DAVID AUGUSTO RIBEIRO

RECONHECIMENTO FACIAL BASEADO NO ALGORITMO DE APRENDIZADO PROFUNDO YOLO

LAVRAS – MG

2021

DAVID AUGUSTO RIBEIRO

RECONHECIMENTO FACIAL BASEADO NO ALGORITMO DE APRENDIZADO PROFUNDO YOLO

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Engenharia de Sistemas e Automação, área de concentração em Sistemas Inteligentes, para a obtenção do título de Mestre.

Prof. Dr. Demóstenes Zegarra Rodríguez

Orientador

LAVRAS – MG

2021

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Ribeiro, David Augusto
Reconhecimento Facial baseado no Algoritmo de Aprendizado Profundo YOLO / David Augusto Ribeiro. – Lavras : UFLA, 2021.
105 p. : il.

Dissertação (Mestrado Acadêmico)–Universidade Federal de Lavras, 2021.
Orientador: Prof. Dr. Demóstenes Zegarra Rodríguez.
Bibliografia.

1. Biometria. 2. Reconhecimento Facial. 3. Redes Neurais Profundas. I. Rodríguez, Demóstenes Zegarra. II. Título.

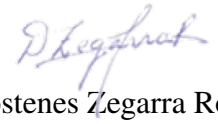
DAVID AUGUSTO RIBEIRO

RECONHECIMENTO FACIAL BASEADO NO ALGORITMO DE APRENDIZADO PROFUNDO YOLO
FACE RECOGNITION BASED ON THE YOLO DEEP LEARNING ALGORITHM

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Engenharia de Sistemas e Automação, área de concentração em Sistemas Inteligentes, para a obtenção do título de Mestre.

APROVADA em 15 de Dezembro de 2021.

Prof. DSc. Demóstenes Zegarra Rodríguez	UFLA
Profa. DSc. Renata Lopes Rosa	UFLA
Prof. DSc. Dante Coaquira Begazo	USP



Prof. Dr. Demóstenes Zegarra Rodríguez
Orientador

LAVRAS – MG
2021

“Dedico este trabalho primeiramente à Deus, que me deu forças para manter firme no propósito. Dedico à minha amada família que sempre me deram suporte para alcançar meus sonhos. Dedico aos professores da Universidade Federal de Lavras que sempre estiveram disponíveis para dúvidas e auxílios em momentos da academia”.

"O temor do Senhor é o princípio da sabedoria, e o conhecimento do Santo a prudência."
Provérbios 9:10

RESUMO

O desenvolvimento tecnológico possibilitou ao homem nas últimas décadas dar um grande salto em técnicas de processamento computacional voltado às Redes Neurais Artificiais (RNA). A biometria facial é uma área que está em crescimento no mundo, e suas aplicações são interessantes, tanto para empresas privadas, quanto públicas. Sua capacidade crescente de autenticação em sistemas de segurança e entretenimento a torna um dos melhores meios para validação de identidade e/ou monitoramento de fluxo de pessoas. No entanto, ainda nos deparamos com sistemas que apresentam certa lentidão no reconhecimento e que demandam alto poder de processamento. Objetiva-se então, suprir essa demanda com uma arquitetura de reconhecimento leve e robusta, e que ao mesmo tempo apresente índices de desempenho otimizados, como precisão média (mAP - Mean Average Precision), inferência, baixo tempo de resposta e interseção sobre união (IoU - Intersection Over Union). O presente trabalho apresenta um processo de aprendizagem supervisionado, no qual faz-se uso de uma arquitetura emergente que vem apresentando destaque no cenário de sistemas de reconhecimento de padrões em processamento de imagens. Programa-se nas linguagens Python e C, com uso de frameworks OpenCV e Darknet em arquitetura YOLOv4 (versão 4). Trabalha-se a RNA em máquina virtual em nuvem com GPU NVIDIA Tesla T4 em ambiente Colab, treinando-a em 3 bases de dados distintas: OIDv4, Personal e Wider Face. Nosso sistema também pode operar localmente em sistemas Linux, como o Ubuntu Minimal, que por sua vez requer ajustes básicos de configuração em IDE Jupyter. Os resultados mostram a capacidade de classificação/detecção otimizada na arquitetura YOLO, bem como obtenção de índices aprimorados no trabalho. O *dataset* OID obteve um mAP de 69,23% para classe de objetos, com inferência média de 82.2%, tempo de detecção de 16 segundos e alcançou um IoU de 52.63%. O segundo *dataset* Personal obteve os incríveis mAP de 99,11% em biometria facial de reconhecimento individual, com inferência média de 98%, tempo de detecção de 3 segundos e alcançou um IoU de 82.56%. E por fim, o terceiro *dataset* Wider Face, obteve um mAP de 86,04% em biometria facial multivariada, com inferência média de 91%, tempo de detecção de 5 segundos e alcançou um IoU de 61.32%. Os resultados, portanto, demonstram a qualidade da rede neural desenvolvida em virtude dos objetivos inicialmente propostos, nos quais mostram-se promissores em relação à comparativos com outros modelos da literatura no estado da arte dos últimos anos.

Palavras-chave: Reconhecimento Facial. Biometria. Redes Neurais Artificiais. Reconhecimento de Padrões. YOLO.

ABSTRACT

Technological development has made it possible for man in recent decades to make a great leap in computational processing techniques aimed at Artificial Neural Networks (ANN). Facial biometrics is a growing area in the world, and its applications are interesting for both private and public companies. Its growing ability to authenticate in security and entertainment systems makes it one of the best means of identity validation and/or people flow monitoring. However, we still come across systems that have a certain slowness in recognition and that demand high processing power. The objective is, then, to supply this demand with a light and robust recognition architecture, and that at the same time presents optimized performance indexes, such as average precision (mAP - Mean Average Precision), inference, low response time and intersection over union (IoU - Intersection Over Union). The present work presents a supervised learning process, in which an emerging architecture is used that has been highlighted in the scenario of pattern recognition systems in image processing. It is programmed in Python and C languages, using OpenCV and Darknet frameworks in YOLOv4 architecture (version 4). The ANN is worked on in a cloud virtual machine with NVIDIA Tesla T4 GPU in Colab environment, training it in 3 different databases: OIDv4, Personal and Wider Face. Our system can also operate locally on Linux systems such as Ubuntu Minimal, which in turn requires basic configuration adjustments in the Jupyter IDE. The results show the optimized sorting/detection capability in the YOLO architecture, as well as achieving improved indices on the job. The *dataset* OID obtained a mAP of 69.23% for object class, with an average inference of 82.2%, detection time of 16 seconds and reached an IoU of 52.63%. The second *dataset* Personal achieved an incredible 99.11% mAP in individual recognition facial biometrics, with an average inference of 98%, detection time of 3 seconds and achieved an IoU of 82.56%. And finally, the third *dataset* Wider Face, obtained a mAP of 86.04% in multivariate facial biometrics, with an average inference of 91%, detection time of 5 seconds and reached an IoU of 61.32% . The results, therefore, demonstrate the quality of the neural network developed in virtue of the objectives initially proposed, in which they are promising in relation to comparisons with other models in the literature in the state of the art in recent years.

Keywords: Facial Recognition. Biometry. Artificial Neural Networks. Pattern Recognition. YOLO.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Biometria

Tecnologias computacionais já fazem parte do cotidiano das pessoas. Uma vez que essas tecnologias demandam segurança de acesso às informações privadas do usuário, faz-se necessário sistemas avançados de segurança cibernética para autenticação no acesso, desde sistemas de autenticação com *login/senha*, até sistemas mais seguros como os sistemas de autenticação com padrões biométricos. O termo biometria vem do grego *bios metron* que significam *vida* e *medida* respectivamente, ou seja *medida da vida*. De acordo com Saeed e Nagashima (2012) há um certo consenso de que o primeiro tipo de biometria a ser utilizado foi a impressão digital, contudo ocorre a inexistência exata da data de sua primeira implementação, variando também em localidades, tais como Egito, China e Babilônia. Alega-se que os primeiros rastros da biometria vieram da Mesopotâmia (atual território do Iraque) depois do povo babilônico há cerca de 4.000 anos (provavelmente entre 1913–1885 a.C.). Komarinski (2004) diz que o primeiro sistema proposto para identificação biométrica, ocorreu na Índia em 1858, no qual se baseava no registro da remuneração dos funcionários por meio das palmas das mãos aplicadas sobre os contratos. Em 1880 a impressão digital foi considerada uma biometria confiável e importante (SAEED; NAGASHIMA, 2012). Outros sistemas baseados em biometria, tais como os antropométricos foram desenvolvido por Alphonse Bertillon e testados por volta de 1883, onde o modelo media de forma precisa comprimentos e larguras da cabeça e corpo de um indivíduo (NUNES, 2015). Posteriormente, em 1892, Francis Galton escreveu um estudo detalhado sobre sistemas de reconhecimento biométrico por meio de impressão digital.

A biometria pode ser utilizada como ferramenta na identificação e autenticação de uma única pessoa. Quando lidamos com sistemas computacionais em biometria, significa que um único usuário possui características biológicas particulares que o destacam de todos os demais. Hoje, as pesquisas têm se direcionado para a robustez de sistemas de detecção em geral, porém diferenciando-as em diferentes tipos de arquiteturas. Muitas fisiologias para características biométricas tem sido usadas, tais como impressão digital dos dedos, íris, face, formato da mão, voz e maneira de andar, dependendo dos tipos de aplicações. As características biométricas são adquiridas por meio de sensores adequados, no qual informações são extraídas para formar um modelo biométrico. Durante a identificação (processável como uma sequência de verificações),

o sistema processa outra medição biométrica que é comparada com os modelos armazenados que produzem aceitação ou rejeição.

1.2 Reconhecimento Facial

O advento e o amplo uso da tecnologia de aprendizado profundo permitiram avanços tremendos na precisão do reconhecimento facial em condições ambientais favoráveis. No campo das tecnologias de reconhecimento facial atual, podemos destacar uma forte tendência em aplicações de inteligência artificial na robótica. De acordo com Xuehong (2009), o método mais imediato e rápido de interação Homem-Máquina (HM) é interagir com robôs por meio da expressão facial. Para tal finalidade é necessário sistemas de reconhecimento facial. Nesse aspecto, um sistema de reconhecimento facial inclui várias partes, como detecção da face, detecção de cor de pele e processamento de imagem. Podemos frisar que nesses sistemas lidamos com diversos fatores de qualidade, nos quais podemos destacar a qualidade da imagem e a taxa de iluminação do ambiente de detecção. A interação HM será uma tendência no campo do desenvolvimento de robôs, e fazer uso da expressão facial para interagir com o robô é o método mais direto e rápido de interação.

Quando tratamos de aplicações de reconhecimento facial, lidamos com uma área em crescimento científico, em que cada arquitetura de rede neural apresenta um relativo desempenho muito específico. Nesse âmbito, podemos citar uma interessante aplicação de Quadros et al. (2017), que trabalhou em um sistema de reconhecimento facial baseado em métodos matemáticos específicos para atuar como identificador de presença de alunos em uma sala de aula. Baseia-se na busca de caraterísticas relevantes para codificar e comparar com outras imagens de um *database*. Tipos de sistemas de reconhecimento facial são configurados como modelos que possuem grau de complexidade com dois padrões de abordagens: extração das características das imagens e outra abordagem que trata da classificação.

O reconhecimento facial se torna um tópico de pesquisa interessante, e isso é comprovado por vários artigos publicados relacionados com o reconhecimento facial, incluindo extração de características faciais, melhorias no algoritmo facial e implementações de reconhecimento facial. O reconhecimento de face começa com a extração das coordenadas de características como largura da boca, largura dos olhos, pupila, e compara o resultado com as medidas armazenadas no banco de dados e retorna o registro mais próximo de acordo com as métricas faciais (SISWANTO; NUGROHO; GALINIUM, 2014).

O reconhecimento facial pode ser entendido como a capacidade de detectar e reconhecer uma pessoa por meio de suas características faciais. A face é multidimensional, e portanto, requer muitos cálculos matemáticos (BORKAR; KUWELKAR, 2017). Os sistemas de reconhecimento tem sido essenciais por fornecerem segurança dos aplicativos, desde os de entretenimento até governamentais. Todas essas aplicações requerem um sistema de reconhecimento facial eficiente. Existem muitos métodos de reconhecimento que já foram propostos, mas que possuem baixa capacidade de reconhecimento e alta taxa de falsos positivos.

As tecnologias de detecção facial tem avançado em ritmo acentuado nos anos recentes com o advento das redes neurais de camadas profundas. Isso tem sido tão evidente que algumas arquiteturas começam a se adaptar a contextos de detecção em videos pré-processados ou em raros casos, detecção em tempo real. Arachchilage e Izquierdo (2019) apresentam uma estrutura ponta a ponta para o reconhecimento facial baseado em vídeo em tempo real. Este sistema detecta, rastreia e reconhece indivíduos de *feed* de vídeo ao vivo. Contudo, a arquitetura abordada pelo autor apresenta limitações no que tange a taxa de FPS (*Frame Per Second*), bem como ao tempo de detecção e precisão, que se comparada a outras arquiteturas no estado da arte.

Para Singh e Goel (2020), quando tratamos de sistemas de reconhecimento biométrico no reconhecimento de qualquer indivíduo, um dos atributos mais importante é o rosto. Ele serve como uma identidade individual de todos e, portanto, o reconhecimento facial ajuda a autenticar qualquer pessoa usando suas características pessoais. Todo o procedimento de autenticação de quaisquer dados faciais é subdividido em duas fases, na primeira fase a detecção facial é feita rapidamente, exceto para os casos em que o objeto é colocado muito longe, seguido desta a segunda fase é iniciada, em que o rosto é reconhecido como um indivíduo. Basicamente, existem dois tipos de técnicas que estão sendo seguidas no padrão de reconhecimento facial, o método Eigenface e o método Fisherface. O método Eigenface basicamente faz uso do PCA (*Principal Component Analysis*) para minimizar o espaço dimensional da face dos traços faciais. Já o método Fisherface é um dos algoritmos mais populares usados no reconhecimento de rosto, baseado no esforço de maximizar a separação entre as classes no processo de treinamento.

A detecção facial e reconhecimento de imagem é um assunto crescente nas pesquisas biométricas. O reconhecimento facial é uma área empolgante e um desafio que cresce rapidamente e tende a avançar visto às demandas de mercado. Para tais finalidades o uso de *frameworks* em reconhecimento podem ser de fundamental importância, uma vez que utilizam ferramentas amplamente testadas por uma comunidade de programadores com objetivos espe-

cíficos. Para o presente projeto utilizaremos o OpenCV e Darknet, uma vez que ambos tem apresentado uma grande variedade de recursos matemáticos aplicados em visão computacional.

Nos dias atuais os smartphones estão sendo agregados com a capacidade de autenticação por reconhecimento facial em vez de senhas convencionais. Outra grande tendência é seu uso em sites de relacionamento social, onde o indivíduo pode ser reconhecido em uma imagem digital por algoritmos específicos (LI et al., 2020). O reconhecimento facial tem ampla perspectivas de aplicação em diversas áreas e sua lógica básica consiste em após o equipamento coletar imagens dos rostos, ele tem a finalidade de processar um algoritmo que busque saber a localização dos rostos na imagem. Utiliza-se uma variedade de algoritmos que identificam o rosto humano em imagens digitais, identificam pessoas e, em seguida, verificam as imagens capturadas comparando-as com as imagens faciais armazenadas em banco de dados. O reconhecimento facial é um tópico importante na visão computacional, e muitos pesquisadores estudaram esse tópico de muitas maneiras diferentes.

Sharma, Bhatt e Sharma (2020) ressaltam que o reconhecimento facial em inteligência artificial é um problema e solução frequente. Essas aplicações estão se tornando amplamente presentes em nosso cotidiano. Um sistema de reconhecimento facial preciso e eficiente é um tópico interessante na maioria das indústrias e setores de pesquisa e desenvolvimento. A informação biométrica é facilmente adaptável em comparação com o tradicional sistema de reconhecimento de cartões. Geralmente, um sistema de reconhecimento de face é precedido por uma técnica de detecção de face, ou seja, a técnica de detecção de face é o estágio preliminar para reconhecimento de uma face em imagens (SAHU; DASH, 2020).

Uma das problemáticas encontradas em sistemas de detecção facial diz respeito à iluminação do ambiente que o indivíduo se encontra, que por sua vez pode causar interferências e reduzir a eficácia da identificação. Podem ocorrer erros de classificação/detecção quando indivíduos estão com óculos ou objetos refletivos na região facial. Sistemas de reconhecimento do tipo generalista também possuem dificuldade em distinguir classes da mesma macro-classe. Como exemplo distinguir a sub-classe *poodle* da classe *pastor alemão*, ambos contidos na macro-classe *cachorros*. O mesmo vale para o sistema de reconhecimento facial que deve ser capaz de distinguir uma pessoa A de outra pessoa B. Os algoritmos devem ser capazes de classificar e detectar a região da classe. Os sistemas no estado da arte ainda se deparam com o desafio de se obter alta taxa de precisão aliada ao tempo de resposta otimizado como veremos na seção sobre comparativos de arquiteturas com YOLO. Para tal combinação necessita-se de modelos

robustos que atendam às novas demandas. Majoritariamente, esses sistemas só conseguem atuar com alto poder de processamento computacional para detectarem com precisão e velocidade a região facial de um indivíduo. Quando tratamos de reconhecimento em *streaming* de vídeos, a demanda por poder computacional é maior. A ausência de sistemas de simples implementação e leveza, tem estimulado a criação de arquiteturas inovadoras como o YOLO que tem apresentado ano após ano índices de desempenho otimizados apesar de sua recente criação.

1.3 Justificativas

Podemos elencar as principais justificativas do presente trabalho, baseando na atual observação do estado da arte em sistemas biométricos. Podemos falar sobre simplicidade. A questão envolvendo sua simplicidade, se dá pelo fato de possuir ambientes de interface de desenvolvimento avançadas e ao mesmo tempo gratuitas e leves, tais como Jupyter, porém, podem ser executadas diretamente dos *terminais*. Uma vez que o sistema YOLO possibilita o bom desempenho em sistemas operacionais leves como *distros* linux (ex. Ubuntu Minimal), não requerem grande quantidade de memória RAM para processamento em CPU e nem grande espaço de armazenamento em *Hard Disc*, se tornando uma ótima opção também para modelos *mobile*. Sua quantidade de camadas ocultas podem ser vistas detalhadamente na seção sobre as arquiteturas YOLO, no qual o número de camadas (161 camadas) otimiza os índices de desempenho deste sistema. Seu tempo de treinamento médio é de 6 horas para cada 500 *epochs* para pesos da rede neural nos parâmetros de *hardware* citados na seção de metodologias. A arquitetura permite alcançar de 30 a 45 FPS em detecção de vídeos nas versões v3 a v4. Porém algumas versões *tiny* do YOLO em *database* COCO podem alcançar detecção em até 220 FPS, no entanto, sacrificam o índice mAP.

- Demanda de arquitetura de classificação/detecção facial simples e leve.
- Demanda de arquitetura que realize reconhecimento facial em altas taxas de FPS.
- Escassez de técnicas de reconhecimento facial com YOLO na literatura.
- Escassez de *datasets* para detecção individual em formato padronizado YOLO.

1.4 Objetivos

1.4.1 Objetivo Geral

O presente trabalho objetiva desenvolver um sistema biométrico de classificação/detecção e de reconhecimento facial. Inicialmente, se moldará o sistema por meio de aplicação em modelos de objetos, com intuito de desenvolver a base de configurações da RNA. Posteriormente, será criado um *dataset* Personal (fotos do rosto do autor), onde a rede neural será inicialmente treinada e testada para aprender classificar/detectar/reconhecer uma face individual, no intuito de verificar se o sistema é capaz de reconhecer indivíduos específicos em uma imagem/-vídeo. O sistema visa também a classificação/detecção de múltiplas faces (várias faces em uma imagem), onde o algoritmo será aplicado a fim de analisar sua capacidade em ambientes com muitas pessoas.

1.4.2 Objetivos Específicos

Deseja-se a otimização de índices de parâmetros de desempenho, em termos de mAP e IoU em relação à soluções similares citadas no presente trabalho. O índice de Inferência (INF) também será avaliado como atributo qualitativo da rede neural. Para o desenvolvimento, se utilizará da arquitetura YOLO v4, uma vez que é a versão considerada mais estável até o momento. O trabalho fará uso de *frameworks* OpenCV/Darknet em TensorFlow CUDNN, para ambiente em *cloud computing*, aplicando-o em *datasets* OID, Personal e Wider Face. Assim, podemos inferir de forma simplificada:

- Implementar o estado da arte em arquiteturas YOLO: *v1 / v2 / v3 / v4*;
- Construir o *Personal Dataset*;
- Treinamento com *Datasets*: *OID* / *Personal* / *WIDER FACE*;
- Analisar diferentes configurações da rede neural desenvolvida;
- Analisar o tempo de detecção do sistema;
- Alcançar índices de desempenho considerados robustos, superiores a:

  IoU em 50%;

  mAP em 70%;

INF em 80%.

- Desenvolver um classificador/detector de imagens faciais em YOLO.

1.5 Estrutura do Trabalho

Este trabalho tem a seguinte estrutura:

- Capítulo 1: Introdução abordando a contextualização geral, destacando-se a importância do tema, e sobretudo estudos em biometria. Finalizando com as justificativas e apresentação dos objetivos geral e específicos;
- Capítulo 2: É apresentado o referencial teórico com os conceitos primordiais das técnicas de reconhecimento de padrões estatísticos e redes neurais artificias, além de apresentar o estado da arte em arquitetura YOLO em todas suas versões;
- Capítulo 3: São apresentadas as metodologias utilizadas no desenvolvimento do trabalho, recursos e técnicas aplicadas para a execução;
- Capítulo 4: São apresentados os resultados e análises pertinentes;
- Capítulo 5: É apresentado a conclusão e expectativas futuras.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão expostos os conceitos principais de reconhecimento de padrões estatístico, redes neurais artificiais, aprendizado, parâmetros de desempenho, arquitetura YOLO, bem como tópicos importantes para o entendimento.

2.1 RPE - Reconhecimento de Padrões Estatístico

De forma geral pode-se considerar o reconhecimento de padrões como a área computacional que lida com classificação. Para tal finalidade o sistema deve aprender com dados que são oferecidos como entrada, de modo a se construir um modelo analítico que represente o reconhecimento de uma classe.

> O termo reconhecimento de padrões abrange uma ampla gama de problemas de processamento de informações de grande significado prático, desde o reconhecimento de voz e a classificação de caracteres manuscritos até a detecção de falhas em máquinas e diagnósticos médicos. Frequentemente, esses são problemas que muitos humanos resolvem de maneira aparentemente fácil. No entanto, sua solução por meio de computadores tem, em muitos casos, se mostrado extremamente difícil. A estrutura mais geral e mais natural para formular soluções para problemas de reconhecimento de padrões é uma estrutura estatística, que reconhece a natureza probabilística tanto da informação que buscamos processar, quanto da forma em que devemos expressar os resultados. O reconhecimento de padrões é um campo bem estabelecido com uma longa história. (BISHOP, 1997).

O sistema classificador deve ser projetado de forma a ser capaz de classificar corretamente um vetor (imagem, texto, áudio, etc) anteriormente não visto. A presença de um grande número de variáveis de entrada pode apresentar alguns problemas graves para os sistemas de reconhecimento de padrões. Uma técnica para ajudar a aliviar tais problemas é combinar variáveis de entrada juntas para formar um número menor de novas variáveis chamadas *features* ou características. Eles podem ser construídos baseado em alguma compreensão do problema específico que está sendo enfrentado ou podem ser derivados dos dados por procedimentos automatizados.

2.1.1 Regressão

A importância das redes neurais neste contexto é que elas oferecem uma estrutura muito poderosa e generalista para representar mapeamentos não-lineares de várias variáveis de entrada para várias variáveis de saída, onde a forma do mapeamento é governada por uma série

de parâmetros ajustáveis. O processo de determinação dos valores para esses parâmetros com base no conjunto de dados é chamado de aprendizagem ou treinamento e, por esta razão, o conjunto de dados de exemplos é geralmente referido como um conjunto de treinamento. Modelos de rede neural, bem como muitas abordagens convencionais para reconhecimento de padrões estatísticos, podem ser vistos como escolhas específicas para as formas funcionais usadas para representar o mapeamento, juntamente com procedimentos específicos para otimizar os parâmetros.

Segundo Bishop (1997), "em problemas de classificação, a tarefa é atribuir novas entradas para um número de classes ou categorias discretas. No entanto, existem muitas outras tarefas de reconhecimento de padrões, às quais nos referiremos como problemas de regressão, nas quais as saídas representam os valores de variáveis contínuas."

Tanto os problemas de regressão quanto de classificação podem ser vistos como casos particulares de funções de aproximação. No caso de problemas de regressão, é a função de regressão que desejamos aproximar, enquanto para problemas de classificação as funções que buscamos aproximar são as probabilidades de filiação das diferentes classes expressas como funções das variáveis de entrada. Muitas das questões chave que precisam ser abordadas ao lidar com problemas de reconhecimento de padrões são comuns tanto na classificação quanto na regressão.

Temos 3 modalidades de aprendizado:

- Supervised learning (Aprendizado Supervisionado);
- Unsupervised learning (Aprendizado Não-Supervisionado);
- Reinforcement learning (Aprendizado por Reforço).

A minimização de uma função de erro, que envolve valores-alvo para saídas da rede é chamada de *supervised learning* ou *aprendizagem supervisionada*, uma vez que para cada padrão de entrada, o valor da saída desejada é especificado. Uma segunda forma de aprendizado em redes neurais, é chamada de *unsupervised learning* ou *aprendizado não-supervisionado*, não envolve o uso de dados alvo, ao invés de aprender um mapeamento de entrada-saída, o objetivo pode ser modelar a distribuição de probabilidade dos dados de entrada ou descobrir *clusters* ou outra estrutura de dados. Há uma terceira forma de aprendizagem, chamada de *reinforcement learning* ou *aprendizagem por reforço*, na qual informações são fornecidas sobre se as saídas da

rede são boas ou ruins, mas novamente nenhum valor desejado atual é dado, sendo este usado principalmente para aplicações de controle.

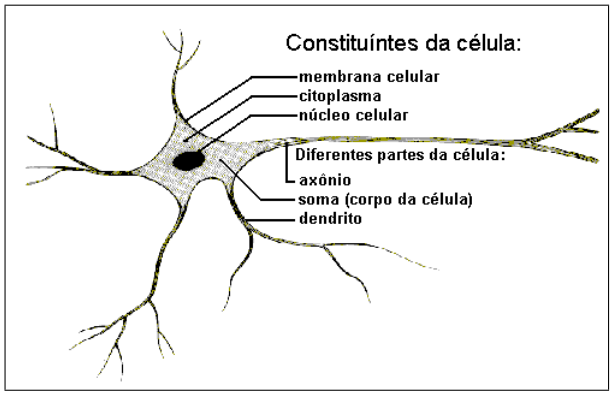
2.2 RNA - Redes Neurais Artificiais

2.2.1 Definição

Como o próprio nome sugere, as RNA são modelos de redes neurais, porem adquirem o caráter artificial devido ao fato de serem criados como modelos não-naturais, assim chamados artificiais. A base por trás das RNA são os os organismos biológicos que apresentam células de neurônios capazes de desempenhar papel no aprendizado do organismo. Seu entendimento nos proporciona replica-lo como modelos em máquinas.

> As RNA ou Redes Neurais Artificiais são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura de rede neural de organismos biológicos inteligentes e que adquirem conhecimento através da experiência. O sistema nervoso é formado por um conjunto extremamente complexo de tipos de células chamadas neurônios. Eles têm um papel essencial na determinação do funcionamento e comportamento do corpo humano e do raciocínio. Os neurônios são formados pelos dendritos que são um conjunto de terminais de entrada pelo corpo central, e pelos axônios que são longos terminais de saída conforme Figura 2.1. (CARVALHO, 2020).

Figura 2.1 – Estrutura celular de um neurônio biológico.



Fonte: Carvalho (2020).

Uma grande RNA pode ter de centenas à milhares de unidades de processamento (neurônios artificiais), porém em comparação ao biológico, o cérebro de um mamífero contém bilhões de neurônios.

As RNA são sistemas paralelos distribuídos compostos por unidades de processamento simples (nós) que calculam determinadas funções matemáticas. Essas unidades são organizadas

em camadas e interligadas por um número significativo de conexões, que por sua vez estão associadas a diferentes valores de pesos, que possuem como objetivo ponderar a entrada recebida por cada neurônio da rede. Nota-se que o funcionamento das RNA é bastante semelhante á estrutura de um cérebro humano, principalmente no que diz respeito à capacidade de aprender através de exemplos e generalizar informações (BRAGA; CARVALHO; LUDERMIR, 2007; SILVA, 2000).

> Os neurônios se comunicam através de sinapses. Sinapse é a região onde dois neurônios entram em contato e através da qual os impulsos nervosos são transmitidos entre eles. Os impulsos recebidos por um neurônio A, em um determinado momento, são processados, e atingindo um dado limiar de ação, o neurônio A dispara, produzindo uma substância neurotransmissora que flui do corpo celular para o axônio, que pode estar conectado a um dendrito de um outro neurônio B. O neurotransmissor pode diminuir ou aumentar a polaridade da membrana pós-sináptica, inibindo ou excitando a geração dos pulsos no neurônio B. Este processo depende de vários fatores, como a geometria da sinapse e o tipo de neurotransmissor. Em média, cada neurônio forma entre mil e dez mil sinapses. O cérebro humano possui cerca de 10 E11 neurônios, e o número de sinapses é de mais de 10 E14, possibilitando a formação de redes muito complexa. (CARVALHO, 2020).

2.2.2 Perceptron

O primeiro modelo computacional de um neurônio foi proposto pelo neurocientista Warren McCulloch e Walter Pitts em 1943. Era dividido em 2 partes. A primeira parte g levava a entrada (dendrito), realizava uma agregação e com base no valor agregado a segunda parte f tomava uma decisão. Essas entradas podiam ser excitatórias ou inibitórias . As entradas inibitórias são aquelas que têm efeito máximo na tomada de decisão, independentemente de outras entradas. McCulloch e Pitts (1943) descreveram esse poderoso e simplificado modelo de neurônio artificial MCP (McCulloch e Pitts) na Figura 2.2, o qual é a base da qual é utilizado atualmente após algumas modificações. Somente após alguns anos foram realizados estudos do trabalho dos autores mencionados - o primeiro que se tem notícia - fazendo ligação direta com o aprendizado de redes neurais artificiais. Pode-se destacar também o feito por Donald Hebb em 1949, que propôs uma teoria para explicar o aprendizado em neurônios biológicos baseada no reforço das ligações sinápticas entre neurônios excitados, segundo relata Braga, Carvalho e Ludermir (2007). Em 1958 Rosenblatt mostrou em seu livro *Principles of Neurodynamics* a base do modelo similar ao *perceptron* da Figura 2.3. Nele, os neurônios eram organizados em camada de entrada e saída, onde os pesos das conexões eram adaptados a fim de se atingir a eficiência sináptica, no qual foi testada no reconhecimento de imagens de caracteres alfabéticos.

Assim o modelo de *perceptron* simples mais conhecido foi proposto por Rosenblatt (1958). A topologia original desse modelo era composta por 3 níveis: o primeiro era composto por unidades de entrada chamadas retina, o nível intermediário formado pelas unidades de associação e por fim o nível de saída composto pelas unidades de resposta. A Figura 2.3 ilustra a topologia do *perceptron* simples. Os sinais da entrada no neurônio são representados pelo vetor

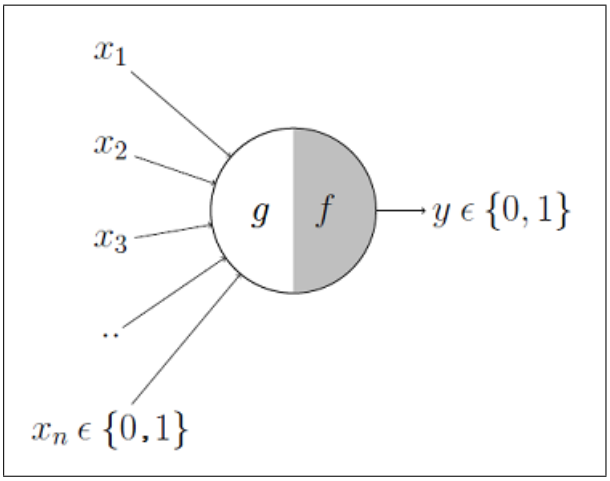
$$x = [x_1, x_2, x_3, ..., x_n]$$

, podendo corresponder aos pixels de uma imagem, por exemplo. Ao chegarem ao neurônio, são multiplicados pelos respectivos pesos sinápticos, que são os elementos do vetor

$$w = [w_1, w_2, w_3, ..., w_n]$$

, é somado ao b (bias) que pode ser entendido como uma constante associada ao peso do neurônio de acordo com sua relevância na rede neural, no qual a função é aumentar ou diminuir a entrada líquida, de forma a transladar a função de ativação no eixo. Gera-se o valor z, comumente denominado potencial de ativação, onde se encaminha para o próximo neurônio como uma nova entrada, de acordo com a equação (2.1).

Figura 2.2 – Modelo de neurônio artificial de McCulloch e Pitts



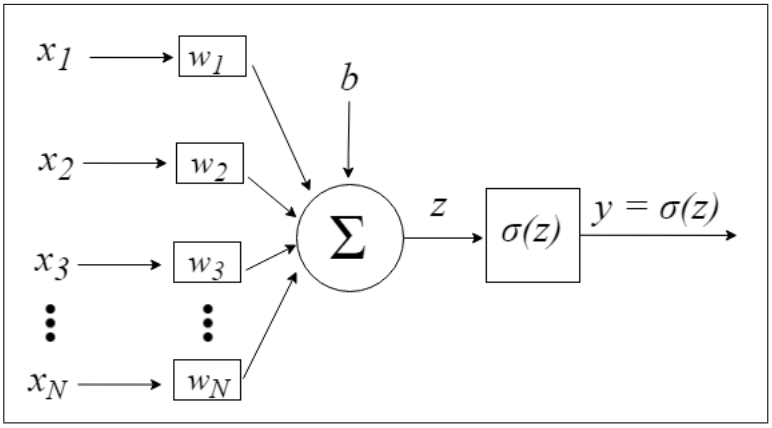
Fonte: Chandra (2018).

$$z = \sum_{i=1}^{N} x_i w_i + b \tag{2.1}$$

Devido ao fato de ter sofrido diversas críticas em relação a sua capacidade computacional, o *perceptron* permaneceu esquecido por muito tempo. Esse cenário mudou por meio

dos estudos de Hopfield (1982), que impulsionaram os estudos nesta área, o que resultou em trabalhos onde os *perceptrons* passaram a ser utilizados em conjunto, formando assim uma rede multicamada, denominadas de Redes MLP - *(Multi Layer Perceptron)* (BRAGA; CARVALHO; LUDERMIR, 2007).

Figura 2.3 – Topologia de um *perceptron* simples.



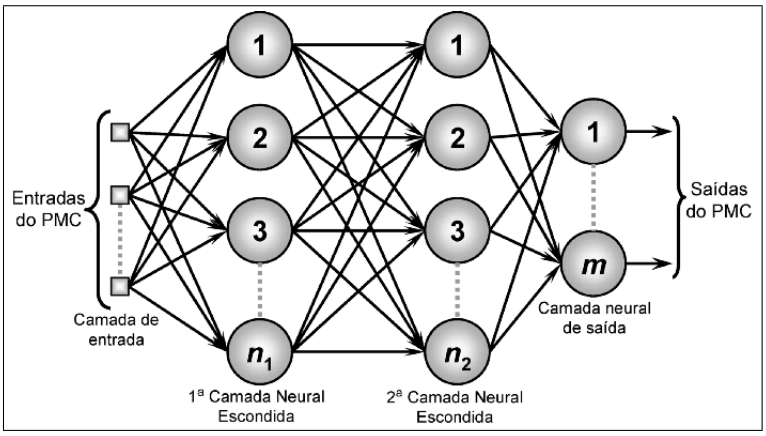
Fonte: Leite (2018)

2.2.3 MLP - Multi Layer Perceptron

De acordo com Minsky e Papert (1970), os anos 70 foram marcados pela estagnação do estudo de RNA, que ocorreu devido a falta de conhecimento de soluções para treinar redes com uma ou mais camadas intermediárias. Em 1986 Rumelhart, Hinton e Williams criaram o algoritmo de treinamento *backpropagation*. Após a descrição do algoritmo, grande parte das pesquisas em RNA foi na tentativa de propor variações desse modelo visando maior eficiência e eficácia de convergência. Para otimizar parâmetros de desempenho, surgiram as Redes MLP ou PMC - Perceptron de Múltiplas Camadas - que segundo Cybenko (1989) apresentam um poder computacional relativamente maior e possibilita o tratamento de dados que não são linearmente separáveis, características não encontradas nas redes sem camadas intermediárias. As Redes MLP consistem em diferentes nós interligados, onde cada nó possui um peso. A capacidade de adaptação em diferentes ambientes consiste no ajuste dos pesos em cada conexão. A Figura 2.4 ilustra o esquema de uma Rede MLP (BRAGA; CARVALHO; LUDERMIR, 2007). Alguns fatores devem ser levados em consideração para a consolidação de uma Rede MLP. A estrutura é divida em três camadas: a primeira camada, recebe os valores de entrada e não há nenhum tipo de processamento. Na segunda camada está a maior parte do processamento. Na terceira camada se encontra a resposta da rede, onde a mesma recebe os resultado das camadas inter-

mediárias e realiza o processamento no algoritmo *backpropagation*. É importante ressaltar que a segunda camada da estrutura pode ser composta por inúmeras camadas intermediárias, porém quanto maior esse número, mais elevada a complexidade do projeto.

Figura 2.4 – Rede Perceptron Multicamadas.



Fonte: Moreira (2018)

Outro fator a ser considerado no projeto de redes neurais é o número de neurônios em cada uma das camadas. Seguindo a mesma ideia das camadas intermediárias, quando maior o número de neurônios, maior a complexidade da rede. O número adequado de neurônios pode depender de diversos fatores, que vão desde a complexidade da função a ser aprendida e treinada pela rede, até a distribuição estatística dos dados. Logo, determinar o valor ótimo do número de neurônios é uma das questões fundamentais em se tratando de redes neurais artificiais. Apesar de não existir uma regra ou fórmula para determinar o número de neurônios necessários para que uma rede neural resolva determinado problema, há na literatura alguns trabalhos que norteiam a estimativa do tamanho da rede dependendo da aplicação.

2.2.4 Funções de Ativação

Na grande maioria dos projetos, os modelos de cada unidade da rede neural possui uma não-linearidade na sua saída, a qual devem ser reduzida. A função de ativação representa o efeito que a entrada interna e o estado atual de ativação exercem na definição do próximo estado de ativação da rede. Dessa maneira, existem funções de ativação lineares e não-lineares. Essas últimas são responsáveis por conferirem à rede neural do tipo MLP a capacidade de mapeamento não-linear. Os dois tipos de funções de ativação não-lineares mais comumente utilizadas são: a função *sigmoidal* e a função *tangente hiperbólica*, que estão representadas nas equações (2.2) e (2.3) respectivamente (HAYKIN, 2005).

$$f(x) = \frac{1}{(1+e^{-x})} \tag{2.2}$$

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3}$$

Haykin (2005) define função *sigmoidal* como uma função crescente, com balanceamento adequado entre o comportamento linear e o não-linear e, além disso, assume um intervalo de variação entre 0 e 1. Dessa maneira, as funções *sigmoidais* padrão produzem saídas próximas de zero, e as mesmas são aplicadas como entradas nas camadas seguintes. A função *sigmoidal* em alguns casos é substituída pela função *tangente hiperbólica*, pois dessa forma a *sigmoidal* da função logística é preservada, porém com capacidade de assumir valores positivos e negativos. Para Fleck et al. (2016), devido à necessidade de tempo de processamento reduzido, a função *tangente hiperbólica* pode ser substituída por uma função matematicamente equivalente chamada *tangente hiperbólica sigmoidal*, de cálculo mais simples e consequentemente rápido. Apesar de apresentar diferenças numéricas em relação à *tangente hiperbólica*, ela é uma boa alternativa para redes neurais onde a forma exata da função não é tão importante, uma vez asseguradas suas propriedades matemáticas. Para problemas de regressão ou estimação, a função de ativação da última camada (camada de saída) pode ser linear, permitindo que a rede retorne quaisquer valores reais na sua saída. Em geral utiliza-se a função puramente linear, dada pela equação (2.4).

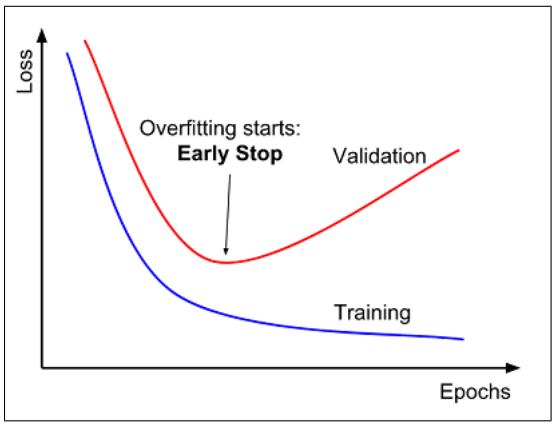
$$f(x) = x \tag{2.4}$$

2.2.5 Early Stopping

No treinamento, é importante ainda que se observe o critério de parada, pois o excesso de iterações de treino faz com que a rede perca sua capacidade de generalização, o que aumenta o erro para amostras desconhecidas. Dessa forma, utiliza-se em geral, a *early stopping* ou *parada antecipada*, para que o treinamento venha cessar antes que o erro de validação comece a aumentar, de acordo com a Figura 2.5. Vale salientar que em geral essa parada será realizada de forma automática, porém poderá ser efetuada de forma manual caso o usuário monitore em tempo real o treinamento e cesse o treinamento. Quando tratamos de redes neurais artificiais, deparamos com alguns conceitos como *epoch* ou épocas. Uma época é igual uma passagem para

frente e uma passagem para trás de todos os exemplos de treinamento. Já o tamanho do lote é o número de exemplos de treinamento em uma passagem para frente e uma trás. Quanto maior o tamanho do lote, mais espaço de memória será necessário. Já o número de iterações é igual ao número de passes, cada passe usando o número de exemplos (tamanho do lote). Para ser claro, uma passagem é igual uma passagem para frente + uma passagem para trás (não contamos a passagem para frente e para trás como duas passagens diferentes). Se usarmos poucas *epochs*, tendemos a ter problemas de *underfitting* (não aprender tudo o que pudermos com os dados de treinamento). Se usarmos muitas *epochs*, tendemos a ter o problema de *overfitting* ("aprender demais", ou seja, ajustar o "ruído" nos dados de treinamento, e não o sinal). Ao treinar uma rede neural, geralmente se está interessado em obter uma rede com desempenho ideal de generalização. A parada precoce é amplamente usada porque é simples de entender e implementar e foi relatada como sendo superior aos métodos de regularização em muitos casos. Usar *early stopping* significa que, no final de cada *epoch* devemos calcular a precisão da classificação nos dados de validação. Quando a precisão parar de melhorar, terminamos o treinamento. Ou seja, a parada antecipada é um método que interrompe o treinamento quando o desempenho do modelo para de melhorar nos dados validação, evitando o treinamento por épocas excessivas, além de evitar um super ajuste dos dados de treino (TITO, 2020; BOOK, 2019). O *avg loss* pode ser entendido como a *função de perda*, a qual é usada para quantificar a diferença entre os dados observados e os valores previstos. A minimização da função de perda é uma maneira de estimar os parâmetros do modelo. Pode ser entendida como a diferença entre os valores de rótulo de treinamento e a previsão feita pelo modelo.

Figura 2.5 – *Early-Stopping*.

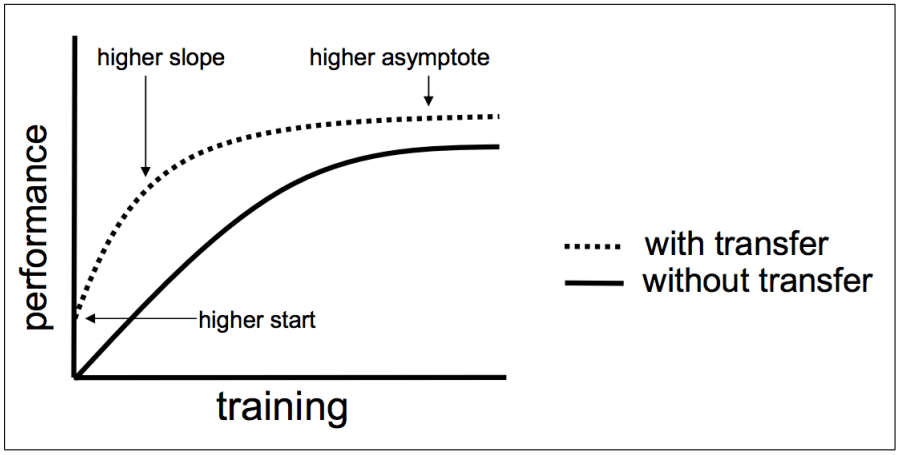


Fonte: Tito (2020).

2.2.6 Transfer Learning

Todas grandes redes neurais que usam grandes *datasets*, podem demorar dias a semanas para serem treinadas mesmo com GPU/TPU e, por isso fica inviável em muitos casos treinar uma CNN (Convolutional Neural Network) do zero. É ai que entra o *Transfer Learning* (TL) ou *Transferência de Aprendizado*. Essa técnica permite fazer uso de um conjunto de pesos iniciais já definidos para modelos personalizados de detecção. Em uma CNN as primeiras camadas de convolução apresentam as informações mais básicas de uma imagem no que tange o processo de treinamento, enquanto que as últimas camadas do modelo CNN apresentam características mais detalhadas do treinamento de objetos específicos. A Figura 2.6 mostra o gráfico hipotético que descreve a resposta para treinamento de um sistema de treinamento com e sem TL no quesito desempenho. A aprendizagem por transferência acelera o processo, aproveitando as semelhanças entre as tarefas (como a detecção de bordas em imagens) e aplicando essas aprendizagens a uma nova tarefa, tornando o aprendizado de máquina mais viável economicamente.

Figura 2.6 – *Transfer Learning*.



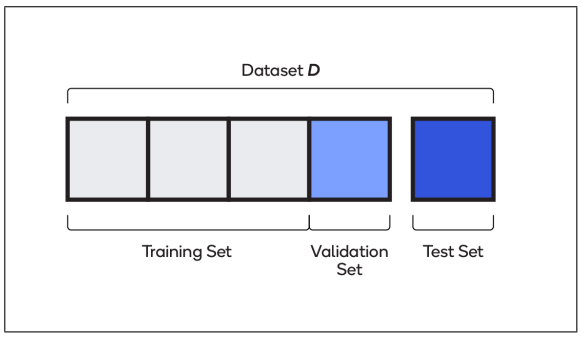
Fonte: Adaptado de Brownlee (2017).

2.2.7 Estrutura: Training - Validation - Test

Quando há uma base de dados de treinamento já definida, para se construir uma *validation base* para a validação do dados, em geral utiliza-se em torno de 20% do número de imagens do *dataset*. Dessa forma indica-se que caso um *dataset* seja composto de 1000 imagens de treinamento, neste caso ele possuirá cerca de 200 imagens de validação. Da mesma forma, em relação à *test base*, para testes de inferência o usuário pode aplicar o número de imagens se-

gundo sua demanda particular (em geral também 20% para ampla análise), como exemplo na Figura 2.7.

Figura 2.7 – Estrutura Proporcional de Projeto.



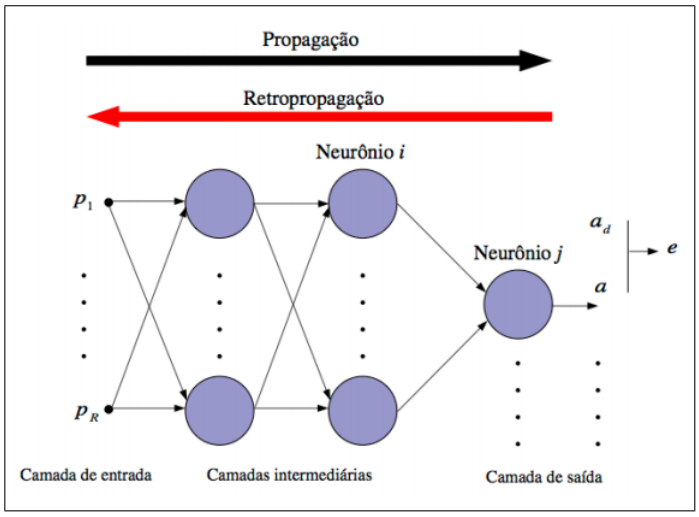
Fonte: Adaptado de Qualcomm (2020).

2.3 Aprendizado

Para Haykin (2005), a aprendizagem de uma RNA define-se como um processo onde os parâmetros livres são adaptados através de um processo de estimulação pelo ambiente em que a rede está inserida. Assim, o tipo de aprendizagem a ser utilizado é determinado levando em consideração a maneira pela qual a modificação dos parâmetros ocorre. De acordo com Miranda, Freitas e Faggion (2009), o treinamento de uma RNA condiz em processos iterativos de ajustes aplicados aos pesos. Considera-se que o aprendizado só ocorre quando a rede neural atinge uma solução generalizada para um determinado problema. Assim, o termo "treinar uma rede neural"consiste em *ajustar os pesos sinápticos* de forma que o vetor de saída coincida com um valor desejado para cada vetor de entrada. Como citamos anteriormente dos estudos de Bishop (1997), as RNA possuem dois tipos principais de aprendizado: supervisionado e não supervisionado. Porém devemos destacar que no aprendizado supervisionado é possível fornecer à RNA a saída desejada em relação a um padrão de entrada. Dessa forma, compara-se a saída da rede neural com a saída desejada, obtendo-se o erro. Feito isso, ajusta-se os pesos de forma a minimizar o erro. Já no aprendizado não supervisionado não existe um supervisor acompanhando o processo de aprendizagem, assim a RNA deve procurar algum tipo de correlação ou redundância nos dados de entrada. Em se tratando de aprendizado, Braga, Carvalho e Ludermir (2007) também destacam que o algoritmo mais conhecido e utilizado para o treinamento de redes neurais do tipo MLP é conhecido como o outrora citado *backpropagation*. O *backpropagation* é um algoritmo supervisionado que utiliza pares para que por meio de um mecanismo de correção de erros ajuste os pesos da rede. Esse algoritmo consiste em duas fases chamadas

de *forward* e *backward* (Figura 2.8). A primeira (*forward*) é utilizada para definir a saída da rede para um dado padrão de entrada. A segunda (*backward*) utiliza a saída desejada e a saída fornecida pela rede para atualizar os pesos de suas conexões.

Figura 2.8 – Estrutura de RNA *backpropagation*.



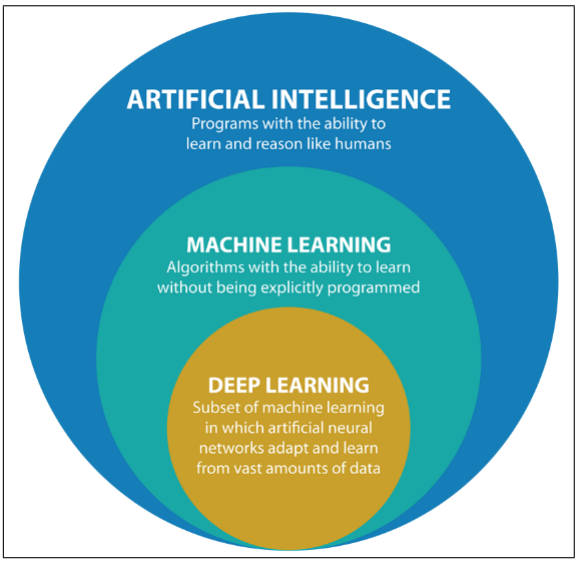
Fonte: Cintra (2018).

Theodoridis e Koutroumbas (2008) relata que existe uma alternativa ao *backpropagation* conhecida como *treinamento de batelada*. Neste algoritmo, o erro para todas as amostras é somado e utilizado para a correção dos pesos, o que é diferente do *backpropagation* apresentado, onde a atualização dos pesos ocorre após a apresentação de cada amostra. De acordo com o autor as características do *backpropagation* são:

1. Inicialização aleatória dos pesos da rede;
2. Apresentação do primeiro vetor de entradas do conjunto de treinamento;
3. Propagação do vetor de entrada pela rede para obter uma saída (*feed-forward*);
4. Calculo do sinal de erro por meio da comparação da saída atual com a saída desejada;
5. Propagação do sinal de erro de volta pela rede (*backpropagation*);
6. Ajuste dos pesos para a minimização do erro geral. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe;
7. Repetição dos passos 2-6 com o próximo vetor de entrada até que o erro geral seja satisfatoriamente pequeno.

A ciência da visão computacional avançou exponencialmente com o advento das GPU(s) com capacidade de processamento paralelo, no qual as redes neurais passaram a apresentar maior quantidade de camadas, dando início à nova era da Machine Learning. Os termos ML (*Machine Learning*) e DL (*Deep Learning*) explodiram junto com a AI (*Artificial Intelligence*). Todos eles fazem parte da evolução que possibilita que máquinas cada vez mais pensem como seres humanos, entretanto, não são a mesma coisa. Podemos compreender que um evolui a partir do outro, sendo que a ML e DL são pilares que sustentam a AI. De um modo bem simplista a Figura 2.9 mostra essa estrutura.

Figura 2.9 – Estrutura AI x ML x DL.



Fonte: Adaptado de Santana (2018).

2.3.1 ML - Machine Learning

De acordo com Salesforce (2018):

> "A Machine Learning ou *Aprendizagem de Máquina*, é o uso de algoritmos para organizar dados, reconhecer padrões e fazer com que computadores possam aprender com esses modelos e gerar *insights inteligentes* sem necessidade de pré-programação. Podemos dizer também que a ML é a área da ciência da computação que permite tornar a AI real. O conceito de AI surgiu faz muito tempo, já em 1956, mas na época ainda faltavam as tecnologias capazes de colocar a teoria em prática. Com o aprendizado de máquina, computadores puderam encontrar respostas sem que fossem especificamente programados para procurá-las. Os algoritmos de ML aprendem a partir dos dados a eles submetidos e, assim as máquinas são treinadas para aprender a executar diferentes tarefas de forma autônoma. Logo, ao serem expostas a novos dados, elas se adaptam a partir dos cálculos anteriores e os padrões se moldam para oferecer respostas confiáveis. O que isso quer dizer, na prática? Em vez de

> programar regras em um computador e esperar o resultado, com ML a máquina aprenderá essas regras de forma autônoma. A AI não é recente, assim como a ML também não é uma ciência nova. No início, suas aplicações eram limitadas pela falta de dados e de tecnologias capazes de processá-los de forma veloz e eficiente. Hoje, a grande quantidade de dados disponíveis e o avanço da computação permitiram o desenvolvimento de algoritmos muito mais complexos e rápidos, que trouxeram um novo impulso para ML. É o caso de *Deep Learning*."

2.3.2 DL - Deep Learning

De acordo com Salesforce (2018):

> "Redes neurais profundas são redes que possuem mais de uma camada intermediária, como as CNN, que são redes profundas (com mais de uma camada intermediária) que utilizam uma técnica chamada de convolução. O *Deep learning* ou *Aprendizagem Profunda*, é a parte do aprendizado de máquina que por meio de algoritmos de alto nível imita a rede neural do cérebro humano (RNA). No entanto, o seu aprendizado não se assemelha totalmente ao neurônio humano pois se utiliza de operações matemáticas. A principal diferença entre o DL e MLP são basicamente no quesito da quantidade de camadas intermediárias, pois a estrutura básica permanece a mesma, sendo que o DL poderá possuir centenas de camadas convolucionais (CNN). Para chegar ao nível de aprendizagem profunda mais avançado, o princípio das RNA foi desenvolvido para suportar camadas discretas (ocultas), conexões e direções de propagação de dados. Assim, os dados são submetidos a várias camadas de processamento não-lineares que simulam a forma de pensar dos neurônios. De forma simplificada, podemos dizer que o DL são esses algoritmos complexos construídos a partir de um empilhamento de diversas camadas de "neurônios", alimentados por quantidades imensas de dados, que são capazes de reconhecer imagens e fala, processar a linguagem natural e aprender a realizar tarefas extremamente avançadas sem interferência humana. A principal aplicação dos algoritmos de DL são as tarefas de classificação, em especial, reconhecimento de imagens. Podemos destacar que a importância do DL se encontra no maior acerto da classificação de imagens, tornando-se difícil de ser ultrapassada por outros modelos de ML. Além disso apresenta um alto desempenho para projetos com cálculos em séries temporais."

2.3.3 Aplicações

As RNA possuem capacidade de solucionar diversos problemas derivados de diversas áreas. Fazem parte da área conhecida como *smart systems* ou sistemas inteligentes (ZADEH, 1992). Podem ser aplicadas em sistemas de controle para tratamento de água, no qual envolve processos físicos e químicos não-lineares que apresentam dificuldades de mapeamento por métodos convencionais de controle. Na medicina, podem ser usadas para classificar e prever câncer baseado no perfil genético de um dado individuo ou pesquisas direcionadas para diagnósticos

de doenças do coração (CIRESAN et al., 2013). Na biologia por exemplo é possível encontrar aplicações usando RNA com o objetivo de identificar espécies de morcegos baseados em seu sinal de localização (bio-sonar) emitido durante o voo (ARMITAGE; K.OBER, 2010). Em economia e finanças há dificuldades de se solucionar problemas de natureza não-linear, o que vem a criar uma forte tendência de crescimento no uso de RNA nas problemáticas do setor. Podemos destacar as habilidades das RNA para classificação de padrões de voz para identificação de emoções humanas. Pode ser aplicada também à classificação de padrões em fontes de corrente harmônicas em sistemas de distribuição de energia. Na área da indústria de alimentos, um exemplo que também foi beneficiado com a aplicação de RNA, foi a classificação de diferentes variedades de chá, bem como pesquisas de Nazario (2007), que empregou RNA combinada com técnicas de ultrassom para classificar e identificar o leite adulterado pela adição de gordura ou água. Na área aeroespacial, temos RNA para métodos de modelagem e estratégias de controle para operações com UAV - *Unmanned Aerial Vehicles* (Veículos Aéreos Não Tripulados). Poderíamos citar avanços na área do entretenimento, onde a exploração das RNA pela indústria do cinema, especialmente na arte cinematográfica em filmes de ficção científica com uso de sistemas inteligentes (efeitos especiais) e de robótica ou mesmo em sites de *streaming de video* que utilizam algoritmos para indicar séries e filmes de acordo com o padrão de gosto do usuário.

2.4 Parâmetros de Desempenho

Podemos considerar os seguintes parâmetros de desempenho em classificação:

2.4.1 TP, TN, FP e FN

De acordo com (LAPIX, 2020), a primeira descrição que deve ser explanada são os parâmetros TP, TN, FP e FN, que são:

1. TP = True Positive (Verdadeiro Positivo).
2. FP = False Positive (Falso Positivo).
3. TN = True Negative (Verdadeiro Negativo).
4. FN = False Negative (Falso Negativo).

- TP – Quando o objeto encontra-se no *ground truth* (verdade fundamental) e o modelo foi capaz de detectá-lo. É considerado como TP quando é superior ao limiar IoU definido.

- FP – Quando o objeto encontra-se no *ground truth* e o modelo não foi capaz de detecta-lo. É considerado como FP quando é inferior ao limiar IoU definido..
- FN – Quando o modelo não gera nenhuma predição para esta marcação.
- TN – Não se aplica em detecção de objetos, pois não faz sentido treinar um modelo com um objeto em *background* (plano de fundo).

2.4.2 Precision x Recall

Precision (Precisão):

O TP é o acerto e o FP é o erro do algoritmo. O *precision* mostra a proporção de positivos previstos corretamente. O cálculo *precision* é dado pela equação (2.5). Podemos assumir então que o *precision* é a % da classificação no modelo de uma imagem desconhecida.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.5}$$

Recall (Revocação):

O TP é o acerto e o FN é o erro do algoritmo. O *recall* mostra a proporção de positivos identificados corretamente. O cálculo é dado pela equação (2.6). Podemos assumir então que o *recall* é a % de classificação no modelo de uma imagem conhecida.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.6}$$

2.4.3 AP - Average Precision

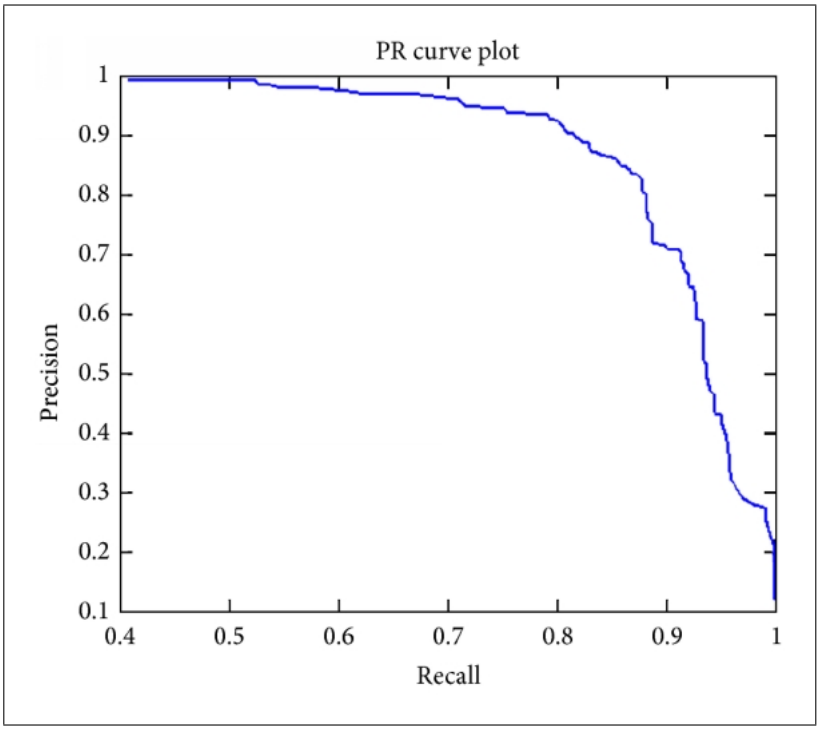
O Average Precision é uma relação entre *precision* x *recall*. Sempre há uma compensação entre as duas métricas, ou seja, uma troca entre *precision* e *recall*, onde aumentar um implica em diminuir o outro. O AP é a métrica utilizada para medir a acurácia de uma classe. Comumente usada como parâmetro de avaliação em detectores como de modelos das famílias: YOLO, SSD, RetinaNet e R-CNN. O YOLO aplica a métrica AP com o *backbone* Darknet - Darknet-19 para YOLO v2 e Darknet-53 para o YOLO v3 e v4. Obs.: Em sistemas de detecções em imagens, quando o *ground truth* está presente na imagem e o modelo não conseguiu detectar o objeto, classifica-o como FN. Já o TN não é útil para a detecção de objetos, portanto ignoramos esse parâmetro (pois ela indica todas as partes da imagem onde não foram previsto nada). Portanto, a definição geral para a precisão média (AP) é encontrar a área sob a curva

de recuperação de precisão. É definido como a área sob a curva de recuperação de precisão (curva PR - *Precision Recall*), onde o eixo *x* é o *recall* e o eixo *y* é o *precision*, também pode ser chamada de *curva AP* (Figura 2.10).

2.4.4 mAP - Mean Average Precision

Os valores de *recall* aumentam à medida que diminui os valores de FN. Entretanto, o *precision* tem um padrão em "zigue-zage", ela sobe com o aumento de TP e desce com o aumento de FP. Por causa desse "zigue-zage"torna-se difícil a comparação entre diferentes modelos em cima de um *dataset* ou conjunto de dados. É por isso que a AP é uma métrica numérica que nos ajuda a comparar diferentes modelos. Na prática, AP é a precisão média em todos os valores de *recall* entre 0 e 1. Portanto, os valores de *precision* e *recall* são para cada classe de cada imagem do *dataset*. Cada classe possui um valor de AP em um *dataset*. Assim concluímos que a mAP (Mean Average Precision) é a média dos AP entre todas as classes (LAPIX, 2020).

Figura 2.10 – Curva PR.



Fonte: Adaptado de Liu (2018).

2.4.5 Threshold

O parâmetro *threshold* pode ser traduzido como limiar ou limite, no qual é usado para encerrar o teste de validação em RNA. O valor determina quantas vezes seguidas o erro do

conjunto de validação pode piorar antes que o treinamento seja encerrado. Ou seja, utilizado como critério de parada, o que especifica o limite para as derivadas parciais da função de erro.

2.4.6 IoU

O IoU ou *Intersection over Union* ou Interseção sobre União é uma métrica de avaliação usada para medir a precisão de um detector de objeto em um conjunto de dados específico. Frequentemente vemos essa métrica de avaliação usada em desafios de detecção de objetos, como o popular desafio *PASCAL VOC*. A intersecção sobre a união é simplesmente uma métrica de avaliação . Qualquer algoritmo que forneça caixas delimitadoras previstas como saída pode ser avaliado usando IoU. O IoU é uma parâmetro de treinamento e, também referenciado como índice de *Jaccard.* A fim de aplicar IoU para avaliar um detector, precisamos:

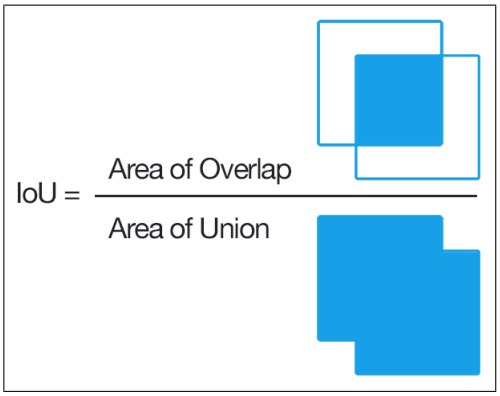
- A *bounding box* da *ground truth.*
- A *bounding box* da previsão do modelo.

Ele quantifica a similaridade entre duas *bounding boxes*. A interseção sobre a união pode ser determinada por meio da equação da Figura 2.11, onde verifica-se que é simplesmente uma razão. No numerador calcula-se a área de sobreposição entre a *bounding box* da *ground truth* e a *bounding box* de previsão. No denominador calcula-se o domínio da união. Dividir a área de sobreposição pela área de união resulta na pontuação final IoU. Se a previsão for perfeita o IoU será = 1, e se falhar completamente o IoU será = 0. Podemos definir um limiar (*threshold*) para o IoU para determinar se a detecção do objeto é válida ou não. Limites IoU sendo fixado em 50% ou 75%, são chamados respectivamente de AP_{50} e AP_{75}. Quando for esse o caso, é simplesmente o valor AP com o limite IoU fixado nesse valor. No caso de detecção de objeções, isso é feito alterando o *cutoff* ou corte de pontuação para o IoU fixo desejado. Qualquer detecção com uma pontuação superior ao *cutoff* é tratada como um TP, e com pontuação inferior como um FP.

2.4.7 Tempo de Processamento

É o tempo estimado para processo completo de classificação/detecção e/ou reconhecimento. O tempo de processamento pode ser de vital importância a depender de sua aplicação. Em sistemas onde o modelo necessita de rápida resposta, como sistemas de autenticação e de controle de tráfego, o fator temporal é fundamental na determinação da segurança e da qua-

Figura 2.11 – Cálculo de Interseção sobre União - IoU.



Fonte: Adaptado de Kovashka (2018).

lidade perceptiva dos usuários. Já em sistemas de entretenimento, modelos mais lentos são aceitáveis em aplicações, uma vez que não demandam uma resposta imediata para o usuário.

2.5 Algoritmo YOLO

Uma das ferramentas de visão computacional que tem ganhado atenção nos últimos anos é o YOLO - You Only Look Once. Após o seu lançamento em 2015, o YOLO foi logo reconhecido como uma técnica inovadora, pois por meio de uma abordagem totalmente nova foi capaz de obter uma precisão igual ou superior ao dos outros métodos de detecção de objetos da época, porém com uma velocidade de detecção muito superior. Foi desenvolvido por Joseph Redmon e Ali Farhadi em 2015, durante o doutorado de ambos. É um método de detecção de objetos de passada única (*single pass*) que utiliza uma rede neural convolucional como extrator de características (*features*). Diferente de algoritmos anteriores de detecção de objetos, como R-CNN ou Faster R-CNN, ele apenas precisa olhar a imagem uma única vez antes de enviá-lo para a rede neural. Por isso ele recebe esse nome You Only Look Once, que significa "Você apenas olha uma vez". E devido a essa característica, foi capaz de conseguir uma velocidade na detecção muito maior do que as técnicas concorrentes, sem perder acurácia (GRANATYR, 2020). Ao longo dos últimos anos alguns algoritmos de destaque foram desenvolvidos na área de classificação/detecção em imagens, tais como:

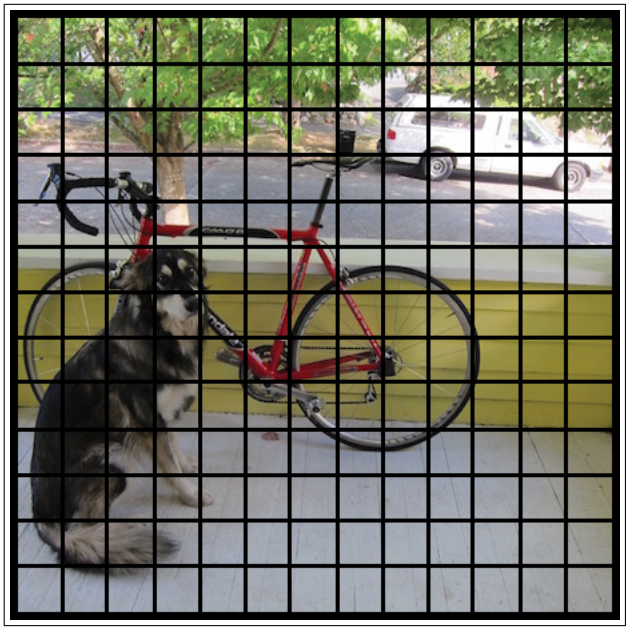
- **HC**: *Haar Cascades* (CUIMEI et al., 2017);
- **HOG**: *Histogram of Oriented Gradients* (SURASAK et al., 2018);
- **CNN**: *Convolutional Neural Networks* (ALOYSIUS; M, 2017);

- **R-CNN**: *Region Based Convolutional Neural Networks* (ABUSHAHMA et al., 2019);
- **SPP-net**: *Spatial Pyramid Pooling* (HE et al., 2015);
- **Fast R-CNN**: *Fast Region Based CNN* (GIRSHICK, 2015);
- **Faster R-CNN**: *Faster Region Based CNN* (GAVRILESCU et al., 2018).

Até o presente momento há 4 versões oficiais do YOLO: v1, v2, v3 e v4 (v5 em fase de testes). O algoritmo divide a imagem em um grid de *S x S* células, de acordo com a versão. Exemplo na Figura 2.12.

- Versão v1 - grid 7 x 7.
- Versão v2 - grid 13 x 13.
- Versão v3 - grid 19 x 19.
- Versão v4 - grid 19 x 19.

Figura 2.12 – Imagem em grid 13 x 13.

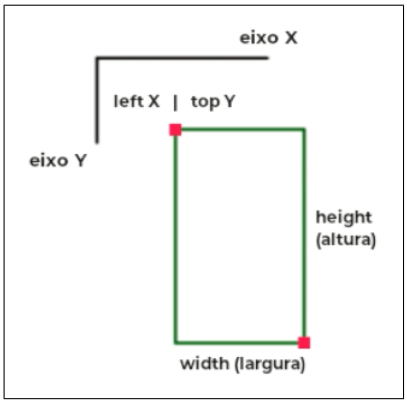


Fonte: Pjreddie (2019).

2.5.1 Arquitetura YOLO v1

O YOLO v1 é a primeira versão dessa arquitetura e é considerado uma nova poderosa abordagem para detecção de objetos extremamente rápida e inovadora. Os modelos iniciais possuíam capacidade de detecção em tempo real com processamento de 45 FPS. Posteriormente o modelo Fast YOLO v1 - uma versão melhorada - atingiu o surpreendente processamento de 155 FPS. Segundo Redmon et al. (2016), em comparação com os sistemas de última geração, o YOLO apresenta mais erros de localização, porém é menos provável de prever FP em segundo plano (*background*). Outra vantagem, é a capacidade de generalização das detecções, o que o faz superior aos sistemas DPM e R-CNN. Tem sido atualmente a arquitetura de rede neural mais avançada para detecção de objetos, com amplo crescimento em aplicações nos setores de carros autônomos, robótica e cyber-segurança. A classificação de imagens tenta prever qual objeto a imagem contém, também chamado de *classification* (classificação), por exemplo, ele prediz se é um gato ou um cachorro em uma imagem. Se a imagem tiver tanto gato quanto cachorro, o modelo vai conseguir prever caso haja treinamento de classificadores multi-classe. A ação de identificar a localização de um objeto em uma imagem é chamado de *localization* (localização). Portanto, pode-se considerar que: Detecção de Objetos = *Classification* + *Localization*. O retângulo contendo a localização do objeto é chamado de *bounding box* (caixa delimitadora). Para descrever a *bounding box* são necessários 4 variáveis de localização + 1 variável de classe, conforme Figura 2.13.

Figura 2.13 – Esquema de uma *bounding box* YOLO v1.



Fonte: Adaptado de Redmon et al. (2016).

1. Número que representa a *label* (rótulo) da classe.

2. Coordenada x de início (*left* X).

3. Coordenada y de início (*top* Y).

4. Largura da caixa (*width*).

5. Altura da caixa (*height*).

Quando compara-se com outros algoritmos de detecção de objetos que trabalham com multi processamentos (ex. R-CNN ou Fast R-CNN), o YOLO processa a imagem uma vez (mono processamento) e envia para a RNA, e assim a rede detecta múltiplas *bounding boxes*, cada uma contendo probabilisticamente um objeto. Antes da versão v1, sistemas de detecção de objetos faziam a detecção por meio da divisão da imagem em várias partes e depois em cada pedaço da imagem se executava um classificador. Nesses sistemas, é necessário executar o mesmo classificador milhares de vezes sobre a mesma imagem. A abordagem convencional até então usa o conceito de *slidind window* (cada janela roda um classificador), que pode ser funcional, mas é extremamente lenta. Se a intenção é implementar uma técnica para ser usada em tempo real, então necessita-se de um computador muito potente devido à alta requisição de processamentos. No caso da versão v1, é treinado uma única rede neural para fazer a detecção completa na imagem uma única vez. Assim, é capaz de produzir todas as *bounding boxes* e respectivas probabilidades dessas detecções em um único *single pass*. Para maior desempenho, o YOLO v1 pode usar um *framework* chamado Darknet, e sobre ele podemos citar:

- O YOLO utiliza-o nativamente apartir do v2.
- É um *Framework* de DCNN (open source).
- Escrito na linguagem C.
- Correspondente ao PyTorch e TensorFlow para DL.
- Suporta os mais recentes CPU/GPU.
- Criado por Redmon e Farhadi (2017).

O algoritmo permite que cada uma dessas células seja responsável por fazer a predição de até 5 *bounding boxes*, para caso haja mais de um objeto naquela célula. Ao contrário das atuais arquiteturas, isso permite ao YOLO ter a capacidade de detectar pequenos objetos em uma imagem. Podemos destacar recursos como:

- **Pontuação de confiança** - P_c: O algoritmo retorna uma pontuação de confiança, ou seja, a probabilidade que aquela *bounding box* contém um objeto. Vale ressaltar que essa pontuação não diz respeito de que tipo de objeto contém ali (classe), e sim ao formato da caixa em si. A Figura 2.14 mostra que quanto maior a confiança, mais grossa a linha, nesse caso 95% de probabilidade de ser um cachorro.

- **Previsão de classe** - C: Para cada *bounding box*, a célula faz a previsão de uma classe, uma vez fornecido o valor de pontuação de confiança (probabilidade) para cada uma das classes treinadas. Essa imagem contida na caixa é recortada e enviada para um CNN onde ela processará a previsão de qual classe esse objeto pertence.

- **Combinação**: A pontuação de confiança + previsão de classe são combinados em uma pontuação final. Por exemplo, na Figura 2.15 a caixa mais grossa amarela possui aproximadamente 85% de certeza que ali contém um cachorro.

- **Ajuste Fino em Threshold**: Com uma grade 13 x 13 há 169 células. Para cada uma dessas células são detectadas 5 *bounding boxes*, o que resulta em 845 caixas no total por imagem na versão v2. A maioria dessas caixas terá um valor de confiança extremamente baixo (linhas finas), por isso geralmente são consideradas apenas as caixas cuja pontuação de combinação seja de pelo menos 30% ou seja 0.3 de parâmetro *threshold*. O resultado final da predição após o ajuste fino com threshold é mostrado na Figura 2.16, no qual o melhor valor de combinação da caixa para cada classe é evidenciado.

Cada *bounding box* pode ser representada a partir de descritores conforme a Figura 2.17:

- 3 descritores de localização $b_x b_y$, b_w e b_h.

- 1 descritor de predição de confiança ou probabilidade P_c.

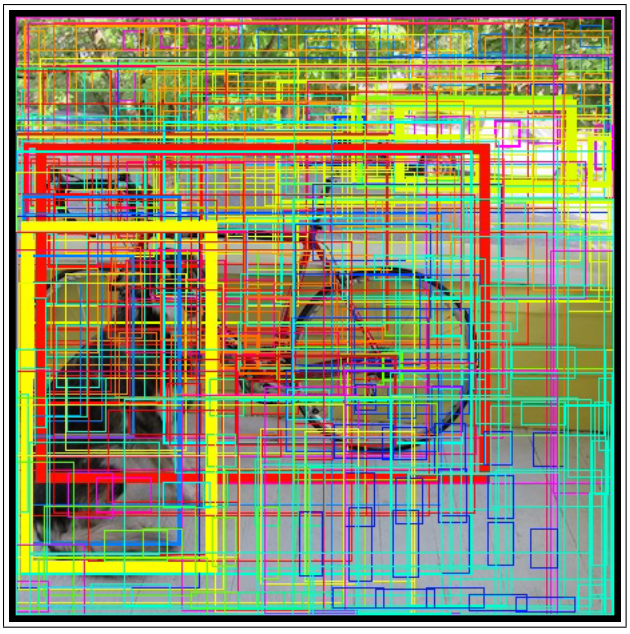
- 1 descritor de predição de classe C.

A versão v4 foi treinada em um dataset chamado COCO que possui 80 classes de objetos. Em um grid 19 x 19 serão 1805 *bounding boxes*. A estrutura de redução para *grid* se dá conforme a Figura 2.18. O *encoding* é o formato dessa estrutura com (19 x 19 x 5 x 85), no qual em cada célula temos 5 *boxes* na horizontal e 85 colunas (5 variáveis para descritores + 80 espaços para probabilidade de cada classe).

Figura 2.14 – Pontuação de Confiança.

Fonte: Pjreddie (2019).

Figura 2.15 – Combinação: Pontuação de Confiança + Previsão de Classe.

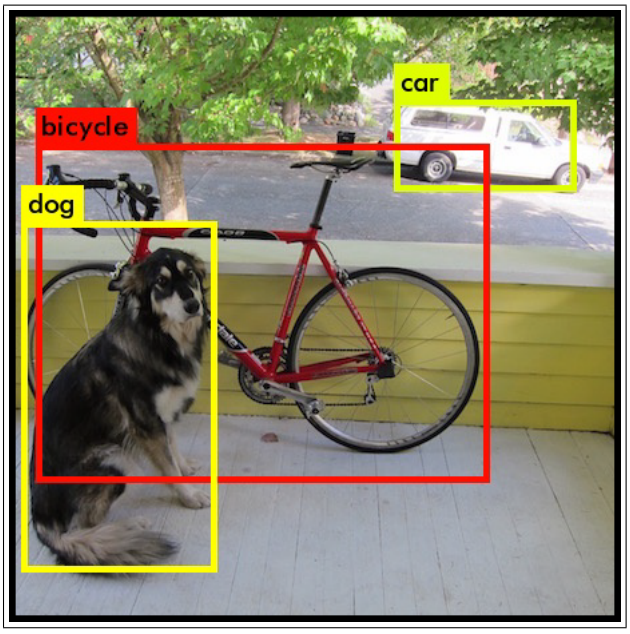


Fonte: Pjreddie (2019).

2.5.1.1 Non-Max Supression

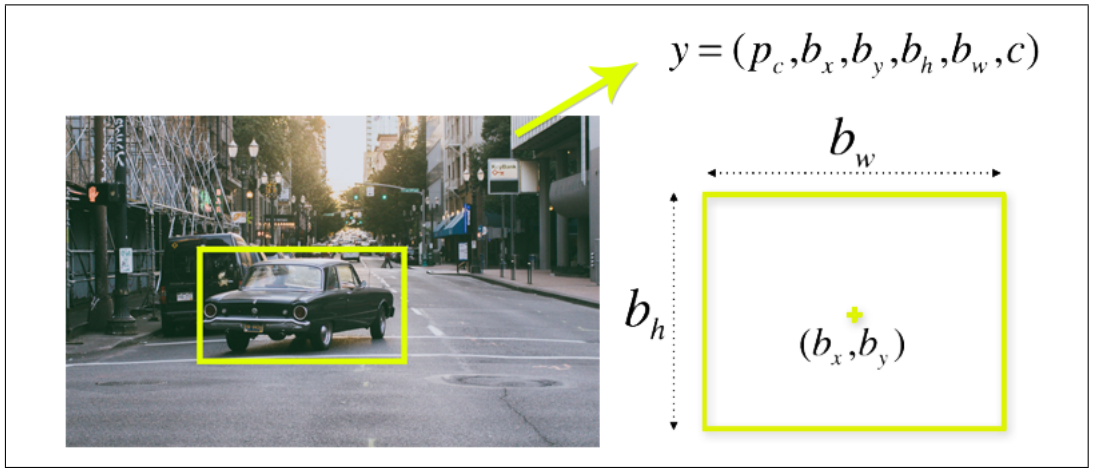
A maioria das células não vão conter um objeto, portanto, é necessário obter o valor P_C que servirá para remover caixas com baixa probabilidade de conter um objeto e também caixas que possuem área compartilhada conforme mostra a Figura 2.19. O modelo é capaz de encontrar várias *bounding boxes* para o mesmo objeto. A NMS (*Non-Max Supression*) ajuda a

Figura 2.16 – Predição Final com Threshold.



Fonte: Pjreddie (2019).

Figura 2.17 – Descritores para *bounding box*.

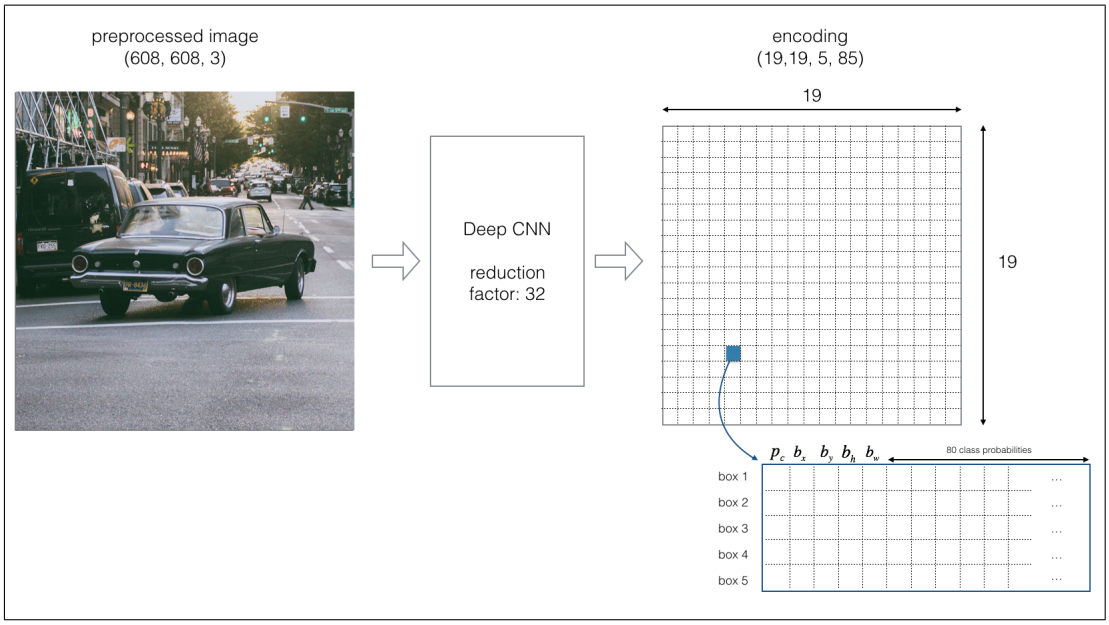


Fonte: Swiezewski (2020).

evitar detecção repetida da mesma instância. Depois de obter um conjunto de *bounding boxes* correspondentes para a mesma categoria de objeto:

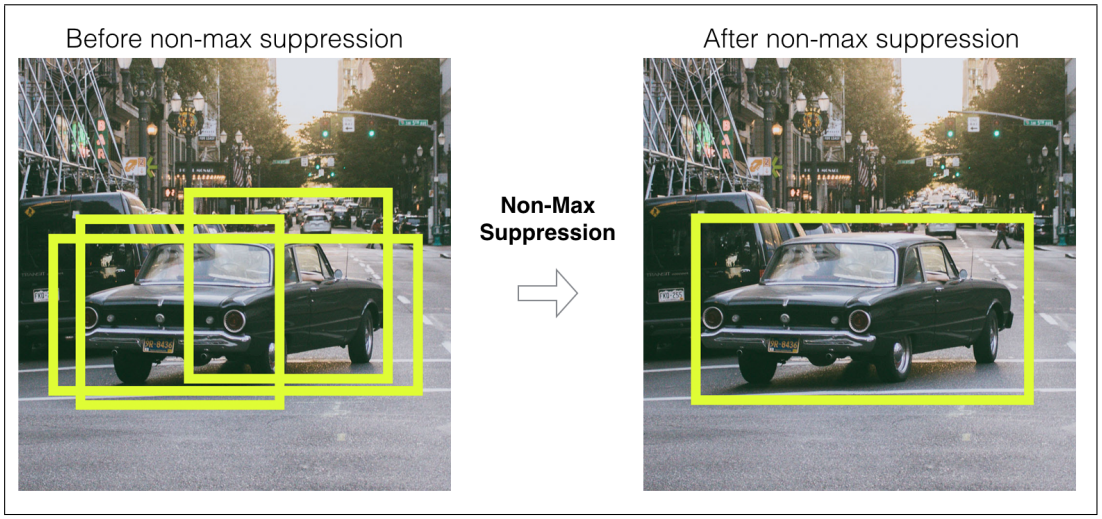
1. Classifica-se todas as *bounding boxes* por pontuação de confiança P_C.
2. Descarta-se as caixas com baixa pontuação de confiança.
3. Seleciona-se avidamente aquele com a pontuação mais alta.

Figura 2.18 – Redução para *Grid*.



Fonte: Swiezewski (2020).

Figura 2.19 – Non-Max Supression.



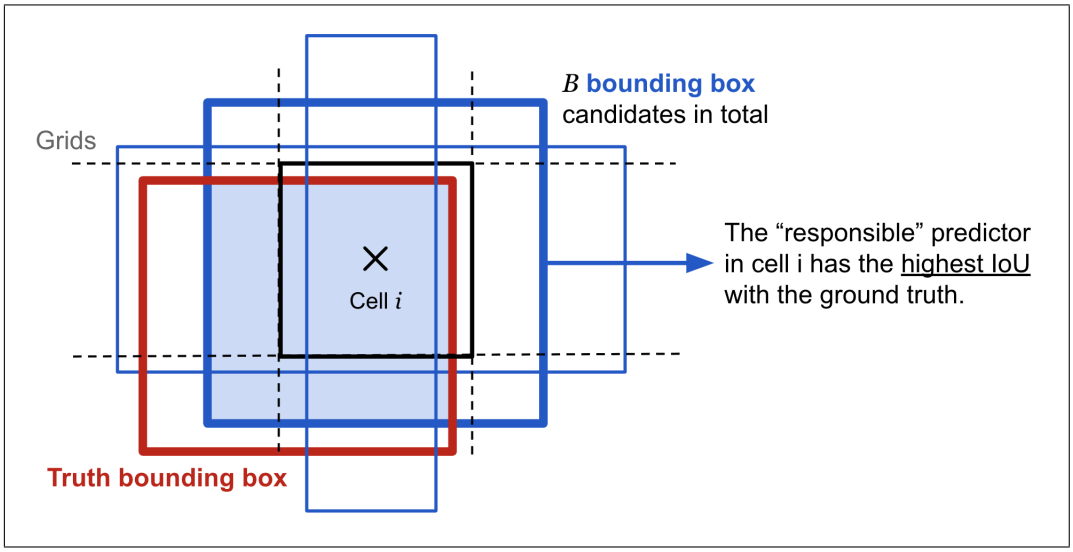
Fonte: Swiezewski (2020).

2.5.1.2 Anchor Boxes

As *anchor boxes* aparecem a partir da versão v2 do YOLO e são retângulos de proporções pré-definidas usados para ter maior correspondência entre as *bounding boxes* esperadas e as previstas. Possuem tamanhos iniciais (largura, altura) próximos aos tamanho dos objetos e são redimensionados para o tamanho do objeto usando algumas saídas da rede neural. A rede neural não deve prever o tamanho final do objeto, mas apenas ajustar o tamanho da *boun-*

ding box mais próxima ao tamanho pré-definido do objeto (Figura 2.20). O uso dessa técnica inovadora torna o processo de detecção mais otimizado.

Figura 2.20 – Anchor Box para aumento de IoU.



Fonte: Adaptado de Weng (2020).

2.5.1.3 Funcionamento

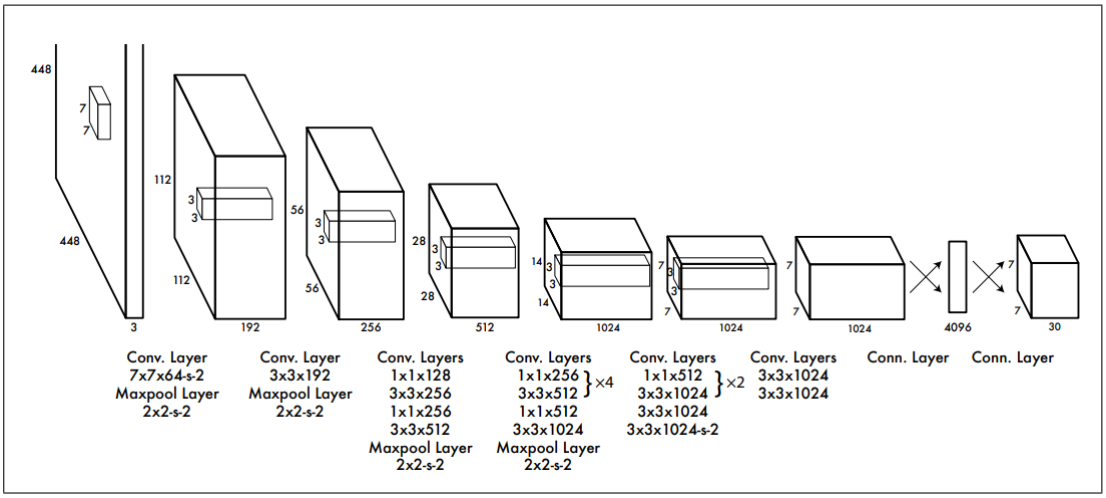
Em YOLO usa-se o conceito de detecção unificada. Uma única CNN prevê simultaneamente no treinamento várias caixas delimitadoras (*bounding box*) e respectivas probabilidades de classe em uma única imagem, ao mesmo tempo que otimiza o desempenho da detecção. Ou seja, unificou-se as componentes separadas da detecção de objetos em uma única rede neural, usando-se recursos para prever cada caixa delimitadora e sua respectiva previsão. Isso significa que a rede neural raciocina globalmente sobre a imagem e todos os objetos na imagem. A arquitetura YOLO permite treinamento de ponta a ponta e velocidades em tempo real, enquanto mantém a precisão média alta. Segundo Redmon et al. (2016) o sistema divide a imagem de entrada em uma grade S x S. Se o centro de um objeto cai em uma célula da grade, essa célula da grade é responsável por detectar esse objeto. Cada célula de grade prevê B caixas delimitadoras e pontuações de confiança para essas caixas. Essas pontuações de confiança refletem a confiança do modelo de que a caixa contém um objeto e também o quão precisa ele pensa que a caixa está prevista. Formalmente, define-se confiança como $Pr(Object) * IoU_{pred}^{truth}$, onde Pr é a probabilidade do objeto pertencer a determinada classe. Se nenhum objeto existir nessa célula, as pontuações de confiança devem ser zero. Caso contrário, a pontuação de confiança (pred - predicted) será igual à IoU entre a caixa prevista e a *ground truth* (IoU_{pred}^{truth}). Cada caixa

delimitadora consiste em 5 previsões: x, y, w, h e confiança. As coordenadas (x, y) representam o centro da caixa em relação aos limites da grade da célula. A largura e a altura são previstas em relação à imagem inteira (w, h). Finalmente, a previsão de confiança representa o IoU entre a caixa prevista e verdadeira. Cada célula da grade também prevê as probabilidades da classe condicional C, $Pr(Class_i|Object)$. Essas probabilidades são condicionadas à célula da grade que contém um objeto. Prevê-se um conjunto de probabilidades de classe por célula da grade, independentemente do número de caixas. No momento do teste multiplica-se as probabilidades de classe condicional e as previsões de confiança de caixa individual, o que dá as pontuações de confiança específicas de classe para cada caixa. Essas pontuações codificam a probabilidade dessa classe aparecer na caixa e o quão bem a caixa prevista se ajusta ao objeto. O sistema modela a detecção como um problema de regressão. Ele divide a imagem em uma grade S x S e, para cada grade a célula prevê B caixas delimitadoras, a confiança para essas caixas e as probabilidades da classe C. Essas previsões são codificadas como um tensor S x S x (B x 5 + C). Para avaliar YOLO em PASCAL VOC (Visual Object Classes), usa-se S = 7, B = 2. PASCAL VOC tem 20 classes rotuladas, então C = 20. A previsão final é um tensor 7 x 7 x 30. Resumindo, podemos dizer: Probabilidade de Classe Condicional * Previsão de Confiança entre as Caixas Prevista e *Ground Truth* = Pontuação de Confiança, representada na equação (2.7).

$$\mathrm{Pr(Class_i|Object)} * \mathrm{Pr(Object)} * \mathrm{IoU}\ {}^{\mathrm{truth}}_{\mathrm{pred}} = \mathrm{Pr(Class_i)} * \mathrm{IoU}\ {}^{\mathrm{truth}}_{\mathrm{pred}} \tag{2.7}$$

O sistema YOLO v1 foi implementado como uma CNN e avaliado sobre os dados do PASCAL VOC. As camadas convolucionais iniciais extraem as características da imagem. As camadas completamente conectadas preveem as probabilidades e coordenadas de saída. A arquitetura foi baseada no modelo GoogLeNet para classificação de imagens, sendo projetado com 24 camadas convolucionais seguidas por 2 camadas totalmente conectadas. Nessa versão usa-se camadas de redução 1 x 1 seguidas por camadas convolucionais 3 x 3. No projeto foram treinadas as camadas convolucionais na tarefa de classificação ImageNet com metade da resolução (imagem de entrada 224 x 224) e nos testes de detecção dobrou-se a resolução para imagens 448 x 448. A versão Fast YOLO v1 usa 9 camadas convolucionais e menos filtros nessas camadas, mas mantém os parâmetros de treinamento e teste do YOLO v1. A saída final da rede é o tensor $7 \times 7 \times 30$ de previsões. A arquitetura pode ser vista na Figura 2.21.

Figura 2.21 – Arquitetura YOLO v1.



Fonte: Redmon et al. (2016).

2.5.1.4 Treinamento

Segundo Ren et al. (2016b), adicionar camadas convolucionais e conectadas a redes pré-treinadas pode melhorar o desempenho. A camada final prevê as probabilidades de classe e as coordenadas da *bounding box*. Normaliza-se a largura e a altura da caixa delimitadora pela largura e altura da imagem para que fiquem entre 0 e 1. Parametriza-se as coordenadas x e y da caixa delimitadora para serem compensações da localização de uma célula específica de grade, de modo que também sejam delimitados entre 0 e 1. Usa-se uma função de ativação linear para a camada final e todas as outras camadas usam a ativação linear retificada (Rectified Linear Unit - ReLU) na equação (2.8).

$$\phi(x) = \begin{cases} x, & se \quad x > 0 \\ 0.1x, & caso\ contrário \end{cases} \tag{2.8}$$

Apesar da possibilidade de otimização do erro quadrático na saída, essa prática não maximiza a precisão. Nesse intuito de melhorar a precisão, o YOLO v1 seta dois parâmetros para aumentar a perda das previsões de coordenadas *bounding box* (λ_{coord}) e diminui a perda de confiança (λ_{noobj}) em caixas que não contém objetos. O erro de soma quadrada pondera erros em caixas grandes e pequenas. A métrica de erro deve refletir que pequenos desvios em caixas grandes são menos importantes do que em caixas pequenas. Para solucionar prevê-se a raiz quadrada da largura e altura da *bounding box* em vez da largura e altura diretamente.

$$\lambda_{coord} = 5 \tag{2.9}$$

$$\lambda_{noobj} = 0.5 \tag{2.10}$$

De acordo com Redmon et al. (2016), o YOLO prevê várias caixas delimitadoras por célula da grade, e no treinamento deseja-se apenas um preditor de *bounding box* responsável para cada objeto. Ou seja, se temos 20 objetos, haverá 20 preditores após o treinamento para predizer um objeto com base no IoU mais alto, sendo que cada preditor prevê aspectos específicos de cada classe abordados no seu treinamento. A função de perda somente penaliza o erro de classificação se um objeto está presente naquela célula da grade. A função também penaliza o erro da *bounding box* se o preditor tiver o IoU mais alto. O YOLO v1 usa por padrão um treinamento com *batch size* = 64, *momentum* = 0.9 e um *decay* = 0.0005 com *learning rate* variável. Seu treinamento na base dados padrão é com o *dataset* PASCAL VOC 2007 e 2012 em 135 *epochs* sendo distribuído a *learning rate* da seguinte maneira:

1. 10^{-2} por 75 *epochs*.
2. 10^{-3} por 30 *epochs*.
3. 10^{-4} por 30 *epochs*.

De modo a evitar o fenômeno de *overfitting* (co-adaptação entre as camadas) usa-se uma camada de *dropout* = 0.5 após a primeira camada conectada. Para o parâmetro de *data augmentation* o modelo trabalha com *random scaling* e *translations* de até 20 % da imagem original e *saturation image* = 1.5 HSV. O YOLO é extremamente rápido no momento do teste, pois requer apenas uma única avaliação de rede, ao contrário dos métodos tradicionais baseados em classificadores. O *Non-Maximal Suppression* pode ser usado para corrigir detecções múltiplas (várias células detectam o objeto). Embora não seja crítico para o desempenho como é para R-CNN ou DPM, o NMS adiciona de 2 a 3 % em mAP. O modelo aprende a prever *bounding box* a partir de dados, ele se esforça para generalizar para objetos em relações de aspecto ou configurações novas ou incomuns. Em se tratando de *loss function*, um pequeno erro em uma grande *bounding box* geralmente é aceitável, mas um pequeno erro em uma pequena *bounding box* tem um efeito muito maior no IoU, sendo que a principal fonte de erro nesse sistema são as localizações incorretas.

2.5.1.5 Vantagens

- Extremamente rápido e preciso. Visto que a detecção de quadros é um problema de regressão, não precisa-se de um *pipeline* complexo. Simplesmente executa-se a rede neural em uma nova imagem no momento do teste para prever detecções. Redmon et al. (2016) testou a rede básica rodando a 45 FPS (quadros por segundo) sem processamento em lote em uma GPU Titan X.
- Possibilidade de reconhecimento de vídeo em tempo real.
- Pode-se trocar velocidade por precisão simplesmente alterando o tamanho do modelo, sem necessidade de novo treinamento, alterando-se o número das camadas de convolução.
- YOLO comete menos da metade do número de erros em *background* (segundo plano) em comparação com Fast R-CNN.
- YOLO aprende representações generalizáveis de objetos. Uma vez que é altamente generalizável, é menos provável que falhe quando aplicado a novos domínios ou entradas inesperadas.
- Capacidade de detectar objetos pequenos.
- Código de treinamento *open source*.
- Unificação dos componentes de detecção de objetos em uma única rede neural.
- Sistema de processamento paralelo das previsões de caixas delimitadoras e previsão de classes de forma simultânea.

2.5.1.6 Desvantagens (versão v2)

- Menos preciso que o F-CNN.

2.5.1.7 Comparativos com outros Algoritmos de Classificação/Detecção

- **DPM**

  De acordo com Felzenszwalb et al. (2010) o DPM (*Deformable Parts Models*) usa uma abordagem de janela deslizante para detecção de objetos, um pipeline separado para extrair características estáticas, classificar regiões, prever *bounding box* para regiões de alta

pontuação. No YOLO o sistema substitui todas essas partes divergentes por uma única CNN. A rede realiza a extração de características, previsão da *bounding box* e NMS simultaneamente. A arquitetura unificada leva a um modelo mais rápido e preciso do que o DPM e R-CNN. Segundo Dean et al. (2013), muitos esforços de pesquisa tem se concentrado em acelerar o pipeline do DPM. Aceleram a computação HOG, usam cascatas e computação baseada em GPU(s). No entanto, apresentam baixo desempenho em tempo real.

- **R-CNN**

  A R-CNN e suas variantes usam propostas de região em vez de janelas deslizantes para localizar objetos nas imagens. A Busca seletiva gera caixas delimitadoras potenciais, uma CNN extrai características, um SVM (*Support Vector Machine*) pontua as caixas, um modelo linear ajusta as caixas delimitadoras e a NMS elimina detecções duplicadas. Cada estágio deste pipeline complexo deve ser ajustado com precisão de forma independente e o sistema resultante é muito lento, levando mais de 40 segundos por imagem no momento do teste. De acordo com Redmon et al. (2016), o YOLO compartilha algumas semelhanças com R-CNN. Cada célula da grade propõe caixas delimitadoras potenciais e pontua essas caixas usando recursos convolucionais. No entanto, o sistema YOLO impõe restrições espaciais nas propostas de células da grade, o que ajuda a reduzir múltiplas detecções do mesmo objeto. O sistema também propõe muito menos *bounding boxes*, apenas 98 por imagem, em comparação com cerca de 2.000 da Busca Seletiva (UIJLINGS et al., 2013). Finalmente, o sistema inovador combina esses componentes individuais em um único modelo otimizado em conjunto.

- **Fast/Faster R-CNN**

  Detectores rápidos como o Fast e Faster R-CNN se concentram em acelerar o *framework* R-CNN, compartilhando computação (pipeline) e usam redes neurais para propor regiões em Busca Seletiva. Embora ofereçam melhorias de velocidade e precisão em relação ao R-CNN, ambos ainda possuem baixo desempenho em tempo real (REN et al., 2016a).

- **Comparação de desempenho**

  YOLO é um detector de uso geral que aprende a detectar uma variedade de objetos simultaneamente. Em vez de tentar otimizar componentes individuais com um grande pipeline

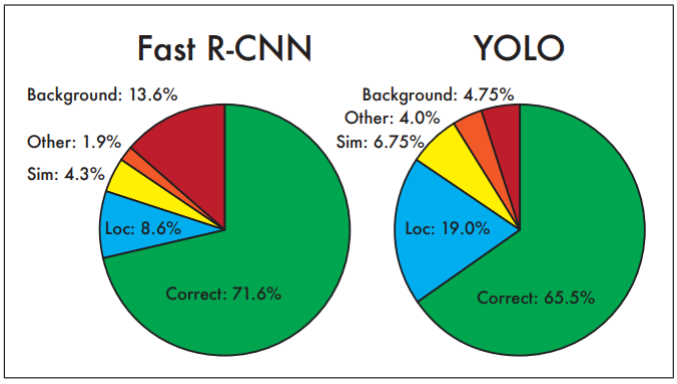
de detecção, ele elimina totalmente o pipeline e apresenta maior robustez. Além disso, em geral detectores para classes únicas, como rostos ou pessoas, podem ser altamente otimizados, uma vez que se lida com menos variação de características Viola e Jones (2001), o que pode ser grande foco de estudos com YOLO. Redmon et al. (2016) comparou o YOLO com outros sistemas de detecção em tempo real usando o PASCAL VOC 2007 e 2012, no intuito de entender as diferenças entre YOLO e as variantes de R-CNN (Fast R-CNN que é considerado método de última geração). O autor nos mostra que o YOLO supera o Fast R-CNN em detecção e redução de falsos positivos de fundo. O autor ressalta a capacidade generalista superior da arquitetura YOLO sob comparativos utilizando o indicador mAP.

Um dos grande objetivos da ciência da detecção de objetos é diminuir o tempo de detecção por meio de Pipelines em sistemas de tempo real. Em comparações com trabalhos de Sadeghi e Forsyth (2014) - um dos poucos autores que conseguiram produzir um sistema de detecção em tempo real de 30 FPS - utilizou-se o indicador mAP, no qual o Fast YOLO foi o método de detecção de objetos mais rápido no PASCAL, apresentando uma taxa de 52,7 % mAP a 63,4 %. Isso possibilitou ser aproximadamente duas vezes mais rápido que seu rival. Em comparativos de arquiteturas YOLO (versão v1) x VGG-16, efetuados por Redmon et al. (2016), a arquitetura VGG-16 oferece maior precisão em detecção, porém apresenta significativa lentidão em relação ao YOLO. Em comparativos de arquiteturas YOLO x DPM, a detecção acelera efetivamente sem sacrificar muito o mAP, porém perde desempenho em tempo real por um fator de 2 e apresenta precisão relativamente baixa (YAN et al., 2014). Em comparativos de arquiteturas YOLO x Fast R-CNN acelera o estágio de classificação do R-CNN, mas depende de uma Busca Seletiva (*Selective Search*) lenta por imagem para gerar caixa delimitadora, ou seja apresenta um mAP alto, porém a 0.5 FPS que não é aceitável para sistemas em tempo real (LENC; VEDALDI, 2015).

De acordo com Redmon et al. (2016), o recente Faster R-CNN substitui a Busca Seletiva por uma rede neural para propor caixas delimitadoras, semelhante ao trabalho de Erhan et al. (2013). Nos testes, seu modelo mais preciso atingiu 7 FPS, enquanto um menor e menos preciso é executado a 18 FPS. A versão VGG-16 do Faster R-CNN é 10 mAP superior, mas também é 6 vezes mais lenta do que o YOLO. O ZeilerFergus Faster R-CNN é apenas 2,5 vezes mais lento que o YOLO, mas também é menos preciso. O autor faz

comparativo entre YOLO x Fast R-CNN de última geração com o VOC 2007, usando a metodologia proposta por Hoiem, Chodpathumwan e Dai (2012) que se baseia em indicadores IoU. A Figura 2.22 mostra a divisão de cada tipo de erro em média em todas as 20 classes. YOLO se esforça para localizar objetos corretamente. Erros de localização são responsáveis por mais erros do YOLO do que todas as outras fontes combinadas. O Fast R-CNN comete muito menos erros de localização, mas muito mais erros de segundo plano. 13,6 % de suas principais detecções são falsos positivos que não contêm nenhum objeto. O Fast R-CNN tem quase 3 vezes mais probabilidade de prever detecções de fundo do que o YOLO. YOLO comete muito menos erros de segundo plano do que Fast R-CNN. Ou seja mostra a porcentagem de erros de localização e de plano de fundo nas N principais detecções de várias categorias (N = objetos nessa categoria), sendo o *Sim* a curva PR já descrita.

Figura 2.22 – Porcentagem de erros de localização e de plano de fundo.



Fonte: Redmon et al. (2016).

O YOLO pode ser combinado com outras arquiteturas. Redmon et al. (2016) combinou YOLO e Fast R-CNN no intuito de diminuir os erros de segundo plano (Background). O método consiste em que para cada caixa delimitadora que o R-CNN prevê, verifica-se se o YOLO prevê uma caixa semelhante. Em caso afirmativo, dá-se a essa previsão um impulso com base na probabilidade prevista pelo YOLO e a sobreposição entre as duas caixas. O melhor modelo Fast R-CNN atinge um mAP de 71,8 % no conjunto de teste VOC 2007. Quando combinado com YOLO, seu mAP aumenta de 3,2 % para 75,0 %. Também tentou-se combinar o modelo Fast R-CNN superior com várias outras versões do Fast R-CNN. Esses conjuntos produziram pequenos aumentos na mAP entre 0,3 e 0,6 % conforme Tabela (2.1). Outras versões do Fast R-CNN fornecem apenas um pequeno benefício, enquanto o YOLO fornece um aumento significativo no desempenho.

Tabela 2.1 – Experimentos de combinação de modelos no VOC 2007

	mAP	**Combined**	**Gain**
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	66.9	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	**63.4**	**75.0**	**3.2**

Fonte: Redmon et al. (2016).

No entanto há poucos benefícios na combinação de diferentes versões do Fast R-CNN. Uma vez que o YOLO comete diversidades de erros nos testes e não erros específicos de grande magnitude como no Fast R-CNN. Como a execução dos modelos é feita separadamente e em seguida a combinação dos resultados, a combinação não se beneficia da velocidade do YOLO, e sim do mAP. De acordo com Redmon et al. (2016) no placar público de 6 de novembro de 2015 - PASCAL VOC 2012 Leaderboard - o YOLO foi o único com capacidade de detecção em tempo real. Apresentando-se na quarta posição em indicador mAP na combinação Fast R-CNN + YOLO. No conjunto de testes VOC 2012, o YOLO pontuou 57,9 % mAP. Isso é inferior ao estado da arte atual, sendo mais próximo do R-CNN original + VGG-16. O YOLO lida com pequenos objetos em comparação com seus concorrentes mais próximos. O modelo combinado Fast R-CNN + YOLO é um dos métodos de detecção de melhor desempenho. A Fast R-CNN obtém uma melhoria de 2,3 % com a combinação com YOLO, aumentando 5 posições na tabela de classificação pública conforme Figura 2.23.

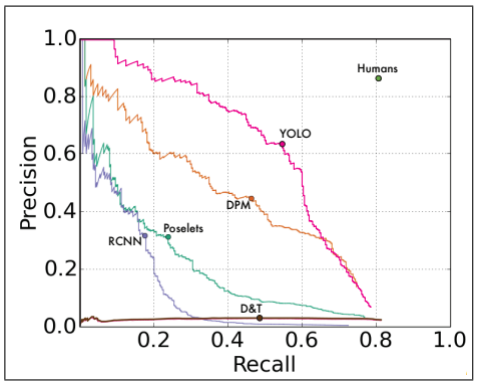
Figura 2.23 – PASCAL VOC 2012 Leaderboard.

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Fonte: Redmon et al. (2016).

Segundo Cai et al. (2015), conjuntos de dados acadêmicos para detecção de objetos extraem os dados de treinamento e teste da mesma distribuição. Ou seja, em aplicações do mundo real é difícil prever todos os casos de uso possíveis e os dados de teste podem divergir do que o sistema viu antes no treinamento da rede neural. Com a finalidade de detectar pessoas em sistemas generalistas como obras de arte, é feito uma comparação do YOLO com outros sistemas de detecção no conjunto de dados Picasso (GINOSAR et al., 2014). Da mesma forma, um conjunto de dados People-Art é utilizado para avaliar o desempenho YOLO (CAI et al., 2015). Para referência, fornece-se o AP de detecção de VOC 2007 por pessoa, onde todos os modelos são treinados apenas em dados de VOC 2007. O R-CNN tem alto AP em VOC 2007. No entanto, o R-CNN cai consideravelmente quando aplicado à arte. O R-CNN usa a Busca Seletiva para propostas de caixa delimitadora ajustadas para imagens naturais. A etapa do classificador no R-CNN vê apenas pequenas regiões e precisa de boas propostas. O DPM mantém seu AP bom quando aplicado à arte. Trabalhos anteriores teorizam que o DPM tem um bom desempenho porque possui modelos espaciais fortes da forma e do layout dos objetos. Embora o DPM não se degrade tanto quanto o R-CNN, ele começa a partir de um AP inferior. YOLO tem um bom desempenho no VOC 2007 e seu AP se degrada menos do que outros métodos quando aplicado à arte, conforme a Tabela (2.2). Assim como o DPM, o YOLO modela o tamanho e a forma dos objetos, bem como as relações entre os objetos e onde os objetos comumente aparecem. A arte e as imagens naturais são muito diferentes em um nível de pixel, mas são semelhantes em termos de tamanho e forma dos objetos, portanto, o YOLO tem a capacidade de prever caixas delimitadoras obtendo alto AP conforme Figura 2.24.

Figura 2.24 – Curvas de recuperação de precisão do conjunto de dados Picasso.



Fonte: Redmon et al. (2016).

Tabela 2.2 – Resultados quantitativos de AP no conjunto de 3 *datasets*.

	VOC 2007	**Picasso**	**People-Art**
	AP	AP	AP
D&T	-	1.9	-
Poselets	36.5	17.8	-
DPM	43.2	37.8	32
R-CNN	54.2	10.4	26
YOLO	**59.2**	**53.3**	**45**

Fonte: Redmon et al. (2016).

A Figura 2.25 mostra o YOLO rodando em ilustrações de arte. Nesse sentido o YOLO aparece como uma das mais poderosas arquiteturas de detecção em redes neurais generalista. YOLO é um detector de objetos rápido e preciso, tornando-o ideal para aplicações de visão computacional. Redmon et al. (2016) conectou o YOLO a uma webcam e verificou se ela mantém o desempenho em tempo real, incluindo o tempo para buscar imagens da câmera e exibição das detecções. O sistema resultante é interativo. Enquanto o YOLO processa imagens individualmente, quando conectado a uma webcam, ele funciona como um sistema de rastreamento, detectando objetos conforme eles se movem e mudam de aparência. Isso faz a arquitetura ser uma ótima escolha para trabalhos de detecção em tempo real com baixo custo computacional. O Fast YOLO é o detector de objetos de uso geral mais rápido da literatura e o YOLO oferece o que há de mais moderno em detecção de objetos em tempo real. YOLO também generaliza bem para novos domínios, tornando-o ideal para aplicativos que dependem de detecção de objetos rápida e robusta.

Figura 2.25 – Detecções em dataset Picasso.

Fonte: Redmon et al. (2016).

2.5.2 Arquitetura YOLO v2

O modelo básico YOLO tem capacidade de processamento de 45 FPS em tempo real. Já a versão menor da RNA - *Fast YOLO* - processa surpreendentes 155 FPS em tempo real.

Contudo, em comparação aos sistemas de última geração, o YOLO comete mais erros de localização, porém com menos probabilidade de FP em *background*.

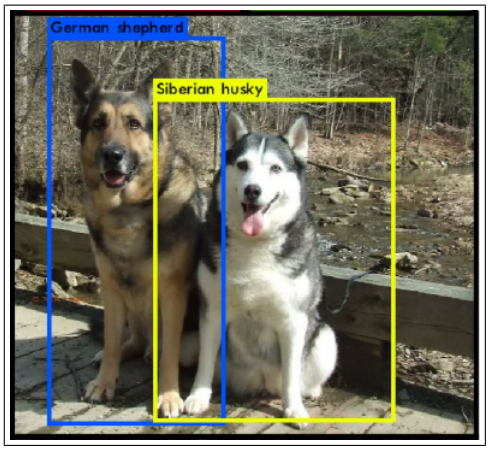
2.5.2.1 Combinação de Conjuntos de Dados

O YOLO v2 - também conhecido como YOLO9000 - supera outros métodos de detecção como DPM e R-CNN quando se trata de capacidade generalista. A cada ano as estruturas de detecção tornam-se cada vez mais rápidas e precisas. No entanto, a maioria dos métodos de detecção ainda está restrita a um pequeno conjunto de objetos. De acordo com Everingham et al. (2010), os conjuntos de dados de detecção mais comuns contêm milhares a centenas de milhares de imagens com dezenas a centenas de marcas. Os conjuntos de dados de classificação têm milhões de imagens com dezenas ou centenas de milhares de categorias (DENG et al., 2009). Rotular imagens para detecção é muito mais dispendioso do que rotular para classificação. O YOLO v2 propõe uma forma de aproveitar a grande quantidade de dados de classificação que já temos e o usa para expandir o escopo dos sistemas de detecção modernos, no qual usa uma visão hierárquica dos objetos e permite combinar conjuntos de dados distintos. Ou seja, ele possui um algoritmo que permite treinar detectores de objetos em dados de detecção-classificação. O método consiste em aproveitar imagens de detecção rotuladas para aprender a localizar objetos com precisão, enquanto usa imagens de classificação para aumentar seu vocabulário. Com esse algoritmo o YOLO9000 foi treinado para detectar mais de 9000 classes. Assim, a versão v1 foi melhorado para uma nova versão v2, que funciona como um detector em tempo real de última geração. Assim o método de combinação de conjuntos de dados distintos e o algoritmo de treinamento conjunto foram usados para treinar modelos de 9000 classes dos *datasets* ImageNet e COCO. A Figura 2.26 mostra a capacidade de detecção de raças diferentes por meio de *datasets* distintos. Ao contrário da tendência de seguimento da visão computacional em redes neurais que tende a redes maiores e mais profundas para melhor desempenho, o YOLO v2 foi simplificado e otimizado na sua capacidade de aprendizagem.

2.5.2.2 Melhorias no YOLO v2

O detalhamento das melhorias serão omitidas do presente trabalho devido ao amplo escopo de informações. De forma simplificada são: *Batch Normalization*, Classificador de Alta Resolução, Caixas de Âncora, Clusters de Dimensão, Previsão Direta de Localização, Recursos Refinados, Treinamento em Multi-Escalas, Darknet-19, *Softmax*, Estrutura de Classificação Hi-

Figura 2.26 – Detecção em raças distintas de cães.



Fonte: Redmon e Farhadi (2017).

erárquica, Combinação dataset - *WordTree* e Combinação de Detecção e Classificação - YOLO 9000.

2.5.3 Arquitetura YOLO v3

Lançado em 2018, o YOLO v3 veio com pequenas mudanças que o tornaram melhor em relação ao v2. Nessa versão treinou-se uma nova rede de classificadores que é melhor do que as anteriores. A rede prevê 4 coordenadas para cada caixa delimitadora, *tx*, *ty*, *tw*, *th*. Se a célula é deslocada do canto superior esquerdo da imagem por (cx, cy) e a caixa delimitadora anterior tem largura e altura (*pw*, *ph*), então as previsões correspondem as Equações (2.11) à (2.14), estas representadas pela Figura 2.27. Durante o treinamento, usa-se a soma da perda de erro quadrático. Este valor da *ground truth* pode ser facilmente calculado invertendo as equações acima. Os estudos de Ren et al. (2016a) agregaram para o desenvolvimento do YOLO v3. A versão v3 prevê uma pontuação de objetividade para cada *bounding box* usando regressão logística. Este deve ser 1 se a *bounding box* anterior se sobrepõe a um objeto de *ground truth* mais do que qualquer outra *bounding box* anterior. Se a *bounding box* anterior não é a melhor, mas se sobrepõe a um objeto da *ground truth* por mais que algum *threshold* tenha sido definido, ignora-se a previsão. Usa-se o *threshold* de 0,5. Ao contrário da linha de atuação de Ren et al. (2016a), a versão v3 atribui apenas uma *bounding box* antes de cada objeto da *ground truth*. Se uma *bounding box* anterior não for atribuída a um objeto da *ground truth*, não haverá perda de coordenadas ou previsões de classe, apenas objetividade (REDMON; FARHADI, 2018).

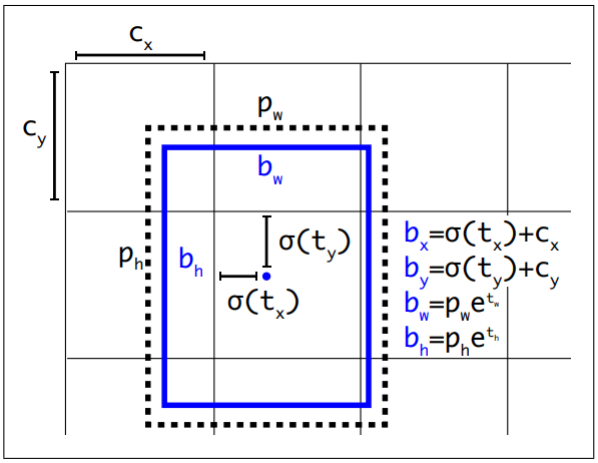
$$b_x = \sigma(t_x) + c_x \tag{2.11}$$

$$b_y = \sigma(t_y) + c_y \tag{2.12}$$

$$b_w = p_w e^{t_w} \tag{2.13}$$

$$b_h = p_h e^{t_h} \tag{2.14}$$

Figura 2.27 – *Bounding box* de dimensão e previsão de localização.



Fonte: Redmon e Farhadi (2017).

$$\hat{t}_* \tag{2.15}$$

$$\hat{t}_* - t_* \tag{2.16}$$

2.5.3.1 Classificação Multi Label

Cada caixa prevê as classes que a caixa delimitadora pode conter usando a classificação de vários rótulos. Não usou-se uma *softmax* porque o criador considerou desnecessário para um bom desempenho, em vez disso, simplesmente usou classificadores logísticos independentes. Durante o treinamento, usa-se a perda de entropia cruzada binária para as previsões de classe. Essa formulação ajuda em domínios mais complexos como o *Open Images Dataset*, nos quais nestes conjunto de dados, há muitos rótulos sobrepostos (ex. Mulher, Pessoa, Ser Humano). Infelizmente o uso de um *softmax* pressupõe que cada caixa tenha exatamente uma classe, o que geralmente não é o caso. Uma abordagem *multi label* modela melhor os dados (REDMON; FARHADI, 2018).

2.5.3.2 Predições entre Escalas

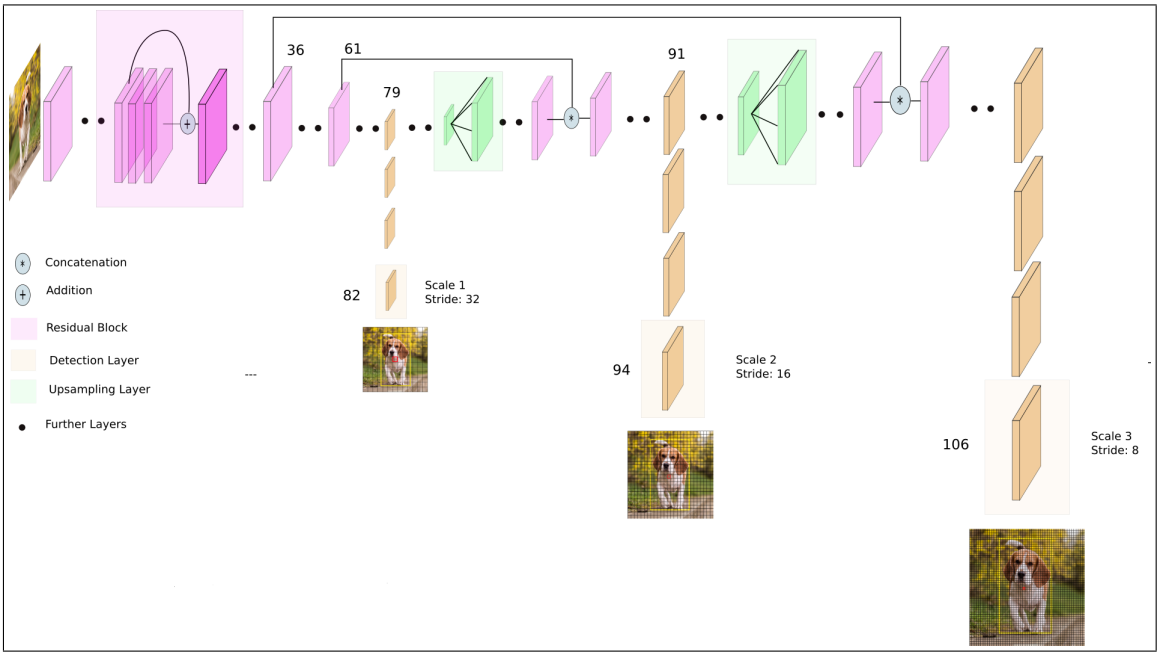
O YOLO v3 prevê caixas em 3 escalas diferentes, no qual o sistema extrai recursos dessas escalas usando um conceito semelhante ao de redes em pirâmide de recursos proposto por Lin et al. (2017). Do extrator de recurso base, adicionou-se várias camadas convolucionais. O último deles prevê uma caixa delimitadora de codificação de tensor 3d, objetividade e previsões de classe. Com o conjunto de dados COCO de Lin et al. (2014), prevê-se 3 caixas em cada escala, então o tensor N x N é $3 \cdot (4 + 1 + 80)$, para os 4 deslocamentos de caixa delimitadora, 1 previsão de objetividade e 80 previsões de classe. Em seguida, pega-se a *feature map* das 2 camadas anteriores e aumenta-se a amostragem em 2 x. Também pega-se uma *feature map* anterior na rede e a mescla com os recursos de *upsampled* usando concatenação. Este método permite obter informações semânticas mais significativas dos recursos *upsampled* e informações mais granuladas da *feature map* anterior. Logo, adiciona-se mais algumas camadas convolucionais para processar essa combinação de *feature map* e, eventualmente, prever um tensor semelhante, embora agora com o dobro do tamanho. Realizou-se o mesmo projeto mais uma vez para prever caixas para a escala final. Assim, as previsões para a 3ª escala se beneficiam de todos os cálculos anteriores, bem como de recursos refinados desde o início na rede. Usou-se o agrupamento *k-means* para determinar os antecedentes de *bounding box*. Nos estudos de Redmon e Farhadi (2018), escolheu-se apenas 9 clusters e 3 escalas arbitrariamente e então dividiu-se os clusters igualmente entre as escalas. No conjunto de dados COCO, os 9 clusters eram: (10 x 13), (16 x 30), (33 x 23), (30 x 61), (62 x 45), (59 x 119), (116 x 90) , (156 x 198), (373 x 326) (REDMON; FARHADI, 2018).

2.5.3.3 Estrutura da Arquitetura da versão v3

O YOLO é uma CNN composta de apenas camadas convolucionais, e não trabalha com camadas densas de redes neurais clássicas em que um neurônio de uma camada está ligado a todos os neurônios da camada seguinte. O processo de concatenação permite à rede neural manter internamente cópias das imagens sem alguns processamentos específicos como filtros de *features map* e *maxpooling*. Nas camadas 79-82, 91-94 e 103-106 se encontram os *detector layer* onde opera-se a detecção que é realizada em 3 escalas no intuito de estabelecer detecção de objetos pequenos na imagem, que é a principal melhoria no YOLO versão v3. A *upsampling layer* ao contrário das camadas convolucionais, aumentam a dimensão da imagem. Por isso

considera-se que a arquitetura v3 possui 3 camadas de saída (layers 82, 94 e 106) e O *residual block* são as próprias camadas convolucionais conforme a Figura 2.28.

Figura 2.28 – Arquitetura de rede neural do YOLO v3.



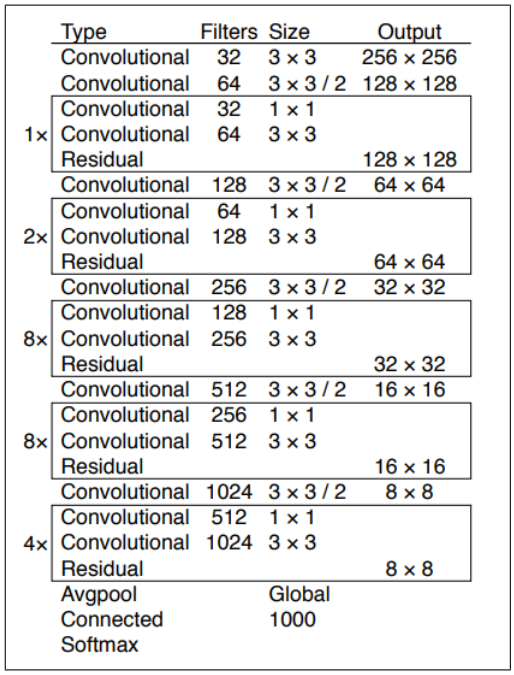
Fonte: Kathuria (2018).

2.5.3.4 Extrator de Recursos - Darknet-53

O YOLO v3 apresenta uma nova rede para realizar a extração de recursos. A nova rede é uma abordagem *hybrid* entre a rede usada em YOLO v2 + Darknet-19 + Rede Residual. Como mencionado, a rede usa sucessivas camadas convolucionais 3 x 3 e 1 x 1, mas na versão v3 também possui algumas conexões de atalho e é significativamente maior. Apresenta 53 camadas convolucionais, e por isso leva o nome de Darknet-53 conforme Figura 2.29. O Darknet-53 é muito mais poderosa e eficiente do que seu antecessor Darknet-19 ou mesmo redes no estado da arte como ResNet atuais.

A Tabela 2.3 mostra alguns resultados comparativos de backbones em *dataset* ImageNet. Cada rede é treinada com configurações idênticas e testada em resolução 256 × 256 (*single crop accuracy*). Os tempos de execução (BFLOP/s e FPS) foram medidos em uma GPU Titan X. O Darknet-53 tem um desempenho semelhante aos classificadores de última geração, mas com mais velocidade (FPS). O *framework* apresenta superioridade de 1,5 x a velocidade do ResNet-101. Em comparação ao ResNet-152 apresenta o desempenho superior em 2 ×. Outra vantagem na versão Darknet-53, é que o *framework* alcança operações de ponto flutuante por segundo

Figura 2.29 – Darknet-53



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fonte: Redmon e Farhadi (2018).

mais altas em relação aos concorrentes (BFLOP/s), conforme a Tabela (2.3). Isso significa que a estrutura da rede utiliza melhor a GPU e a torna mais eficiente. Segundo Redmon e Farhadi (2018) isso ocorre principalmente porque os *framework*s ResNet possuem camadas excessivas e não são muito eficientes em função dos estudos de He et al. (2016).

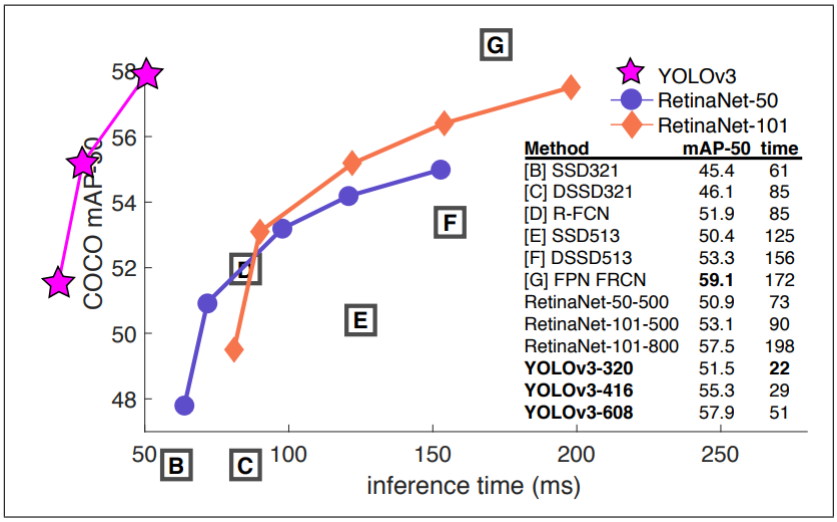
Tabela 2.3 – Comparação de backbones em *dataset* ImageNet (mAP).

Backbone	**Top-1** (mAP)	**BFLOP/s**	**FPS**
Darknet-19	74.1	1246	171
ResNet-101	**77.1**	1039	53
ResNet-152	**77.6**	1090	37
Darknet-53	**77.2**	**1457**	**78**

Fonte: Redmon e Farhadi (2018).

O *framework* de rede neural Darknet foi projetado em linguagem C. É necessário ressaltar que o desempenho cai significativamente conforme o limite do indicador IoU aumenta (acima de AP75), o que implica que o YOLO v3 se esforça para alinhar as caixas perfeitamente com o objeto. Quando a análise é para a "antiga" métrica de detecção de mAP em IoU = 0,5 (ou AP50) o YOLO v3 é muito forte. A Figura 2.30 mostra o gráfico Precisão x Velocidade na métrica AP50, onde mostra-se os benefícios significativos para a arquitetura emergente baseada em YOLO v3 + Darknet-53.

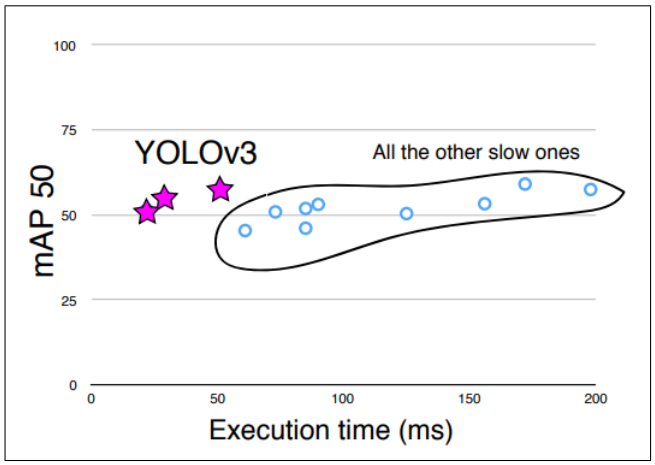
Figura 2.30 – Precisão x Velocidade em AP50.



Fonte: Redmon e Farhadi (2018).

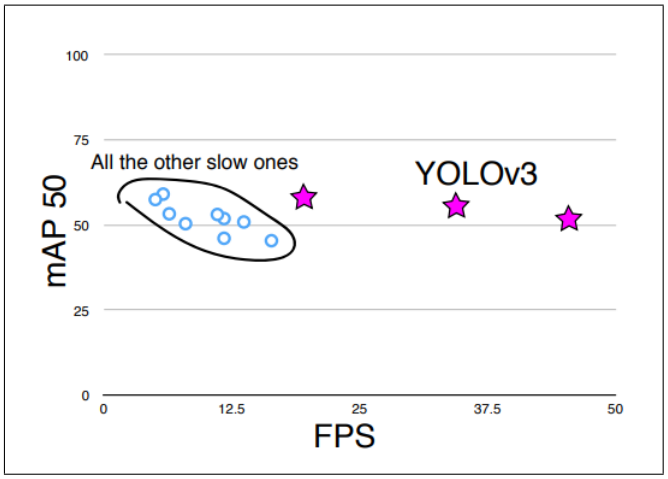
Não é tão bom no AP médio (mAP) do COCO entre 0,5 e 0,95 IoU métrico, ou seja, equilíbrio entre Precisão x Velocidade no mAP na métrica de 0,5 IoU. Nessa versão foi corrigido um *bug* de carregamentos de dados do YOLO v2, que permitiu mais 2 pontos em mAP. Detectores robustos como o YOLO v3 e outros, começam a ser utilizados em pesquisas de identificação de animais em projetos de censo populacional como nos estudos de Parham et al. (2017). As Figuras 2.31 e 2.32 mostram uma visão mais generalista da capacidade da arquitetura YOLO frente aos detectores citados anteriormente sem detalha-los. A métrica COCO enfatiza melhores *bounding boxes*, mas essa ênfase deve significar que ela tira a ênfase de outra coisa, neste caso a precisão da classificação (REDMON; FARHADI, 2018).

Figura 2.31 – Análise Precisão x Tempo de Execução.



Fonte: Redmon e Farhadi (2018).

Figura 2.32 – Análise Precisão x FPS.



Fonte: Redmon e Farhadi (2018).

2.5.4 Arquitetura YOLO v4

2.5.4.1 YOLO v4

Até abril de 2020, o YOLO v4 é o detector de objetos com mais acurácia que permite ser executado em tempo real. Algumas características que podem ser destacadas nessa versão:

1. Melhor velocidade de inferência
2. Melhor acurácia.
3. Melhor eficiência em GPU (demanda menos memória).

2.5.4.2 Formato de Coordenada

O formato compatível com o YOLO versão v4 é dada por:

$$< \text{classe} - \text{id} >< \text{x} >< \text{y} >< \text{largura} >< \text{altura} > \tag{2.17}$$

onde:

- **<classe-id>**: um número inteiro associado à classe, iniciando do 0 (o número para cada classe vai ser equivalente à ordem dos nomes em classes.txt).
- **<x><y>**: valores do tipo float relativos à largura e altura do objeto na imagem. Variam entre 0.0 e 1.0.
- **<largura>**: <largura _obj>/<largura _img>

- **<altura>**: <altura _obj>/<altura _img>
- **<x><y>**: pontos centrais do retângulo de detecção do objeto (e não o canto esquerdo superior do retângulo)

2.5.4.3 Comparativo: YOLO v3 x YOLO v4

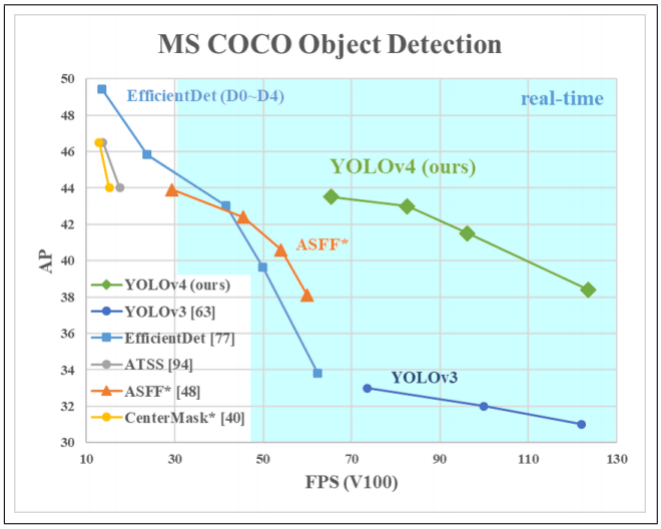
Em abril de 2018, o YOLO v3 de Redmon e Farhadi (2018) demonstrou uma grande melhora na eficiência da predição. No entanto, no geral ele não é mais rápido que a versão anterior. A principal novidade é a predição da imagem em 3 diferentes escalas, o que resolveu o principal problema da versão anterior que era a dificuldade para reconhecer objetos muito pequenos na imagem. Essa novidade também é o principal motivo de não ser mais rápido que o seu antecessor, já que tal função exigiu mudanças na arquitetura e funcionamento, que tornaram o processo mais pesado.

O YOLO v4 foi lançado em abril de 2020, sendo oficializada após a publicação de Bochkovskiy, Wang e Liao (2020). As principais características que podem ser destacadas nessa versão são a melhoria no tempo de inferência e processamento para acurácia. Isso torna o v4 a melhor opção quando o quesito desejado é velocidade e precisão em classificação/detecção até o ano de 2020. Outra característica importante é o fato de ser mais eficiente para rodar em GPUs, pois foi otimizada para utilizar menos memória. A versão 4 demonstrou ser o melhor detector de objetos para testes em tempo real de acordo com as métricas do famoso dataset MS COCO, comumente utilizado para avaliar os sistemas de detecção de objetos. Além de ser mais rápido e mais preciso que o EfficientDet, ele também supera o RetinaNet/MaskRCNN (Facebook Pytorch/Detectron) no dataset COCO. Umas das novidades mais importantes foi a introdução de um novo método de data augmentation chamado de Mosaic e Self-Adversarial Training (SAT). Outro fator inovador é que o YOLO v4 passa a trabalhar com o formato de imagens JPG, que vem a somar com o PNG até o presente momento (GRANATYR, 2020). Há um grande número de recursos que supostamente melhoram a precisão da Rede Neural Convolucional (CNN). Alguns recursos operam exclusivamente em determinados modelos e problemas ou apenas para conjuntos de dados de pequena escala. Enquanto alguns recursos, como normalização em lote e conexões residuais, são aplicáveis à maioria dos modelos, tarefas e conjuntos de dados. O YOLO v4 apresenta alguns recursos universais que podemos citar:

1. Conexões Residuais Ponderadas (WRC - *Weighted Residual Connections*);

2. Conexões Parciais de Estágios Cruzados (CSP - *Cross Stage Partial*);
3. Normalização de Mini-Lote Cruzado (CmBN - *Cross mini-Batch Normalization*);
4. Treinamento de autocontrole (SAT - *Cross mini-Batch Normalization*);
5. Ativação *Mish*.

Figura 2.33 – Melhorias: YOLO v3 x YOLO v4 e outras arquiteturas.



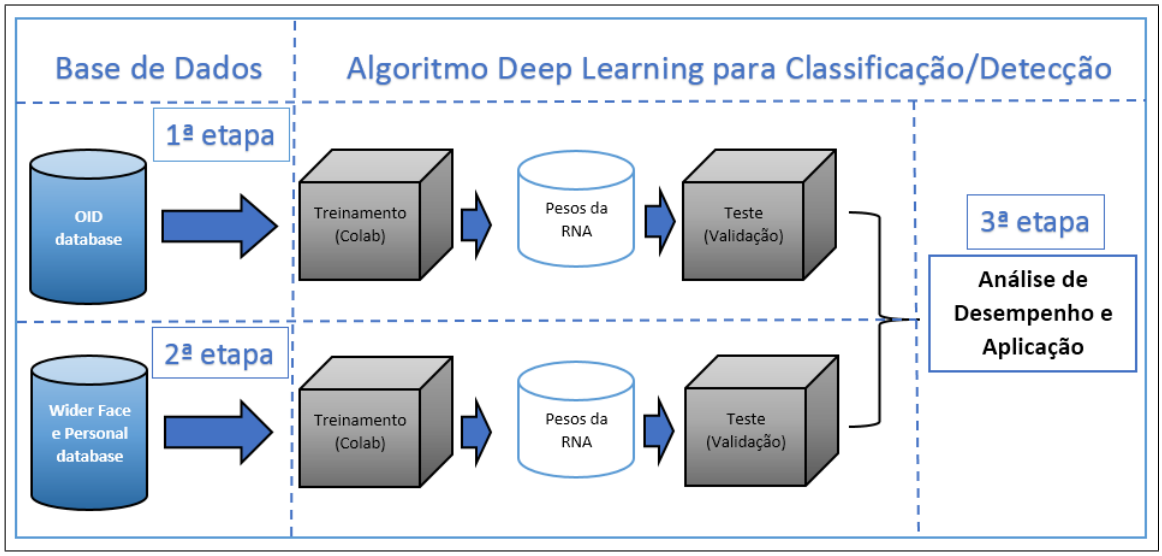
Fonte: Bochkovskiy, Wang e Liao (2020).

O presente trabalho não aprofundará no detalhamento de cada recurso, uma vez que extrapolaria os limites do trabalho. Bochkovskiy, Wang e Liao (2020) usou os novos recursos citados, e combina alguns deles para obter resultados de última geração: 43,5% AP (65,7% AP50) para o conjunto de dados MS COCO a uma velocidade em tempo real de 65 FPS no Tesla V100. A Figura 2.33 mostra a comparação do YOLO v4 com outros detectores de objetos de última geração. YOLOv4 é executado duas vezes mais rápido do que EfficientDet com desempenho comparável. Apresenta melhora do AP e FPS do YOLOv3 em 10% e 12%, respectivamente.

3 METODOLOGIA

Este capítulo descreve a metodologia adotada no presente trabalho de forma sequencial, abordando as ferramentas de projeto utilizadas e detalhando seu funcionamento. A metodologia está estruturada de acordo com a Figura 3.1. Primeiro obtemos uma *database* voltada para os objetivos do presente trabalho. A database OID que veremos a seguir com mais detalhes será utilizada a fim de obter detecções mais generalistas de objetos, com intuito de programar o algoritmo com especificações otimizadas para melhor desempenho. A database WIDER FACE já é mais específica, e focará em detecções de apenas faces de pessoas. A primeira etapa do projeto consiste em adquirir desempenho regular na detecção de objetos com a primeira *database* e, a segunda etapa em detecção facial. Note pela Figura 3.1 que ambas databases serão treinadas e validadas em ambiente *Colab*, gerando assim os pesos de nossa rede neural. A terceira etapa da metodologia se foca em analisar o desempenho do algoritmo em aplicações.

Figura 3.1 – Diagrama metodológico.



Fonte: Do Autor (2020).

3.1 Databases

3.1.1 OID Database

Nosso projeto utiliza na primeira etapa de projeto o OID *database*, no intuito de executar detecção de objetos e programar a rede neural para desempenho regular. O OID (*Open Images Dataset*) - com OIDv4 Toolkit (versão v4) - do Google, é o maior conjunto de dados existente com anotações de localização de objeto com aproximadamente 9 milhões de imagens para 600

classes de objeto que foram anotadas com rótulos em nível de imagem e caixas delimitadoras de objeto. As imagens são muito diversas e geralmente contêm cenas complexas com vários objetos (em média 8,4 por imagem). Ele contém anotações de rótulos no nível da imagem, caixas delimitadoras de objetos e segmentações de objetos.

3.1.2 WIDER FACE Database

Na segunda etapa do presente trabalho, nos propomos a executar treinamento e testes na *database* WIDER FACE. É um conjunto de dados de referência mundial para detecção de rosto, cujas imagens são selecionadas do conjunto de dados WIDER disponível publicamente (SHUO et al., 2016). São 32.203 imagens e rotulação de 393.703 faces com um alto grau de variabilidade em escala, pose e oclusão, conforme representado nas imagens de amostra (Figura 3.2).

Figura 3.2 – WIDER FACE dataset.

Fonte: Shuo et al. (2016).

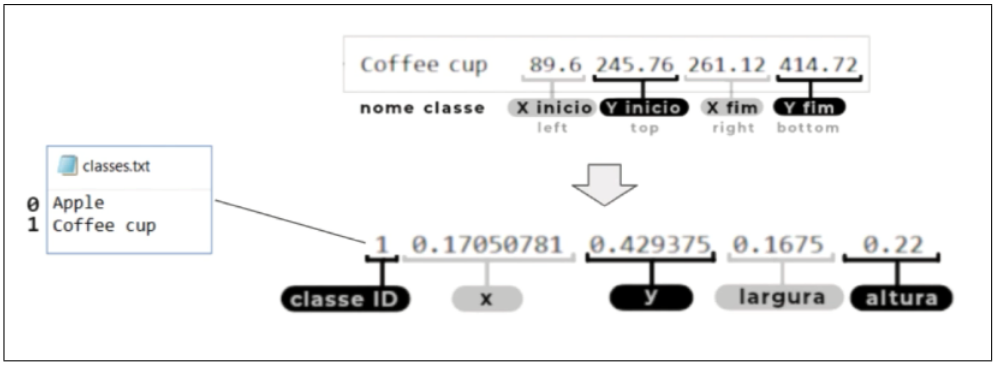
O conjunto de dados é organizado com base em 61 classes de eventos. Para cada classe de evento no modelo padrão do fornecedor, selecionou-se aleatoriamente, 80% e 20% dos dados como conjuntos de treinamento e validação respectivamente. O conjunto de teste aloca a mesma proporção do conjunto de validação, para efeito estatístico nos resultados. Neste *dataset* adota-se a mesma métrica de avaliação de parâmetros empregada no conjunto de dados PASCAL VOC, tais como os índices citados no objetivo do trabalho.

3.1.3 Personal Database (Autor)

Como se discutiu rapidamente na teoria do presente trabalho, o YOLO apresenta uma característica única de representação de coordenadas. Para tal representação, faz-se necessário

um sistema que possa converter coordenadas de outras arquiteturas para a do YOLO, ou usar alguma ferramenta que dê suporte à criação do seu próprio *dataset* diretamente nas coordenadas que desejamos. Para o desenvolvimento do *dataset* de identificação pessoal (Personal), foi utilizado o labelImg, que é uma ferramenta gráfica de *annotation* (anotação) de coordenadas de *bounding boxes* em imagens e aplicação de rótulos em imagens, com a finalidade de se criar imagens *training base* e *test base*. A mudança de representação tradicional para a YOLO pode ser observada na Figura 3.3. As imagens são compostas de imagens faciais do próprio autor em diferentes ângulos e expressões, afim de obter uma maior representatividade facial do indivíduo. É composto por 240 imagens de treinamento e 60 para validação.

Figura 3.3 – Estrutura de coordenadas de localização gerada pelo labelImg.



Fonte: Granatyr (2020).

3.2 Google Colaboratory

Para o presente trabalho, optou-se por utilizar os recursos de *cloud computing* oferecidas pela Google Inc. Isso se dá em virtude das restrições impostas pela limitação de *hardware* em computadores domésticos, que dependendo do tipo de aplicação em RNA, poderão não suportar determinados processamentos. Quando se trata de processamento de imagens e videos em Visão Computacional, há a necessidade de sistemas que possam dar suporte aos bilhões de cálculos que serão executados em processos de treinamento. O código é executado em uma máquina virtual particular. As máquinas virtuais são excluídas quando estão inativas por muito tempo e têm um ciclo de vida máximo restringido pelo serviço do Colab. As GPU e TPU algumas vezes são priorizadas para usuários que usam o Colab interativamente, e não para cálculos de longa duração, ou para aqueles usuários que utilizaram menos recursos do Colab recentemente. Como resultado, os usuários que utilizam o Colab para cálculos de longa duração ou que recentemente usaram mais recursos do Colab provavelmente alcançarão seus limites de uso e terão o acesso

às GPU e TPU temporariamente restringido. Outra boa ideia é usar a IU (Interface de Usuário) do Colab com um ambiente de execução local no próprio hardware quando não houver grandes demandas de processamento gráfico. As GPUs disponíveis incluem: Nvidia K80s, T4s, P4s e P100s. Os notebooks são executados conectando-se a máquinas virtuais que têm ciclos de vida máximos de até 12 horas. Eles também se desconectam das Virtual Machines (VM) quando permanecem inativos por muito tempo. O ciclo de vida máximo da VM e o tempo limite de inatividade podem variar com o tempo ou de acordo com seu uso. A quantidade de memória disponível nas máquinas virtuais variam com o tempo, mas são estáveis durante todo o ciclo de vida da VM (GOOGLE, 2020).

3.2.1 Configuração de GPU no Colab

Trabalhamos com o *framework* TensorFlow para manipular a GPU, em modalidade para rede de camadas profundas. Foram necessárias configurações específicas para openCV, GPU e CUDA em *Deep Learning*. Para o presente projeto alocamos uma GPU NVIDIA Tesla T4 16GB DDR6 (equivalentes a 260 teraflops de desempenho), conforme Figura 3.4. Foi configurado o driver CUDA para executar nosso treinamento (Makefile). Para trabalhar com grandes quantidades de dados, é inevitável que se utilize recursos de *cloud computing* no intuito de ler, mover e copiar arquivos frequentemente. Assim montou-se um ambiente virtual no Google Drive (GD) de modo a lidar com esses dados de forma rápida.

3.2.2 Criação de Ambiente de Detecção

A arquitetura do presente trabalho usa o YOLO v4 no intuito de utilizar recursos otimizados como descrito no estado da arte. Para execução de detecção, escolheu-se o processamento de CPU no Colab no intuito de economizar uso de GPU. Estamos em ambiente nativo Linux e trabalhando com linguagens *C* e *Python*. É necessário realizar um clone do Darknet por meio da plataforma *Github* diretamente para o Google Drive. Para trabalhar com detecção no YOLO, precisamos dos seguintes parâmetros:

- Configuração YOLO.
- Pesos da CNN.
- Imagem.

Figura 3.4 – Especificação GPU.

```
Sat Feb  6 00:56:45 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.39       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   47C    P0    27W /  70W |    227MiB / 15079MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Fonte: Do Autor (2020).

Para visualização de imagens trabalhamos com o *openCV*, biblioteca esta, muito conhecida em pesquisas de Visão Computacional, ou *Computer Vision* (CV). Seu uso foi no intuito de criar funções específicas que dessem capacidade de visualização e conversões de canais de cores em imagens. Usamos também para criar *bounding boxes* com características específicas de projeto. Após a execução do treinamento, o sistema gera um arquivo chamado *predictions* que representa a imagem com as coordenadas de detecção. Outro recurso adotado no presente trabalho é o parâmetro *ext_output*, cuja finalidade é se obter o posicionamento do objeto. Não podemos ter certeza do posicionamento do objeto real em uma imagem desconhecida, porém teremos um valor muito aproximado caso a rede neural tenha sido bem treinada, uma vez que se refere às coordenadas de localização de nossa *bounding box*. Em muito casos, esse parâmetro pode ser útil para localização e posicionamento de sistemas de controladores baseados em geolocalização, outra vantagem desse recurso é que ele nos permite inferir as coordenadas de formação do parâmetro IoU visualmente.

3.3 Treinamento com Transfer Learning

Apesar do presente projeto ter inicialmente trabalhado com os pesos pré-treinados do YOLO v4, optou-se por criar o próprio arquivo de pesos. Para tal finalidade foi extraído do arquivo de pesos das primeiras camadas convolucionais por meio da técnica TL. Esse processo

parte do princípio que o modelo de pesos já conhece características como bordas, cantos e linhas. Isso permite que o processo de treinamento seja muito mais rápido do que realizar o treinamento completo do início. No projeto, utilizamos um TL de um arquivo de peso na camada 137 para treino na database OID, acelerando-se o processo de aprendizado da rede neural. Essa camada foi escolhida baseando-se no fato que de acordo com a arquitetura, até essa camada finaliza-se o aprendizado generalizado base da imagem de acordo com a versão v4.

3.4 Reconhecimento de Objetos

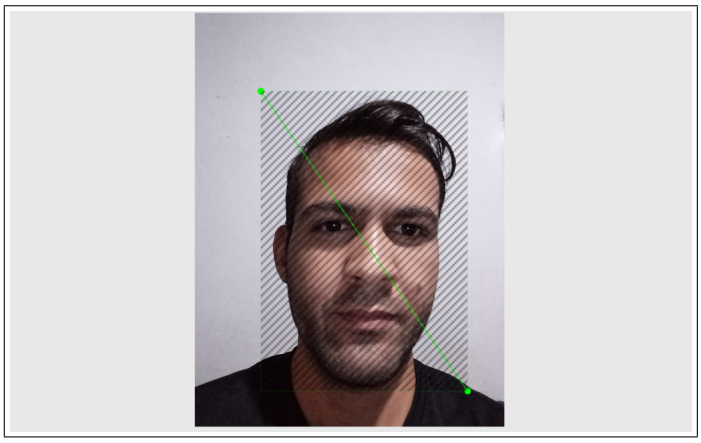
Foi criado uma *database* personalizada no intuito de testar o algoritmo de detecção YOLO v4 modificado (TL). Essa *database* é uma base da dados própria de imagens com suas respectivas configurações (*annotation*), de modo a fazer um treinamento desde o início e depois buscar modelos de sistemas pré-treinados para efetuar comparativos. Essa abordagem nos possibilitou também definir qual tipo de classes iríamos querer classificar e detectar em imagens. Escolheu-se as seguintes classes para treinamento: Apple (Maçã), Coffee cup (Xícara de café) e Horse (Cavalo). Essas classes foram escolhidas aleatoriamente a fim de testar a capacidade de nossa rede neural em classificar/detectar e estabelecer um ponto de partida da configuração da rede neural, com um número limitado e distinto de objetos. Todas essas classes definidas inicialmente para o projeto devem conter os arquivos de treinamento bem como os de validação para análise de resultado. Para tal finalidade utilizou-se dos *dataset* da OID (*Open Images Dataset*), no qual é obtida pela ferramenta OIDv4 (Github) para trabalharmos com YOLO v4. Para trabalhar com o treinamentos generalista em OIDv4, necessitamos instalar algumas bibliotecas específicas do *python*, no qual não entraremos em detalhes de seu funcionamento. Quando se obtém o OIDv4 nativamente. ele já se apresenta com os requerimentos de bibliotecas para seu funcionamento. Essas bibliotecas são na versão v4 as seguintes: *pandas*, *numpy*, *awscli*, *urllib3*, *tqdm* e *opencv-python*. Como trabalhamos em *python*, precisamos chamar o *main.py* da pasta raiz. Inicialmente para os testes de treinamento foram alocadas 500 imagens de cada classe para treinamento, e 100 imagens de cada classe para teste de validação, ou seja uma proporção de 80% e 20% respectivamente. É importante frisar que os dados de treinamento e validação da base de dados do OID vem nativamente com seus arquivos *annotations* em padrão VOC (*Visual Object Classes*) para pontos iniciais e ponto finais de suas coordenadas. Nesse sentido, tivemos que alterar os nome das classes nativas manualmente e converter essas coordenadas ao

padrão YOLO. Para trabalhar com os arquivos de treino em *cloud computing* é recomendável comprimir os arquivos de treinamento e de validação na nuvem, uma vez que o ambiente Colab é suscetível a suspender as sessões após um espaço de tempo, então comprimimos todo nosso *dataset* na nuvem.

3.5 Reconhecimento de Faces

Os testes efetuados de detecção facial irão se basear em duas databases: Personal e WIDER FACE. Ambos os conjuntos de dados serão voltados para a pesquisa de classificação/-detecção facial em imagens. Essa etapa de projeto focamos no reconhecimento facial. Para tal finalidade criamos um *dataset* (Personal) que é composto de 300 imagens de uma única pessoa (Autor do projeto), sendo 240 para treinamento e 60 imagens para validação, ou seja 80% e 20% respectivamente. A Figura 3.5 mostra a imagem sendo trabalhado manualmente com a ferramenta de *annotation*.

Figura 3.5 – Procedimento *annotation* em labelImg.



Fonte: Do Autor (2020).

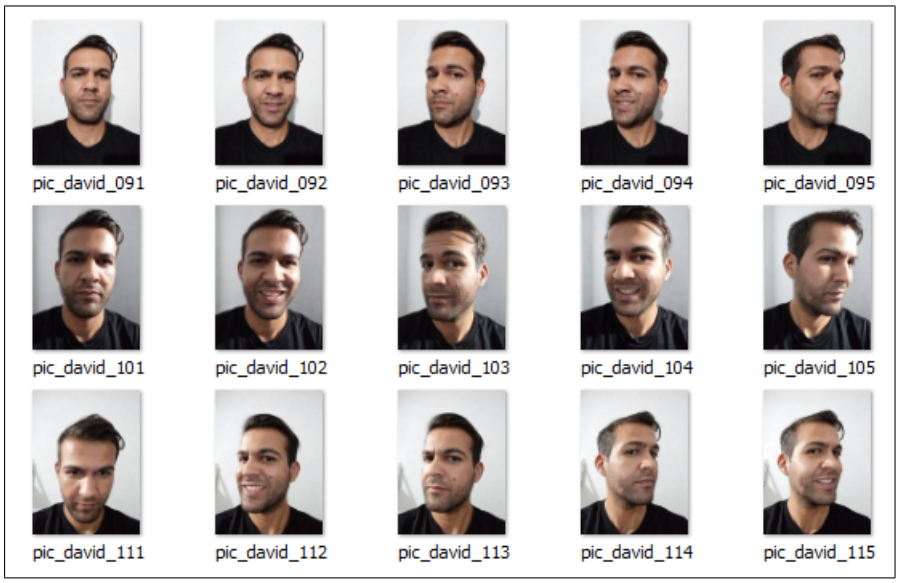
Para se criar um *dataset* específico, o indivíduo deverá fazer as anotações manualmente uma por uma, e com atenção nas limitações daquilo que se quer identificar nos procedimentos de treinamento. A Figura 3.5 mencionada gerou as seguintes coordenadas de *bounding box*:

[0 0.547135 0.551562 0.668229 0.725000]

Observamos que está de acordo com o padrão aceito de reconhecimento do algoritmo YOLO v4 conforme discutido anteriormente. Para tal finalidade os testes entrarão também em regime de *overfitting* a fim de avaliar aumento de detalhamento para identificação individual, uma vez que o YOLO ainda não apresenta tecnologia de segmentação ou *key points*. A segunda

etapa que consiste no treinamento de imagens da WIDER FACE, que contará com as mesmas ferramentas enunciadas anteriormente e com a geração do arquivo de pesos. Testaremos e avaliaremos os parâmetros de desempenho. Verificaremos também os nichos de aplicações que nosso algoritmo poderá ser utilizado. Entre as possíveis aplicações podemos destacar detecção de faces em ambientes de aglomerados de pessoas e sistemas de segurança que pautam em reconhecimento de humanos. Trabalha-se com treinamentos de 2000 *epochs* para avaliação de desempenho. A Figura 3.6 mostra exemplos de imagens contidos no Personal *database*.

Figura 3.6 – Personal *dataset*.



Fonte: Do Autor (2020).

3.5.1 Códigos de Transfer Learning (TL)

O projeto fez implementação e testes com versões TL de diversos autores, no qual verificou-se múltiplos modelos com distintas configurações, e ideias para otimizar os índices de desempenho abordados. Analisou-se os modelos de algoritmos em desenvolvimento dos seguintes autores: Nguyen (2020), Lin (2020), Purohit (2020), Xi (2020), Daybreak (2020), Abars (2020), JMU (2020) e Zhu (2020). Os modelos abordados referenciam sistemas de classificação/detecção facial, bem como aplicações específicas. No entanto, apesar de sua ampla aplicabilidade, os modelos citados não apresentam índices de desempenho compatíveis com os objetivos do projeto, e não foram utilizados como peso de treinamento contínuo para nosso modelo. A nomeação de cada algoritmo, atributos e seus respectivos testes serão omitidas, uma vez que extrapolam os limites do presente trabalho.

3.5.2 Criação da Rede Neural

Realizamos testes com 3 classes, nos quais representam uma diversidade de modelos visuais com características bem distintas, nos quais podem ser encontradas facilmente no *dataset* OID. Para essa finalidade tivemos que modificar o arquivo nativo da rede neural do YOLO v4 para nosso cenário. O primeiro parâmetro que modificamos na rede neural foi o número *classes* que cai de 80 (*dataset* COCO) para 3 (Maçã, Cavalo e Caneca). O número de *filters* ou filtros foi alterado seguindo o manual do YOLO v4, no qual diz que o valor ideal deve ser: (número de classes + 5) * 3. Portanto, reduzimos de 255 para 24 *filters*. Nossa rede neural passa a ter o parâmetro *subdivisions* de 8 para 64, que indicará o consumo de memória (um valor muito baixo o sistema consumirá uma quantidade maior de memória), neste caso, como estamos trabalhando com IDE Colab e Jupyter, necessitamos dessa especificação para início do treinamento, caso contrário o sistema instabiliza. Em relação a esse último parâmetro, conforme testes realizados, caso o valor seja muito baixo o arquivo pode nem mesmo ser executado no ambiente de *cloud computing*. Conforme o YOLO v3, a resolução de entrada era de 416 x 416. No intuito de trabalharmos com maior resolução e do suporte inovador do YOLO v4 alteramos para 608 x 608 (resolução máxima suportada). Os parâmetros *decay* para decaimento de gradiente foi configurado para 0.0005 e o *momentum* para 0.949, ambos com valores nativos (*default*) da versão v4. A *learning rate* ou taxa de aprendizado foi mantida em 0.0013. O *max batches* que se refere ao número total de registro em cada um dos *batches* vem nativamente em 500500 (*database* COCO), porém como temos em nosso teste apenas 3 classes de teste configuramos em 6000, respeitando a tendência *convencional* de 2 mil *batches* para cada classe. Como cada *epoch* executa um número específico de *steps* ou passos, foi definido um *steps* de 4800 até 5400, uma vez que representa por convenção 80% e 90% respectivamente do número de *max batches*. É importante observar que os arquivos nativos "obj.names"e "obj.data"também devem ser modificados, uma vez que seus algoritmos estão projetados para o sistema COCO (relacionados respectivamente aos nomes das classes e aos *path* de arquivos de treinamento). Desenvolvemos um algoritmo para fazer *backups* a cada 100 *epochs* de treinamento, eliminando a possibilidade de *perda de projeto* caso a VM encerre a sessão. O YOLO também necessita de um arquivo de treino “train.txt” e o de validação "test.txt", que ele usará para percorrer uma lista. Esse código de geração de aquivos ".txt"foi criado em *python* e não detalharemos o código por ser muito extenso. O resumo da rede neural criada pode ser vista na Tabela 3.1.

Tabela 3.1 – Características da Rede Neural.

classes	filters	subdivisions	decay	momentum	learning rate	max batches
3	24	64	0.0005	0.949	0.0013	6000

Fonte: Do Autor (2020).

3.5.3 Treinamento

Usamos o TensorFlow com finalidade de configurar o OpenCV, GPU e CUDA para treinamento em Deep Learning. Os detalhes do plano de fundo das configurações específicas de cada notebook para cada PC não serão detalhadas no presente trabalho, uma vez que extrapolam o objetivo do projeto. Porém, vale salientar que cada GPU/CUDA apresenta uma configuração bem específica para trabalho com redes neurais profundas (neste caso particular o nvidia Tesla T4). Nessa etapa executamos a *transferência de aprendizado* utilizando as 137 primeiras camadas convolucionais dos pesos. Assim o treinamento da rede neural foi efetuado com as configurações particulares de projeto citadas na Tabela 3.1.

4 RESULTADOS

O presente capítulo aborda os resultados dos testes efetuados por meio de análise de tabelas e gráficos. Como o objetivo principal do projeto é otimização dos parâmetros de desempenho para estabelecer melhores resultados, focamos nossos esforços em traçar métricas que possibilitem uma visão das vantagens do algoritmo desenvolvido. Para tais objetivos foi utilizado também recursos visuais de classificação/detecção em imagens de modo a facilitar a compreensão.

4.1 Reconhecimento de Objetos

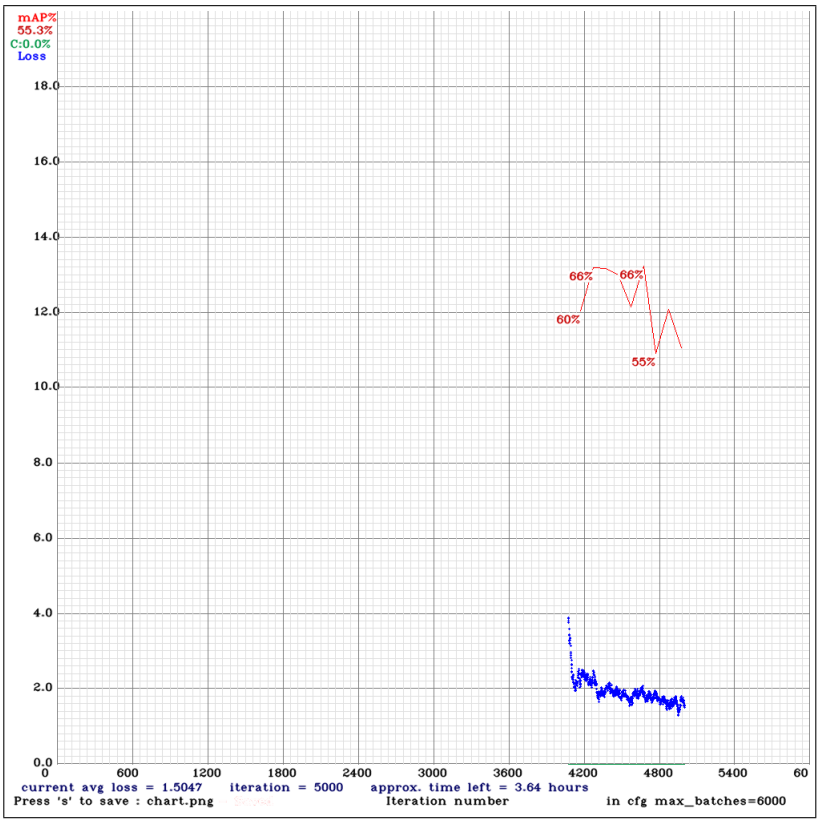
As seções a seguir apresentam os resultados e análises pertinentes relacionados aos parâmetros definidos na seção de metodologia. Os detalhes do código fonte podem ser analisados no link disponível do projeto.

4.1.1 Dinâmica *AVG Loss x Epochs*

Nossa rede fez 1000 iterações por epoch. Um dos principais objetivos numa rede neural é diminuir o erro ou função de perda (*avg loss*). Nosso sistema salvou o arquivo de pesos parciais da rede neural a cada 100 epoch executadas, possibilitando assim treinar um número específico de horas por dia. É interessante que nosso sistema possua um número aceitável de *epochs* para aumentar a representatividade do modelo. Nosso algoritmo em openCV permitiu que o parâmetro mAP do treinamento fosse analisado por meio de um gráfico de saída. Após 100 epoch geramos um gráfico *loss x epochs* de pouca representatividade, e portanto omitida. É interessante que o modelo apresente taxa de decrescimento, uma vez que isso implica em pesos da rede neural cada vez mais otimizados. Nesse primeiro treinamento taxa de erro manteve-se alta, uma vez que nesse tipo de dados (imagens), 100 *epochs* foi insuficiente para diminuir essa taxa. Portanto não notou-se nenhum decaimento do *avg loss*. Para continuar o treinamento de onde paramos utilizamos *backup* de pesos parciais no intuito de começar o treino apartir da *epoch* 100. Assim o sistema iniciou a partir da *epoch* 100 até 300. Após 300 epochs geramos um gráfico *loss x epochs* de pouca representatividade, e portanto omitida. A Figura 4.1 mostra o treinamento em estágio final para a primeira etapa do projeto, onde chegamos a 5000 *epochs* variando seu mAP constantemente a cada 200 *epochs*. Vale salientar que podemos escolher o melhor peso dentre os últimos blocos de treinamento, preservando assim um melhor mAP

mesmo que venha ocorrer oscilações. Observe também que o *avg loss* apresentou um valor de aproximadamente 1.5%, o que mostra um decaimento expressivo no erro de 4.5% até o treinamento de 5000 *epochs*. Vale salientar que o eixo x do gráfico é setado em código fonte de acordo com o *max batches* total.

Figura 4.1 – Treinamento - 5000 *epochs*.



Fonte: Do Autor (2020).

Pela análise dos gráficos fica claro que com o aumento das *epochs* o desempenho da rede aumentou, uma vez que a nossa taxa *avg loss* (erro) diminuiu drasticamente. Assim nosso próximo passo é analisar os resultados numéricos da saída desse gráfico. Para verificar o mAP do modelo, precisamos chamar um recurso do YOLO que irá testar nosso modelo de pesos para as classes de nosso *dataset*. Esse recurso nos permitiu analisar o percentual de acertos do nosso modelo, uma vez que com os recursos de *accuracy* podemos verificar quantos registros o algoritmo classificou corretamente e quantos e incorretamente.

A Tabela 4.1 mostra os resultados numéricos para o treino de 300 *epochs*. A Tabela 4.2 mostra os resultados para treinamento de 4000 *epochs*. Pela análise das Tabelas 4.1 e 4.2, verificamos que os parâmetros de detecção do mAP com IoU em 50% [mAP@0.50], melhoram substancialmente por um fator de aproximadamente 10x. Outro fator notável é o IoU que praticamente triplicou em nossos testes. O nível de precisão é aceitável para grande parte de

aplicações. Em testes de última etapa do projeto com 5000 *epochs*, mostrou-se resultados mais otimizados uma vez que nossa precisão subiu de 39% para 57%. O mAP como parâmetro mais vital da detecção, teve aumento considerável de 62.21% para 66.80% (apesar de oscilações). Os pequenos descaimentos locais que se verifica ao longo do gráfico, se deve ao fato da adaptação do algoritmo em seus blocos de iterações, e pequenos *overfitting* locais antes da atualização de pesos da rede. Isso mostra que nosso sistema adquiriu a capacidade de classificação e detecção. Caso a atualização visual seja mais rápida, o gráfico poderá ser acompanhado em tempo real com curvatura mais suave. Após obter resultados considerados aceitáveis para os pesos da CNN pode-se configurar o *batch* (Batch Gradient Descent) de 64 para 1, e *subdivisions* da CNN de 64 para 1, uma vez que não há necessidade de verificação em lotes de 64 em 64 imagens para detecção. O valor 64 será usado em casos de contínuo treinamento. A Tabela 4.4 nos mostra uma melhora significativa na capacidade de generalização da rede neural para as 3 classes, mantendo um mAP geral satisfatório de acordo com as referências abordadas.

Tabela 4.1 – Análise mAP com arquivo de pesos de 300 *epochs*.

Precision	**TP**	**FP**	**FN**	**IoU**	**mAP [mAP@0.50]**	**Tempo [s]**	**Inferência**
0.20	22	89	466	11,90 %	5.90 %	39	0 %

Fonte: Do Autor (2020).

Tabela 4.2 – Análise mAP com arquivo de pesos de 4000 *epochs*.

Precision	**TP**	**FP**	**FN**	**IoU**	**mAP [mAP@0.50]**	**Tempo [s]**	**Inferência**
0.39	366	574	122	29.18 %	62.21 %	9	77.1 %

Fonte: Do Autor (2020).

Tabela 4.3 – Análise mAP com arquivo de pesos de 5000 *epochs*.

Precision	**TP**	**FP**	**FN**	**IoU**	**mAP [mAP@0.50]**	**Tempo [s]**	**Inferência**
0.57	333	252	155	47.03 %	66.80 %	17	79.4 %

Fonte: Do Autor (2020).

Tabela 4.4 – Análise mAP com arquivo de pesos de 6000 *epochs*.

Precision	**TP**	**FP**	**FN**	**IoU**	**mAP [mAP@0.50]**	**Tempo [s]**	**Inferência**
0.63	339	197	149	52.63 %	69.23 %	16	82.2 %

Fonte: Do Autor (2020).

4.1.2 Classificação e Detecção

Após o desenvolvimento da rede neural de projeto, usamos o modelo para prever objetos em imagens de exemplo de suas respectivas classes. O desempenho foi satisfatório, uma vez que foi possível classificar e detectar com ótimo nível de previsibilidade. Nota-se pelas detecções vistas nas Figuras 4.2, 4.3 e 4.4, que o modelo teve maior dificuldade em detectar a classe Cavalo do que as classes Maçã e Caneca, o que reforça a necessidade de quantidades maiores de imagens de treinamento dessas classes, bem como maior número de *epochs* para o modelo. Outro fator, é que erros de redundância podem ocorrer, devido a imagens cortadas e de baixa qualidade que foram alocadas no *dataset*, como no caso da detecção de duas maçãs no mesmo local, o que é raro de ocorrer. Porém, mesmo com o erro, o algoritmo apresentou alta previsibilidade. Nos testes efetuados com GPU, obteve-se um ganho de velocidade considerável em relação a CPU local. Nos testes em GPU, algumas imagens tiveram uma média de 300 vezes o tempo de previsão do que em CPU. Isso se deve ao fato de serem realizados testes de detecção tanto em ambiente de processamento local com CPU, quanto em *cloud computing* com CPU/GPU.

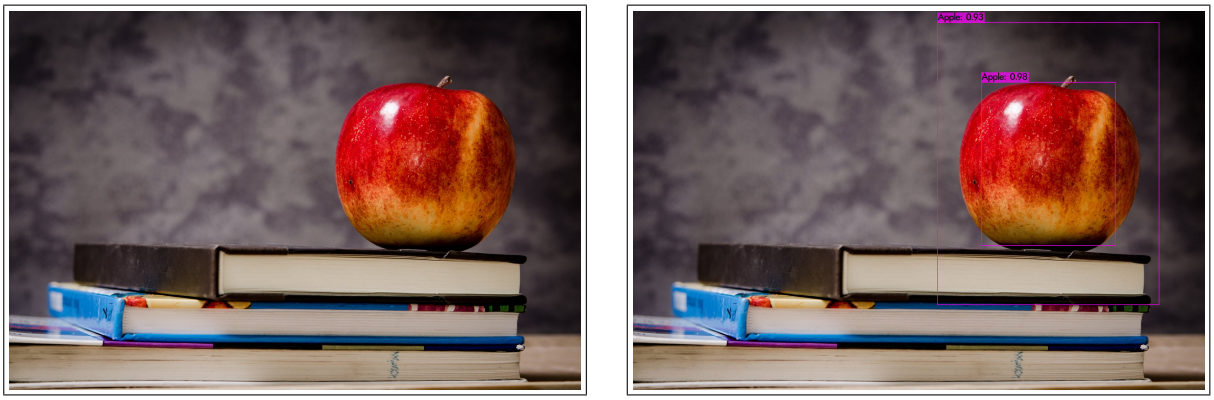
Figura 4.2 – Classificação/Detecção de Xícaras

Fonte: Do Autor (2020).

4.1.3 Impacto da Variação *Threshold*

O parâmetro *threshold* para limiar de previsibilidade de detecção indicou basicamente a qualidade da nossa detecção. Como seus valores variam entre 0 e 1, quando o limiar é baixo demais podem ocorrer centenas FP (False Positives) como na Figura 4.5. Portanto se faz necessário chegar a um limiar adequado para a finalidade do projeto. Para sistemas onde deseja-se um

Figura 4.3 – Classificação/Detecção de Maçãs



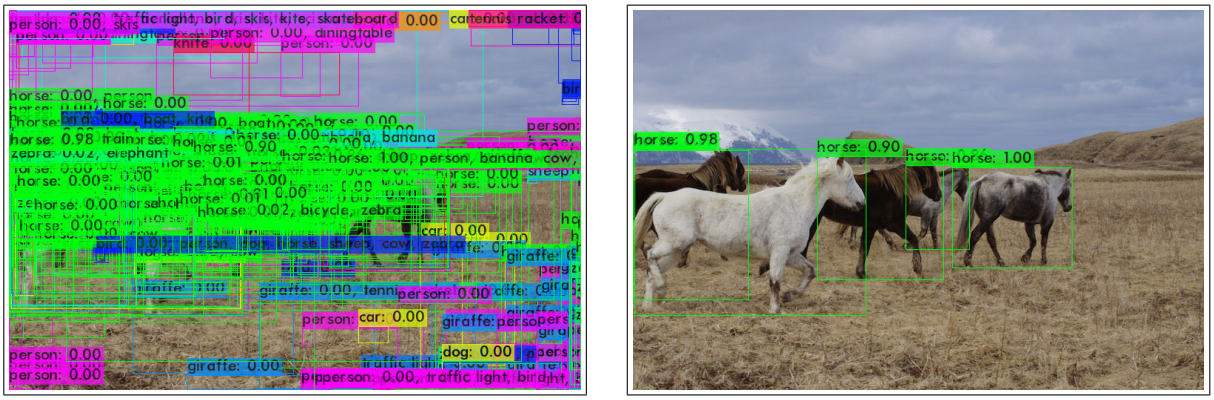
Fonte: Do Autor (2020).

Figura 4.4 – Classificação/Detecção de Cavalos

Fonte: Do Autor (2020).

grau de confiança elevado, é interessante que o valor de *threshold* seja de pelo menos 0.7 ou seja 70% de acurácia. Nessa caso, a título de testes de avaliação, testou-se o modelo com *threshold* em 0.3, 0.9 e 0.98 e comparou-se pelas Figuras 4.5 e 4.6. Isso mostra que quando deseja-se detecções com maior acurácia e imagens mais limpas, o aumento do parâmetro *threshold* é vital. Pode-se notar pelo comparativo visual das imagens que o algoritmo detecta menos cavalos na Figura 4.6, uma vez que seu nível de *threshold* é de 98%, o que pode ser considerado um alto nível de previsibilidade. Esse limiar fez com que apenas dois cavalos fossem detectados na imagem, sendo necessário uma redução do limiar para maiores detecções. Assim nossa rede neural já pode ser empregada com sucesso para um certo grau de confiança em sistemas que trabalhem com essas classes abordadas.

Figura 4.5 – Problemática de baixo limiar *threshold* e *Threshold* de 0.3.



Fonte: Do Autor (2020).

Figura 4.6 – *Threshold* de 0.9 e *Threshold* de 0.98.

Fonte: Do Autor (2020).

4.2 Reconhecimento de Faces

Após os resultados preliminares para definição da rede neural para classificação/detecção de objetos, seguiu-se para o desenvolvimento da nova rede neural para classificação/detecção de faces. A estrutura interna da rede neural pode ser verificada na Tabela 4.5. Nota-se que o número de filtros reduziu de 24 para 18 em virtude da limitação do número de classes, bem como o *max batches* que seguirá uma proporção em função do número de imagens disponíveis. Após testes realizados para primeira etapa de projeto, foi obtido os seguintes resultados da Tabela 4.6 para o Personal database. Como nosso mAP ainda está relativamente baixo para 1000 *epochs*, o nosso parâmetro de precisão está com 0.44, o que nos leva a crer que com maior número de epochs podemos chegar a melhores indicadores antes de alcançar o *overfitting*. Nos testes de inferência conseguimos bons resultados preliminares, alcançando detecção com 54 % de acurácia de acordo com a Figura 4.7. Isso mostra que para algumas imagens em virtude da

qualidade e resolução, podemos alcançar boas métricas de detecção. Os testes visam etapas de treinamento de 100 em 100 *epochs* até 1500, afim de avaliar o crescimento do mAP.

Tabela 4.5 – Parâmetros internos estruturais da rede neural Personal.

classes	**filters**	**subdivisions**	**decay**	**momentum**	**learning rate**	**max batches**
1	18	64	0.0005	0.949	0.0013	2000

Fonte: Do Autor (2020).

Tabela 4.6 – Análise mAP com peso 1000 *epochs*.

Precision	**TP**	**FP**	**FN**	**IoU**	**mAP [mAP@0.50]**	**Tempo [s]**	**Inferência**
0.44	25	32	35	32.59 %	34.70 %	5	54%

Fonte: Do Autor (2020).

Figura 4.7 – Reconhecimento facial em teste de inferência média para 1000 *epochs*.

Fonte: Do Autor (2020).

Assim os testes realizados já possuem alta capacidade de inferência facial, o que leva a crer que os próximos testes podem convergir para uma detecção específica em ambiente de mais de um indivíduo. Após o treinamento da rede neural por 1500 *epochs*, foi obtido os resultados da Tabela 4.7 representado pela inferência da Figura 4.8. Finalizando os testes do Personal database podemos obter a Tabela 4.8 e Figura 4.9 e discutir seus resultados antes de partir para análise de detecção específica.

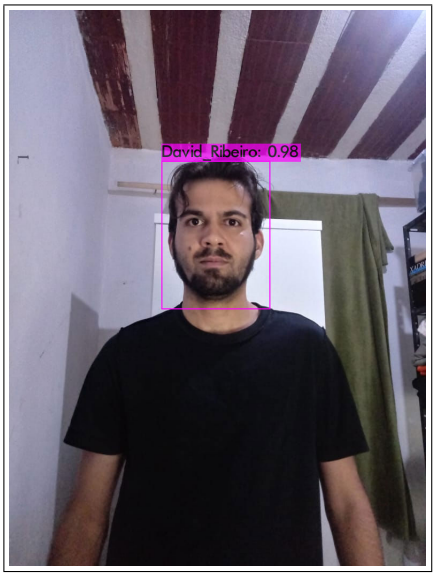
Os resultados obtidos pelo treinamento da rede neural em *dataset Personal* se mostram otimizados, uma vez que apresenta alta capacidade na identificação de face do indivíduo. O índice mAP alcançou 99.11 % antes de entrar em estado de *overfitting*, e apresentou o tempo de 3

Tabela 4.7 – Análise mAP com peso 1500 *epochs*.

Precision	TP	FP	FN	IoU	mAP [mAP@0.50]	Tempo [s]	Inferência
0.80	56	14	4	60.64 %	95.81 %	4	98 %

Fonte: Do Autor (2020).

Figura 4.8 – Reconhecimento facial em teste de inferência média para 1500 *epochs*.



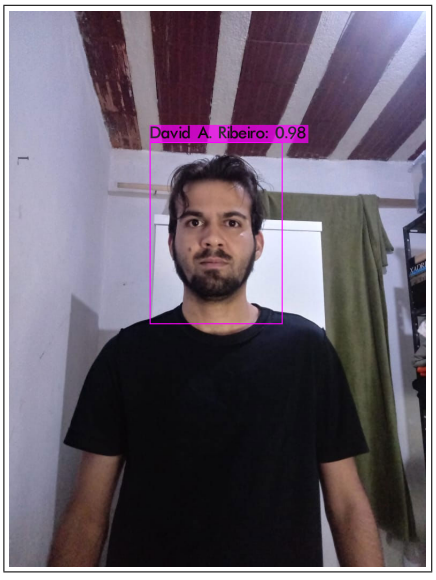
Fonte: Do Autor (2020).

Tabela 4.8 – Análise mAP com peso 2000 *epochs*.

Precision	TP	FP	FN	IoU	mAP [mAP@0.50]	Tempo [s]	Inferência
0.97	56	2	2	82.56 %	99.11 %	3	98 %

Fonte: Do Autor (2020).

Figura 4.9 – Reconhecimento facial em teste de inferência média para 2000 *epochs*.



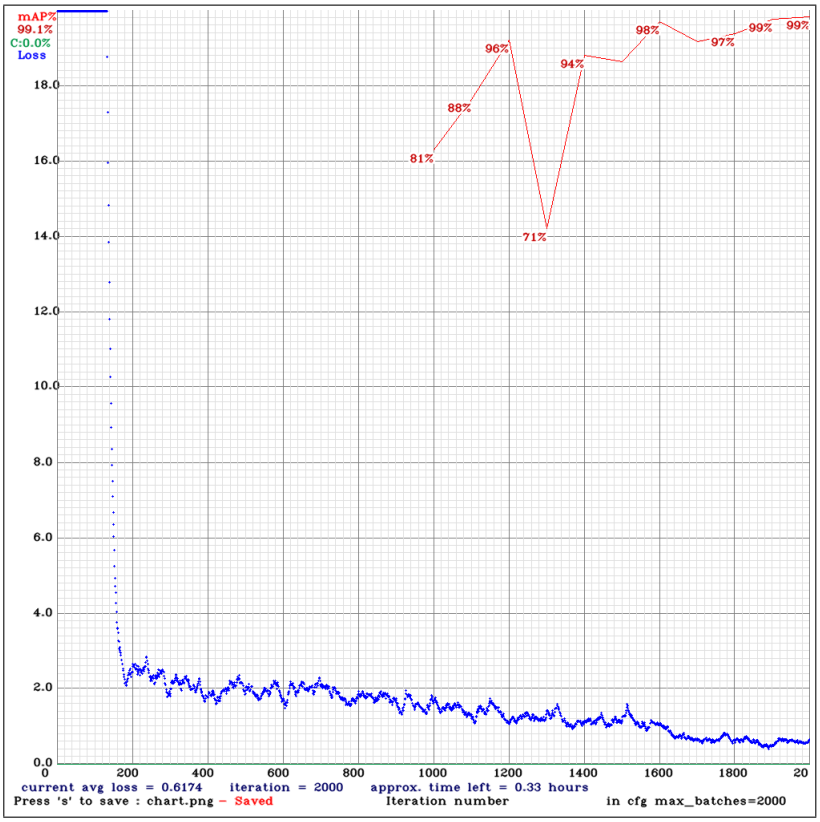
Fonte: Do Autor (2020).

segundos no reconhecimento. Comparativamente aos modelos atuais, o nosso modelo apresentou mAP superior, uma vez que os modelos analisados na literatura do presente trabalho variam entre 69 % a 78 % em precisão média. A partir dessa métrica, agora podemos aplicar o modelo no reconhecimento individual em público, avaliando sua capacidade de reconhecimento específico não generalista para o mesmo modelo, porém agora com o indivíduo inserido com outras pessoas na imagem. Utilizamos um algoritmo que possibilitou o salvamento do arquivo da tabela de pesos em blocos, permitindo assim a unificação de cada treinamento (100 *epochs*) em um único gráfico. Porém, vale salientar que para isso ocorra é necessário que o sistema retorne ao ponto final de treinamento do modelo de pesos da rede neural anterior, caso contrário não pode ser efetuada. O gráfico da Figura 4.10 mostra o desempenho do nosso algoritmo ao longo de um *max batche* de 2000 *epochs*, onde podemos perceber um baixo erro em estado de regime permanente estabilizado do *avg loss*. Note que o modelo finaliza o processo de treinamento a uma taxa de erro bem pequena, aproximadamente 0.62 Esse índice é otimizado, uma vez que os sistemas atuais amplamente aceitos trabalham com valores em torno de 2 %. Assim temos um modelo que pode representar a classificação/detecção facial de uma pessoa e analisar sua aplicação em reconhecimento individual. Vale salientar, porém, que no processo de treinamento na *epoch*) 1300, o sistema apresentou uma queda no mAP para 71%, o que trouxe uma anomalia no processo de crescimento gradativo do mAP. Essa anormalidade momentânea pode ocorrer em momentos que a atualização de pesos se deu no momento de *backup* do sistema de dados, que nos momentos iniciais pode indicar uma redução de mAP antes do peso ser calculado pela rede neural. Portanto, poderá haver oscilações ao longo do processo de treinamento quando o lote de *backup* for extenso.

4.2.1 Reconhecimento de Face Individual

Aliado aos índices de desempenho otimizados obtido nos treinamentos, o modelo foi aplicado a fim de analisar a capacidade da nossa rede neural em não apenas detectar faces com o *dataset* Personal, mas também ter a capacidade de reconhecer e discernir pessoas em um determinado contexto com mais pessoas. Para tal, a RNA foi aplicada com inferência em dois ambientes de testes, uma com imagem recente e outro com imagens antigas aleatórias.

Figura 4.10 – Avg Loss Function para 2000 *epochs*.



Fonte: Do Autor (2020).

4.2.1.1 Reconhecimento de Face Individual em Imagens Antigas

As inferências a seguir fazem parte dos testes em imagens aleatórias obtidas pelo autor a fim de análises. As imagens a seguir dessa sub-seção possuem data de criação a aproximadamente de 10 anos atrás. Na Figura 4.11 o modelo obteve uma detecção de 29% mesmo para imagens antigas produzidas a mais de 10 anos. Isso nos mostra que o modelo teve uma capacidade de discernir pessoas, mesmo para uma arquitetura considerada como generalista segundo as referências. Nessa imagem, houve a detecção de outro membros na foto para um *threshold* abaixo de 6%, o que nos leva a crer que o sistema deve ser projetado com limiar baseado na média de detecções efetuadas, ou seja analisar qual média de limiar o modelo erra na detecção. Na Figura 4.12 apresentamos um resultado de inferência (detecção) em 45% com *threshold* mínimo de 11%. O que é satisfatório uma vez que essa imagem apresenta objetos de oclusão, neste caso o óculos que é uma das problemáticas quando não está contida em *datasets* de treino. Já na Figura 4.13 obtém-se uma inferência de 49% com *threshold* 16% (com erro de duplicata inferência, ocasionado por falha no NMS), o que é bastante satisfatório, podendo já ser empregado em sistemas de entretenimento que não necessitem de modelos de alta exatidão.

Figura 4.11 – Inferência em imagem antiga de aproximadamente 10 anos atrás.

Fonte: Do Autor (2020).

Figura 4.12 – Inferência em imagem antiga de aproximadamente 10 anos atrás.

Fonte: Do Autor (2020).

Figura 4.13 – Inferência em imagem antiga de aproximadamente 10 anos atrás.

Fonte: Do Autor (2020).

4.2.1.2 Reconhecimento de Face Individual em Imagens Recentes

A imagem 4.14 é uma imagem recente de aproximadamente 2 anos atrás, no qual foi aplicado ao modelo da rede neural a fim de analisar seu desempenho. Como esperado, o modelo obteve resultados otimizados assim como foi encontrado na Figura 4.9 (imagem do ano atual), onde o modelo obteve alta acurácia. Observe que o modelo alcança os incríveis 89% em inferência na imagem contendo mais pessoas, outro fator é seu limiar que foi um *threshold* de 6%, o que é otimizado uma vez que permite que o modelo opere com maior margem de erro sem comprometer o reconhecimento. Esse erro pode ser visto na Figuras 4.15, onde outro integrante masculino na foto foi identificado como "David A. Ribeiro", mostrando que o modelo erra nessa margem limiar. A detecção do indivíduo feminino na foto só foi observado para limiares abaixo de 1%, o que mostra que o modelo atingiu a capacidade e distinção na mesma categoria (pessoas).

Figura 4.14 – Reconhecimento em imagem com alcance de 89 % em inferência.

Fonte: Do Autor (2020).

4.2.2 Detecção de Faces Wider Face

A presente seção foca-se nos resultados de detecção faciais (faces humanas) em modelo de múltiplas pessoas em *dataset* Wider Face. Das 2 mil imagens do *dataset*, foram alocados 1000 imagens para treino e 200 para validação (seguindo a lógica de aproximadamente 80% e 20% como nos modelos anteriores, de modo não haver ampla divergência), no intuito de obter capacidade mais generalista do modelo, porém, sacrificando seu índice mAP. Foi aplicado nosso algoritmo de conversão de coordenadas, no qual adaptamos o *dataset* citado aos nossos requisitos de treinamento. Essa fase do projeto escolheu 4000 *epochs*, uma vez que, pela quantidade

Figura 4.15 – Erro de reconhecimento em baixo limiar *threshold* de detecção.

Fonte: Do Autor (2020).

de imagens selecionadas necessitaríamos de interações proporcionalmente maior. Em sistemas de detecção, podemos nos deparar com imagens de baixa qualidade ou até mesmo com seus atributos *annotation* incorretos. Visando corrigir tais problemas, foi feito uma análise manual das imagens, de modo a mitigar possíveis problemas de *false image*, no qual, há a composição de imagens desconexas com a classe do *dataset*. Vale salientar que o sistema foi treinado diariamente uma quantidade de 200 *epochs* em conta *free* do Colab. Seu mAP atingiu 86%, uma vez que as imagens utilizadas se distinguem amplamente uma das outras pelas análises manuais visuais, o que leva a crer que o *dataset* pode ser mais homogêneo para melhores índices. O gráfico da Figura 4.16 mostra o desempenho do nosso algoritmo ao longo de um *max batche* de 4000 iterações, onde podemos perceber um erro em estado de regime permanente do *avg loss* em 0.24%. Esse índice é otimizado, uma vez que os sistemas atuais amplamente aceitos trabalham com valores até 2% para aplicações profissionais. Assim temos um modelo que pode representar a classificação/detecção facial e analisar sua aplicação generalista. O modelo apresentou de modo geral os seguintes índices de desempenho da Tabela 4.9, no qual mostra uma alta capacidade generalista em inferência, e alto mAP para modelos de *dataset* considerados não homogêneos.

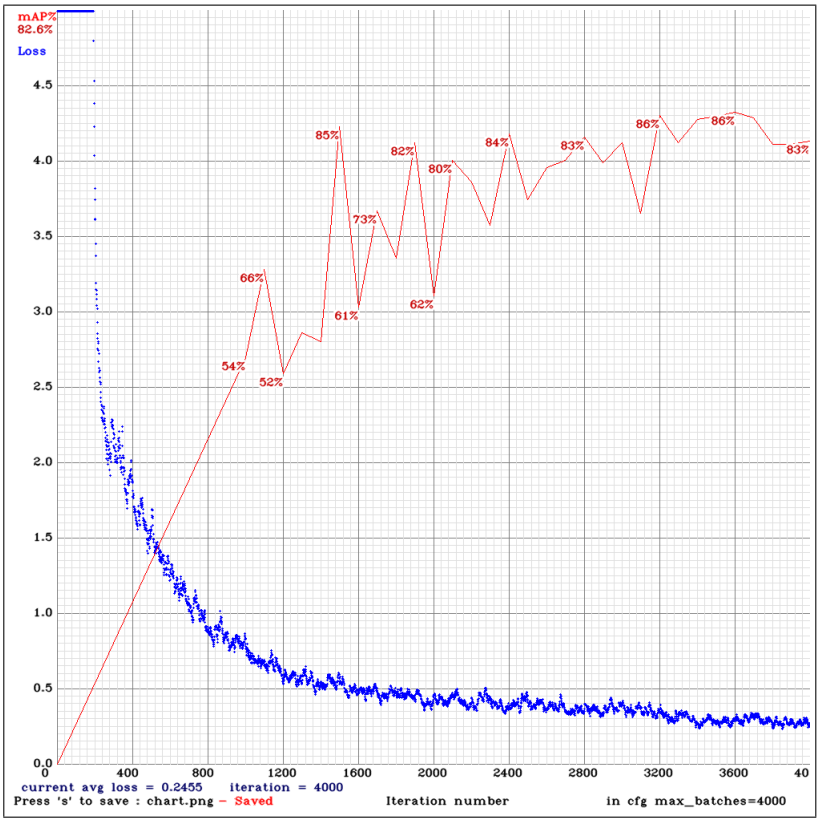
Tabela 4.9 – Análise mAP com peso 4000 *epochs*.

Precision	**TP**	**FP**	**FN**	**IoU**	**mAP [mAP@0.50]**	**Tempo [s]**	**Inferência**
0.81	290	70	84	61.32 %	86.04 %	5	91 %

Fonte: Do Autor (2020).

O modelo apresenta resultados otimizados em *dataset* para inferência, uma vez que alcança detecções que variam da ordem 82% até 100% em sua mediana. Isso vem do fato

Figura 4.16 – Avg Loss Function para 4000 *epochs*.



Fonte: Do Autor (2020).

do modelo apresentar alta capacidade de detecção generalista, porém com um aumento de FP devido às faces distintas contidas na base de dados. Assim, percebemos que o modelo pode apresentar uma mAP relativamente robusta, e com alta capacidade de detecção. Nessa fase do projeto, esse estado é otimizado, uma vez que não se deseja identificar um individuo específico, mas sim várias faces distintas que são apresentadas à rede neural. Observando o índice de detecção, o modelo foi aplicado à vídeos com biblioteca *open CV* adaptado ao *darknet*. Os resultados igualmente foram satisfatórios e podem ser encontrados online no link do presente projeto (<github.com/davidaugustoribeirobrazil/facial_recognition>). A primeira aplicação de detecção, foi na imagem da base de teste do Personal *dataset*, onde verificamos um alcance de aproximadamente 100% como vemos na Figura 4.17. Observamos que o modelo possui uma facilidade em detectar faces, não apenas em imagens isoladas, mas imagens com grande quantidade de pessoas, mantendo o mesmo tempo de detecção, uma vez que o YOLO passa imagem uma única vez pela rede neural, fazendo com que até pequenos detalhes faciais sejam detectados em plano de fundo, como na aplicação do nosso modelo à Figura 4.18, com *threshold* em 11% para eliminação de ruídos. O modelo também foi aplicado novamente à imagens de teste abordados na seção anterior, a fim de comparar o índice de detecção em inferência nas

mesmas imagens. As Figuras 4.19, 4.20 e 4.21 nos mostram que novamente o algoritmo se comportou de forma satisfatória, onde apresenta índices de detecção próximos à 100% com *threshold* em 25%.

Figura 4.17 – Detecção em imagem com alcance de 99% em inferência.

Fonte: Do Autor (2020).

Figura 4.18 – Detecção em imagem com alcance de 99% em inferência.

Fonte: Do Autor (2020).

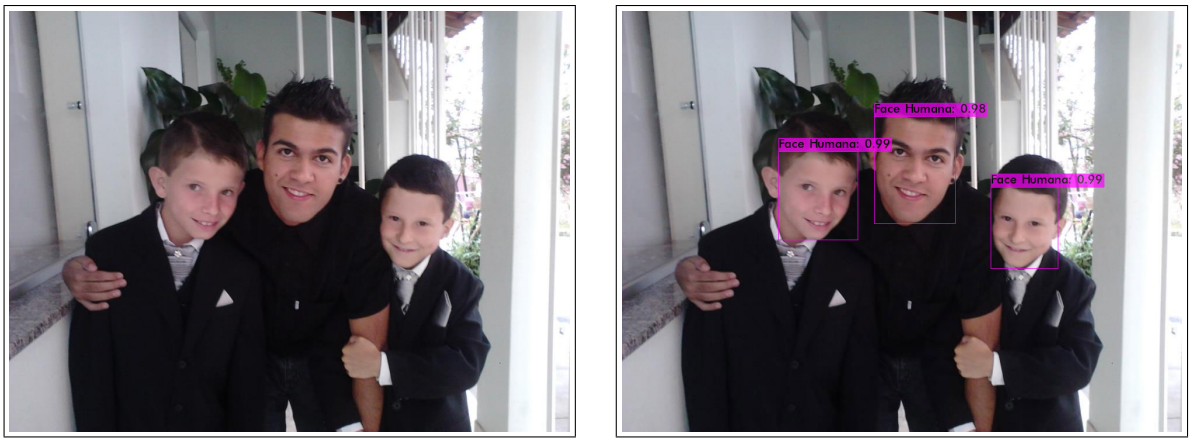
4.2.2.1 Detecções em Ambientes Multivariados

As Figuras 4.22, 4.23, 4.24, 4.25 e 4.26 apresentam os resultados do nosso modelo para prever faces em ambientes com mais de 10 pessoas. Um dos objetivos do projeto foi manter um alto índice de inferência para permitir a detecção de todas faces humanas em aglomerados contendo muitas pessoas. Para ambientes assim o modelo não precisa de uma alto mAP, mas

Figura 4.19 – Detecção em imagem com alcance de 99% em inferência.

Fonte: Do Autor (2020).

Figura 4.20 – Detecção em imagem com alcance de 99% em inferência.



Fonte: Do Autor (2020).

Figura 4.21 – Detecção em imagem com alcance de 99% em inferência.

Fonte: Do Autor (2020).

sim uma alta capacidade generalista de detecção, e nos prover a localização e o número de

pessoas em um ambiente determinado. Tanto para as imagens de teste, quanto aplicação em video, foram otimizadas e permitiram detecções com o modelo de 4000 *epochs*. Vale observar que o modelo não detecta muito bem recém nascidos (bebês), uma vez que a base dados não contempla muitas dessas faces (apenas 4 imagens). Outro fator observado, é que nosso modelo foi incapaz de detectar pessoas de costas, o que leva crer que uma base de dados de imagens nessa posição deve ser fundida ao *dataset* atual para nos prover a detecção de todas as pessoas em uma ambiente monitorado.

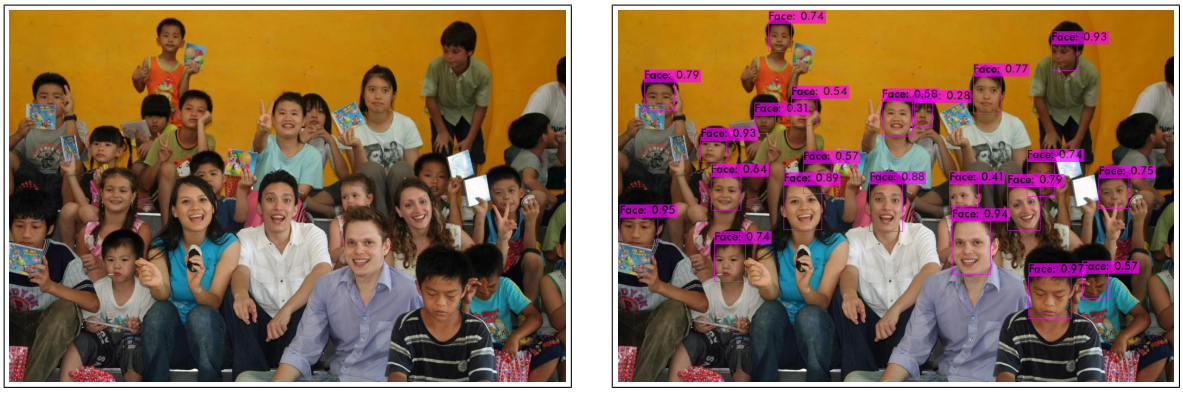
Figura 4.22 – Detecções em ambientes multivariados.

Fonte: Do Autor (2020).

Figura 4.23 – Detecções em ambientes multivariados.

Fonte: Do Autor (2020).

Figura 4.24 – Detecções em ambientes multivariados.

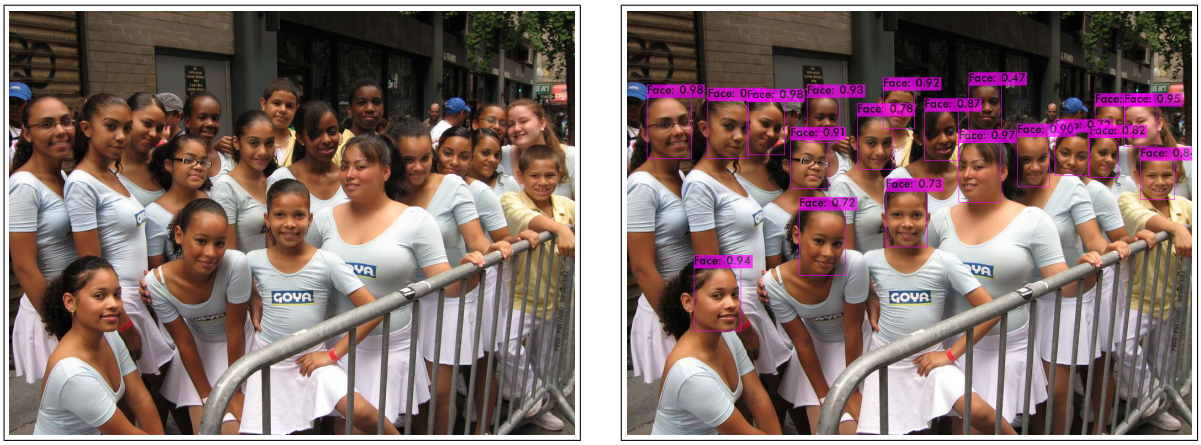


Fonte: Do Autor (2020).

Figura 4.25 – Detecções em ambientes multivariados.

Fonte: Do Autor (2020).

Figura 4.26 – Detecções em ambientes multivariados.



Fonte: Do Autor (2020).

4.3 Análise Comparativa

A análise de detecção de objetos foi desenvolvida no intuito de entender o funcionamento YOLO e adotar métricas e configurações para os dois modelos posteriores (Personal e Wider Face), e analisa suas diferenças. Esse comparativo se dá em virtude de se verificar quais aplicações são interessantes e específicas. A rede neural artificial YOLO v4 adaptada, obteve ótimos resultados em detecção e reconhecimento facial de indivíduo (o autor) nos testes realizados em *dataset* Personal. O mAP de 99.11% nos mostrou que o algoritmo é capaz de reconhecer em imagens de múltiplas pessoas, um indivíduo específico, mesmo este pertencendo à macro classe *Face Humana*. Aliado à uma mediana de testes com inferência de 98%, o torna um modelo altamente robusto para aplicações de segurança/autenticação, sistemas estes que carecem de alta precisão, agregado à uma validação otimizada para aplicação no cotidiano. Nesse quesito o modelo *Personal* apresentou resultados otimizados e estão de acordo com os objetivos propostos no presente projeto. Quando analisamos o segundo modelo de *dataset* Wider Face para *Face Humana*, percebemos que este possui um mAP também otimizado e um alto índice de inferência em detecção. Com um mAP de 86,04% e uma taxa de inferência média de 91%, o modelo alcança um bom desempenho. Essa alta capacidade o coloca como um modelo muito bom em sistemas que necessitem apenas detectar pessoas em ambientes e ao mesmo tempo contar e/ou controlar tráfego de pessoas. Certamente, o segundo modelo desenvolvido não é capaz de detectar uma pessoa específica, uma vez que foi treinada com milhares de faces distintas. Nota-se contudo que, o índice mAP não pôde ser elevado na atual configuração, pois apartir de um momento no treinamento (antes de concluir 4000 iterações), o modelo começou a tornar-se detalhista e entrou em *overfitting*. Isso provocou uma estagnação no *avg loss* em 0.24%, e ao mesmo tempo diminuiu a capacidade generalista do modelo, devido ao aumento do detalhamento no processo, e naturalmente, quanto maior for o *dataset*, maior será a capacidade de generalização da rede neural para a mesma quantidade de *epochs*. O tempo de treinamento deve ser analisado, para que haja o *early stopping* como critério de parada no menor erro possível, algo que no YOLO infelizmente é uma tarefa manual, mesmo que a arquitetura possibilite indicar ao longo de todo treinamento qual dos pesos foi o melhor, atribuindo à este como (*best weight*). Vale salientar porém que, um fator observado, foi que algumas imagens do *dataset* Wider Face não correspondem a *Faces Humanas*, como por exemplo, foram encontrados imagens de desenhos animados, bonecos, ou quadros. Essas imagens, no entanto, não são interessantes para a finalidade e aplicação propostas, o que faz-se necessário uma pré análise de todo *dataset*

manualmente no intuito de excluir imagens desconexas, pois imagens dessa natureza proporcionam aumento de FP no processo de validação. Portanto, nosso modelo alcançou um mAP igual ou superior para detecção específica individual e facial multivariada em relação aos índices de detecção citados no presente projeto que estavam em média 77.2% em mAP (VOC). O que nos mostra que a rede neural pode ser aprimorada e aperfeiçoada. Nota-se pelos estudos que, seu uso como modelo de reconhecimento individual é algo raro em aplicações até então, levando a crer que pode ser um dos um melhores modelos atuais para tal finalidade, em função da sua facilidade de adaptação e manipulação e possuir código fonte aberto, ao mesmo tempo que nos permite obter altos índices de desempenho em sistemas de classificação e detecção.

5 CONCLUSÃO

Em virtude das muitas arquiteturas de redes neurais existentes na atualidade, faz-se importante saber filtrar e indicar quais delas são mais apropriadas para cada tipo de nicho de aplicação. As conclusões vem a indicar uma forte necessidade de expandirmos as pesquisas na área de reconhecimento de padrões, mesmo em vista aos otimizados resultados. Há também a possibilidade de uso de validação cruzada no *dataset* Personal que por sua vez contém pequena quantidade de imagens, o que pode ser interessante para trabalhos futuros.

O presente projeto, focou-se inicialmente na verificação do uso arquitetura YOLO como uma poderosa ferramenta de detecção de objetos, uma vez que seus criadores a princípio o projetaram para essa finalidade. Contudo, foi necessário adquirir o conceito fundamental da arquitetura, bem como aprender aplicá-la a outros nichos. Percebeu-se que, os resultados para detecção de objetos possuíram os menores índices de desempenho entre os 3 *datasets* abordados. Isso se deve ao fato do número de iterações insuficientes, frente ao tamanho do banco de dados. Contudo, nota-se que os índices foram satisfatórios de acordo com os objetivos propostos, o que nos leva a crer que para um sistema mais simplista o primeiro modelo fica em nível de igualdade com algumas arquiteturas atuais.

Quando analisamos a segunda etapa do projeto, já notamos um resultado robusto, e que agrega à literatura nos estudos de reconhecimento facial para indivíduos específicos. Como vimos pelos resultados, alcançar um mAP de 99.11% com IoU em 82.56% não é uma tarefa trivial, e isso nos mostra que o modelo é altamente aplicável. O *dataset* Personal contudo, apresenta um banco de dados ligeiramente "pequeno", o que poderia ser ampliado em testes futuros. Os testes e resultados de inferência, nos mostraram que a quantidade de iterações (*epochs*) ao longo de aproximadamente um mês de treinamento foram suficientes segundo os cálculos teóricos definidos no trabalho. O sistema pode futuramente ser utilizado em modelos de de autenticação que venham a trabalhar com alto *threshold* de 90% (sistemas robustos) para sistemas de ponto em empresas ou de aplicações *mobile*, bem como implementar em sistemas de monitoramento de segurança em aeroportos a fim de identificar uma pessoa específica. Vale salientar que nosso modelo precisou filtrar ruídos de detecção em imagens, portanto, deve-se definir um limiar mínimo de acurácia em *threshold* de pelo menos 11%, uma vez que abaixo desse valor nosso sistema não foi capaz de realizar detecção adequadamente.

Os resultados de detecções em *dataset* Wider Face se mostraram satisfatórios também, em função dos objetivos propostos inicialmente, alcançando um mAP de 86,04% com IoU em

61.32%. Contudo houve alguns problemas referente ao tipo de de detecção, no qual crianças e recém-nascidos não foram identificados, o que leva a crer que o banco de dados não possui imagens o suficiente dessa natureza para corrigir. Uma solução seria adicionar um banco de dados adicional destes, e reiniciar um modelo nas configurações direcionadas a ela. Essas problemáticas podem ser facilmente resolvidas, desde que os recursos para treinamento estejam disponíveis.

Portanto, destes resultados podemos inferir que o sistema YOLO pode ser generalizável (mais de uma classe) ou específico (uma classe) em seu modelo, e isso depende especificamente de sua aplicação. Nosso modelo se tornou modular, podendo aplicá-los em diferentes contextos de forma simplista e atuando sistemas operacionais relativamente leves. Os modelos podem ser compartilhados e usados nos mais diversos sistemas, desde que citado a fonte. Em trabalhos futuros o modelo pode ser implementado em softwares de forma a facilitar a manipulação dos dados, bem como implementado em aplicativos de celular visando soluções. A sua capacidade de modularidade também deve ser aperfeiçoada, uma vez que isso agregará a toda comunidade que trabalha nessa arquitetura, mais autonomia em seus algoritmos.

REFERÊNCIAS

ABARS. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/abars/YoloKerasFaceDetection>.

ABUSHAHMA, R. I. H. et al. Region-based convolutional neural network as object detection in images. **In: 2019 IEEE 7th Conference on Systems, Process and Control (ICSPC)**, IEEE, Melaka, Malaysia, p. 264–268, Dec. 2019. Disponível em: <https://doi.org/10.1109/ICSPC47137.2019.9068011>.

ALOYSIUS, N.; M, G. A review on deep convolutional neural networks. **In: 2017 International Conference on Communication and Signal Processing (ICCSP)**, IEEE, Chennai, India, p. 588–592, Apr. 2017. Disponível em: <https://doi.org/10.1109/ICCSP.2017.8286426>.

ARACHCHILAGE, S. W.; IZQUIERDO, E. A framework for real-time face-recognition. **In: 2019 IEEE Visual Communications and Image Processing (VCIP)**, IEEE, Sydney, NSW, Australia, p. 1–4, Dec. 2019. Disponível em: <https://doi.org/10.1109/VCIP47243.2019.8965805>.

ARMITAGE, D. W.; K.OBER, H. A comparison of supervised learning techniques in the classification of bat echolocation calls. **Ecological Informatics**, Elsevier, v. 5, n. 6, p. 465–473, Nov. 2010. Disponível em: <https://doi.org/10.1016/j.ecoinf.2010.08.001>.

BISHOP, C. M. **Neural Networks for Pattern Recognition**. New York: Oxford University Press, 1997.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. **Computer Vision and Pattern Recognition**, arXiv, p. 1–17, Apr. 2020. Disponível em: <https://arxiv.org/abs/2004.10934>.

BOOK, D. L. Usando early stopping para definir o número de épocas de treinamento. 2019. Acesso em: 10 dez. 2020. Disponível em: <http://deeplearningbook.com.br>.

BORKAR, N. R.; KUWELKAR, S. Real-time implementation of face recognition system. **In: 2017 International Conference on Computing Methodologies and Communication (ICCMC)**, IEEE, Erode, India, p. 249–255, Jul. 2017. Disponível em: <https://doi.org/10.1109/ICCMC.2017.8282685>.

BRAGA, A. de P.; CARVALHO, A. P. de Leon F. de; LUDERMIR, T. B. **Redes Neurais Artificiais - Teoria e Aplicações**. 2. ed. [S.l.]: LTC, 2007.

BROWNLEE, J. A gentle introduction to transfer learning for deep learning. **Deep Learning for Computer Vision**, Machine Learning Mastery, Dec. 2017. Disponível em: <https://machinelearningmastery.com/transfer-learning-for-deep-learning>.

CAI, H. et al. The cross-depiction problem: computer vision algorithms for recognising objects in artwork and in photographs. **Computer Vision and Pattern Recognition**, arXiv, p. 1–12, May. 2015. Disponível em: <https://arxiv.org/abs/1505.00110>.

CARVALHO, A. C. P. de Leon Ferreira de. Redes neurais artificiais. USP, 2020. Acesso em: 15 jan. 2021. Disponível em: <https://sites.icmc.usp.br/andre/research/neural>.

CHANDRA, A. L. Mcculloch pitts neuron — mankind's first mathematical model of a biological neuron. Jul. 2018. Acesso em: 17 jan. 2021. Disponível em: <https://towardsdatascience.com/mcculloch-pitts-model>.

CINTRA, R. Introdução à neurocomputação. INPE, 2018. Acesso em: 05 dez. 2020. Disponível em: <http://www.inpe.br/elac2018>.

CIRESAN, D. C. et al. Mitosis detection in breast cancer histology images with deep neural networks. **In: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013**, Lecture Notes in Computer Science - Springer, Berlin, Heidelberg, v. 8150, p. 411–418, 2013. Disponível em: <https://doi.org/10.1007/978-3-642-40763-5_51>.

CUIMEI, L. et al. Human face detection algorithm via haar cascade classifier combined with three additional classifiers. **In: 2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)**, IEEE, Yangzhou, China, p. 483–487, Oct. 2017. Disponível em: <https://doi.org/10.1109/ICEMI.2017.8265863>.

CYBENKO, G. Approximation by superpositions of a sigmoid function. **Mathematics of Control, Signals and Systems**, Springer, v. 2, p. 303–314, Dec. 1989. Disponível em: <https://doi.org/10.1007/BF02551274>.

DAYBREAK. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/DayBreak-u/yolo-face-with-landmark>.

DEAN, T. et al. Fast, accurate detection of 100,000 object classes on a single machine. **In: 2013 IEEE Conference on Computer Vision and Pattern Recognition**, IEEE, Portland, OR, USA, p. 1814–1821, Jun. 2013. Disponível em: <https://doi.org/10.1109/CVPR.2013.237>.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. **In: 2009 IEEE Conference on Computer Vision and Pattern Recognition**, IEEE, Miami, FL, USA, p. 248–255, Jun. 2009. Disponível em: <https://doi.org/10.1109/CVPR.2009.5206848>.

ERHAN, D. et al. Scalable object detection using deep neural networks. **Computer Vision and Pattern Recognition**, arXiv, p. 1–8, Dec. 2013. Disponível em: <https://arxiv.org/abs/1312.2249>.

EVERINGHAM, M. et al. The pascal visual object classes (voc) challenge. **International Journal of Computer Vision**, Springer, v. 88, p. 303–338, Jun. 2010. Disponível em: <https://doi.org/10.1007/s11263-009-0275-4>.

FELZENSZWALB, P. F. et al. Object detection with discriminatively trained part-based models. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 32, n. 9, p. 1627–1645, Sep. 2010. Disponível em: <https://doi.org/10.1109/TPAMI.2009.167>.

FLECK, L. et al. Redes neurais artificiais: Princípios básicos. Revista Eletrônica Científica Inovação e Tecnologia, Medianeira, Paraná, Brasil, v. 7, n. 15, p. 47–57, Jun. 2016. Acesso em: 11 nov. 2020. Disponível em: <https://periodicos.utfpr.edu.br/recit/article/view/4330/Leandro>.

GAVRILESCU, R. et al. Faster r-cnn:an approach to real-time object detection. **In: 2018 International Conference and Exposition on Electrical And Power Engineering (EPE)**, IEEE, Iasi, Romania, p. 165–168, Oct. 2018. Disponível em: <https://doi.org/10.1109/ICEPE.2018.8559776>.

GINOSAR, S. et al. Detecting people in cubist art. **In: Computer Vision - ECCV 2014 Workshops**, Lecture Notes in Computer Science - Springer, v. 8925, p. 101–116, Mar. 2014. Disponível em: <https://doi.org/10.1007/978-3-319-16178-5_7>.

GIRSHICK, R. Fast r-cnn. **In: 2015 IEEE International Conference on Computer Vision (ICCV)**, IEEE, Santiago, Chile, p. 1440–1448, Dec. 2015. Disponível em: <https://doi.org/10.1109/ICCV.2015.169>.

GOOGLE. Colaboratory. Google Colaboratory - Google Inc., 2020. Acesso em: 03 dez. 2020. Disponível em: <https://research.google.com/colaboratory/intl/pt-BR/faq>.

GRANATYR, J. Yolo. 2020. Acesso em: 04 jan. 2021. Disponível em: <https://iaexpert.academy>.

HAYKIN, S. **Neural networks: A comprehensive foundation**. 2. ed. [S.l.]: Prentice Hall. Pearson Education, 2005.

HE, K. et al. Spatial pyramid pooling in deep convolutional networks for visual recognition. **In: IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 37, n. 9, p. 1904–1916, Sep. 2015. Disponível em: <https://doi.org/10.1109/TPAMI.2015.2389824>.

HE, K. et al. Deep residual learning for image recognition. **In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, IEEE, Las Vegas, NV, USA, p. 770–778, Jun. 2016. Disponível em: <https://doi.org/10.1109/CVPR.2016.90>.

HOIEM, D.; CHODPATHUMWAN, Y.; DAI, Q. Diagnosing error in object detectors. **In: Computer Vision – ECCV 2012**, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, v. 7574, p. 340–353, 2012. Disponível em: <https://doi.org/10.1007/978-3-642-33712-3_25>.

HOPFIELD, J. J. Neural networks and physical systems with emergent collective properties. **In: Proceedings of the National Academy of Sciences of the United States of America**, National Academy of Sciences, USA, v. 79, p. 2554–2558, Apr. 1982. Disponível em: <https://doi.org/10.1073/pnas.79.8.2554>.

JMU. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/jmu201521121021/FaceDetector-Base-Yolov3-spp>.

KATHURIA, A. What's new in yolo v3 ? Towards Data Science, Apr. 2018. Acesso em: 11 out. 2020. Disponível em: <https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>.

KOMARINSKI, P. **Automated Fingerprint Identification Systems (AFIS)**. 1. ed. [S.l.]: Academic Press, 2004.

KOVASHKA, A. Intersection over union (iou) for object detection. Department of Computer Science - University of Pittsburgh, 2018. Acesso em: 02 out. 2020. Disponível em: <https://people.cs.pitt.edu/~kovashka/teaching>.

LAPIX. Visão computacional - métricas - mean average precision. **The Cyclops Research Group - LAPIX - Image Processing and Computer Graphics Lab**, Universidade Federal de Santa Catarina – UFSC, Florianópolis, SC, Brazil, Out. 2020. Acesso em: 28 nov. 2020. Disponível em: <http://www.lapix.ufsc.br>.

LEITE, T. M. Redes neurais, perceptron multicamadas e o algoritmo backpropagation. 2018. Acesso em: 22 nov. 2020. Disponível em: <https://medium.com/ensina-ai>.

LENC, K.; VEDALDI, A. R-cnn minus r. **Computer Vision and Pattern Recognition**, arXiv, p. 1–9, Jun. 2015. Disponível em: <https://arxiv.org/abs/1506.06981>.

LI, L. et al. A review of face recognition technology. **IEEE Access**, IEEE, v. 8, p. 139110 – 139120, Jul. 2020. Disponível em: <https://doi.org/10.1109/ACCESS.2020.3011028>.

LIN, T.-Y. et al. Feature pyramid networks for object detection. **Computer Vision and Pattern Recognition**, arXiv, p. 1–10, Apr. 2017. Disponível em: <https://arxiv.org/abs/1612.03144>.

LIN, T.-Y. et al. Microsoft coco: Common objects in context. **In: Computer Vision – ECCV 2014**, Springer, v. 8693, 2014. Disponível em: <https://doi.org/10.1007/978-3-319-10602-1_48>.

LIN, V. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/VictorLin000/YOLOv3_mask_detect>.

LIU, Y. The confusing metrics of ap and map for object detection. Oct. 2018. Acesso em: 09 dez. 2020. Disponível em: <https://medium.com/@yanfengliux/the-confusing-metrics-of-ap-and-map-for-object-detection>.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, Springer, v. 5, p. 115–133, Dec. 1943. Disponível em: <https://doi.org/10.1007/BF02478259>.

MINSKY, M.; PAPERT, S. **Perceptrons: an introduction to computational geometry**. New York: MIT Press, 1970. v. 17. 501-522 p. Department of Theoretical and Applied Mechanics, Cornell University. Disponível em: <https://core.ac.uk/download/pdf/82206249.pdf>.

MIRANDA, F. A.; FREITAS, S. R. C. de; FAGGION, P. L. Integração e interpolação de dados de anomalias ar livre utilizando-se a técnica de rna e krigagem. Boletim de Ciências Geodésicas, v. 15, n. 3, p. 428–443, Jun. 2009. Acesso em: 27 nov. 2020. Disponível em: <https://revistas.ufpr.br/bcg/article/view/15507/10358>.

MOREIRA, S. Rede neural perceptron multicamadas. Dec. 2018. Acesso em: 21 nov. 2020. Disponível em: <https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas>.

NAZARIO, S. L. S. Caracterização de leite utilizando técnicas de ultra-som e redes neurais. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Estadual Paulista, UNESP, p. 102, Nov. 2007. Disponível em: <https://repositorio.unesp.br/bitstream/handle/11449/87250/nazario_sls_me_ilha.pdf?sequence=1&isAllowed=y>.

NGUYEN, T. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/sthanhng/yoloface>.

NUNES, F. T. Técnicas de biometria baseadas em padrões faciais e sua utilização na segurança pública. UFSC - Programa de Pós Graduação da Universidade Federal de Santa Catarina, p. 65, Jul. 2015. Disponível em: <https://repositorio.ufsc.br/handle/123456789/180402>.

PARHAM, J. et al. Animal population censusing at scale with citizen science and photographic identification. **AI**, AI Access Foundation, United States, p. 37–44, Jan. 2017.

PJREDDIE. Images of yolo. Dec. 2019. Acesso em: 13 set. 2020. Disponível em: <https://pjreddie.com>.

PUROHIT, A. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/adityap27/face-mask-detector>.

QUADROS, J. R. de T. et al. Facial recognition system for automatic presence control in a classroom. **In: 2017 12th Iberian Conference on Information Systems and Technologies (CISTI)**, IEEE, Lisbon, p. 1–6, Jun. 2017. Disponível em: <https://doi.org/10.23919/CISTI.2017.7975712>.

QUALCOMM. Training, testing and evaluating machine learning models. **Learning Resources**, Machine Learning Mastery, Dec. 2020. Disponível em: <https://developer.qualcomm.com>.

REDMON, J. et al. You only look once: Unified, real-time object detection. **In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, IEEE, Las Vegas, NV, USA, p. 779–788, Jun. 2016. Disponível em: <https://doi.org/10.1109/CVPR.2016.91>.

REDMON, J.; FARHADI, A. Yolo9000: Better, faster, stronger. **In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, IEEE, Honolulu, HI, USA, p. 6517–6525, Jul. 2017. Disponível em: <https://doi.org/10.1109/CVPR.2017.690>.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. **Computer Vision and Pattern Recognition**, arXiv, p. 1–6, Apr. 2018. Disponível em: <https://arxiv.org/abs/1804.02767>.

REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. **Computer Vision and Pattern Recognition**, arXiv, p. 1–14, Jan. 2016. Disponível em: <https://arxiv.org/abs/1506.01497>.

REN, S. et al. Object detection networks on convolutional feature maps. **Computer Vision and Pattern Recognition**, arXiv, p. 1–8, Aug. 2016. Disponível em: <https://arxiv.org/pdf/1504.06066>.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, n. 6, 1958. Disponível em: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf>.

SADEGHI, M. A.; FORSYTH, D. 30hz object detection with dpm v5. **In: Computer Vision – ECCV 2014**, Lecture Notes in Computer Science. Springer, v. 8689, p. 65–79, 2014. Disponível em: <https://doi.org/10.1007/978-3-319-10590-1_5>.

SAEED, K.; NAGASHIMA, T. **Biometrics and Kansei Engineering**. 1. ed. Springer-Verlag New York, 2012. ISBN 978-1-4614-5608-7. Disponível em: <https://doi.org/10.1007/978-1-4614-5608-7>.

SAHU, M.; DASH, R. Study on face recognition techniques. **In: 2020 International Conference on Communication and Signal Processing (ICCSP)**, IEEE, Chennai, India, p. 613–616, Jul. 2020. Disponível em: <https://doi.org/10.1109/ICCSP48568.2020.9182358>.

SALESFORCE. Machine learning e deep learning. 2018. Acesso em: 12 out. 2020. Disponível em: <https://www.salesforce.com/br/blog/2018/4>.

SANTANA, M. Deep learning: do conceito às aplicações. Jul. 2018. Acesso em: 13 out. 2020. Disponível em: <https://medium.com/data-hackers/deep-learning-do-conceito>.

SHARMA, S.; BHATT, M.; SHARMA, P. Face recognition system using machine learning algorithm. **In: 2020 5th International Conference on Communication and Electronics Systems (ICCES)**, IEEE, Coimbatore, India, p. 1162–1168, Jun. 2020. Disponível em: <https://doi.org/10.1109/ICCES48766.2020.9137850>.

SHUO, Y. et al. Wider face: A face detection benchmark. **In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, 2016. Acesso em: 12 fev. 2021. Disponível em: <http://shuoyang1213.me/WIDERFACE>.

SILVA, D. **Rede Neural Artificial Aplicada ao Reconhecimento de Padrões usando o Método Back-Propagation**. [S.l.]: Monografia (Bacharel em Ciência da Computação) - Centro Universitário do Triângulo, Unit, 2000. 51 p.

SINGH, G.; GOEL, A. K. Face detection and recognition system using digital image processing. **In: 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)**, IEEE, Bangalore, India, p. 348–352, Mar. 2020. Disponível em: <https://doi.org/10.1109/ICIMIA48430.2020.9074838>.

SISWANTO, A. R. S.; NUGROHO, A. S.; GALINIUM, M. Implementation of face recognition algorithm for biometrics based time attendance system. **In: 2014 International Conference on ICT For Smart Society (ICISS)**, IEEE, Bandung, Indonesia, p. 1–6, Sep. 2014. Disponível em: <https://doi.org/10.1109/ICTSS.2014.7013165>.

SURASAK, T. et al. Histogram of oriented gradients for human detection in video. **In: 2018 5th International Conference on Business and Industrial Research (ICBIR)**, IEEE, Bangkok, Thailand, p. 172–176, May 2018. Disponível em: <https://doi.org/10.1109/ICBIR.2018.8391187>.

SWIEZEWSKI, J. What is yolo object detection? May. 2020. Acesso em: 10 Set. 2021. Disponível em: <https://appsilon.com/object-detection-yolo-algorithm>.

THEODORIDIS, S.; KOUTROUMBAS, K. **Pattern Recognition**. 4. ed. [S.l.]: Academic Press, 2008.

TITO, N. Como escolher o número de neurônios na camada oculta de uma rede neural? Apr. 2020. Acesso em: 05 ago. 2020. Disponível em: <https://nathaliatito.medium.com>.

UIJLINGS, J. R. R. et al. Selective search for object recognition. **International Journal of Computer Vision**, Springer, v. 104, p. 154–171, Apr. 2013. Disponível em: <https://doi.org/10.1007/s11263-013-0620-5>.

VIOLA, P.; JONES, M. Robust real-time face detection. **In: Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001**, IEEE, Vancouver, BC, Canada, p. 1–1, Jul. 2001. Disponível em: <https://doi.org/10.1109/ICCV.2001.937709>.

WENG, L. Anchor box. Github, Dec. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/lilianweng>.

XI, X. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/xialuxi/yolov5_face_landmark>.

XUEHONG, T. Face recognition system and it's application. **In: 2009 First International Conference on Information Science and Engineering**, IEEE, Nanjing, China, p. 1244–1245, Dec. 2009. Disponível em: <https://doi.org/10.1109/ICISE.2009.583>.

YAN, J. et al. The fastest deformable part model for object detection. **In: 2014 IEEE Conference on Computer Vision and Pattern Recognition**, IEEE, Columbus, OH, USA, p. 2497–2504, Jun. 2014. Disponível em: <https://doi.org/10.1109/CVPR.2014.320>.

ZADEH, L. Fuzzy logic, neural networks and soft computing. **In: Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks**, p. 13–14, 1992.

ZHU, C. 2020. Acesso em: 01 mar. 2021. Disponível em: <https://github.com/Chenyang-ZHU/YOLOv3-Based-Face-Detection-Tracking>.