento. Paralelamente, aplicações de visão computacional demandam crescente capacidade de processamento e apresentam maior consumo energético, o algoritmo Yolo mesmo para atingir excussão em tempo real gera apenas alguns *frames* por Watt (DING et al., 2019). Assim, *offloading* surge como uma solução para a performance extra exigida por essas aplicações que rodam nos clientes da rede.

Aplicações 12 fatores *12 Factor Apps*

Na era moderna, software é comumente entregue como um serviço: denominados web apps, ou software-como-serviço. A aplicação doze-fatores é uma metodologia para construir softwares-como-serviço (WURSTER et al., 2017) que:

- Usam formatos declarativos para automatizar a configuração inicial, minimizar tempo e custo para novos desenvolvedores participarem do projeto;
- Tem um contrato claro com o sistema operacional que o suporta, oferecendo portabilidade máxima entre ambientes que o executem;
- São adequados para implantação em modernas plataformas em nuvem, evitando a necessidade por servidores e administração do sistema;
- Minimizam a divergência entre desenvolvimento e produção, permitindo a implantação contínua para máxima agilidade; E podem escalar sem significativas mudanças em ferramentas, arquiteturas, ou práticas de desenvolvimento.

Esses objetivos são atingidos através da observância das seguintes práticas:

1. Codebase: O primeiro passo é colocar todos o código em um sistema de controle, como Git ou Mercurial. Geralmente só existe uma base de código e ela é compartilhada com vários repositórios. Ou seja, é basicamente um local para produção e vários para testes, todo desenvolvedor tem uma cópia da base em seu ambiente

local, onde saem vários deploys. Mesmo que o ambiente local tenha códigos que não tem em produção e vice-versa, não há problema, pois todos compartilham da mesma base de código, portanto não haverá conflitos ao passar os desenvolvimentos de local para produção.

2. Dependencies: Em diversas aplicações as bibliotecas são adicionadas diretamente no diretório, porém o 12 Factor não permite que isso ocorra, ele declara todas as dependências de acordo com o manifesto de declaração de dependência. Independente das ferramentas, as dependências devem ser declaradas e isoladas, fazendo com que a configuração da aplicação para futuros desenvolvedores fique mais fácil. Portanto, todas as dependências são isoladas, fazendo com que não haja vazamentos pelo sistema, deixando a aplicação mais segura e confiável
3. Config: Configurações podem variar entre ambientes diferentes, a ideia é armazenar as configurações no ambiente, ou seja, não colocar configurações no código-fonte. O 12 Factor armazena todas as configuração em variáveis de ambiente. Fazendo com que se possa alternar facilmente entre implementações sem precisar alterar o código. Isso faz com que os arquivos de configuração sejam dispensáveis e que a aplicação se torne independente da linguagem. Desse modo, conforme a aplicação cresce com os deploys, é evitado problemas com a configuração, sem contar que fica mais fácil de gerenciar o mesmo.
4. Baking Services: A base de código do aplicativo 12 Factors não difere bibliotecas e APIs. Para o aplicativo, os dois são ferramentas incorporadas, acessíveis por URLs ou outros localizadores salvos no diretório. Um aplicativo 12 Factor é capaz de trocar um banco de dados local por um controlado por terceiros sem que seja necessário alterar o código, sendo que apenas a configuração (método 3) precisa ser alterado.
5. Build, release, run: Separar estritamente os builds e executar em estágios.A separação dessas etapas é importante para garantir que a automação e a manutenção do sistema seja o mais simples possível.
6. Processes: Processos de 12 Factors não gravam estado, quaisquer dados que precisam ser armazenados devem ser gravados em um serviço de apoio, geralmente em um banco de dados. Tais dados podem ser usados como um cache curto. Isto é de extrema importância, pois permite que a aplicação execute de forma mais rápida, porém em contrapartida quando o sistema é reiniciado, todos os estados locais são destruídos.

7. Port Binding: Ao contrário de alguns aplicativos da web que são executados de um servidor da web, o 12 Factor age como serviço independente, o que significa que não depende de nenhum servidor de aplicativos para ser executado. Isso significa que a porta à qual o aplicativo está conectado também é salva no Config (Método 3). Essa abordagem também ajuda na troca de serviços de suporte.
8. Concurrency: Neste método, pode-se dizer que que, devido aos Processos, a simultaneidade é simples e confiável, isso se torna muito benéfico, considerando que o dimensionamento vertical pode ser limitado de acordo com a execução do servidor, ou seja, é sempre recomendado obter o dimensionamento horizontal.
9. Disposability: Aplicativos de 12 Factors devem ser iniciados ou parados o mínimo possível, pois assim evita erros e bugs que poderiam afetar todo o desempenho do sistema. Também deve-se fazer, sempre que possível, com que as solicitações HTTP sejam breves. Afinal, se a conexão cair no meio de uma troca de dados, isso fará com que dados sejam perdidos e fará com que o cliente tenha de se reconectar manualmente.
10. Dev/prod parity: Aplicativos 12 Factors são projetados para implantações contínuas, mantendo menos disparidades entre os ambientes de produção e desenvolvimento. Isso é feito para evitar problemas imprevistos quando o aplicativo estiver funcionando bem no ambiente de desenvolvimento. Isso não significa necessariamente ter o SO em ambos os ambientes. Portanto, o ambiente de teste e produção deve ser mantido o mais semelhante possível. É bem melhor para passar o desenvolvimento de uma alteração ou a implantação para o ambiente de produção.
11. Logs: Tratar logs como fluxo de eventos.Dessa forma é possível acompanhar os logs do sistema distribuído em tempo real e até criar alertas para monitoramento.
12. Admin Process: Executar tarefas de gerenciamento, como processos únicos (migração de banco de dados ou execução de scripts). Isso parece correto se realmente for uma tarefa única, mas e se a migração do banco de dados for uma tarefa periódica e você precisar lidar com agendadores e executá-la automaticamente. Então deve-se automatizar as tarefas administrativas, uma vez que a aplicação pode rodar em diversos servidores, contemplando diversos processos. Tarefas como limpar cache, carregar dados, atualizar bases de dados, precisam ser facilitadas.

A metodologia doze-fatores pode ser aplicada a aplicações escritas em qualquer linguagem de programação, e que utilizem qualquer combinação de serviços de suportes.

No caso da aplicação desenvolvida a propriedade da implementação deu-se nos fatores disponibilidade, processes e logs.

Disponibilidade é requisito inerente de uma aplicação projetada para atender demandas de utilização na escala de um serviço disponível a toda sociedade brasileira. O conceito de processes é chave no ganho de escala da aplicação, o processo de inferência da rede neural é dependente apenas da entrada sendo tratada e o resultado de sua computação não tera nenhuma dependência de dados com os processamentos futuros ou passados. Em um ambiente como *Graphical Processing Units* GPUs cujo custo operacional é fator crítico a eficiente utilização do recurso memória é chave na viabilidade econômica de uma aplicação.

Optamos por sua utilização de forma a pomover o reuso das instancias da aplicação em GPU, permitindo o atendimento de múltiplos clientes com diferentes modelos de detecção em tempo real. A parametrização do modelo de execução da-se na requisição da aplicção e o objeto de computação é gerado em tempo real.

4.3 Contêineres e Nuvens sob demanda

Um dos mais recorrentes problemas em provisionamento de recursos computacionais é a capacidade de determinar precisamente a quantidade de recursos para atender a carga de processamento de uma aplicação, dado um certo acordo de nível de serviço (ANS) expresso em algum objetivo de performance. Dada a característica dinâmica da alocação de recursos durante a execução de um programa recursos são geralmente alocados baseados nos requerimentos de pico de processamento. Tal super provisionamento é um dos principais geradores de ineficiência energética de sistemas em nuvem (ALZAHRANI et al., 2017).

Paralelamente, o crescimento da conteinerização de aplicações abre as portas para novas soluções servindo computação de borda. Contêineres são serviços rodando em um dispositivo usando seu sistema operacional para disponibilizar recursos. Isso os torna menos complexos que máquinas virtuais as quais usam uma cópia completa do sistema operacional (FELTER et al., 2015).

A conteinerização de aplicações vem crescendo como técnica para enfrentar esse desafio. A presença de contêineres na edge pode ser utilizada como uma maneira de mitigar o tempo de resposta de aplicações. Estudos anteriores demonstram que a conteinerização das aplicações junto a rede de borda provêm uma queda do tempo de resposta

médio de processamento frente a nuvem tradicional de ordens de magnitude conforme o número de *requests* a estes recursos cresce (SAMI; MOURAD, 2020); a simulação comparando uma nuvem Amazon (AWS) com um sistema edge rodando um raspbery pi mostra que conforme o número de requisições cresce o tempo de resposta da edge chega a ser 93% menor que a nuvem tradicional (SAMI; MOURAD, 2020).

Com o surgimento de ferramentas como Docker e Kubernetes (HIGHTOWER; BURNS; BEDA, 2017) e as chamadas aplicações de 12 fatores (NEWMAN, 2015) o provisionamento de recursos computacionais tomou uma nova face. Docker e Kubernetes provêm um novo paradigma para encapsular e executar código com abstração de plataforma. Ao mesmo tempo a metodologia de 12 fatores assegura a construção de microserviços desacoplados e *stateless*. Essa combinação permite a alocação dinâmica de recursos na rede de borda para executar as tarefas geradas nos dispositivos energeticamente limitados.

Essa classe de aplicações é caracterizada por uma grande quantidade de computação efetuada sob um conjunto limitado de dados também chamada de Multiple Instruction Single Data (MISD). Adicionalmente sabemos que a diferença de performance entre processadores de propósito geral e *Graphical Processing Units* é de cerca de duas ordens de magnitude (HADJIS et al., 2015). Assim, uma arquitetura de *offloading* entre o processamento local e uma nuvem de processamento pode aumentar a autonomia de bateria enquanto reduz os tempos computacionais.

Essas idéias são exploradas tomando vantagem dos três benefícios da conteinerização: reuso em tempo real da infraestrutura, diminuição da alocação dos recursos de memória e processamento, já que as instâncias de computação são invocadas apenas para o processamento de uma dada tarefa. A plataforma de execução é abstraída conforme recursos heterogêneos são gerenciados pelo escalonador de containers, isso prove reuso de *hardware* enquanto abstrai configurações específicas de plataforma. Por fim, o custo de processamento é amplamente diminuído, visto que, a alocação de uma nova instância de execução em um container é restrita a tarefa de criar um novo processo (Singh; Singh, 2016).

Especificamente para tarefas de visão computacional a construção *stateless* dos contêineres *Docker* é de interesse, visto que cada tarefa de detecção necessita de um conjunto específico de pesos a ser carregado na rede neural. Adicionalmente, não há dependência de dados entre as diferentes requisições ao sistema. Uma vez efetuada a classificação através do vetor de classes da rede neural todos os recursos utilizados no

processamento podem ser devolvidos ao ambiente original.

Dessa forma é possível aumentar a entrega de serviço sem a necessidade de escalar horizontalmente a plataforma (IBM, 2010).

5 ESTUDO DE CASO ESCOLHIDO

5.1 Contexto do Estudo de Caso

No contexto da pandemia de 2020 surge uma crescente necessidade de extração e verificação automática de dados em fontes documentais. Os exemplos mais evidentes são os cadastros de auxílio-emergencial, um programa de renda básica, e o gerenciamento do processo de vacinação. Ambos os casos mencionados possuem em seu fluxo a atividade manual de identificação e registro dos dados apresentados pelo cidadão de forma documental no momento de acessar o respectivo serviço público.

A combinação desses fatores fomentou o esforço da criação de uma aplicação capaz de utilizar visão computacional tomando como cenário os ganhos já mencionados e provendo celeridade ao processo de cadastro, validação e registro dos documentos apresentados. A aplicação deveria: ser performática para executar a validação em tempo real sem impactar no fluxo do serviço, ser executada dentro da nuvem do governo de forma permitir o consumo de seu processamento em todo o território nacional a partir dos dispositivos mais simples e prover grande grau de precisão de forma que as revisões e intervenções manuais fossem rápidas e pontuais.

Nesse contexto a aplicação do algoritmo YOLO dentro de uma nuvem privada compostas por GPUs foi o modelo escolhido de forma a dar atendimento a essa demanda. A aplicação foi desenvolvida como uma *Abstract Programmers Interface*(API) de forma a prover interoperabilidade entre os diferentes contextos já mencionados ao mesmo que oferece um processo claro de fácil acoplamento em variadas soluções. O processo de construção foi dividido em duas etapas principais; a preparação dos dados a serem reconhecidos pela rede neural e a implementação do serviço da nuvem.

5.2 Preparação dos dados

Para o funcionamento da rede neural é fundamental o trabalho de treinamento dos pesos da rede. *Overfitting*, explosão do gradiente e desbalancemaneto de classes são os maiores desafio quando executa-se o treinamento de uma Rede Neural Convolucional. Esses problemas podem trazer problemas como diminuição da performance performance e aumento da taxa de falsos positivos, o correto entendimento de medidas preventivas é fundamental para evitar essa ocorrência (JOSHI et al., 2019). No caso da extração de

dados documentais o *overfitting* é uma preocupação fundamental.

Overfitting em modelagem matemática pode ser definido como a produção de um modelo que corresponde demasiadamente a um conjunto específico de dados, podendo assim falhar ao reconhecer outros dados ou predizer observações futuras corretamente (EVERITT, 2002). No caso do modelo construído, o algoritmo YOLO deve reconhecer a estrutura dos documentos sem excluí-los devido a divergências em seu preenchimento.

Foi explorada a homogeneidade da estrutura dos documentos. A capacidade do algoritmo YOLO em detectar objetos semelhantes de uma mesma classe, sem a necessidade que estes sejam exatamente iguais, mostrou-se fundamental no processo de segmentação e detecção dos campos de interesse dos documentos. Isso deve-se ao fato que apesar dos documentos possuírem a mesma estrutura, os valores de seus campos variam conforme o titular.

Um conjunto inicial de 50 imagens de cada classe foi selecionado. As imagens não sofreram nenhuma forma de tratamento à exceção da aplicação do algoritmo HoG (HBALI; SADGAL; FAZZIKI, 2013). HoG é uma técnica já amplamente reconhecida para detecção de rostos e olhos. Sua aplicação permite que o documento seja orientado à partir da posição relativa da foto do portador. Uma vez alinhadas as imagens, foi efetuado o processo manual de marcação dos campos de interesse.

Para atingir esse objetivo foi construída uma aplicação que utiliza o algoritmo YOLO para a classificação e segmentação dos documentos. Os RGs e CNHs foram segmentados nos seguintes campos: foto_CNH, nome_CNH, identidade_CNH, cpf_CNH, nascimento_CNH, filiacao_CNH, registro_CNH, validade_CNH, pri_habilitacao_CNH, local_emissao_CNH, data_emissao_CNH, CNH, nome_RG, foto_RG, assinatura_RG, digital_RG, registro_geral_RG, data_expedicao_RG, filiacao_RG, naturalidade_RG, nascimento_RG, doc_origem_RG, CPF_RG, RG_verso, RG_frente e CNH_frente.

Uma representação dos itens acima mencionados pode ser visualizada na figura 5.1 que apresenta a marcação dos campos por parte do algoritmo de detecção em um documento real.

Uma vez finalizado, e efetuado o treinamento utilizamos esse primeiro conjunto de pesos para efetuar a marcação automática de uma massa de 1000 documentos. Tal abordagem deve-se a necessidade de reconhecer RGs e CNHs de diferentes portadores, expedidas em diferentes unidades da federação brasileira as quais possuem alguns diferencias em sua formatação como os brasões e marcas da água dos respectivos entes expedidores. O processo de treinamento levou cerca de 10 horas em uma NVidia RTX

Figura 5.1: Campos identificados no documento

3600.

Essa capacidade de generalização do algoritmo YOLO a solução é capaz de identificar utilizar um conjunto de RGs e CNHs limitado para o treinamento sendo capaz de classificar corretamente novos documentos os quais não pertenciam ao conjunto inicial. Assim, partindo de um conjunto de 50 documentos de cada classe foi possível obter um resultado final com precisão superior a 95%.

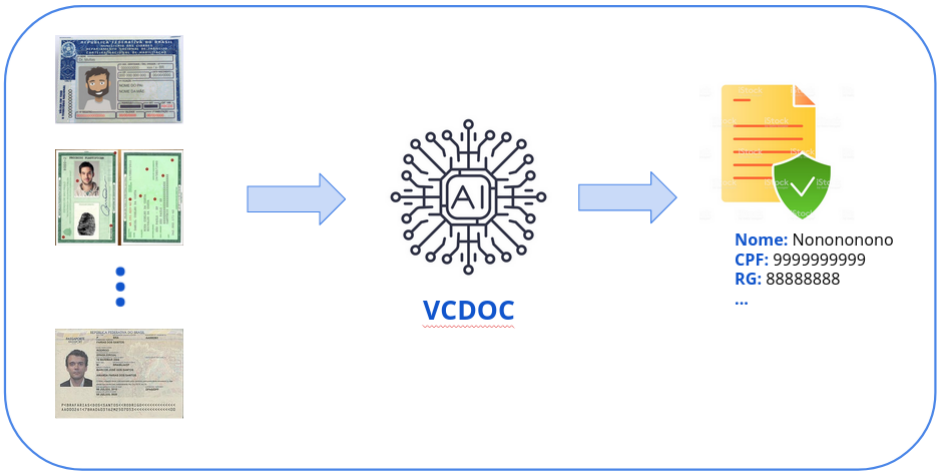
5.3 A ferramenta Visual Checker

Nesse contexto projetamos uma ferramenta à partir das imagens CUDA da plataforma Docker. Seu objetivo é efetuar a verificação em tempo real dos documentos apresentados pelos cidadãos. O fluxo definido para o tratamento de dados é descrito a seguir:

- Identificação do documento e extração dos dados: o governo federal definiu um série de aplicativos responsáveis pelo cadastramento nos programas já mencionados. Em um primeiro momento é necessário reconhecer qual documento foi apresentado como identificação. De forma a saber em quais bases de dados deve ser feita a validação dos dados apresentados.
- Validação dos dados: as bases documentais de Registro de Identidade no Brasil não são centralizadas. Dessa forma é possível que um mesmo cidadão tenha 28 RGs, um para cada ente da federação. A verificação manual e aferição das informações ali contidas tornam-se de alta complexidade devido a situação apartada das bases de dados.

A imagem 5.2 apresenta o fluxo básico da aplicação. Os documentos são submetidos ao processamento da rede neural e seus campos componentes consolidados no formado JSON para futura verificação. Note-se que o processo de inferência opera sobre uma imagem a qual não precisa ser tratada simplificando os custos tanto da aplicação como da plataforma cliente. O formato JSON facilita a interação com diferentes dispositivos, isso deve-se ao tratamento dos dados na forma de um *string* (CROCKFORD; MORNINGSTAR, 2017) estruturado.

Figura 5.2: Funcionamento Validador



Uma visão mais detalhada da arquitetura pode ser vista na figura 5.3. O serviço de reconhecimento e segmentação dos dados recebe requisições de serviços de uma nuvem pública. As requisições são capturadas por um componente chamado *dispatcher*, responsável por orquestrar e persistir a chegada e os resultados das computações. As imagens cujo processamento já findou são armazenadas em um CEPH, um *storage* de baixa performance e grande volume. Já o resultado das requisições é armazenado em um banco relacional PostgreSQL para fins de futura auditoria.

A criação destes serviços em máquinas separadas do núcleo de processamento da rede neural visa não apenas aumentar performance como diminuir custos, utilizando recursos computacionais mais simples que as GPUs que executam a Rede Neural.

Note-se que dada essa arquitura o container responsável pelo processo de inferência funciona como uma máquina SIMD (Single Instruction Multiple Data) sendo aderente ao conceito de 12 factor apps.

O Componente VCDOC GPU é um contêiner que contem a execução da rede neural. Foi implementado um *listener*, ou seja, um serviço que escuta as chamadas à porta 8081 responsável por receber as submissões de imagens para serem processadas. Sua execução dá-se em uma plataforma TeslaV100 cujas especificações encontram-se na

Figura 5.3: Topologia da Solução

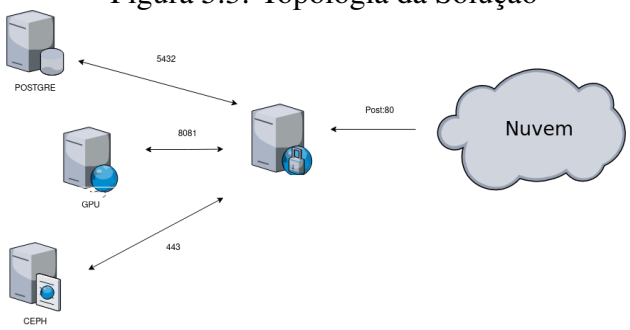


figura 5.4

Figura 5.4: Especificação TeslaV100

	V100 PCIe	V100 SXM2	V100S PCIe
GPU Architecture	NVIDIA Volta		
NVIDIA Tensor Cores	640		
NVIDIA CUDA® Cores	5,120		
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS	8.2 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS	16.4 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS	130 TFLOPS
GPU Memory	32 GB /16 GB HBM2		32 GB HBM2
Memory Bandwidth	900 GB/sec		1134 GB/sec
ECC	Yes		
Interconnect Bandwidth	32 GB/sec	300 GB/sec	32 GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink™	PCIe Gen3
Form Factor	PCIe Full Height/Length	SXM2	PCIe Full Height/Length
Max Power Comsumption	250 W	300 W	250 W
Thermal Solution	Passive		
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC®		

Fonte:(MARKIDIS et al., 2018)

Note-se que a plataforma Tesla tem capacidade computacional compatível com as velocidades presentes no cenário 5G. O *throughput* acima de 100 TFLOPS e a banda de memória acima de 900GB/s são mais que suficientes para assegurar o nível de serviço requerido para um sistema de extração documental em tempo real. O processamento interno requisições não impacta na reatividade da solução criada. Dessa maneira nosso serviço é capaz de atender as requisições de diversos aplicativos que acessam esse barramento de serviço.

Um desafio inicial foi orientar os documentos de forma a auxiliar a rede Darknet a aumentar sua acurácia. Para isso foi utilizada a função de reconhecimento de faces disponível na biblioteca DLib (DOMINGUES; ROSáRIO, 2019) Ao obter-se a posição correta do campo foto, pelo alinhamento dos olhos da imagem, foi possível simplificar o processo de tratamento das imagens.

A DLib opera buscando e alinhando os olhos presentes em uma imagem. conforme pode ser visto na imagem 5.5.

Figura 5.5: Algoritmo HoG em funcionamento

Fonte:(BOYKO; BASYSTIUK; SHAKHOVSKA, 2018)

O resultado da execução de um processo de detecção pode ser observado na imagem 5.6. Note-se que conforme a descrição do algoritmo YOLO as marcações são levemente maiores que os campos identificados. Isso é de grande valia no momento de efetuar os recortes onde será executado o algoritmo OCR.

Figura 5.6: Resultado execução

Dada a constituição *stateless* dos contêineres orientados à serviço, a camada de persistência dos dados processados é feita em duas etapas. Em um banco relacional PostgreSQL que armazena os dados da requisição e um ponteiro para um *storage* de alta capacidade e baixa performance que armazena as imagens já processadas. Assim é possível a manutenção das combinações chave e valor de cada requisição sem modificações na camada de aplicação.

Essa escolha dá-se para reduzir custos uma vez que imagens já processadas dificilmente serão buscadas à exceção de possíveis auditorias. Logo o local de armazenamento das requisições precisa de alta capacidade de escrita e baixo custo de armazenamento.

Os campos uma vez classificados são submetidos a um processo de extração de dados através da ferramenta EasyOCR (JAIDEDAI,). Esta além de operar em GPUs sua implementação trabalha em imagens multi linha o que é necessário no caso de campos de tamanho variável como os dos documentos. Uma das principais dificuldades na extração de informações documentais é a variabilidade de tamanho, fonte e formatação no preenchimento dos campos.

Um exemplo de requisição pode ser observado no trecho de código abaixo:

```
0  {
1  "key": {
2    "cpf": "11295395045"
3  },
4  "answer": {
5    "documento": {
6      "formato": "JPG",
7      "base64": "/9j/4AAQSkZJRgABAQEBLAEsAAD/4Te8
           RXhpZgAASUkqAAgAAAAFABoBBQABAAAASgAAABsBBQABAAAAU
8    },
9    "documento_verso": {
10     "formato": "JPG",
11   },
12   "biometria_face": {
13     "formato": "JPG",
14   }
15 }
16 }
```

O campo base64, aqui truncado para fins de anonimização carrega a informação da imagem a ser analisada.

6 ANÁLISE DE RESULTADOS

6.1 Avaliação da performance

Quinhentas imagens teste foram selecionadas aleatoriamente, de uma base disjunta do conjunto de treinamento sendo verificada a acurácia dos os campos detectados. As imagens foram coletadas de bancos de dados públicos. Não houve interferência manual no processo de análise dos dados.

O gabarito dos dados de cada documento foi extraído das bases oficiais usando como chave dados identificados pela ferramenta VisualChecker. A combinação de chaves como cpf, filiação e nome permitiu recuperar todos os documentos presentes no experimento. Foram implementadas duas técnicas complementares ao processo de extração de forma a aumentar a acurácia da solução.

EasyOcr_MunicOrgExp: uso de rapid fuzzy para identificar município e órgão mais similar.

EasyOcr_MunicOrgExp2: uso de rapidfuzz e Regex para identificar órgão mais similar.

Rapid fuzz (Max Bachmann , 2021) é uma biblioteca de *string matching* em Python e C++ baseada em distância de Levenshtein (LEVENSHTEIN et al., 1966). Esta é a distância de edição entre dois "strings"(duas sequências de caracteres) é dada pelo número mínimo de operações necessárias para transformar um string no outro. Entendemos por "operações"a inserção, deleção ou substituição de um carácter.

A imagem 6.1 apresenta o número de comparações e matches que a biblioteca consegue realizar por segundo. Uma descrição completa de cada *benchmark* pode ser encontrada em (MAXBACHMANN,).

Primeiramente, destaca-se o alto percentual de acerto por parte dos campos extraídos. A ferramenta EasyOCR opera sob um conjunto de linhas. Assim o recorte retangular provido pelo YOLO torna-se ideal para prover uma área de fácil segmentação. Isso aliado a velocidade de correção da técnica rappid fuzz permite grande acurácia em tempo real no processo de extração do texto.

Foi testada a performance com linhas de 48 e 120 pixels de resolução. À partir da chave CPF dos documentos da base de treinamento e validação foi possível comparar os resultados extraídos com os contidos nas bases oficiais do governo a acurácia média dos campos foi de 95,42% para um conjunto de imagens de uma base pública de dados. Com

Figura 6.1: Performance Rappid Fuzz

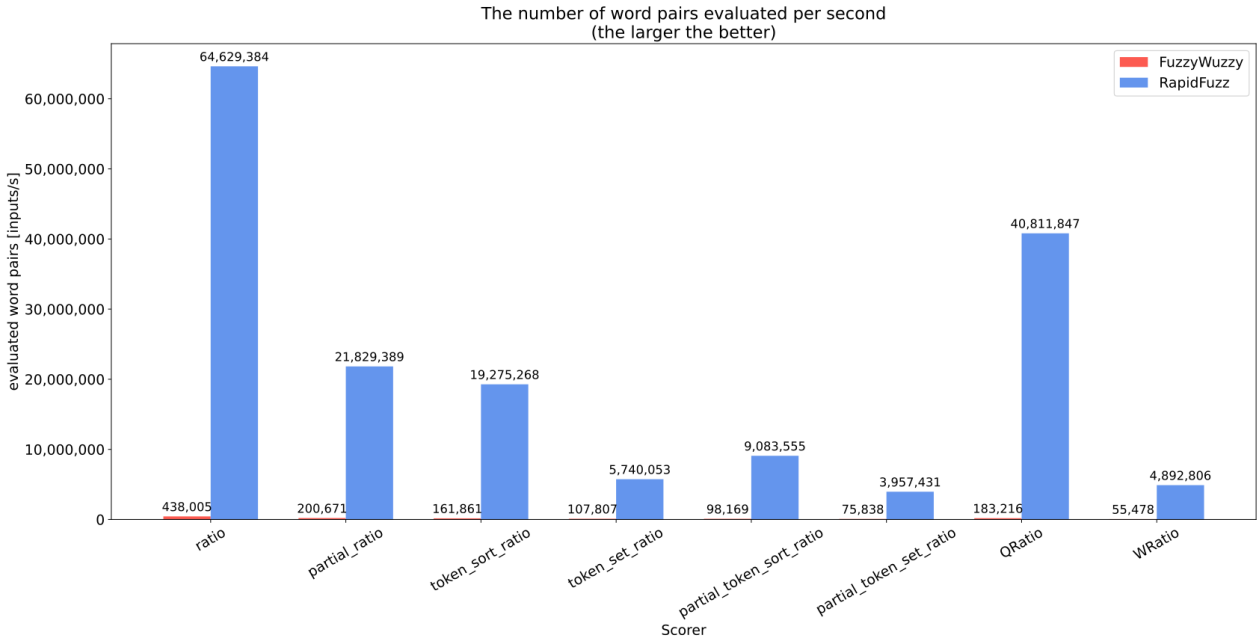


Tabela 6.1: Performance de processamento

	A	B	C	D	E	F	G	H	I	J	K
1	Arquivo	Nome CNH	Identidade CNH	CPF CNH	Dt.Nasc. CNH	Filiação CNH	Nº Registro CNH	Dt. Validade CNH	Dt. 1ª Habil. CNH	Local Emissão CNH	Dt. Emissão CNH
2	MÉDIA (cro	96,99%	96,54%	98,75%	99,03%	95,43%	98,57%	98,82%	98,97%	98,18%	99,23%
3	MÉDIA (cro	96,99%	96,54%	98,75%	98,59%	95,43%	98,48%	98,66%	98,82%	99,03%	99,20%
4	DIFEREN	0,00%	0,00%	0,00%	-0,45%	0,00%	-0,10%	-0,17%	-0,15%	0,86%	-0,02%
5											
6	Nome RG	Nº Reg. Geral RG	Dt. Exped. RG	Filiação RG	Naturalidade de RG	Dt. Nasc. RG	CPF RG	Tempo Predição	Tempo Detecção	Tempo OCR	Tempo Total
7	96,30%	90,97%	88,28%	92,71%	97,17%	93,33%	84,55%	0,42	0,48	3,46	3,94
8	96,30%	90,27%	91,03%	92,35%	93,93%	93,33%	84,55%	0,42	0,49	4,19	4,68
9	0,00%	-0,69%	2,76%	-0,36%	-3,24%	0,00%	0,00%	0,13%	0,54%	73,79%	74,33%

um tempo total de processamento médio de 1,45 segundos, dos quais 1,2 correspondem ao processo de OCR.

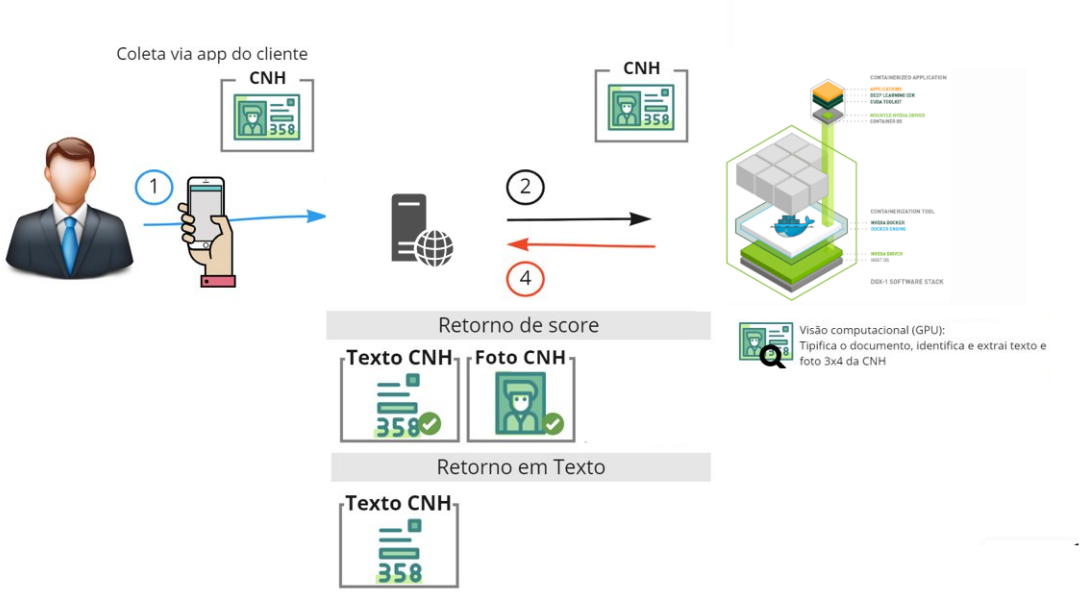
O campo com a taxa de identificação mais baixa trata-se da data de expedição do RG, esse problema ocorre pela falta de padronização no preenchimento desse campo. Diversos separadores como "."ponto, "/", barra e -"hífen são usados indiscriminadamente pela falta de uma padronização.

Considerando o tempo médio de processamento das imagens obtido de 1,45 segundos fica evidente a vantagem em efetuar o offloading não apenas no quesito energético mas também no tocante a responsividade da solução. A Figura 6.2 apresenta o funcionamento completo da ferramenta.

Primeiramente analisamos o desvio padrão do tempo total de processamento dos documentos.

Os gráficos a seguir apresentam os desvio padrão na extração dos principais cam-

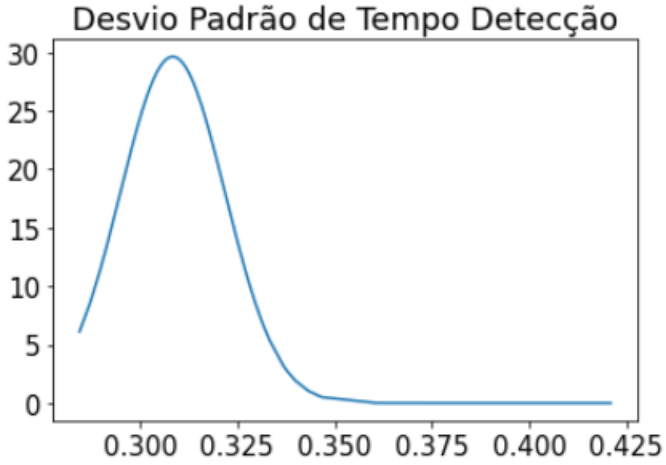
Figura 6.2: Funcionamento Request



pos. No primeiro conjunto de gráficos vemos o desvio padrão do tempo de detecção 6.3. Vemos que a grande maioria das detecções ocorrem em até 0.325 segundos.

Os casos onde o tempo necessário foi superior deveram-se ao desalinhamento do documento que necessitou de rotações antes de ser identificado.

Figura 6.3: Desvio Padrão campo Data de Nascimento



Foi obtido um desvio padrão de 1,18519031521468 segundos na detecção. Esse desvio alto em relação a média é oriundo do processo de orientação do documento. Tais números foram obtidos escalando a solução até 1000 requisições.

6.2 Avanços da Tecnologia

Na data da primeira implementação do protótipo a execução do algoritmo era repassada do wrapper python para o binário da aplicação Darknet. Com o avanço das APIs de desenvolvimento, a execução de CNNs passou a ser uma método nativo da API python.

Isso demonstra o avanço no esforço do provimento de algoritmos de visão como serviço. Na versão atual da aplicação através do método

```
cv::dnn::readNet (const String &model,
const String &config="", const String &framework="")
```

é possível identificar o esforço no provimento de uma tecnologia mais aderente ao novo modelo de provimento de serviços.

Essa modificação torna os projetos de visão computacional mais simples de serem instanciados na nuvem. A ausência de artefatos intermediários de compilação e não necessidade de manipulação de componentes binários simplifica o tratamento das versões da aplicação. Todo o conjunto do projeto pode ser versionado como código.

Isso reduz necessidades de espaço em disco e memória de execução e armazenamento. Além de facilitar a criação de *pipelines* de provisionamento da aplicação.

7 CONCLUSÃO E TRABALHOS FUTUROS

7.1 Conclusão

As premissas levantadas na fase de projeto do trabalho se mostraram verdadeiras. A utilização de SaaS como plataforma para a entrega de serviços de visão computacional mostrou-se tecnicamente promissora. No presente momento a chegada do 5G no Brasil abre um leque de oportinidades para serviços de visão computacional. Mercados imediatos são processos visuais de inspeção de peças e estruturas de grande porte.

Como trabalhos futuros é de interesse a expansão da solução para outros modelos de documentos do Mercosul, de forma a possuir um serviço centralizado de verificação documental para o trânsito no bloco, visto que o uso de passaporte é dispensado por acordos. Adicionalmente, o provimento do serviço de validação dos dados pode ser vendido como adicional ao processo de *on-boarding* de instituições como bancos, provedores de serviço (Über, Dash, iFood..) de forma a prover maior segurança e rastreabilidade aos agentes envolvidos nas transações.

7.2 Desafios Encontrados

A migração de uma aplicação tradicional para o modelo 12 Factor App utilizado em micro-serviços requereu a implementação de um padrão *listener*. Este é um *loop* que executa a espera por novas requisições ao invés de receber um comando direto do sistema. Essa decisão implicou na criação de um *fork* da versão original da rede.

Essa solução posteriormente foi substituída pela remoção do binário da aplicação à partir da disponibilização do módulo Deep Neural Networks (dnn module) do OpenCV. Sua implementação disponibilizou novos métodos para a execução de DNNs, incluindo um método específico para execução do algoritmo YOLO. Outros métodos de interesse também inclusos no pacote são a execução de modelos OCR e a utilização de modelos para o *framework* Caffee.

A disponibilização desse módulo nesse formato reforça a crença no modelo de aplicação desenvolvido. Uma vez que as bibliotecas como o OpenCV disponibilizado inicialmente em 2000 começam a disponibilizar métodos compatíveis com a necessidade das aplicações em nuvem.

Adicionalmente, a definição das bibliotecas de forma a prover um ambiente que

provisse compatibilidade entre todos os componentes da solução mostrou-se uma tarefa onerosa. Embora a linguagem Python provenha grande grau de abstração, as bibliotecas de visão computacional requerem atenção na compatibilidade entre sua implementação, os drivers do sistema e a versão do mecanismo de virtualização. A construção de ambientes de visão computacional ainda requer grande intervenção manual.

Nesse contexto novos produtos como Zendo (ZENDO,) e Amazon Rokognition (AMAZON...,) estão sendo anunciados de forma a prover aos criadores de serviços na tecnologia 5G um conjunto de ferramentas de aobnstração que diminua os custos de construção dos modelos através da remoção da curva de aprendizagem envolvida nas técnicas de visão computacionais atuais.

REFERÊNCIAS

Adit Deshpande . *A Beginner's Guide To Understanding Convolutional Neural Networks*. 2019. [Online; accessed August 20, 2019]. Available from Internet: <https://adeshpande3.github.io/assets/Cover.png>.

Max Bachmann . *Rapid fuzzy string matching in Python using various string metrics*. 2021. [Online; accessed August 20, 2021]. Available from Internet: <https://github.com/maxbachmann/RapidFuzz>.

AIRLINES, A. **American Airlines Boarding Regulations**. [S.l.]: American Airlines, 2023. <https://www.aa.com/i18n/travel-info/boarding-process.jsp>.

AKHERFI, K.; GERNDT, M.; HARROUD, H. Mobile cloud computing for computation offloading: Issues and challenges. **Applied Computing and Informatics**, King Saud University, v. 14, n. 1, p. 1–16, 2018. ISSN 22108327. Available from Internet: <http://dx.doi.org/10.1016/j.aci.2016.11.002>.

AKTER, R.; DOAN, V.-S.; LEE, J.-M.; KIM, D.-S. Cnn-ssdi: Convolution neural network inspired surveillance system for uavs detection and identification. **Computer Networks**, v. 201, p. 108519, 2021. ISSN 1389-1286. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1389128621004503>.

ALSBATIN, L.; OZ, G.; ULUSOY, A. H. An Overview of Energy-Efficient Cloud Data Centres. **2017 International Conference on Computer and Applications, ICCA 2017**, n. June 2019, p. 211–214, 2017.

ALZAHRANI, E. J.; TARI, Z.; ZEEPHONGSEKUL, P.; LEE, Y. C.; ALSADIE, D.; ZOMAYA, A. Y. SLA-Aware Resource Scaling for Energy Efficiency. **Proceedings - 18th IEEE International Conference on High Performance Computing and Communications, 14th IEEE International Conference on Smart City and 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016**, IEEE, p. 852–859, 2017.

AMAZON Rekognition. Available from Internet: <https://aws.amazon.com/pt/rekognition/>.

ANALYSIS, A.; MANUAL, U. User Manual User Manual. n. January, p. 1–148, 2012.

BALAN, R.; FLINN, J.; SATYANARAYANAN, M.; SINNAMOHIDEEN, S.; YANG, H.-I. The case for cyber foraging. In: **Proceedings of the 10th Workshop on ACM SIGOPS European Workshop**. New York, NY, USA: ACM, 2002. (EW 10), p. 87–92. Available from Internet: <http://doi.acm.org/10.1145/1133373.1133390>.

BARNETT, T. J.; SUMITS, A.; JAIN, S.; ANDRA, U. Cisco Visual Networking Index (VNI) Update Global Mobile Data Traffic Forecast. **Vni**, p. 2015–2020, 2015. ISSN 1553-877X. Available from Internet: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html>.

BELAJAR Pembelajaran Mesin Indonesia. 2020. <https://indoml.com>.

BOCHKOVSKIY, A. **OpenCV**. [S.l.]: GitHub, 2017. <https://github.com/opencv/opencv/>.

BONI, M. F.; LEMEY, P.; JIANG, X.; LAM, T. T.-Y.; PERRY, B. W.; CASTOE, T. A.; RAMBAUT, A.; ROBERTSON, D. L. Evolutionary origins of the sars-cov-2 sarbecovirus lineage responsible for the covid-19 pandemic. **Nature microbiology**, Nature Publishing Group, v. 5, n. 11, p. 1408–1417, 2020.

BOUREAU, Y.-L.; PONCE, J.; LECUN, Y. A theoretical analysis of feature pooling in visual recognition. In: **Proceedings of the 27th International Conference on International Conference on Machine Learning**. USA: Omnipress, 2010. (ICML'10), p. 111–118. ISBN 978-1-60558-907-7. Available from Internet: <http://dl.acm.org/citation.cfm?id=3104322.3104338>.

BOYKO, N.; BASYSTIUK, O.; SHAKHOVSKA, N. Performance evaluation and comparison of software for face recognition, based on dlib and opencv library. In: **2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP)**. [S.l.: s.n.], 2018. p. 478–482.

BRASIL, G. F. do. **Auxílio Brasil**. <https://www.gov.br/cidadania/pt-br/auxilio-brasil#oque>.

BUXMANN, P.; HESS, T.; LEHMANN, S. Software as a service. **Wirtschaftsinformatik**, Springer, v. 50, n. 6, p. 500–503, 2008.

CHAE, M.; LEE, H.; LEE, K. A performance comparison of linux containers and virtual machines using docker and kvm. **Cluster Computing**, v. 22, 01 2019.

CHANDAKKAR, P. S.; LI, Y.; DING, P. L. K.; LI, B. Strategies for re-training a pruned neural network in an edge computing paradigm. In: . [S.l.: s.n.], 2017. p. 244–247.

CHUN, B.-G.; IHM, S.; MANIATIS, P.; NAIK, M.; PATTI, A. Clonecloud: Elastic execution between mobile device and cloud. In: . [S.l.: s.n.], 2011. p. 301–314.

CISCO Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper. 2017. Available from Internet: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html>.

COMPUTER Vision – Thematic Research. 2020.

CROCKFORD, D.; MORNINGSTAR, C. Standard ecma-404 the json data interchange syntax. 12 2017.

DASIOPOULOU, S.; MEZARIS, V.; KOMPATSIARIS, I.; PAPASTATHIS, V.; STRINTZIS, M. Knowledge-assisted semantic video object detection. **Circuits and Systems for Video Technology, IEEE Transactions on**, v. 15, p. 1210 – 1224, 11 2005.

DENG, J.; DONG, W.; SOCHER, R.; LI, L. jia; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: **In CVPR**. [S.l.: s.n.], 2009.

DIGITAL, C. **Infraero e Serpro oficializam o embarque biométrico**. 2022. <https://www.convergenciadigital.com.br/Inovacao/Infraero-e-Serpro-oficializam-o-embarque-biometrico-59422.html?UserActiveTemplate=mobile>.

DING, C.; WANG, S.; LIU, N.; XU, K.; WANG, Y.; LIANG, Y. REQ-YOLO: A resource-aware, efficient quantization framework for object detection on FPGAS. **FPGA 2019 - Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays**, p. 33–42, 2019.

DOCKER. Understanding the Business Value of Docker Enterprise Edition. n. June, 2017.

DOMINGUES, P.; ROSáRIO, A. F. a. Deep learning-based facial detection and recognition in still images for digital forensics. In: **Proceedings of the 14th International Conference on Availability, Reliability and Security**. New York, NY, USA: Association for Computing Machinery, 2019. (ARES '19). ISBN 9781450371643. Available from Internet: <https://doi.org/10.1145/3339252.3340107>.

EVERITT, B. **The Cambridge dictionary of statistics**. Cambridge, UK; New York: Cambridge University Press, 2002. ISBN 052181099X 9780521810999. Available from Internet: <http://www.worldcat.org/search?qt=worldcat_org_all&q=052181099X>.

Fan, K.; Pan, Q.; Wang, J.; Liu, T.; Li, H.; Yang, Y. Cross-domain based data sharing scheme in cooperative edge computing. In: **2018 IEEE International Conference on Edge Computing (EDGE)**. [S.l.: s.n.], 2018. p. 87–92. ISSN null.

FELTER, W.; FERREIRA, A.; RAJAMONY, R.; RUBIO, J. An updated performance comparison of virtual machines and linux containers. In: . [S.l.: s.n.], 2015. v. 00, p. 171–172.

FELZENSZWALB, P. F.; GIRSHICK, R. B.; MCALLESTER, D.; RAMANAN, D. Object detection with discriminatively trained part-based models. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 32, n. 9, p. 1627–1645, Sept 2010. ISSN 0162-8828.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. Thesis (PhD), 2000.

FORCZMAńSKI, P.; SMOLINSKI, A.; NOWOSIELSKI, A.; MAłECKI, K. Segmentation of scanned documents using deep-learning approach. In: ____. [S.l.: s.n.], 2020. p. 141–152. ISBN 978-3-030-19737-7.

FOWLER, M.; LEWIS, J. Microservices. 2014. Available from Internet: <http://martinfowler.com/articles/microservices.html>.

GERDES, R.; OTTERBACH, R.; KAMMÜLLER, R. Fast and robust recognition and localization of 2-d objects. In: . [s.n.], 1995. v. 8, n. 6, p. 365–374. ISSN 1432-1769. Available from Internet: <https://doi.org/10.1007/BF01213498>.

GIRSHICK, R. **Fast R-CNN**. 2015.

GIRSHICK, R. B.; DONAHUE, J.; DARRELL, T.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. **CoRR**, abs/1311.2524, 2013. Available from Internet: <http://arxiv.org/abs/1311.2524>.

HADJIS, S.; ABUZAID, F.; ZHANG, C.; Ré, C. Caffe con troll: Shallow ideas to speed up deep learning. In: **Proceedings of the Fourth Workshop on Data Analytics in the Cloud**. New York, NY, USA: Association for Computing Machinery, 2015. (DanaC'15). ISBN 9781450337243. Available from Internet: <https://doi.org/10.1145/2799562.2799641>.

HBALI, Y.; SADGAL, M.; FAZZIKI, A. E. Object detection based on hog features: Faces and dual-eyes augmented reality. In: **2013 World Congress on Computer and Information Technology (WCCIT)**. [S.l.: s.n.], 2013. p. 1–4.

HIGHTOWER, K.; BURNS, B.; BEDA, J. **Kubernetes: Up and Running Dive into the Future of Infrastructure**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2017. ISBN 1491935677, 9781491935675.

HTTPS://WWW.SECUNET.COM/EN/ABOUT-US/PRESS/ARTICLE/GERMAN-FEDERAL-POLICE-AND-SECUNET-PRESENT-AN-APP-FOR-MOBILE-IDENTITY-CHECKS. 2018.

HU, Y. C.; PATEL, M.; SABELLA, D.; SPRECHER, N.; YOUNG, V. ETSI White Paper #11 Mobile Edge Computing - a key technology towards 5G - etsi_wp11_mec_a_key_technology_towards_5g.pdf. **ETSI White Paper No. 11 Mobile**, n. 11, p. 1–16, 2015. Available from Internet: <http://www.etsi.org/images/files/ETSIWhitePapers/etsi{_}wp11{_}mec{_}a{_}key{_}technology>.

IBM. **IBM Red Books Power Systems and SOA Synergy**. [S.l.]: IBM, 2010. <http://www.redbooks.ibm.com/redbooks/pdfs/sg247607.pdf>.

JACCARD, P. Étude comparative de la distribution florale dans une portion des alpes et des jura. **Bulletin del la Société Vaudoise des Sciences Naturelles**, v. 37, p. 547–579, 1901.

JAIDEDAI. **EasyOCR**. Available from Internet: <https://github.com/JaidedAI/EasyOCR>.

JAIN, S.; VAIBHAV, A.; GOYAL, L. Raspberry pi based interactive home automation system through e-mail. In: **2014 International Conference on Reliability Optimization and Information Technology (ICROIT)**. [S.l.: s.n.], 2014. p. 277–280.

JOSHI, S.; VERMA, D.; SAXENA, G.; PARAYE, A. Issues in training a convolutional neural network model for image classification. In: ____. [S.l.: s.n.], 2019. p. 282–293. ISBN 978-981-13-9941-1.

JSON Data Interchange Syntax COPYRIGHT PROTECTED DOCUMENT. 2017.

JSON.ORG. **JSON**. [S.l.]: Json.org, 2015. <https://json.org/json-pt.html>.

KAEHLER, A.; BRADSKI, G. **Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2016. ISBN 1491937998.

KANG, Y.; HAUSWALD, J.; GAO, C.; ROVINSKI, A.; MUDGE, T.; MARS, J.; TANG, L. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In: . [S.l.: s.n.], 2017. p. 615–629.

KEMP, R.; PALMER, N.; KIELMANN, T.; BAL, H. Cuckoo: A computation offloading framework for smartphones. In: GRIS, M.; YANG, G. (Ed.). **Mobile Computing, Applications, and Services**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 59–79. ISBN 978-3-642-29336-8.

KOSTA, S.; AUCINAS, A.; HUI, P.; MORTIER, R.; ZHANG, X. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. **Proceedings - IEEE INFOCOM**, v. 945-953, p. 945–953, 03 2012.

LECUN, Y.; JACKEL, L.; BOTTOU, L.; BRUNOT, A.; CORTES, C.; DENKER, J.; DRUCKER, H.; GUYON, I.; MULLER, U.; SACKINGER, E.; SIMARD, P.; VAPNIK, V. Comparison of learning algorithms for handwritten digit recognition. In: . [S.l.: s.n.], 1995.

LEVENSHTEIN, V. I. et al. Binary codes capable of correcting deletions, insertions, and reversals. In: SOVIET UNION. [S.l.], 1966. v. 10, n. 8, p. 707–710.

LIYANAGE, M.; PORAMBAGE, P.; DING, A. Y.; KALLA, A. Driving forces for multi-access edge computing (mec) iot integration in 5g. **ICT Express**, v. 7, n. 2, p. 127–137, 2021. ISSN 2405-9595. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S2405959521000631>.

LV, H.; CHEN, D.; WANG, Y. Deployment of edge-computing in 5g nfv environment and future service-based architecture. **2018 IEEE 4th International Conference on Computer and Communications, ICCC 2018**, IEEE, p. 811–816, 2018.

MARKIDIS, S.; CHIEN, S. W. D.; LAURE, E.; PENG, I. B.; VETTER, J. S. NVIDIA tensor core programmability, performance & precision. **CoRR**, abs/1803.04014, 2018. Available from Internet: <http://arxiv.org/abs/1803.04014>.

MAXBACHMANN. **RappidFuzz**. Available from Internet: <https://maxbachmann.github.io/RapidFuzz/fuzz.html>.

MOBILETIME. **58exclusivamente pelo celular**. [S.l.]: Mobiletime, 2021. <https://www.mobiletime.com.br/noticias/18/08/2021/tic-domicilios-2020-81-da-populacao-brasileira-tem-acesso-a-internet/>.

NEWMAN, S. **Building Microservices**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 1491950358, 9781491950357.

OKUN, O.; DOERMANN, D.; PIETIKAINEN, M. Page segmentation and zone classification: The state of the art. p. 37, 11 1999.

OPENCV. **OpenCV**. [S.l.]: GitHub, 2020. <https://github.com/opencv/opencv/wiki/ChangeLog#version420>.

QORVO. **Getting to 5G: Comparing 4G and 5G System Requirements**. 2020. Available from Internet: <https://www.qorvo.com/design-hub/blog/getting-to-5g-comparing-4g-and-5g-system-requirements>.

REDMON, J. **Darknet: Open Source Neural Networks in C**. 2013–2016. <http://pjreddie.com/darknet/>.

Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016. p. 779–788.

REDMON, J.; FARHADI, A. Yolo9000: Better, faster, stronger. **arXiv preprint arXiv:1612.08242**, 2016.

REDMON, J.; FARHADI, A. Yolo9000: Better, faster, stronger. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 6517–6525, 2016.

REDMOND, J. **YOLO: Real-Time Object Detection**. Available from Internet: <https://pjreddie.com/darknet/yolo/>.

REPORT, G. GR MEC 031 - V2.1.1 - Multi-access Edge Computing (MEC) MEC 5G Integration. v. 1, p. 1–47, 2020.

RESEARCH, G. V. **Computer Vision Market Size, Share Trends Analysis Report By Component, By Product Type (Smart Camera-based Computer Vision System, PC-based Computer Vision System), By Application, By Vertical, By Region, And Segment Forecasts, 2022 - 2030**. [S.l.]: Grind View Research, 2022. <https://www.grandviewresearch.com/industry-analysis/computer-vision-market>.

Sahni, Y.; Cao, J.; Zhang, S.; Yang, L. Edge mesh: A new paradigm to enable distributed intelligence in internet of things. **IEEE Access**, v. 5, p. 16441–16458, 2017. ISSN 2169-3536.

SAMI, H.; MOURAD, A. Towards Dynamic On-Demand Fog Computing Formation Based On Containerization Technology. **2018 International Conference on Computational Science and Computational Intelligence (CSCI)**, IEEE, n. 2, p. 960–965, 2020.

SAMSUNG. USER MANUAL Â Table of Contents. n. February, p. 1–17, 2019.

SARFARAZ, A.; HäMMäINEN, H. 5g transformation: How mobile network operators are preparing for transformation to 5g ? In: . [S.l.: s.n.], 2017. p. 1–9.

SATYANARAYANAN, M. Edge Computing. **Computer**, v. 50, n. 10, p. 36–38, 2017. ISSN 00189162.

SHAH, J.; DUBARIA, D. Building modern clouds: Using docker, kubernetes google cloud platform. In: **2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)**. [S.l.: s.n.], 2019. p. 0184–0189.

Singh, S.; Singh, N. Containers docker: Emerging roles future of cloud technology. In: **2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)**. [S.l.: s.n.], 2016. p. 804–807.

SMITH, S. W. **The Scientist & Engineer's Guide to Digital Signal Processing**. California Technical Pub, 1997. ISBN 0966017633. Available from Internet: <https://www.amazon.com/Scientist-Engineers-Digital-Signal-Processing/dp/

0966017633?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0966017633>.

Strigl, D.; Kofler, K.; Podlipnig, S. Performance and scalability of gpu-based convolutional neural networks. In: **2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing**. [S.l.: s.n.], 2010. p. 317–324. ISSN 2377-5750.

SUFYAN, F.; BANERJEE, A. Computation offloading for distributed mobile edge computing network: A multiobjective approach. **IEEE Access**, IEEE, v. 8, p. 149915–149930, 2020.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: **Computer Vision and Pattern Recognition (CVPR)**. [s.n.], 2015. Available from Internet: <http://arxiv.org/abs/1409.4842>.

TALEB, T.; SAMDANIS, K.; MADA, B.; FLINCK, H.; DUTTA, S.; SABELLA, D. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. **IEEE Communications Surveys and Tutorials**, v. 19, n. 3, p. 1657–1681, 2017. ISSN 1553877X.

Uetz, R.; Behnke, S. Large-scale object recognition with cuda-accelerated hierarchical neural networks. In: **2009 IEEE International Conference on Intelligent Computing and Intelligent Systems**. [S.l.: s.n.], 2009. v. 1, p. 536–541. ISSN null.

W., S.; H., S.; J., C.; Q., Z.; W., L. Edge computing-an emerging computing model for the internet of everything era. **Jisuanji Yanjiu yu Fazhan/Computer Research and Development**, v. 54, p. 907–924, 05 2017.

WURSTER, M.; BREITENBüCHER, U.; FALKENTHAL, M.; LEYMANN, F. Developing, deploying, and operating twelve-factor applications with tosca. In: . [S.l.: s.n.], 2017. p. 519–525. ISBN 978-1-4503-5299-4.

ZENDO. Available from Internet: <https://deepai.org/zendo>.

ZHANG, Q.; ZHANG, M.; CHEN, T.; SUN, Z.; MA, Y.; YU, B. Recent advances in convolutional neural network acceleration. **Neurocomputing**, v. 323, p. 37 – 51, 2019. ISSN 0925-2312. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0925231218311007>.

ZHU, Y.; SAMAJDAR, A.; MATTINA, M.; WHATMOUGH, P. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. **2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)**, IEEE, Jun 2018. Available from Internet: <http://dx.doi.org/10.1109/ISCA.2018.00052>.