UNIVERSIDADE FEEVALE

PABLO GILVAN BORGES

APLICAÇÃO PRÁTICA DE TÉCNICAS DE DEEP LEARNING NA DETECÇÃO DE RACHADURAS EM PAREDES DE ALVENARIA

Novo Hamburgo
2022

PABLO GILVAN BORGES

APLICAÇÃO PRÁTICA DE TÉCNICAS DE DEEP LEARNING NA DETECÇÃO DE RACHADURAS EM PAREDES DE ALVENARIA

Trabalho de Conclusão de Curso, apresentado como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação pela Universidade Feevale.

Orientador: Profa. Dra. Marta Rosecler Bez

Novo Hamburgo
2022

RESUMO

A construção civil é um dos pilares que sustentam o avanço da sociedade. Compondo tais avanços surgiram, no decorrer da história, produtos, técnicas e todo um conjunto de ferramentas com a finalidade de impulsionar a sociedade através da engenharia. Entre estes surgiu o conceito de SHM (*Structural Health Monitoring*), cujo principal objetivo é monitorar toda e qualquer mudança que uma estrutura possa apresentar, com a finalidade de verificar possíveis dados sobre a integridade desta estrutura. Os sistemas de SHM se apoiaram fortemente no uso de sensores e dispositivos de monitoramento específicos para este fim, o que tornava tais sistemas com um custo muito elevado. Decorrente das conquistas tecnológicas começou-se a pesquisar outras formas mais simples e baratas no monitoramento de estruturas, entre várias abordagens, o uso de captação de imagens de estruturas a fim de realizar análises de forma não intrusiva. Tais técnicas fazem uso de Visão Computacional, tendo como base algoritmos de *Deep Learning* para fazer esta tarefa. Este trabalho tem por finalidade aplicar tais técnicas e metodologias levantadas em pesquisas na área, a fim de desenvolver uma aplicação prática para detecção de rachaduras em paredes de alvenaria, fazendo o uso de imagens de tais estruturas. A abordagem utilizada para obtenção do resultado desejado foi empregar uma técnica de *deep learning* conhecida como segmentação semântica. O trabalho obteve resultados parciais onde a acuraria foi de 98.8%. Demonstra, assim, que o uso de visão computacional apresenta um grande avanço para inspeção e monitoria predial.

Palavras-chave: *Deep Learning*; *Structural Health Monitoring*; *Computer Vision*; Detecção; Alvenaria.

ABSTRACT

Civil construction is one of the pillars that support the advancement of society. Composing through construction sets, throughout the history of technical tools and the entire society of construction projects. Among these emerged the concept of SHM (Struct Health Monitoring), whose main data on the simplified structure can be presented to all changes that a presentation, with the purpose of possible control to this structure. SHM has selected sensor systems and monitoring devices specific for this purpose, which makes monitoring systems very specific with an elevator. As a result of technological achievements, other simpler and cheaper ways of monitoring structures began to be researched, among approaches, the use of capturing images of structures to perform exams in a non-intrusive way. Such techniques make use of Computer Vision, based on Deep Learning algorithms to do this task. This work aims to apply such techniques and methodologies raised in research in the area, to develop an application for the detection of cracks in masonry walls, making use of practices of such structures. The approach used to obtain the desired result was a deep learning technique known as semantics. The partial work results where the curatorship obtained 98.8%. Thus, proving that computer vision presents a great advance for the use of building monitoring.

Keywords: Deep Learning; Structural Health Monitoring; Computer Vision; Detection; Masonry.

LISTA DE FIGURAS

Figura 1 - Exemplos de uma rede neural 22
Figura 2 - Estrutura de um Perceptron 23
Figura 3 - Separação linear do conjunto de dados 24
Figura 4 - Multilayer Perceptron 24
Figura 5 - Separação linear de dados por AND e OR 25
Figura 6 - Representação de um problema XOR, linearmente inseparável por uma única linha 25
Figura 7 - Problemas de classificação 26
Figura 8 - Exemplo interpretativo de como se comportaria o aprendizado de um modelo 27
Figura 9 - Representação visual de uma RN para classificação de dígitos 28
Figura 10 - Fórmula MSE 28
Figura 11 - Cálculo da MSE considerando todos os possíveis valores 29
Figura 12 - Ponto representando o mínimo de uma função 29
Figura 13 - Função com vários mínimos 30
Figura 14 - Gradient Descent sendo aplicado com learning rate baixo 31
Figura 15 - Gradient Descent sendo aplicado com learning rate muito alto 32
Figura 16 - Função Logística 32
Figura 17 - Problemas para convergir ao mínimo da função 34
Figura 18 – Exemplo de downsample 35
Figura 19 - Regra para classificação de imagens 39
Figura 20 - Arquitetura do projeto de Flah et al.(2020) 39
Figura 21 - Tabela comparativa dos resultados obtidos 41
Figura 22 - Resultados obtidos com o modelo de segmentação 42
Figura 23 - Resultados a nível de objeto 43
Figura 24 - Resultados a nível de objetos planos 43
Figura 25 - Resultados a nível de estruturas 44
Figura 26 - Arquitetura do projeto de Jenkins *et al*. (2018) 46
Figura 27 - Resultados obtidos por Jenkins *et al*. (2018) 46
Figura 28 - Resultados obtidos (Phi-Net) 49
Figura 29 - Resultados obtidos (Pohang) 49
Figura 30 - Resultados obtidos (Purdue University) 49
Figura 31 - Fluxo de desenvolvimento rede Flah et al. (2020) 52
Figura 32 - Configurações do Ambiente de desenvolvimento 55
Figura 33 - Informações do artigo de Flat *et al*. (2021) 56
Figura 34 - Configuração da rede neural com Keras 56
Figura 35 - Sumário da configuração da rede 57
Figura 36 - Código de configuração da rede em Keras 57
Figura 37 - Acurácia e Taxa de erro 58
Figura 38 - Exemplo de resultado de detecção de objetos 60
Figura 39 - Exemplo de resultado com múltiplas áreas identificadas 61
Figura 40 - Exemplo do emprego de detecção de objetos 62
Figura 41 - Ferramenta LabelIMG 62
Figura 42 - Exemplo de aplicação de segmentação 63
Figura 43 - Fluxo de desenvolvimento de Segmentação 65

Figura 44 - Imagem do conjunto de dados após redimensionamento 66
Figura 45 - Máscara da imagem 66
Figura 46 - Configuração modelo de Flah et al. (2020) 68
Figura 47 - Bloco de código da nova configuração da rede..... 68
Figura 48 - Bloco de mesmo nível na etapa de *upsampling* 69
Figura 49 - Sumário da rede 69
Figura 50 - Resultados da rede neural(acurácia) 69
Figura 51 - Log de treinamento da rede 71
Figura 52 - Resultados parciais 72

LISTA DE ABREVIATURAS E SIGLAS

ACI	*American Concrete Institute*
ASCE	*American Society of Civil Engineers*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CNN	*Convolutional Neural Network*
COCO	*Common Objects in Context*
CPU	*Central Processing Unit*
DNN	*Deep Neural Networks*
FHWA	*Federal Highway Administration*
FPN	*Feature Pyramid Network*
GB	*Gigabyte*
GPU	*Graphic Processing Unit*
LTU	*Linear Threshold Unit*
MLP	*Multi-Layer Perceptron*
MSE	*Mean Squared Error*
RAM	*Random Access Memory*
ReLU	*Rectified Linear Unit*
RGB	*Red, Green, Blue*
RNN	*Recurrent Neural Network*
R-CNN	*Region Based Convolutional Neural Networks*
SHM	*Structural Health Monitoring*
SVM	*Support Vector Machines*
TAHN	*The Hyperbolic Tangent Function*

SUMÁRIO

1 INTRODUÇÃO 12
2 NECESSIDADE DE MONITORAMENTO ESTRUTURAL 15
2.1 SHM 15
2.1.1 Aquisição de dados 16
2.1.2 Comunicação de dados 17
2.1.3 Processamento de dados 17
2.1.4 Armazenamento/retenção de dados 17
2.1.5 Diagnóstico 17
2.2 ABRANGÊNCIA DE HSM 18
2.3 CONSIDERAÇÕES DO CAPÍTULO 18
3 DEEP LEARNING 20
3.1 REDES NEURAIS 21
3.2 PERCEPTRON 22
3.3 APRENDIZADO 26
3.4 FUNÇÃO DE PERDA 28
3.5 GRADIENT DESCENT (GRADIENTE DESCENDENTE) 30
3.7 OTIMIZADOR 33
3.8 DROPOUT 34
3.9 POOLING LAYERS 34
3.10 VARIAÇÕES DE REDES NEURAIS 35
3.11 CONSIDERAÇÕES DO CAPÍTULO 35
4 TRABALHOS CORRELATO 37
4.1 CLASSIFICATION AND QUANTIFICATION OF CRACKS IN CONCRETE STRUCTURES USING DEEP LEARNING IMAGE-BASED TECHNIQUES (FLAH ET. AL., 2020) 37
4.1.1 Dados utilizados no trabalho de Flah *et al*. (2020) 37
4.1.1.1 Anotação dos dados no trabalho de Flah et al.(2020) 38
4.1.2 Arquitetura utilizada no trabalho de Flah *et al*. (2020) 38
4.1.3 Conclusão do trabalho de Flah *et al*. (2020) 40
4.2 DEEP CASCADED NEURAL NETWORKS FOR AUTOMATIC DETECTION OF STRUCTURAL DAMAGE AND CRACKS FROM IMAGENS (BAI ET AL., 2020) 40
4.2.1 Dados utilizados no trabalho de Bai *et al*. (2020) 40
4.2.2 Arquitetura utilizada no trabalho de Bai *et al*. (2020) 41
4.2.3 Conclusão do trabalho de Bai *et al*. (2020) 44

4.3 A DEEP CONVOLUTIONAL NEURAL NETWORK FOR SEMANTIC PIXEL-WISE SEGMENTATION OF ROAD AND PAVEMENT SURFACE CRACKS (JENKINS *ET AL*., 2018) 45
4.3.1 Dados utilizados por Jenkins *et al*. (2018) 45
4.3.2 Arquitetura utilizada por Jenkins *et al*. (2018) 45
4.3.3 Resultados obtidos por Jenkins *et al*. (2018) 46
4.3.4 Conclusão do trabalho do Jenkins *et al*. (2018) 47
4.4 DETECTING CRACKS AND SPALLING AUTOMATICALLY IN EXTREME EVENTS BY END-TO-END DEEP LEARNING FRAMEWORKS (BAI ET AL., 2021) 47
4.4.1 Dados utilizados por Bai *et al*. (2021) 47
4.4.2 Arquitetura utilizada por Bai *et al.* (2021) 48
4.4.3 Resultados obtidos por Bai *et al*. (2021) 48
4.4.4 Conclusão do trabalho de Bai *et al*. (2021) 49
5 DESENVOLVIMENTO 51
5.1 APLICAÇÃO DA REDE DE FLAH *ET AL*. (2020) 51
5.1.1 Fluxo de atividades 51
5.1.2 Conjunto de dados 52
5.1.3 Python 53
5.1.4 Keras 53
5.1.5 Tensorflow 54
5.1.6 Pandas 54
5.1.7 Numpy 55
5.1.8 Ambiente 55
5.1.9 Codificação 55
5.1.10 Alterações realizadas 58
5.1.11 Resultados obtidos 58
5.1.12 Considerações 59
5.2 NOVA ABORGAGEM 59
5.2.1 Detecção de objetos 60
5.2.2 Limitações no uso de detecção de objetos 62
5.2.3 Segmentação semântica 63
5.2.4 Upsampling 63
5.2.5 Definições de passos a seguir 64
5.2.6 Desenvolvimento de aplicação com Segmentação Semântica 64
5.2.7 Fluxo de atividades para nova rede 64
5.2.8 Conjunto de dados 66
5.2.9 Equalizando o conjunto de dados 67

5.2.10 U-Net 67
5.2.11 Codificação da rede 67
5.2.12 Resultados parciais 70
6 Conclusão 73
REFERÊNCIAS BIBLIOGRÁFICAS 75

1 INTRODUÇÃO

A vida útil de qualquer construção está na sua manutenção periódica, visto que todo e qualquer material sofre mudanças com o tempo, seja dilatação, efeitos de trepidação, clima e outros fatores que acabam impactando na preservação de uma estrutura (BALAGEAS, FRITZEN, 2006). Somente no Brasil, tem-se vários casos de desabamentos, sejam estas obras públicas (GLOBO G1, 2021) ou privadas (GLOBO G1, 2022), caso de danos estruturais que acabam causando até mesmo a perda de vidas. Muitos desses casos são resultados de investimentos em materiais de baixa qualidade (Correio do Povo, 2010) ou mesmo por negligência com a manutenção da construção (Correio Braziliense, 2021). Sabe-se que muitas das obras públicas que acabam acometendo em desastres são frutos de negligência e falta de manutenção periódica, fato este não exclusivo do Brasil (Jovem Pan, 2020). Por mais que muitos municípios estejam aprovando leis municipais para vistorias, seja com a finalidade de assegurar qualidade na obra empreendida ou mesmo de manutenção preventiva de obras públicas (Correio do Estado, 2019), isto ainda não é uma realidade na esfera federal. Como apontado no documento do XIX COBREP (MATOS JR. *et al*., 2017), somente um projeto de lei tramita no senado.

Com o intuito de fornecer um controle e análises não intrusivas em inspeções aeroespaciais, civis e mecânicas, formou-se o conceito de SHM (*Structural Health Monitoring*). A SHM foca em manter uma análise constante de uma estrutura, seja através da captação de dados de sensores ou de outras ferramentas eletrônicas (KAYA; SAFAK, 2014). Até certo tempo, como apontado por Ozer e Feng (2022), os custos para se manter um sistema de SHM não eram de baixo investimento, dependia-se do uso de sensores sensíveis e pessoal especializado no uso de tais ferramentas. Graças aos avanços que surgiram na área da engenharia e da computação, hoje dispõe-se de vários tipos de dispositivos com a finalidade de auxiliar no monitoramento e manutenção de obras de engenharia. Mesmo *smartphones* já possuem muitos dos recursos que podem ser empregados em sistemas SHM (OZER; FENG, 2022). Este avanço tecnológico proporcionou levar a SHM para outro patamar, onde vem sendo adotado amplamente nas duas últimas décadas (YE *et al*., 2019).

A pesquisa por uso de técnicas computacionais, especialmente as voltadas para o uso de inteligência artificial (IBM, 2022), como o caso de *Deep Learning*

(GOODFELLOW *et al*., 2016), tem procurado fornecer meios não intrusivos no emprego de análise de estruturas. Trabalhos como os apresentados por Wang *et al*. (2019) empregam técnicas de *Deep Learning* em imagens capturadas por dispositivos móveis (ou mesmo câmeras estáticas) para observação e detecção de danos em construções históricas. Outros trabalhos sugerem um modelo de *Deep Learning* para detecção de danos em estruturas de concreto, como é o caso do modelo apresentado por Flah *et al*. (2020). Jr. *et al.* (2019) também apresenta um *overview* sobre conjuntos de técnicas de Visão Computacional (Intel, 2021) utilizando-se de imagens, ou mesmo vídeo, obtidas através de câmeras estáticas ou drones. YE *et al*. (2019) faz um compilado de vários trabalhos que visam o emprego de *Deep Learning* no processamento de imagens para controle e auxílio na manutenção de estruturas.

Visando a necessidade de um sistema não intrusivo e de custo relativamente baixo em relação ao controle periódico do estado de estruturas públicas, este trabalho tem como finalidade realizar uma aplicação prática, com base nos trabalhos encontrados. Foi aplicada uma das técnicas estudadas, comparando com os achados de diversos autores, tendo como fonte de dados o processamento de imagens obtidas por câmeras. Sendo assim, capaz de detectar rachaduras em paredes de alvenaria, possibilitando no futuro uma forma de observar a progressão de rachaduras.

O objetivo geral deste trabalho é desenvolver uma aplicação que se utilize de técnicas de *Deep Learning* para detecção de rachaduras em paredes de alvenaria através de captura de imagens, tendo como foco os seguintes itens:

- Validar projetos de pesquisa que apresentaram modelos de algoritmos e técnicas que fazem uso de *deep learning*, para detecção de rachaduras através de captura de imagens.
- Verificar qual dos trabalhos apresenta os melhores resultados, comparando recursos utilizados e resultados obtidos.
- Analisar a melhor forma de fazer a captação de imagens, se via *streaming* de vídeo ou fotos.
- Desenvolver uma aplicação que faça uso dos itens estudados a fim de servir como modelo de uma aplicação real.
- Validar a aplicação desenvolvida.

No Capítulo 2 é apresentada a definição de SHM e o motivo de sua aplicação estar cada vez mais sendo instigada por governos e agências reguladoras nos setores de engenharia. Também são descritos os principais pilares de um sistema SHM, fornecidos pelo órgão do governo canadense chamada ISIS Canada. O capítulo termina com uma breve descrição de alternativas para os sistemas que, até então, se baseavam somente em sensores.

O Capítulo 3 apresenta o conceito de *Deep Learning*. Apresenta desde suas origens e as pesquisas que iniciaram e formaram a base para o que se tem hoje. São descritos os conceitos de Redes Neurais e detalhada sua estrutura.

O Capítulo 4 apresenta quatro trabalhos selecionados, cujo foco é a aplicação de *Deep Learning* e captura de imagens para detecção de danos estruturais, para serem estudados e comparados. A finalidade deste capítulo é apresentar, de forma resumida, a proposta de cada trabalho, como os pesquisadores abordaram o problema em questão, as técnicas utilizadas e os resultados obtidos.

O Capítulo 5 apresenta o desenvolvimento do trabalho. Quais técnicas foram empregadas e os caminhos escolhidos para alcançar o objetivo proposto. Também são apresentadas as dificuldades encontradas. No mesmo capítulo são abordadas outras técnicas de *deep learning* que foram aprendidas no decorrer do estudo. Os resultados são exibidos de forma visual. Na sequência apresenta-se as conclusões do trabalho.

2 NECESSIDADE DE MONITORAMENTO ESTRUTURAL

O conceito de inspeção de estruturas não é algo recente, casos datam do século IXX (ISIS Canadá, 2001). Assim como o corpo humano deveria receber uma avaliação periódica para detecção e prevenção de possíveis problemas de saúde, toda e qualquer estrutura está sujeita a sofrer deterioração dos seus componentes e materiais com o tempo e uso, o que pode resultar em perdas financeiras e mesmo fatalidades (Torti *et al*. 2021), com isso a necessidade de vistoria periódica.

Segundo Reagan *et al*. (2017), em 2013 a ASCE (*American Society of Civil Engineers*, Sociedade Americana de Engenheiros Civis) classificou o estado das pontes americanas como medíocre, quase 145 mil pontes foram classificadas com funcionalidade obsoleta e quase 70 mil com estrutura deficiente.

Em termos de infraestrutura governamental, o custo para se ter uma manutenção periódica é altíssimo (Cidade de São Paulo, 2021). Obviamente o custo de não se ter uma manutenção periódica é ainda maior. Segundo estudos realizados nos Estados Unidos entre 1989 e 2000, mais de 500 falhas ocorreram somente em pontes (Reagan *et al*., 2017). Ainda segundo o mesmo autor, pontes de concreto tendem a experienciar vários tipos de degradação, seja material de má qualidade, cargas excessivas, corrosão, rachaduras verticais etc.

Em 2017 a ASCE publicou o *Infrastructure Report Card* que levantou a informação de que deveria ser investido 123 bilhões de Dólar Americanos (USD) para reabilitar cerca de 56 mil pontes que apresentavam deficiência (Spencer *et al*., 2019). O processo de vistoria tem sua complexidade e custos relativos ao tamanho da estrutura que irá ser analisada, podendo se tornar complexa, exigir muito tempo de mão-de-obra qualificada (somente inspetores treinados) e também muito arriscada (Spencer *et al*. 2019). Inspeções convencionais (visual) e avaliação usando práticas conservadoras não são suficientes para determinar a conservação de estruturas mais velhas (ISIS Canada, 2001).

2.1 SHM

Em 2001 o governo do Canadá publicou o Guidelines for SHM (*Structural Health Monitoring*) onde apresentava um guia de implementação de padrão sobre

coleta, observação e diagnóstico de estruturas de concreto reforçado (ISIS Canadá, 2001). Até então o governo do Canadá já vinha fazendo um estudo utilizando sistemas de HSM nos últimos 5 anos anteriores à publicação do guia. A ideia com a publicação do manual era mostrar as vantagens de se implementar sistemas SHM.

Segundo o guia apresentado, o objetivo do SHM é o constante monitoramento de estruturas de forma a se ter em mãos dados precisos sobre as estruturas (o manual focava em monitoramento de pontes), seus comportamentos mediante a vários tipos de eventos a fim de detectar danos ou deterioração, assim sendo possível determinar a saúde e/ou condição da mesma e apresenta quatro tipos de classificação de sistemas SHM (ISIS Canada, 2001): Testes de Campo Estáticos (*Static Field Testing*), Testes de Campo Dinâmicos (*Dynamic Field Testing*), Monitoramento Periódico (*Periodic Monitoring*) e Monitoramento Contínuo (*Continuous Monitoring*).

SHM é o desenvolvimento e implementação de métodos e técnicas que são utilizadas na supervisão e manutenção contínua de estruturas (Sofi *et al*., 2021). O manual da ISIS Canadá (2001) define que sistemas SHM devem ser capazes de providenciar os dados *on demand* e define que tal informação deve poder ser obtida através de uma rede local ou transmitida diretamente para uma rede remota. Além disso, o guia define que sistemas SHM são divididos nas seguintes etapas: aquisição de dados, comunicação dos dados, processamento dos dados, armazenamento, diagnósticos e retenção dos dados.

2.1.1 Aquisição de dados

O manual da ISIS Canadá (2001) define que a aquisição de dados deve provir de dispositivos que podem medir valores absolutos ou detectar alterações dos seguintes tipos: deformações, acelerações, temperaturas, emissões acústicas, tempo, potencial elétrico, carga, e outros atributos da estrutura monitorada. O guia também descreve que muitos dos sensores disponíveis comercialmente não são recomendáveis para sistemas SHM, dada a necessidade de sensores com uso contínuo e de longo prazo. Tais dispositivos comerciais não apresentam a qualidade desejada. Ozer e Feng (2020) apresentam SHM como sistemas altamente baseados na coleta de dados de sensores.

2.1.2 Comunicação de dados

Durante a fase de testes de campo estático a coleta se dá de forma manual, onde a pessoa fazendo a inspeção vai fornecer os dados. O guia enfatiza que a comunicação dos dados com a fonte de destino deve evitar ser via fios, uma conexão *wireless* é menos indicada, pois além de ser mais difícil tende a resultar em erros.

2.1.3 Processamento de dados

Todos os dados são sujeitos a ruído, ainda mais vindo de sensores expostos ao clima e o ambiente como um todo. Além disso, para um certo tipo de análise de dados pode ser necessário combinar dados de múltiplos sensores distintos e diferentes a fim de ter uma única informação. Essa etapa é responsável por limpar os dados e agrupá-los.

2.1.4 Armazenamento/retenção de dados

O guia defende que tais dados devem ser armazenados por muitos anos, a fim de servir como base histórica. Os dados devem também ser armazenados após serem “limpos”, e os dados crus podem ser descartados, ainda que se livrar dos dados crus tenha um possível ponto negativo: se no futuro pesquisadores quiserem cruzar os dados de outra maneira, isso não será possível. Tanto dados coletados de sensores quanto coletados em testes de campo devem ser armazenados. Os dados devem ser de fácil acesso para que possam ser usados para pesquisas e reinterpretações no futuro.

2.1.5 Diagnóstico

A parte mais importante de um sistema SHM é onde a interpretação dos dados coletados acontece. Nesta etapa os dados crus se convertem em informação quantitativa, que podem, entre outras coisas, indicar possíveis perigos para a estrutura. Por exemplo, como no caso deste trabalho, aumento significativo da

quantidade de rachaduras e/ou o aumento da dimensão de rachaduras já registradas em dados históricos.

O guia apresenta muito mais informações sobre sensores e testes de campo. Tais itens não são o foco deste trabalho.

2.2 ABRANGÊNCIA DE HSM

Governos, como o Chinês, já incorporaram SHM em suas diretrizes para construção de pontes com sistemas SHM já incorporados nas estruturas (Moreu *et al.* 2018). Outros países como Austrália, Estados Unidos, Inglaterra, Suíça e outros já possuem diretrizes e padrões para implementação de sistemas SHM em pontes (Moreu *et al.,* 2018). Muitos dos projetos, e mesmo guias governamentais, como os apresentados por Moreu *et al*. (2018), se baseiam fortemente no uso de sensores. O problema com o uso de sensores é o custo, tanto de investimento inicial como com manutenção e substituição.

Órgãos como a FHWA (*Federal Highway Administration*) dos Estados Unidos têm continuamente solicitado que novas formas de detecção de danos sejam desenvolvidas a fim de conseguir prever falhas em estágios iniciais (Reagan *et al*., 2017). Trabalhos com o foco no uso de captação de imagens através de drones (Reagan *et al*., 2017), captação de forma coletiva de dados de sensores de *smartphones* (Ozer e Feng, 2020) e demais trabalhos têm surgido a fim de não depender somente de sensores desenvolvidos somente com este fim.

2.3 CONSIDERAÇÕES DO CAPÍTULO

A engenharia civil já vem trabalhando com o uso de tecnologia a anos, e muito esforço tem sido empregado para aperfeiçoar o trabalho, seja de planejamento, construção ou monitoria aliada com tecnologias. Com o crescimento populacional e expansão territorial uma demanda maior de profissionais para inspecionar tais construções se faz necessário.

HSM tem como finalidade definir padrões de como melhor realizar esta tarefa. Sistemas convencionais de HSM tem se mostrado custosos e complexos, por isso a

necessidade de buscar alternativas que facilitem o trabalho, mas ao mesmo tempo, não se tornem mais caras.

O uso de visão computacional tem se mostrado bem promissor, ainda mais com o avanço alcançado por técnicas de deep learning, que devido ao aumento do poder computacional dos dias de hoje, acabou atraindo a atenção de diversos setores.

3 DEEP LEARNING

Quando o sistema AlphaGo, da então DeepMind Technologies, derrotou o campeão mundial de Go (jogo Chines que envolve uma complexa gama de decisões estratégicas) (NYTimes, 2017) chamou muita atenção para o assunto da inteligência artificial (Tecnoblog, 2017). Para os mais leigos no assunto, era a prova de que os computadores em breve iriam estar substituindo os humanos em seus ofícios. Para pessoas envolvidas na área da computação um nome começou a se destacar: *Deep Learning*. Foi graças a modelos de *deep learning* que foi possível o supercomputador AlphaGo derrotar um ser humano em um jogo que era considerado muito complexo para ter todos os possíveis movimentos mapeados por um sistema computacional (MIT Technology Review, 2017).

Deep Learning é uma *branch* de *Machine Learning*, já *Machine Learning* é uma *branch* dentro do conceito de inteligência artificial. Algoritmos de *Machine Learning* fazem uso de modelos matemáticos ajustados para prever um dado resultado (*output*) a partir de um conjunto de dados de entrada (*input*). Estes modelos podem ser treinados a partir de um conjunto de dados de testes, com os quais a saída do modelo é então comparada, se a saída do modelo não for a esperada, sabe-se que o modelo precisa ser ajustado, isso chama-se de aprendizado supervisionado. Mas também têm-se modelos não supervisionados, são modelos que não possuem um conjunto de testes e nos quais o objetivo é a clusterização, ou agrupamento de tais dados baseados em características similares entre si.

Deep Learning vem de um conceito não tão recente, a primeira rede neural foi descrita em 1943 por McCulloh e Pitts (Markus, 2021). Nessa época esse termo ainda não era usado, mas foram os primeiros a descrever o conceito de uma rede neural. Segundo Markus (2021) em seguida têm-se o conceito de Perceptron, sendo apresentado por Rosenblatt, em 1957. Em 1986 o trabalho de Rumelhart descreveria o conceito de *backpropagation*, este trabalho ajudou a impulsionar as pesquisas relacionadas a redes neurais, pois mostrava que através de *backpropagation* poderia-se superar as limitações do perceptron (Markus, 2021). Em 1989 surgiria LeNet, que aplicaria o conceito de *convolutional neural network* (CNN) que Lecun usaria em 1990 para desenvolver uma aplicação capaz de reconhecer CEPs escritos por mão humana (Markus, 2021). Apesar dos esforços em pesquisa, *Deep Learning* acabou saindo de

cena por ter sido ofuscada por outras técnicas de *Machine Learning*, que pareciam mais promissoras naquele tempo, técnicas como SVM (*Support Vector Machines*), visto que apresentaram melhores resultados e tinham uma base teórica bem fundamentada (GÉRON, 2017). Mais tarde, com o aumento do poder computacional, *Deep Learning* voltou a se destacar. Em 2012 a implementação de uma CNN chamada AxelNet atingiu o maior *score* em uma competição de visão computacional conhecida como ImageNet (Marcus, 2021). Seguindo o raciocínio apresentado por Géron (2018), esta nova onda parece ser diferente das demais, pois agora têm-se outros fatores que mudaram o cenário para técnicas de *Deep Learning*. Entre tais pontos Géron (2018) aponta:

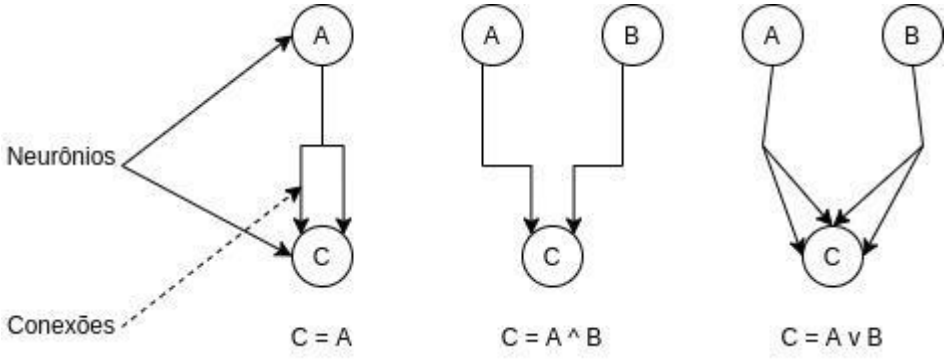
- O aumento dos dados disponíveis. Somente no ano de 2020 estima-se que cada pessoa tenha gerado em média 1.7mb de dados por segundo (Tech Jury, 2022). Tem-se como uma das necessidades fundamentais quando treinando um modelo de *deep learning*: uma quantidade de dados expressiva.
- Também citado por Chollet (2018), outro fator foi o aumento computacional expressivo, e muito disso se deve a indústria dos games, quando empresas como Nvidia e AMD começaram a desenvolver as GPUs (*graphic processing units*, unidades de processamento gráfico) para fornecer maior poder computacional na renderização de jogos e ferramentas 3D em tempo real. A comunidade científica viu uma oportunidade de utilizar tais GPUs quando a NVIDIA apresentou uma interface de programação para tais chamada CUDA, com um poder de processamento até 15x maior que uma CPU (Towards Data Science, 2020). As GPUS começaram a ser adotadas fortemente para projetos de *Deep Learning*.
- Os modelos de treinamento foram aperfeiçoados.

Limitações teóricas, como a que defendia que um Rede Neural ficaria limitada ao *local optima* ao calcular o erro do modelo se provaram errôneos, até então são raros os casos em que os modelos ficaram presos no local mínima e mesmos estes, ainda estavam muito próximos do global mínima.

3.1 REDES NEURAIS

As redes neurais estão no coração de muitas das aplicações que existem hoje quando os assuntos são processamento de imagens, reconhecimento de fala, sistema de recomendações de vídeos do YouTube (Géron, 2018). As redes neurais tiveram seu conceito inicial se espelhando em como um conjunto de neurônios biológicos se comporta (este trabalho não irá detalhar uma estrutura básica de um neurônio biológico e nem procura mostrar uma relação entre tais conceitos). Quando McCulloh e Pitts (1943) propuseram a primeira rede neural, eles apresentaram o que seria uma representação de um modelo de neurônio biológico, conhecido como neurônio artificial (Géron, 2018), onde este neurônio é ativado mediante a ativação de um de seus *inputs*, como no exemplo apresentado na Figura 1.

Figura 1 - Exemplos de uma rede neural



Fonte: do autor

O exemplo fornecido por Géron (2018), exibido na Figura 1, a esquerda, a ativação do neurônio A ativa o neurônio B. Ao centro, a ativação do neurônio C somente acontece se os neurônios A e B estiverem ativos. Na direita, a ativação do neurônio C depende somente da ativação de um dos dois casos. Assim é mostrado ser possível a criação de uma rede de neurônios artificiais para cálculos de proposição lógica.

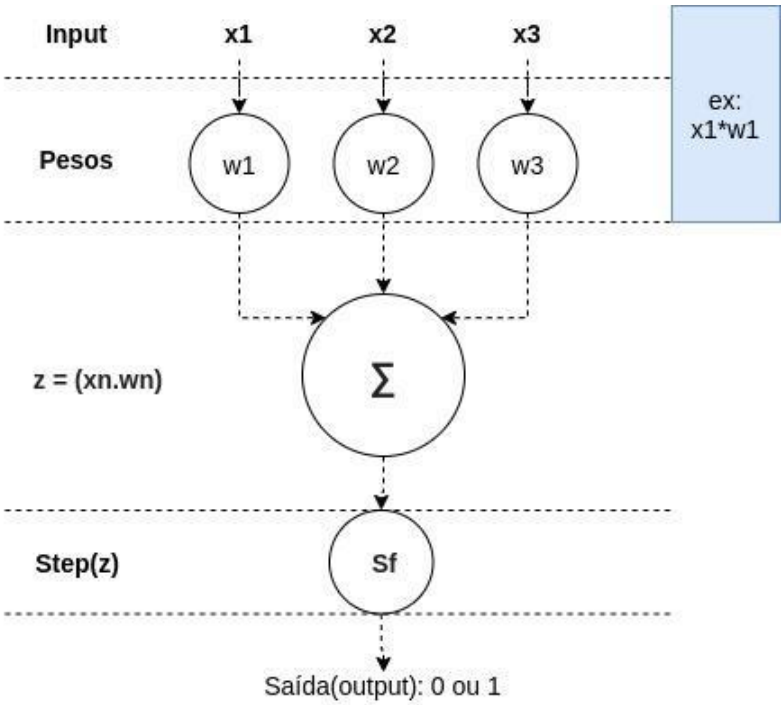
3.2 PERCEPTRON

O perceptron é um modelo de rede neural mais simples, seu modelo de neurônio é chamado de LTU (*Linear Threshold Unit*). Diferentemente do exemplo da Figura 1, seus *inputs* não são binários (0 ou 1) e sim numéricos (Géron, 2018). O

Perceptron é composto por uma unidade computacional, um número de *inputs*, o viés, seus pesos e uma única saída. Os *inputs* são geralmente nomeados de *x1, x2, x3*…etc. Cada *input* tem associado a ele um peso (*w0, w1, w2*). Antes dos *inputs* serem calculados na unidade computacional, eles são multiplicados pelos seus pesos. A função dos pesos é servir como os potenciómetros citados acima, quando falou-se sobre aprendizado supervisionado. São os pesos que servirão para ajustar o modelo para representar corretamente os dados fornecidos.

Após feito o somatório dos *inputs* tem-se o valor de saída. Com isso é chamada a função de ativação *y=f(z)*, onde *z* é o somatório dos *inputs* (já multiplicados pelos seus pesos). A função de ativação também é conhecida como *step function,* onde se o valor de entrada for menor que zero ele vai resultar em 0, senão, o resultado será 1. O Perceptron tem somente duas saídas, 1 ou 0 (Markus, 2021), como pode ser visto na Figura 2.

Figura 2 - Estrutura de um Perceptron

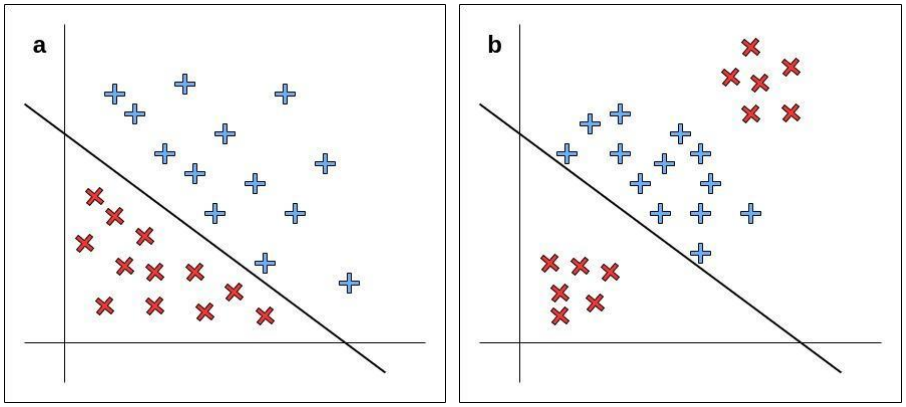


Fonte: do autor

O viés é adicionado através de um neurônio especial chamado de neurônio de viés, onde ele sempre terá como saída 1. Um perceptron é composto de uma única camada de LTUs, e um único LTU pode ser utilizado para uma classificação linear

(Géron, 2018). O problema com classificações lineares é que as mesmas não podem ser utilizadas (propriamente) para realizar uma classificação não-linear, ou seja, serem aplicadas em modelos mais complexos (Markus, 2021).

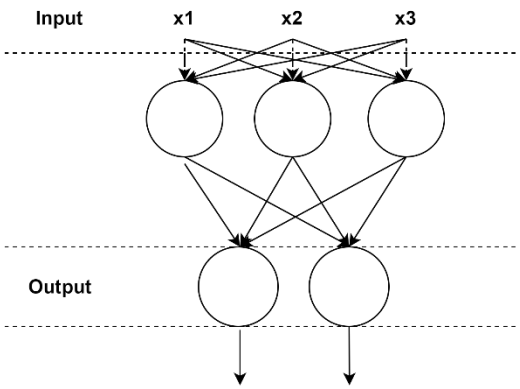
Figura 3 - Separação linear do conjunto de dados



Fonte: do autor, baseado no material apresentado por Udacity

Como exemplo, a Figura 3, caso a, é um problema que pode ser resolvido por um modelo linear. Já o segundo caso, Figura 3 - b, é um problema onde os dados não podem ser separados linearmente por uma única linha. Esta limitação do Perceptron foi um grande desmotivador para muitos pesquisadores, pois impossibilitava que o Perceptron pudesse aprender padrões mais complexos (Géron, 2018). Para solucionar problemas mais complexos, como o citado no caso da Figura 3-b, idealizou-se então a MLP (*Multilayer Perceptron*). Uma MLP é composta por uma camada de *input*, uma ou várias camadas de LTUs e uma camada final de saída (Géron, 2018). Estas camadas de LTUs são conhecidas como *hidden layers*. Outro ponto interessante é que em redes onde cada neurônio em uma camada "escondida" recebe *input* da camada anterior é conhecido como uma rede totalmente conectada (Markus, 2011), como apresentado na Figura 4.

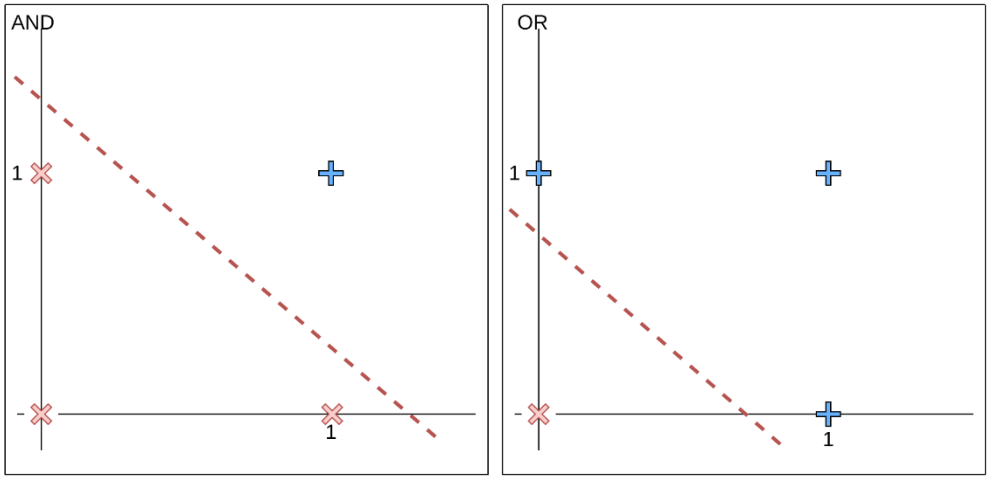
Figura 4 - Multilayer Perceptron



Fonte: do autor, baseado no material apresentado por Markus (2021)

As limitações do perceptron ficam bem claras quando se observa o problema apresentado por Markus (2021). Imagine o caso de um modelo com uma função de ativação que aplica a função XOR. Para modelos que aplicam regras AND ou OR ambos são linearmente separados, como na Figura 5.

Figura 5 - Separação linear de dados por AND e OR

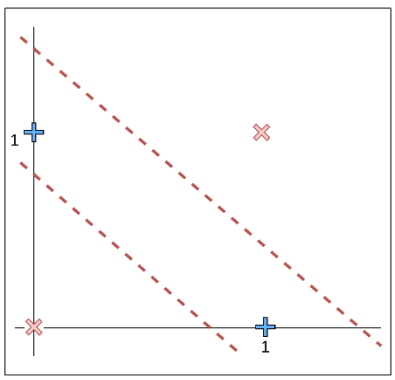


Fonte: - Separação linear de dados por AND e OR

Para resolver o problema XOR, ilustrado na Figura 5, seriam necessárias duas linhas separando os dados. Markus (2021) simplifica o caso quando aponta que para tal situação seriam necessárias mais camadas na rede apresentada. E aqui tem-se o conceito de *Deep Neural Networks* (Redes Neurais Profundas, DNN). Chama-se DNN uma rede neural que possui duas ou mais camadas escondidas (*hidden layers*) de neurônios (Géron, 2018).

Figura 6 - Representação de um problema XOR, linearmente inseparável por uma única linha

XOR	True	False
True	0	1
False	1	0



Fonte: do autor, baseado no material apresentado pela Udacity

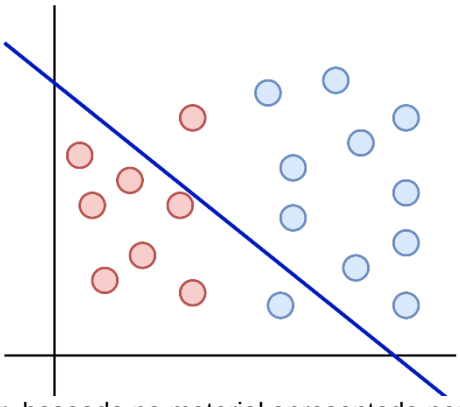
Na prática, até então ninguém tinha conseguido aplicar o conceito de MLP, pelo simples fato de que não era possível treinar o modelo. Este modelo de MLP se define como uma *fully connected feedforward network*. *Fully connected* é uma rede onde um neurônio recebe como entrada os valores de todos os neurônios da camada anterior. *Feedforward network* significa que a rede não possui ciclos (Markus, 2021).

Para entender as limitações do que foi apresentado até então é preciso dar um passo atrás e entender o processo de aprendizagem do perceptron, e aprendizado de máquina em um conceito mais geral.

3.3 APRENDIZADO

O processo de aprendizagem de um perceptron consiste em comparar a saída de um dado modelo para um conjunto de dados de treinamento. Este conjunto de dados deve ser anotado (*labeled*) com o valor correto de classificação. Após o modelo ser executado, partindo com valores de pesos aleatórios – processo conhecido como *random initialization* – então o valor de saída é comparado com o valor esperado. Se um dado foi classificado erroneamente, é preciso ajustar o modelo para que classifique corretamente o dado de entrada (Chollet, 2018). No caso da Figura 7, onde dois elementos estão classificados erroneamente, o modelo precisa ser ajustado corretamente para fazer a classificação dos dois itens.

Figura 7 - Problemas de classificação



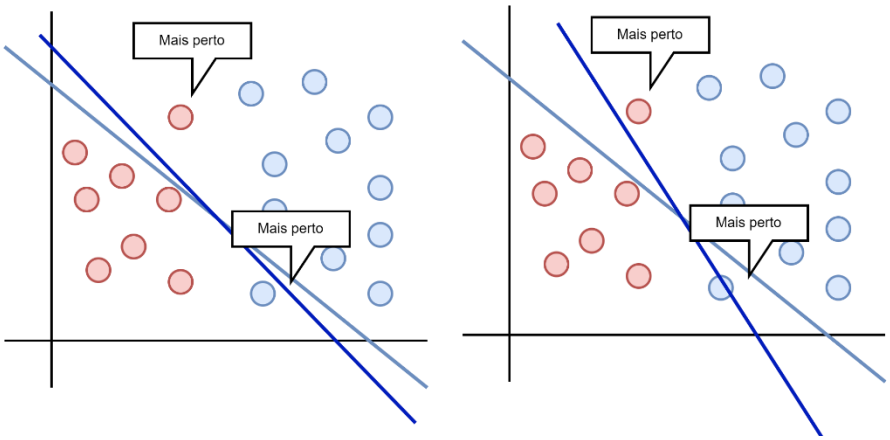
Fonte: do autor, baseado no material apresentado por Udacity

Para ajustar o modelo precisa-se ajustar os pesos. Ao ajustar o peso, o modelo será alterado, mas para tal o mesmo deve ser ajustado de forma a não ter uma

alteração drástica na sua classificação. Com isso é importante a utilização do *learning rate* (taxa de aprendizagem). O *learning rate* é quem dirá quanto o modelo deve ser ajustado para que os dois itens sejam classificados corretamente.

Para que o modelo saiba o quão distante está do esperado é preciso calcular a distância do modelo e o ponto erroneamente classificado, ou seja, é preciso calcular o erro do modelo (*Udacity, Deep Learning*). A Figura 8 apresenta um exemplo.

Figura 8 - Exemplo interpretativo de como se comportaria o aprendizado de um modelo



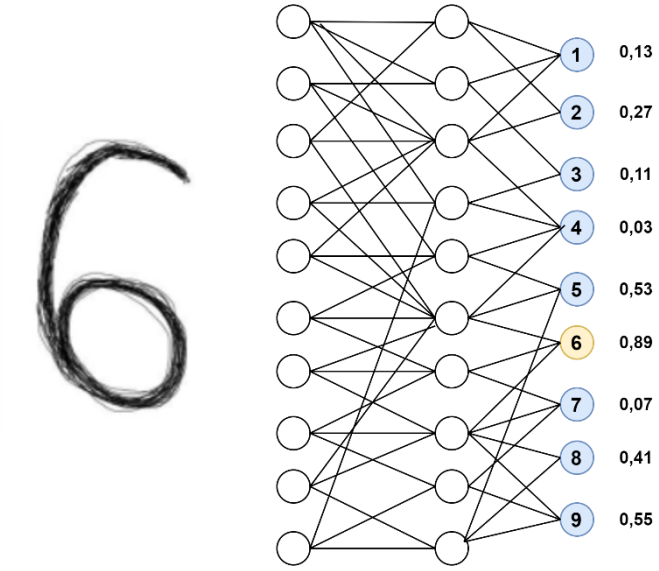
Fonte: do autor, baseado no material apresentado por Udacity

A função de perda é quem diz o quão distante o modelo está da solução ideal. No modelo atual é utilizada a quantidade de pontos classificados erroneamente, no caso a direita da Figura 8. A partir do *learning rate* o modelo vai sendo ajustado para que possa então diminuir a quantidade de erros. O problema com o modelo atual é a utilização de uma função discreta (*step function*), a função retorna somente 1 ou 0 para classificar o valor de entrada (Chollet, 2018). Para ajustar o modelo de forma que classifique os resultados de maneira que o *learning rate* e a função de perda possam performar melhor e mais precisamente, se faz necessário substituir a função de ativação por uma função contínua (*Udacity, Deep Learning*).

Com a função contínua, a função de ativação resultará na probabilidade de um dado *input* pertencer a uma dada classificação. Com isso a função de perda é calculada, medindo o quão errado um modelo está na sua classificação. Note-se que geralmente em resultados de modelo de *machine learning* estes são expressos alegando que um dado modelo *x* obteve 87% de acurácia na sua classificação, ou

seja, o modelo *x* teve uma taxa de erro de 13%. O objetivo do modelo é então achar o menor valor de erro possível. No exemplo fornecido por Grant Sanderson (Figura 9) tem-se uma rede neural para classificar e reconhecer dígitos escritos à mão humana.

Figura 9 - Representação visual de uma RN para classificação de dígitos



Fonte: do autor, baseado no material de Grant Sanderson(2017)

O modelo classificou que tem 89% de certeza que o caractere é o valor 6, como também 55% de certeza que o caractere é o valor 9. Note que o modelo não conseguiu classificar perfeitamente o caractere, por mais que o valor correto tenha obtido uma percentagem maior, ainda assim a taxa de certeza de que o valor escolhido é o correto não foi tão alta.

3.4 FUNÇÃO DE PERDA

Entre funções de perda de *deep learning* tem-se a MSE (*Mean Squared Error*, Média Quadrática de Erros). A equação a seguir apresenta este cálculo.

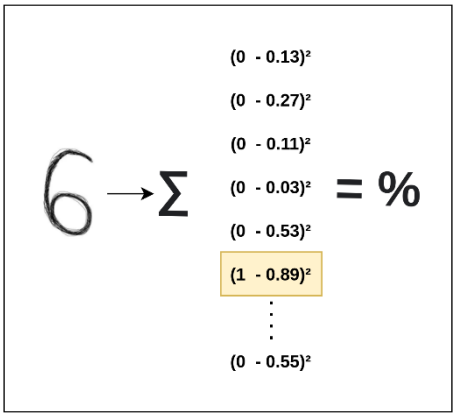
Figura 10 - Fórmula MSE

$$\mathrm{MSE} = \frac{\sum_{i=0}^{n}(y-y')^2}{n}$$

Fonte: Géron (2018)

A MSE é o somatório dos valores onde *y* é o valor de saída do modelo e *y'* é o valor esperado. A Figura 11 apresenta o cálculo usando a fórmula.

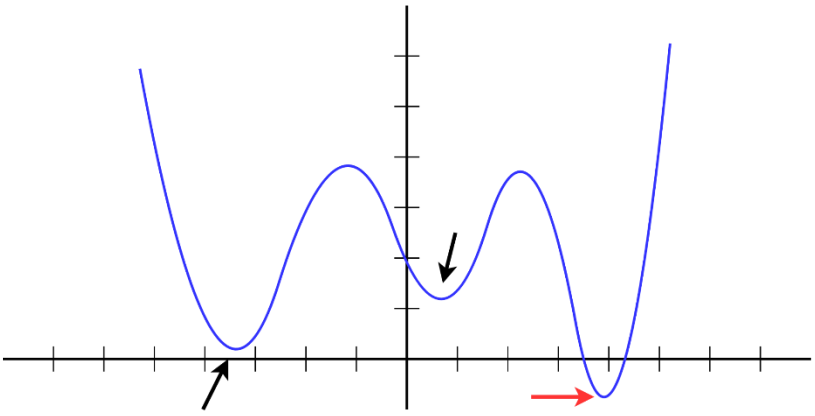
Figura 11 - Cálculo da MSE considerando todos os possíveis valores



Fonte: do autor, baseado no material de Grant Sanderson (2017)

Após a função retornar o valor de perda do atual modelo, o modelo deve ser ajustado de forma a achar o cenário que melhor reflete os dados de saída esperados, onde a perda (erro) é menor, e isso é alcançado ajustando-se pesos e viés. Para que o modelo saiba quando a menor taxa de erro possível foi encontrada é preciso encontrar o mínimo da função. Imagine um único *input*, para este *input* o modelo precisa ajustar seus pesos e viés de forma achar o mínimo da função, como na Figura 12.

Figura 12 - Ponto representando o mínimo de uma função

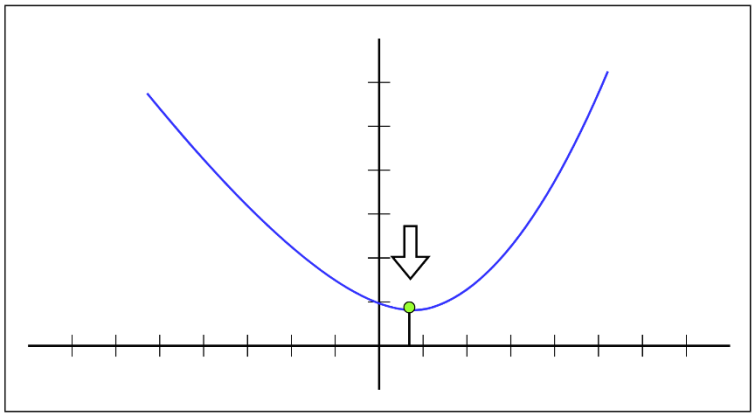


Fonte: do autor, baseado no material de Grant Sanderson(2017)

Para esta função só existe um valor mínimo. A Figura 13 apresenta um cenário mais complexo. A função apresenta três mínimos, dois indicados pelas setas pretas,

sinalizando o que é conhecido como *local minimum* (mínimo local) e um indicado pela seta vermelha que apresenta o *global mínima* (mínimo global).

Figura 13 - Função com vários mínimos



Fonte: do autor, baseado no material de Grant Sanderson(2017)

Para otimizar-se o modelo de forma a achar o menor valor possível para a função de perda utiliza-se um algoritmo de otimização genérico capaz de achar a melhor solução para uma grande gama de problemas chamado *Gradient Descent* (Géron, 2018).

3.5 GRADIENT DESCENT (GRADIENTE DESCENDENTE)

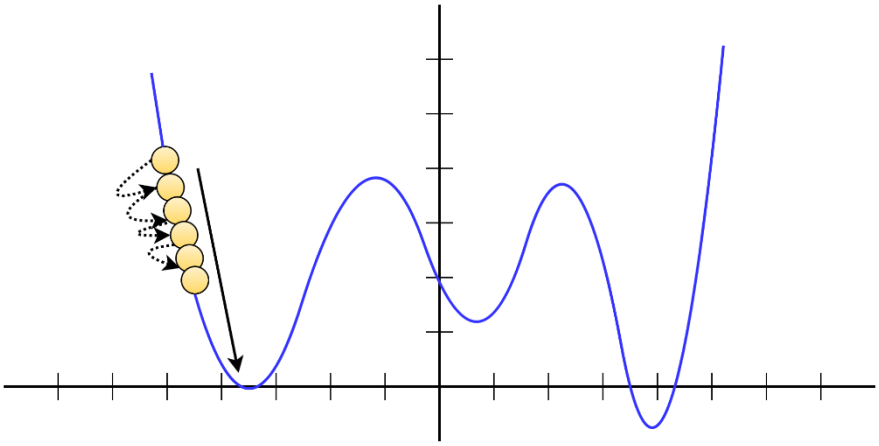
Ao aplicar a derivada de uma função sabe-se como alterar o valor de *x* – os pesos do modelo nesse caso – da função em ordem a alterar o valor de *y* (saída). Isso é conhecido como o gradiente da função. Para reduzir o valor da função aplica-se o sinal oposto da derivada (Goodfellow, 2016). Para casos de *deep learning* que os modelos possuem inúmeras entradas, se faz necessário aplicar derivadas parciais, visto que uma derivada parcial reflete a mudança de *Y* em relação a uma entrada *Xn*. O algoritmo de Gradiente faz isso, ele generaliza a noção de derivada trabalhando com a ideia de que o gradiente de uma função engloba todos os *inputs* da mesma (Goodfellow, 2016).

No exemplo da Figura 12 se o modelo encontrar um dos dois mínimos locais da função não irá refletir o melhor cenário, pois a taxa de erro não será a melhor possível, não estará otimizado (Goodfellow, 2016). A saída do gradiente é um vetor de cada derivada parcial apontando para os máximos da função, aqui entra o termo

Gradient Descent que faz o caminho inverso, matematicamente falando, ele inverte o sinal da saída. Um exemplo simples de imaginar para explicar o que é o gradiente seria imaginar-se em uma montanha cercado por uma neblina densa, para descer tal montanha o mais rápido possível a melhor estratégia é achar o maior decline possível e seguir em direção a ele, essa é a ideia por trás do *Gradient Descent*. Ele mede o gradiente local em relação a função de erro e segue na direção descendente (negativa) do gradiente da mesma até chegar ao valor onde o gradiente é zero (Géron, 2018).

Dois outros fatores se fazem importantes, entre eles o *learning rate* (taxa de aprendizado). Se o *learning rate* for definido muito baixo, o modelo vai demorar muito para convergir para o mínimo, tomando muitas iterações (Géron, 2018). Essa realidade pode ser observada na Figura 14.

Figura 14 - Gradient Descent sendo aplicado com learning rate baixo

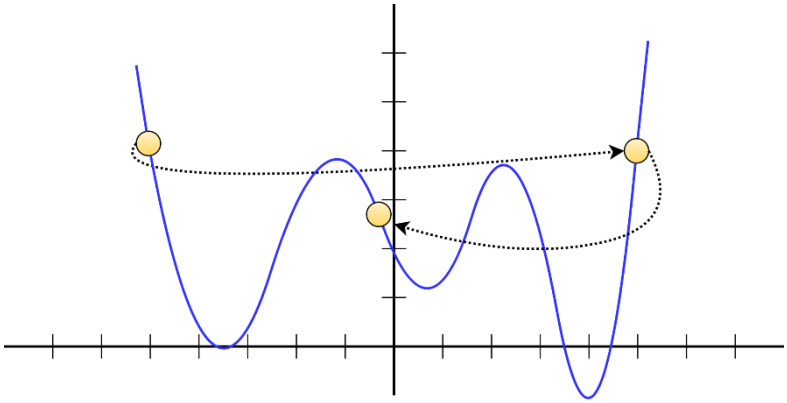


Fonte: do autor, baseado no material de Géron (2018)

Se o *learning rate* for muito alto, o modelo pode acabar extrapolando e pode acabar caindo em um mínimo local em outra parte da função (Géron, 2018), como demonstrado na Figura 15.

Outro ponto é que como os valores iniciais dos pesos são gerados de forma aleatória, isso pode proporcionar que o modelo comece em um certo valor da função que logo venha a convergir para um mínimo local. Para modelos de Regressão Linear a função de perda MSE atua como uma função convexa, não havendo mínimos locais, somente um único mínimo global (Géron, 2018).

Figura 15 - Gradient Descent sendo aplicado com learning rate muito alto



Fonte: do autor, baseado no material de Géron (2018)

3.6 BACKPROPAGATION

Como citado no tópico de MLP (Multilayer *Perceptron*), pesquisadores não conseguiram treinar uma MLP até que, em 1986, Rumelhart publicou um algoritmo introduzindo o conceito de *backpropagation* (Géron, 2018). Cada vez que o modelo roda sobre o conjunto de treinamento, cada camada de neurônios tem sua saída computada e encaminhada para a próxima, até que a rede tem a saída, a qual é comparada com o valor esperado. O algoritmo então calcula quanto cada neurônio da camada anterior contribuiu na taxa de erro e em cada camada anterior até chegar na camada de entrada, fazendo a propagação da medida de erro em todas as camadas da rede (Géron, 2018). Géron (2018) também enfatiza que os autores do algoritmo tiveram que trocar a função de ativação (*Step Function*) pela *Logistic Function* (Função Logística), como demonstrado na fórmula a seguir.

Figura 16 - Função Logística

$$\text{Logistic sigmoid, } \frac{1}{1+\exp(-x)}$$

Fonte: Géron(2018)

Como citado anteriormente, a *Step Function* somente resultava em 1 ou 0, o que impossibilita o gradiente de funcionar. Géron (2018) aponta que não necessariamente precisa-se utilizar da *Logistic Function* como função de ativação. Segundo o autor, existem duas outras funções populares como função de ativação:

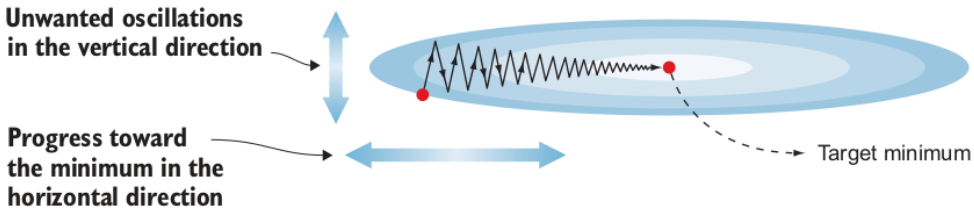
- *The Hyperbolic Tangent Function (tanh)*: tem forma de S, contínua e diferenciável. Os valores de saída variam de -1 a 1 (*Logistic Function* é 0 e 1). Dado os valores de saída, faz com que cada camada tenha os valores mais ou menos normalizados.
- ReLU: é uma função contínua e não é diferenciável. É rápida de computar. Também não possui um valor máximo, que segundo Géron (2018), ajuda a reduzir alguns problemas ao aplicar o *Gradient Descent*.

Como recomendado por Markus (2021), escolher a função de ativação é algo que deve ser feito mediante ao problema a ser solucionado. Segundo o mesmo autor, uma boa estratégia é utilizar funções *tanh* para as camadas escondidas, pois o limite vai ser centrado em torno de zero. E utilizar a *logistic function* na camada de saída, assim os dados de saída podem ser interpretados como probabilidade, como a de um dado item ser classificado como X ou não.

3.7 OTIMIZADOR

Um dos problemas comuns do *gradient descent* é quando o algoritmo tenta chegar ao mínimo da função e acaba ou oscilando de mais entre mínimos e máximos (alto *learning rate*) ou acaba levando tempo demais para convergir ao mínimo da função. Segundo Elgendy (2021), quando o *gradient descent* começa a oscilar demais, no sentido vertical, ele acaba demorando muito para chegar ao mínimo da função. O autor exemplifica de forma visual os problemas de se ter uma oscilação vertical muito alta na Figura 17.

Figura 17 - Problemas para convergir ao mínimo da função



Fonte: Elgendy (2021)

Ainda segundo Elgendy (2021), a fim de reduzir tais oscilações criou-se uma técnica chamada de *momentum*, que faz com que o aprendizado seja menor em termos de oscilações verticais e torne mais rápido o progresso em sentido horizontal. Além do *momentum*, desenvolveu-se outro otimizador chamado Adam (*adaptive moment estimation*), que é uma combinação de *momentum* com *adaptative learning*.

3.8 DROPOUT

Dropout é uma técnica de regularização de dados que tem como finalidade ajudar a reduzir o *overfit* dos dados. Entre as mais populares estão L2 e *Dropout*. Quando aplicado *Dropout*, a rede simplesmente tem uma probabilidade, informada pelo parâmetro conhecido como *dropout rate*, onde cada neurônio pode ter o seu valor completamente ignorado e assim não ser encaminhado para a rede seguinte (Elgendy, 2021, p. 177).

3.9 POOLING LAYERS

Segundo Elgendy (2021) redes neurais tendem a se tornarem muito complexas sendo necessário reduzir a quantidade de parâmetros, pois a medida que a complexidade da rede aumenta, maior é a quantidade de filtros e com isso de parâmetros. Elgendy (2021) sugere pensar em camadas de *Pooling* como programas de compressão, pois eles reduzem a resolução da imagem enquanto consegue mandar as características importantes dessa. A Figura 18 apresenta um exemplo de *downsample*.

Figura 18 – Exemplo de downsample

Fonte: Elgendy(2021)

3.10 VARIAÇÕES DE REDES NEURAIS

Já foi citado neste trabalho o que é uma *fully connected feedforward network*. Restam serem descritos outros dois tipos de redes neurais. Os dois exemplos são descritos por Markus (2021):

> ***Convolutional Neural Network* (CNN):** um dos principais elementos de uma CNN é que os pesos de uma camada são compartilhados entre os neurônios daquela camada, isso é conhecido como *weight sharing* (compartilhamento de peso). Diferente dos exemplos dados até agora, uma CNN não tem uma rede onde todos os neurônios (de uma mesma camada) se conectam com todos os neurônios da seguinte. Este tipo de rede tem excelentes resultados na classificação de imagens.
>
> ***Recurrent Neural Network* (RNN):** Os modelos de redes anteriores eram treinados somente com os *inputs* de entrada atual, e não os outros *inputs* que foram utilizados no treinamento. Esse tipo de rede possui um conceito de memória (*inputs* anteriores) para produzir uma nova saída.

3.11 CONSIDERAÇÕES DO CAPÍTULO

Deep learning tem apresentado resultados excelentes em distintas áreas, como apresentado neste trabalho. Mas a complexidade no desenvolvimento de redes neurais e escolher a melhor abordagem demandam tempo e experimentação. Tempo, pois, para a utilização de redes neurais são necessários conjuntos de dados com um volume considerável (ainda que seja difícil definir o que seria considerável) e experimentação pois a definição de parâmetros e melhores abordagens necessitam serem testadas no conjunto de dados obtidos e validadas em relação ao objetivo a serem atingidos.

Por estes motivos iniciar um projeto embasado por estudos prévios se torna necessário. Utilizar material já validado por outros estudos ajuda a evitar os mesmos erros já encontrados por outros autores e possibilita partir de um ponto já validado, entre eles conjuntos de dados válidos e modelos já treinados, por exemplo.

4 TRABALHOS CORRELATO

Foram selecionados quatro trabalhos que possuem como foco a detecção de danos estruturais. Para se chegar aos trabalhos selecionados, consultou-se a base da CAPES, buscando por trabalhos que datam no máximo 5 anos de sua publicação, trabalhos que possuem como foco detecção de rachaduras ou danos estruturais utilizando *deep learning*, ordenou-se a busca para listar os 50 trabalhos mais significantes. Dos resultados obtidos foram selecionados os trabalhos que fazem uso de processamento de imagens em seu estudo. Os seguintes trabalhos foram escolhidos.

4.1 CLASSIFICATION AND QUANTIFICATION OF CRACKS IN CONCRETE STRUCTURES USING DEEP LEARNING IMAGE-BASED TECHNIQUES (FLAH ET. AL., 2020)

O estudo apresenta um trabalho onde modificaram uma técnica de processamento de imagem conhecida como *Otsu* e combinaram com *Deep Learning* com o objetivo de classificar, localizar e quantificar rachaduras em estruturas com base em cimento.

4.1.1 Dados utilizados no trabalho de Flah *et al*. (2020)

Para o estudo, os autores utilizaram um conjunto de imagens públicas disponíveis na web. O conjunto é composto por 40 mil imagens, anotadas com a finalidade de treinamento de algoritmos de *machine learning*. São imagens de estruturas de concreto com e sem rachaduras, 20 mil imagens para cada caso. Cada imagem tem um tamanho de 227*227 pixels com os canais RGBs presentes. O estudo tem como foco trabalhar em imagens que foram tiradas em proximidade da área, entre 25 e 50cm de distância. Isso torna o conjunto de dados ideal, pois as imagens são partes de imagens maiores. Foram 458 imagens de 4032*3024 pixels que geraram o conjunto de dados. Isso é bom pois o modelo não fica com rachaduras centralizadas na imagem. Em um cenário onde a imagem obtida possui rachaduras que acabam saindo da área da imagem capturada, o modelo precisa saber trabalhar com esse tipo

de situação. Imagens de cantos não foram selecionadas por este motivo. As imagens também perdem os canais RGB a fim de reduzir o tempo de processamento, visto que as cores, segundo os autores, não são necessárias no processo de detecção.

4.1.1.1 Anotação dos dados no trabalho de Flah et al.(2020)

Os autores utilizaram-se de uma rede CNN (*convolutional neural network*) para realizar a classificação. A rede foi treinada utilizando uma amostra de 10 mil imagens, 50% com rachaduras e 50% não. Desse total, dividiram o conjunto de dados novamente em uma fração de 60% para treinamento, 20% para validação e 20% para testes.

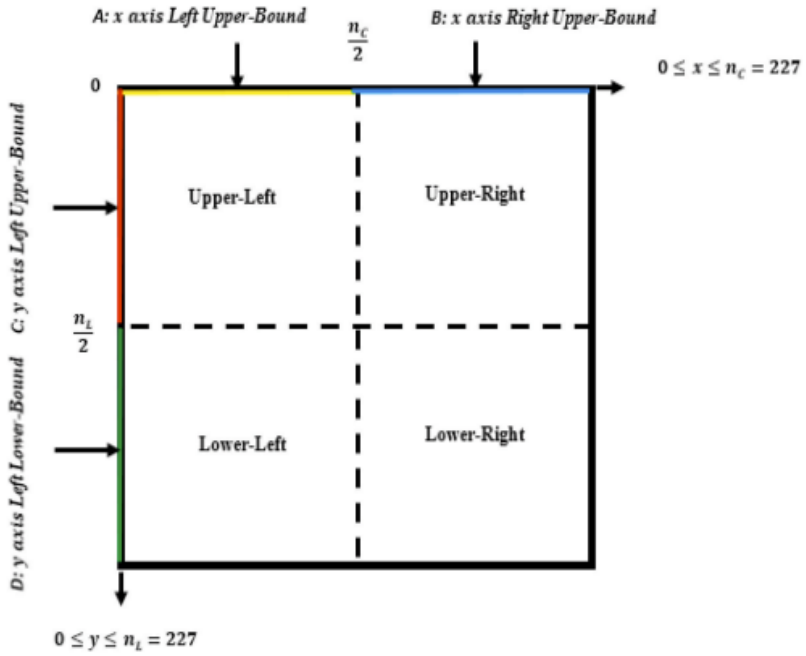
4.1.2 Arquitetura utilizada no trabalho de Flah *et al*. (2020)

Os autores projetaram uma rede neural convolucional (CNN) com um grupo de camadas além das camadas de *input* (entrada dos dados) e *output* (saída). Além disso, utilizaram duas camadas auxiliares. De forma simplificada descrevem que: a camada de entrada (*Input layer*) recebe uma imagem em tons de cinza de 227*227 (uma matriz de 227 linhas e 227 colunas). Sendo reduzida em um vetor de 1x1x64. A camada seguinte (*ReLU layer*) recebe o vetor de 64 elementos, processa os dados e envia para a seguinte (*Softmax layer*) que faz a predição dos dados utilizando os classificadores citados a seguir.

O processo de classificação utilizou três conjuntos de algoritmos. O primeiro tem o foco em classificar se a imagem apresenta rachaduras (*c, cracks*) ou não (*safe, s*). O segundo faz a classificação das imagens em quatro categorias: *vertical-left(VL), vertical-rigth(VR), horizontal-left(HL) e horizontal-rigth(HR)*. O terceiro classificador engloba todos os anteriores em um único classificador. Selecionaram um conjunto de 1200 imagens que satisfaçam um dos grupos para cada conjunto. Para selecionar cada imagem de cada um dos grupos do segundo classificador, os autores aplicaram a regra expressa na Figura 19.

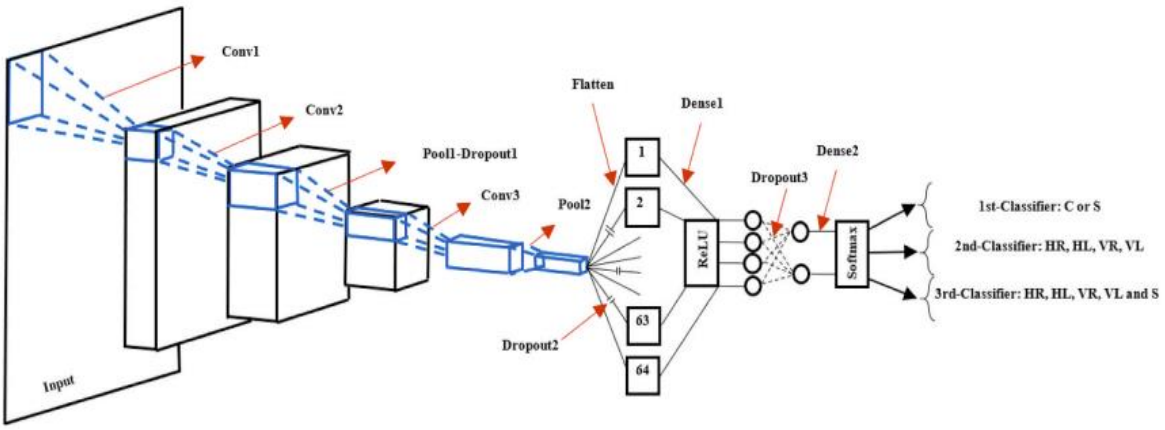
Dependendo da saída do primeiro classificador (se apresenta rachadura ou não), a imagem é então enviada para um *script* em MATLAB para que seja calculada a posição da rachadura utilizando a técnica de segmentação de *thresholding* (denominada Otsu), determinando suas propriedades geométricas.

Figura 19 - Regra para classificação de imagens



Fonte: Flah *et al*. (2020)

Figura 20 - Arquitetura do projeto de Flah et al.(2020)



Fonte: Flah *et al*. (2020)

Alguns pontos levantados pelos autores que são interessantes de destacar em relação a classificação:

- O *learning rate* escolhido foi de 0,0001.
- Aplicaram a técnica de *shuffling* (uma técnica que tem como objetivo diminuir o *overfitting* dos dados).

- Utilizaram o *Nesterov Accelerated Gradient*, uma deformação do *Gradient Descent*.
- O tempo de treinamento da rede apresentado pelos autores (e com o conjunto de dados descrito) foi de 2 a 3 horas utilizando uma CPU e cerca de minutos quando utilizando uma GPU.

4.1.3 Conclusão do trabalho de Flah *et al*. (2020)

O modelo apresentado pelos autores tem uma acurácia de mais de 97%. Ainda segundo os autores, o algoritmo apresenta uma performance melhor que muitos dos trabalhos considerados estado da arte até o momento da sua publicação. Todo o processo de classificar uma imagem corretamente (se possui rachaduras, identificar a posição e quantificar (tamanho, espessura) pode ser realizado em menos de 1 minuto. Segundo os autores, a taxa de erro para o primeiro classificador (binário) foi de 1,5%, para o classificador usando ReLU foi de 5% e para o terceiro foi de 2%.

4.2 DEEP CASCADED NEURAL NETWORKS FOR AUTOMATIC DETECTION OF STRUCTURAL DAMAGE AND CRACKS FROM IMAGENS (BAI ET AL., 2020)

O trabalho apresentado se baseia nos trabalhos de Yeum *et al*. (2018) e Gao e Mosalam (2018). Diferencia-se em além de identificar danos e rachaduras, definir a localização destes dentro das imagens. O trabalho faz uso de uma rede *ResNet* para classificar rachaduras e utiliza *U-Net* com a finalidade de conseguir definir a localização dela. Uma diferença importante entre este trabalho e o de Flah et al. (2020) é que neste trabalho os autores se propõem em aplicar a análise a nível de pixel, objeto e estrutura. O trabalho anterior tinha sua aplicação focada a nível de pixel.

4.2.1 Dados utilizados no trabalho de Bai *et al*. (2020)

O estudo faz uso das imagens disponíveis no *PEER Hub ImageNet*. É um projeto *open source* da universidade de Berkeley (Estados Unidos, CA) que disponibiliza vários conjuntos de imagens. Também fizeram uso de um outro conjunto de dados obtidos de um projeto *open source* da Middle East Technical University, com imagens de vários prédios. O conjunto de dados de imagens com rachaduras em

concreto é composto por um total de 40 mil imagens, 50% com rachaduras e 50% não. Este conjunto de imagens é de 224*224 píxeles.

Os autores também criaram um conjunto de dados próprios seguindo o mesmo padrão que encontraram no projeto COCO (*Common Objects in Context*). Utilizaram a ferramenta *COCO Annotator* para anotar o conjunto de dados com várias escalas. Imagens criadas para o treinamento de dados tem o tamanho de 256*256 píxeles.

4.2.2 Arquitetura utilizada no trabalho de Bai *et al*. (2020)

Na etapa de classificação os autores escolheram utilizar uma *Phi-Net*. Segundo eles, diferentemente das demais CNNs que sofrem problemas de performance conforme a profundidade da rede aumenta, uma *Phi-Net* consegue superar esse problema e ainda apresenta uma alta acurácia em problemas envolvendo reconhecimento de imagens. No projeto apresentado os autores usaram uma *Phi-Net* de 152 camadas. O *learning rate* foi definido em 0.001. Para a classificação de imagens com rachaduras quatro são as saídas possíveis: *Non Damage* (Sem rachaduras); *Flexural Damage*; *Shear Damage* e *Combined Damage*. Na Figura 21 é apresentada uma tabela fornecida pelos autores com o resultado do modelo de classificação.

Figura 21 - Tabela comparativa dos resultados obtidos

Cracking Type	Number of images		
	Noncracking	Cracking	Total
Non damage	19,184	2,699	21,883
Flexural damage	228	4,490	4,718
Shear damage	84	7,702	7,786
Combined damage	504	5,109	5,613
Accuracy	95.92%	86.505%	91.21%

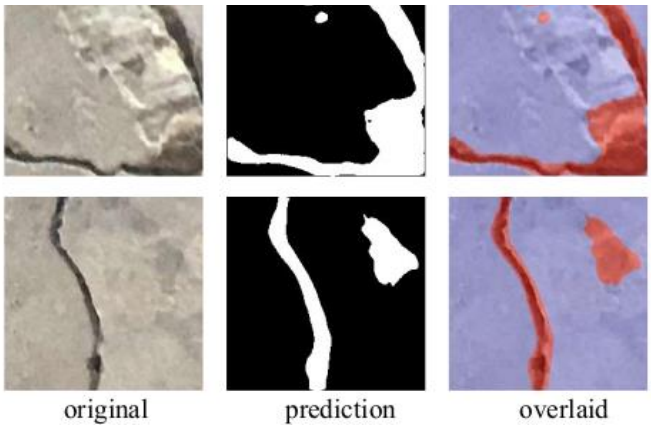
Fonte: Bai *et al*. (2020)

Para a localização das rachaduras nas imagens os autores utilizaram uma *U-Net*, que segundo eles, é uma rede que vem sendo aplicada com sucesso em segmentação de imagens em biomedicina. Os autores não detalham muito a estrutura da rede. Ela é composta por várias camadas convolucionais em uma ponta da rede e na outra, várias camadas de *up-sampling layers* (camadas que não possuem pesos atribuídos a elas). Não defendem a escolha do modelo, sinalizando apenas trabalhos consultados. Para o treinamento da *U-Net* dos autores, utilizaram um conjunto de

1000 imagens a nível de pixel, 853 a nível de objeto e estrutural. Cada conjunto de dados é treinado separadamente. O *Learning rate* da rede foi definido em 0,0001 e para função de perda utilizaram uma função de entropia cruzada (*binary cross-entropy*).

Os autores apresentam o resultado da rede *U-Net* para superfícies de concreto, na detecção de rachaduras, com uma acurácia de 96,48%. Foram 1000 imagens selecionadas e utilizaram o *COCO Annotator* para fazer a anotação. A rede conseguiu marcar com excelente precisão as áreas que apresentavam rachaduras.

Figura 22 - Resultados obtidos com o modelo de segmentação

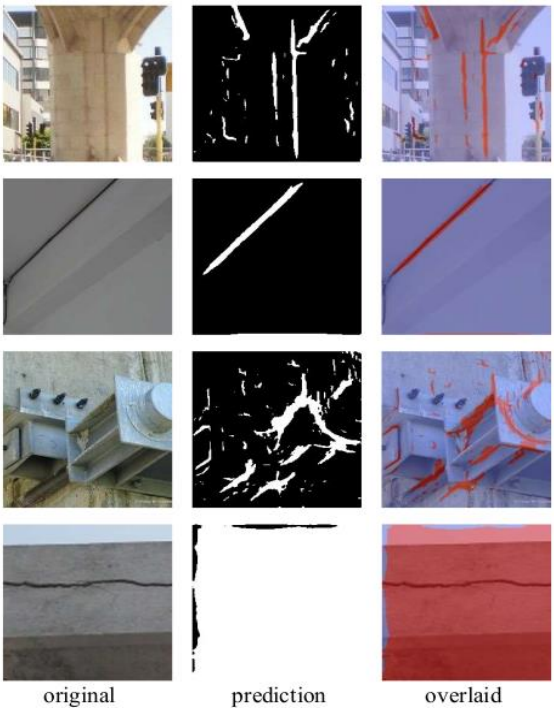


Fonte: Bai *et al*. (2020)

Os resultados apresentados na Figura 22 são a níveis de pixel. Os autores então apresentam os resultados das redes a nível de objeto. Dado que a imagem está mais distante e as rachaduras agora, quando presentes, aparentam ser bem mais finas do que a nível de pixel, ambas as redes não performaram bem. O resultado final, após a classificação de imagens com rachaduras e sem, e então a segmentação, foi de apenas 26,15% de acurácia.

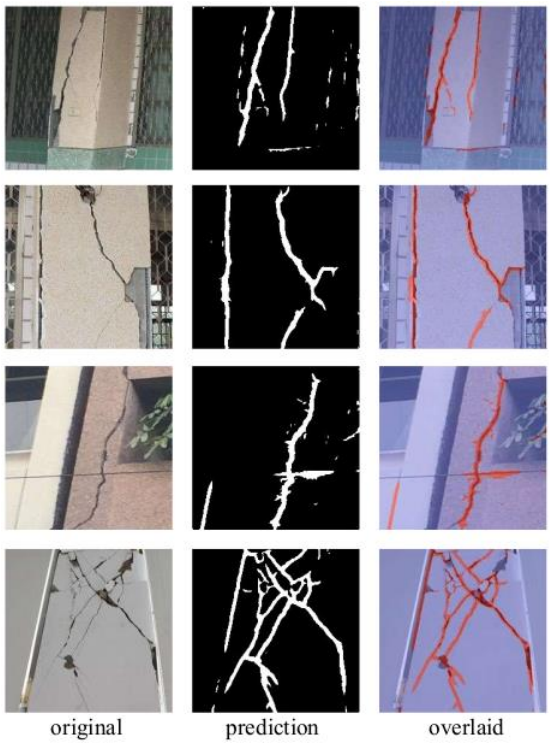
A Figura 23, retirada do material dos autores, mostra um cenário onde a rede exibiu um resultado ruim na classificação a nível de objeto. Em outro resultado (Figura 24) o modelo performou melhor, com uma acurácia de 59,55%. Os autores ressaltam que o fato dos últimos resultados terem apresentado uma acurácia baixa se deve ao fato das imagens possuírem objetos 3D, onde o conjunto de dados utilizado era mais apoiado em imagens de planos.

Figura 23 - Resultados a nível de objeto



Fonte: Bai *et al*. (2020)

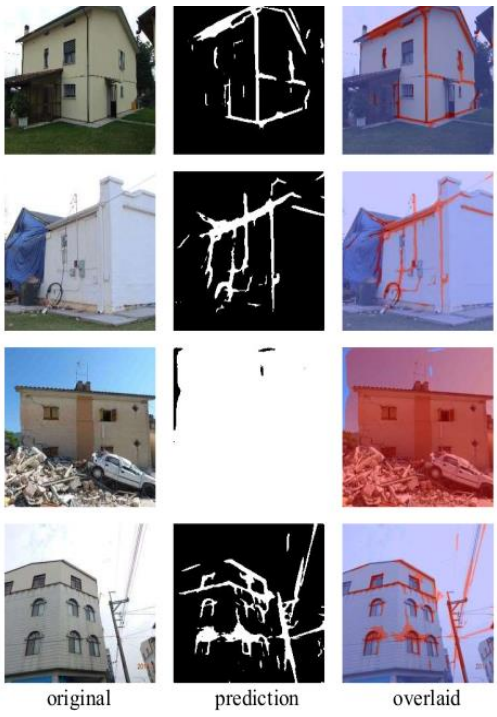
Figura 24 - Resultados a nível de objetos planos



Fonte: Bai *et al*. (2020)

A presença de diversos ruídos nos dados (fios, pessoas, plantas, e outros objetos) torna a tarefa de classificação mais complicada, segundo os autores. De um total de 5832 imagens, somente 500 foram classificadas corretamente. Uma acurácia de 8,57%. Um dos motivos pela baixa acurácia, segundo os autores, é que as imagens foram redimensionadas para uma qualidade bem menor quando submetidas para treinar as redes, então uma imagem grande, com uma resolução bem melhor, acabou em uma imagem de 224*224 pixels. Outro fator é que mesmo um conjunto de dados anotados não consegue representar todas as nuances de um cenário real em sua forma mais complexa, como formas geométricas, texturas, escalas, objetos que se assemelham a rachaduras etc.

Figura 25 - Resultados a nível de estruturas



Fonte: Bai *et al*. (2020)

4.2.3 Conclusão do trabalho de Bai *et al*. (2020)

Os autores inicialmente tentaram utilizar duas redes *U-Net* para classificar e segmentar os dados, mas as mesmas performaram pobremente. Quando optaram por usar a rede *Phi-Net* para classificar as imagens e então segmentar utilizando a *U-Net,* os resultados a nível de pixel foram excelentes. A nível de objetos e estrutura das

redes enfrentaram problemas com a classificação. Os autores terminam os estudos confiantes de que o método proposto é a solução para o problema de segmentação.

4.3 A DEEP CONVOLUTIONAL NEURAL NETWORK FOR SEMANTIC PIXEL-WISE SEGMENTATION OF ROAD AND PAVEMENT SURFACE CRACKS (JENKINS *ET AL*., 2018)

Neste trabalho os autores propõem uma rede neural *fully convolutional* para análise de imagens de pavimentos e estradas.

4.3.1 Dados utilizados por Jenkins *et al*. (2018)

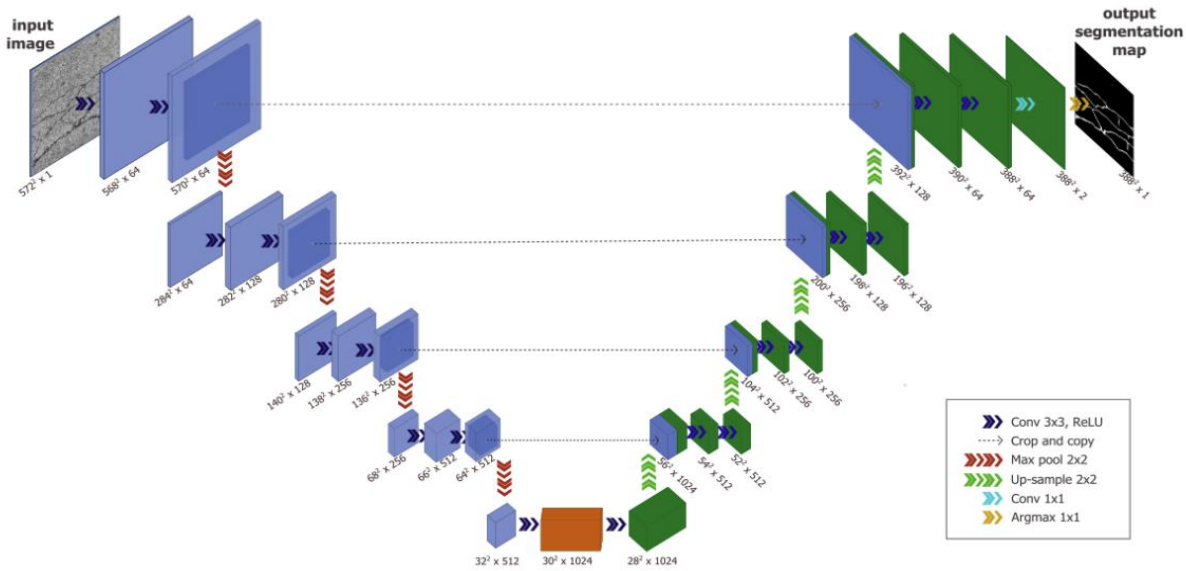
O estudo conta com poucas imagens de testes, somente 118 imagens capturadas por câmeras móveis, segundo os autores. O estudo faz uma análise das imagens a nível de pixel. As imagens são em escala de cinza, possuem o formato de 480*320 pixels. O conjunto de imagens conta também com uma imagem, em formato binário (preto e branco) correspondente, destacando as rachaduras.

4.3.2 Arquitetura utilizada por Jenkins *et al*. (2018)

Os autores utilizaram em seu algoritmo uma rede *U-Net*. Segundo eles, tal tipo de rede é ideal para casos que possuam poucos dados para treinamento. A rede *U-Net* foi desenvolvida para trabalhar com segmentação de imagens microscópicas de células, bactérias e outros. Nesse tipo de pesquisa a quantidade de dados não é grande, mas as imagens são bem detalhadas. A rede é dividida em duas unidades: a unidade de *encoding* e *decoding*.

A primeira seção, *encoding* dos dados, possui uma série de camadas convolucionais com funções de ativação *ReLU*. A segunda etapa, de decodificação, é composta de operações *up-sampling*.

Figura 26 - Arquitetura do projeto de Jenkins *et al*. (2018)



Fonte: Jenkins et al. (2018)

4.3.3 Resultados obtidos por Jenkins *et al*. (2018)

Os autores fizeram a comparação entre a acurácia do algoritmo proposto com outros seis algoritmos pesquisados. Os resultados são dados utilizando-se três métricas de comparação: Precisão, *Recall* e *F1-Score*. Abaixo a tabela (Figura 27) é exibida com os resultados.

Figura 27 - Resultados obtidos por Jenkins *et al*. (2018)

Method	Precision	Recall	F1-Score
Canny [18]	12.23%	21.15%	15.76%
CrackTree [19]	73.22%	76.45%	70.80%
CrackIT [20]	67.23%	76.69%	71.64%
FFA [21]	78.56%	68.43%	73.15%
CrackForest (KNN) [16]	80.77%	78.15%	79.44%
CrackForest (SVM) [16]	*82.28%*	**89.44%**	*85.71%*
Proposed (U-Net)	**92.46%**	*82.82%*	**87.38%**

Fonte: Jenkins et al. (2018)

Os autores utilizaram uma máquina com 12GB de RAM e uma GPU da NVIDIA Titan XP. O treinamento levou aproximadamente três horas.

4.3.4 Conclusão do trabalho do Jenkins *et al*. (2018)

Por mais que o foco do trabalho esteja em detecção de rachaduras em superfícies de estradas e pavimentação, levou-se em consideração o mesmo pelas semelhanças: plano 2D em superfícies planas com o foco na detecção de rachaduras. O trabalho data de 2018, o conjunto de dados de testes é pequeno. Mas vale destacar que mais uma vez o modelo de rede *U-Net* apareceu em outro trabalho, com um excelente desempenho, comparado com outros algoritmos. Um detalhe importante é que as imagens de testes não apresentam variações para cenários com ruído.

4.4 DETECTING CRACKS AND SPALLING AUTOMATICALLY IN EXTREME EVENTS BY END-TO-END DEEP LEARNING FRAMEWORKS (BAI ET AL., 2021)

O trabalho apresentado tem como finalidade desenvolver um *framework* ponta a ponta para automaticamente detectar rachaduras e fragmentação em prédios e pontes após eventos extremos, como no caso de terremotos. Para isso os autores apresentam três modelos de *Mars Regional Convolutional Neural Networks* (*Marsk R-CNNs*).

4.4.1 Dados utilizados por Bai *et al*. (2021)

Dados da base *Phi-Net, Pohang* (terremotos) e *Mexico City earthquake* são utilizados no estudo. Os dados são anotados utilizando-se a ferramenta *COCO Annotator*, onde é definida uma área com polígonos que fazem a demarcação das áreas danificadas. Os autores utilizaram imagens de vários tamanhos, de 147*288 até 4600*3070. Ainda utilizaram o método de *albumentation* (albumização), que segundo os autores, foi desenvolvido por Buslaev *et al*. (2020) com a finalidade de aumentar a quantidade de dados para treinamento, visto que a técnica utiliza as mesmas imagens, mas aplicando rotação, alteração de tamanho, *cropping*, etc.

4.4.2 Arquitetura utilizada por Bai *et al.* (2021)

Os autores utilizaram o trabalho de Chen *et al*. (2019) para desenvolver seu estudo, *MMDetection*. Tal projeto utiliza uma rede *Mask R-CNN*. Uma *Mask R-CNN* é uma rede neural convolucional (CNN) estado-da-arte na segmentação de imagens, é uma extensão de uma *Faster R-CNN*. No estudo apresentado os autores fazem uso de três modelos de *Mask R-CNN*:

- Uma *Mask R-CNN* faz uso da combinação de *ResNet* com *Feature Pyramid Network* (FPN) com a finalidade de fazer a abstração de características das imagens. Nesse modelo os autores substituíram a FPN pela *PANet*, seguindo o estudo de Liu *et al*. (2018), que visava o aumento de performance. Chamaram este modelo de *APANet Mask R-CNN*.
- A estrutura central de uma *Mask R-CNN*, foi inicialmente projetada fazendo uso de uma *ResNet* de 101 camadas. Para este modelo os autores optaram por seguir o estudo apresentado por Sun *et al.* (2019) que desenvolveu uma nova rede chamada *HRNet*. Chamaram de H*RNet Mask R-CNN*.
- O terceiro modelo é baseado no estudo apresentado por Cai e Vasconcelos (2019) que propõe uma *Cascade Mask R-CNN*, cuja finalidade é resolver problemas de *overfitting* quando o limite usado para computar o IoU (*Intersection of Union*) e a desproporção da qualidade sobre a inferência e o treinamento.

4.4.3 Resultados obtidos por Bai *et al*. (2021)

Todos os testes dos modelos foram executados em um computador com uma GPU GTX 2080, o *learning rate* utilizado foi de 0,002, a função de perda utilizada é a de entropia cruzada (*cross-entropy*).

Os autores apresentam três resultados, um para cada conjunto de imagens. Começando com o conjunto de imagens do projeto *Phi-Net*, um total de 33413

imagens de diversas resoluções. Os resultados são apresentados na Figura 28 pela tabela fornecida pelos autores.

Figura 28 - Resultados obtidos (Phi-Net)

Methods	Accuracy	Recall	Precision
Cascade Mask R-CNN	78.9%	70.7%	**88.7%**
APANet Mask R-CNN	**81.1%**	84.8%	83.8%
HRNet Mask R-CNN	58.6%	**95.5%**	57.7%

Fonte: Bai et al. (2021)

O segundo é um conjunto de imagens de Pohang, na Coréia do Sul, completado pela ACI (*American Concrete Institute*). Um total de 4109 imagens com resoluções de 2600*3890 e 5180*3460 foi utilizado (Figura 29).

Figura 29 - Resultados obtidos (Pohang)

Methods	Accuracy	Recall	Precision
Cascade Mask R-CNN	66.0%	37.3%	**91.8%**
APANet Mask R-CNN	67.6%	57.6%	79.3%
HRNet Mask R-CNN	**68.1%**	**67.0%**	75.6%

Fonte: Bai et al. (2021)

O terceiro conjunto de imagens foi coletado pela universidade de Purdue, fotos obtidas na investigação de um terremoto de escala 7.1 que atingiu a Cidade do México em 2017. Os resultados são apresentados na Figura 30.

Figura 30 - Resultados obtidos (Purdue University)

Methods	Accuracy	Recall	Precision
Cascade Mask R-CNN	69.4%	45.5%	**90.9%**
APANet Mask R-CNN	**74.7%**	70.4%	85.7%
HRNet Mask R-CNN	69.1%	**73.5%**	73.4%

Fonte: Bai et al. (2021)

4.4.4 Conclusão do trabalho de Bai *et al*. (2021)

O foco do trabalho está mais a nível de objeto e construção do que a nível de pixel, isso pode também ajudar a explicar a diferença dos resultados quando comparados com outros trabalhos aqui apresentados, dado a quantidade de ruídos

presentes nas imagens testadas. Mesmo em um cenário com tamanha quantidade de ruídos e variedades de tamanhos e resoluções, o estudo apresenta um bom resultado. Os autores terminam o estudo de forma confiante e com a intenção de, através de outros conjuntos de dados, melhorar a acurácia e precisão na segmentação dos dados.

5 DESENVOLVIMENTO

O processo do desenvolvimento da aplicação é descrito neste capítulo. O trabalho não tem como finalidade desenvolver uma rede neural do zero para o objetivo proposto, porém, visa a aplicação da arquitetura apresentada por um dos trabalhos selecionados e, caso necessário, ajustes a fim de melhorar os resultados. São descritas as duas abordagens escolhidas para o projeto, inicialmente seguindo a estrutura proposta por Flah *et al*. (2020) e depois uma nova arquitetura obtida a partir de materiais literários consultados. Nas duas abordagens o mesmo conjunto de tecnologias e ambientes se mantiveram iguais.

5.1 APLICAÇÃO DA REDE DE FLAH *ET AL*. (2020)

O fluxo de desenvolvimento da aplicação baseada no trabalho de Flah *et al*. (2020) é apresentado a seguir. Durante o desenvolvimento do projeto algumas alterações foram necessárias.

5.1.1 Fluxo de atividades

Abaixo estão descritas as etapas necessárias para o desenvolvimento e treinamento da aplicação.

Obtenção dos dados: os dados devem ser obtidos da fonte informada pelos autores, após isso os dados devem ser separados em duas categorias: imagens com rachaduras e imagens sem rachaduras.

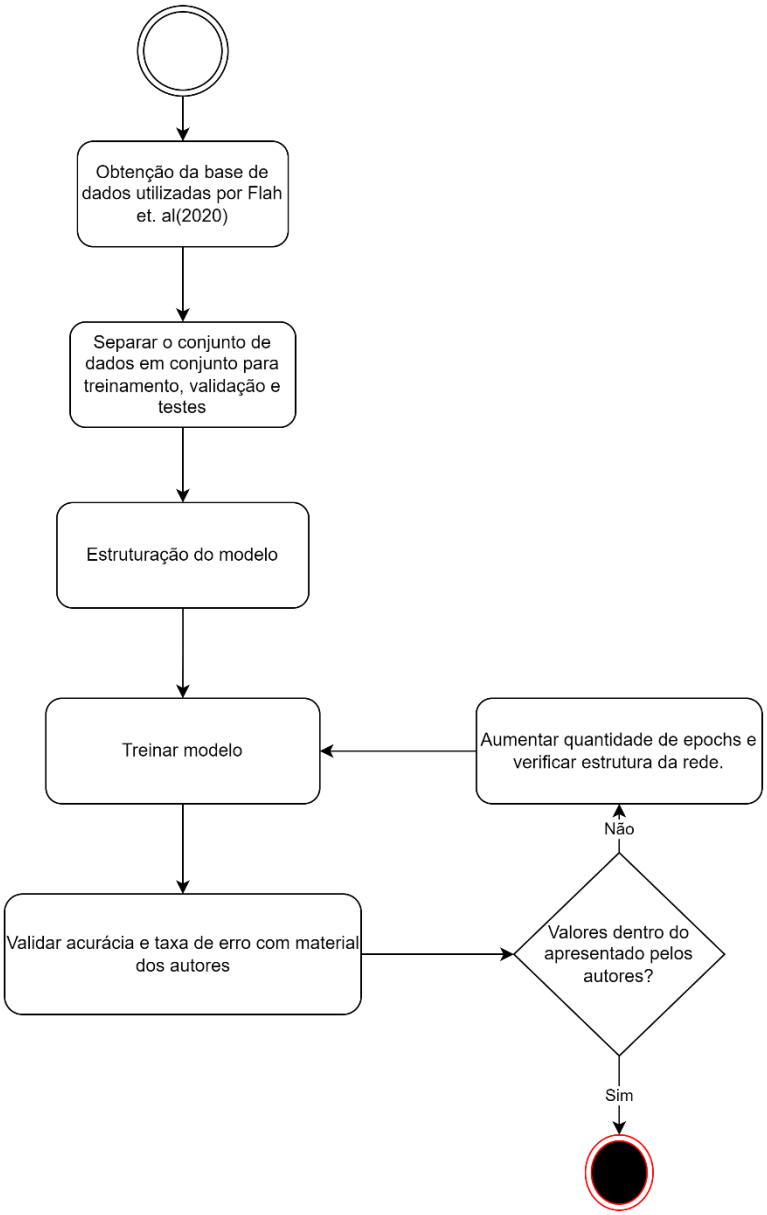
Separação do conjunto de dados: os dados devem serem separados antes do início do treinamento da rede neural. Flah et al. (2020) apontam em seu trabalho que a separação dos dados foi realizada da seguinte forma: 60% do conjunto foi usado para treinamento, outros 20% para validação e o restante dos outros 20% para testes.

Estruturação do modelo: nesta etapa deve-se seguir o material pesquisado para realizar a estruturação da rede.

Treinamento e validação: após estruturação é necessário passar pelo processo de treinamento da rede. Obtido o resultado é necessária uma validação da acurácia da rede e verificar se os resultados estão dentro do

apresentado pelos autores. Do contrário é necessário realizar ajustes na rede e revalidar.

Figura 31 - Fluxo de desenvolvimento rede Flah et al. (2020)



Fonte: do autor

5.1.2 Conjunto de dados

A base de dados fornecida por Flah *et al*. (2021) apresenta algumas modificações na forma de distribuição dos dados, estes foram separados em mais de duas categorias. Para seguir com o desenvolvimento optou-se por utilizar a base

original, que somente difere na estruturação das pastas. A base de dados é um compilado de imagens coletadas de vários prédios do campus da METU (Mendeley Data, 2022). Este conjunto de dados é composto de 458 imagens em alta resolução (4032x3024 pixels), dos quais foram reprocessadas em 40 mil imagens de 227x227 pixels de tamanho.
Flah et al. (2020) indicam que o conjunto de dados devem vir de fotos tiradas de uma distância próxima (entre 25cm e 50cm de distância) do objeto (no caso a estrutura cujo as imagens estão sendo coletadas).

5.1.3 Python

O trabalho foi desenvolvido utilizando-se da linguagem Python, uma das linguagens mais utilizadas no desenvolvimento de algoritmos de *machine learning* (Towards Data Science, 2022) e com uma das maiores variedades de ferramentas no auxílio ao processamento de dados, como é o caso de NumPy, Pandas, Keras entre outras.

5.1.4 Keras

Segundo Chollett (2017), criador do Keras e um dos principais contribuidores do Tensorflow, Keras é um *framework* desenvolvido para linguagem Python que providencia formas convenientes e ágeis de definir e treinar quase todo tipo de modelo de *deep learning*. Chollett (2017) ainda cita algumas das características chave do Keras, tais como:

- Permitir que o modelo desenvolvido rode tanto em uma arquitetura de CPU como em uma GPU;
- API amigável para facilitar o trabalho com modelos de *deep learning*;
- Tem suporte nativo para trabalhar com redes convolucionais (*convolutional neural networks*) e redes recorrentes (*recurring neural networks*);
- Keras permite a composição de redes com múltiplas entradas (*inputs*), múltiplas saídas (*outputs*), entre outros. Segundo o próprio autor, Keras é apropriada para o desenvolvimento de qualquer modelo de *deep learning*.

Mas segundo Chollet (2017), Keras não tem o poder de trabalhar no uso de *tensors* e outras operações de baixo nível, como diferenciação. Ao invés disso, Keras faz uso de uma outra camada, atuando como se fossem módulos que se compõem. Keras atuaria na camada fazendo uma interface entre a definição dos modelos e uma espécie de *back-end*, que seriam as bibliotecas de tensores. Ainda segundo Chollet (2017), no tempo da escrita do livro, três implementações se destacam: Tensorflow, Theano e Microsoft Cognitive Toolkit (CNTK). É através do uso de tais bibliotecas que Keras consegue rodar os modelos tanto usando a arquitetura do processador (CPU), como em uma arquitetura de GPU.

5.1.5 Tensorflow

Segundo Géron (2017), Tensorflow é uma poderosa biblioteca *open-source* para computação numérica. Ela interpreta a estrutura computacional em Python e converte para um código otimizado em C++ (linguagem de programação de baixo nível). Tensorflow ainda suporta computação distribuída. Géron (2017) atribui o sucesso do Tensorflow não somente ao seu design de fácil entendimento, escalabilidade, flexibilidade e excelente documentação, mas principalmente por ter sido desenvolvido e mantido por times de engenheiros do Google. Outra característica importante para projetos como este é que o Tensorflow roda além dos sistemas operacionais conhecidos (Linux, Windows, macOS), também nas plataformas Android e iOS, tornando possível seu uso em aplicações mobile.

5.1.6 Pandas

Pandas é uma biblioteca de código aberto desenvolvida em Python. Tem como finalidade a manipulação de dados e análise. Pandas permite trabalhar com dados de forma fácil, sendo um dos seus usos mais populares a limpeza e pré-processamento dos dados (Nvidia, 2022).

5.1.7 Numpy

Segundo o site da Nvidia (2022), Numpy é uma biblioteca utilizada para processamento e computação numérica de *arrays n dimentionais* (tensores). Ao utilizar Tensorflow se faz necessário o uso de Numpy, pois ele faz uso de *arrays* do Numpy como uma representação de tensores. Quando for providenciar os tensores aos modelos, eles devem estar em formato de *arrays* Numpy multidimensionais (Markus, 2021).

5.1.8 Ambiente

O computador utilizado para o treinamento das redes neurais possui as seguintes configurações:

Figura 32 - Configurações do Ambiente de desenvolvimento

	Versão
Sistema Operacional	Ubuntu 21.04
Processador	Intel® Core™ i5-8400 CPU @ 2.80GHz × 6
Memória	16GB
GPU	Nvidia RTX 3060 12GB
CUDA	11.7
Tensorflow	2.10.0
Keras	2.10.0

Fonte: do autor

5.1.9 Codificação

Para aplicar a arquitetura desenvolvida pelos autores Flat *et al*. (2021), utilizou-se as informações contidas no artigo publicado, conforme apresentado na Figura 33.

O artigo não apresenta detalhamento em níveis de código. Por exemplo, como a rede seria configurada via Keras. Ao tentar replicar a arquitetura utilizando Keras, chegou-se à arquitetura apresentada na Figura 34.

Figura 33 - Informações do artigo de Flat *et al*. (2021)

Layer	Height	Width	Depth	Num of Parameters
Input	227	227	1	–
Conv1	227	227	32	320
ReLU	227	227	32	0
Conv2	225	225	32	9248
ReLU	225	225	32	0
Pool1	112	112	32	0
Dropout1	112	112	32	0
Conv3	110	110	64	18,496
ReLU	110	110	64	0
Pool2	55	55	64	0
Dropout2	55	55	64	0
Flatten	1	1	193,600	0
Dense1	1	1	64	12390464
ReLU	1	1	64	0
Dropout3	1	1	64	0
Dense2	1	1	2	130
Softmax	1	1	2	0
Total Parameters				12,418,658
Trainable Parameters				12,418,658
Non-Trainable Parameters				0

Fonte: Flat et al.(2021)

As duas arquiteturas apresentam quase os mesmos valores na quantidade de parâmetros, mas o resultado diverge na quantidade total de parâmetros treináveis. Isso ocorre provavelmente devido a alguma configuração no carregamento das redes com o Keras.

Figura 34 - Configuração da rede neural com Keras

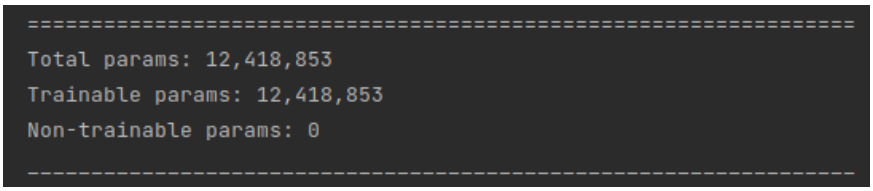
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 225, 225, 32)      896

 conv2d_1 (Conv2D)           (None, 225, 225, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 dropout (Dropout)           (None, 112, 112, 32)      0

 conv2d_2 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 56, 56, 64)        0

 flatten (Flatten)           (None, 200704)            0

 dense (Dense)               (None, 64)                12845120

 dropout_2 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 2)                 130

=================================================================
Total params: 12,873,890
Trainable params: 12,873,890
Non-trainable params: 0
_________________________________________________________________
```

Fonte: do autor

Solicitou-se então auxílio dos autores do artigo via e-mail, os quais responderam prontamente fornecendo o código por eles desenvolvido. A fim de não utilizar todo o código fornecido pelos autores, optou-se por somente observar a forma que eles estavam carregando as redes do modelo, buscando encontrar problemas de configuração.

Após consulta ao material de Flah *et al*. (2020), obteve-se a configuração presente na Figura 35, onde se pode notar ainda uma pequena diferença na quantidade de parâmetros treináveis.

Figura 35 - Sumário da configuração da rede



Fonte: do autor

A Figura 36 apresenta o trecho de código que configura a arquitetura do modelo.

Figura 36 - Código de configuração da rede em Keras

```
model.add(Conv2D(32, kernel_size=(3, 3), strides=1, padding='same', activation='relu', input_shape=(225,225,3)))

model.add(Conv2D(32, kernel_size=(3, 3), strides=1, padding='same', activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(rate=0.5))

model.add(Conv2D(64, kernel_size=(3, 3), strides=1, padding='same', activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(rate=0.5))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(2, activation='softmax'))

model.summary()

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Fonte: do autor

5.1.10 Alterações realizadas

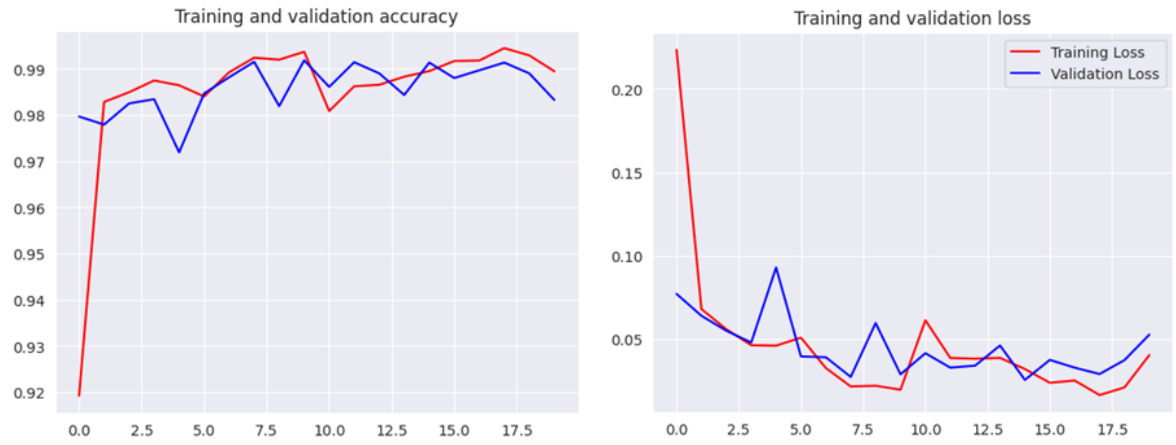
Algumas alterações foram feitas, entre elas a função de perda, taxa de *dropout* e otimizador.

A função de perda (*loss function*), que já foi descrita neste trabalho, escolhida por Flah *et al*. (2020) foi '*categorical cross entropy*', o que faz sentido, pois o trabalho dos autores já apresenta o resultado dividido em categorias, como citado acima. Para o resultado deste trabalho a mesma não apresenta ser a melhor abordagem, pois o foco do trabalho é identificar se a imagem contém rachaduras ou não, ou seja, uma classificação binária. Portanto, optou-se por alterar a função de perda para *'binary_crossentropy'*.

O outro ponto que foi alterado é que na definição da rede de Flah *et al*. (2020) os autores optaram por um *dropout* de 50%. Rodando o modelo com um *dropout* de 30% parece ter melhorado os resultados obtidos.

5.1.11 Resultados obtidos

Figura 37 - Acurácia e Taxa de erro



Fonte: do autor

Os dois gráficos apresentados na Figura 37 demonstram os resultados da fase de testes e validação, tanto para a taxa de acurácia (gráfico a esquerda) quanto para a taxa de erro (gráfico a direita). Pode-se notar que o comportamento da rede no conjunto de testes é semelhante ao comportamento da rede no conjunto de treinamento. Segundo Elgendy (2021) este comportamento é o desejado pois sinaliza

que a rede não está apresentando *overfiting* (onde o modelo acaba assimilando o conjunto de testes e não aprendendo com ele) e nem *underfiting* (onde a rede não está conseguindo aprender os padrões).

A rede apresentou uma acurácia de 99,10%, compara com o melhor resultado apresentado por Flah *et al*. (2020), pode-se afirmar que os resultados foram expressivos e apresentaram uma acurácia dentro do apresentado pelos autores. A comparação foi realizada em relação a capacidade da rede conseguir classificar corretamente as imagens entre contendo rachaduras ou não. Flah *et al.* (2020) apresentam 3 classificadores em seus resultados, o primeiro sendo o classificado que indica a presença de rachaduras ou não, este tendo uma acurácia de 98,25% no conjunto de testes.

5.1.12 Considerações

Flah *et al*. (2020) realizam a categorização dos resultados entre as 5 categorias descritas por eles: Imagens sem rachaduras, com rachaduras na horizontal a esquerda (HL), com rachaduras na horizontal a direita (HR), vertical a direita (VR) e vertical a esquerda (VL).

Optou-se por não seguir com esta implementação, pois o objetivo do trabalho é apresentar de forma visual e automática onde, na imagem, se localiza a rachadura. Flah *et al*. (2020) apresentam este resultado também, mas esse resultado de segmentação é obtido não através do uso de redes neurais e sim do uso de um novo processamento, utilizando-se de outra ferramenta em uma etapa posterior a classificação da imagem (Matlab). Outro detalhe é que os autores fazem a conversão das imagens para escalas de cinza, removendo assim os canais RGBs necessários para a utilização em um processo de segmentação utilizando redes neurais, descrito mais à frente. Seria necessária uma nova abordagem para o problema.

5.2 NOVA ABORGAGEM

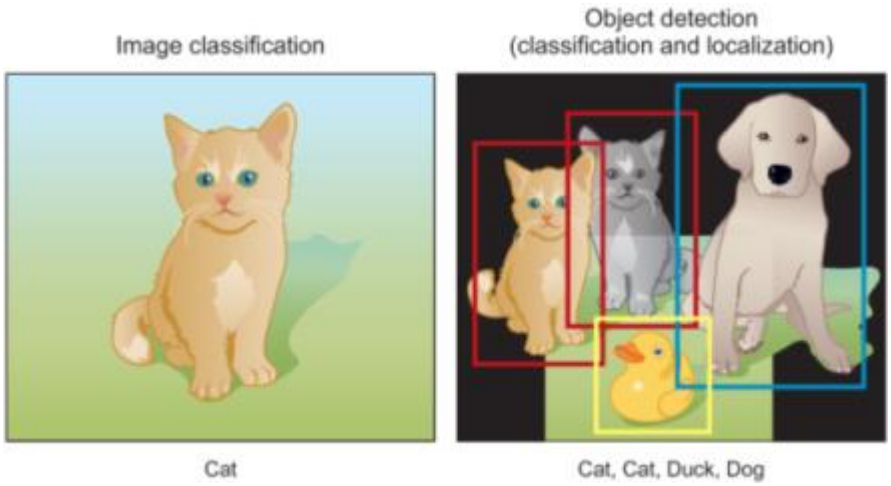
O projeto tem como objetivo ser capaz de ajudar engenheiros e equipes de inspeção a identificar, de forma automática, a presença de rachaduras. Somente fornecer uma aplicação que apresenta o percentual de certeza de uma certa imagem

conter ou não rachaduras não é de grande utilidade. Para que a aplicação possa ter um valor de entrega, necessita apresentar de forma visual onde estão localizadas tais rachaduras nos conjuntos de imagens ou mesmo filmagens dos locais a serem inspecionados. Duas possíveis formas de atingir tal objetivo são utilizando-se de outras duas técnicas de *deep learning*: detecção de objetos ou de segmentação semântica.

5.2.1 Detecção de objetos

Segundo Elgendy (2021) uma tarefa de detecção de objetos envolve tanto localizar objetos na imagem quanto classificar cada objeto detectado dentro da imagem. O resultado obtido de tais técnicas são imagens com os objetos detectados contornados com um retângulo e, em alguns casos, uma anotação identificando a classe de tal objeto.

Figura 38 - Exemplo de resultado de detecção de objetos



Fonte: Elgendy(2021)

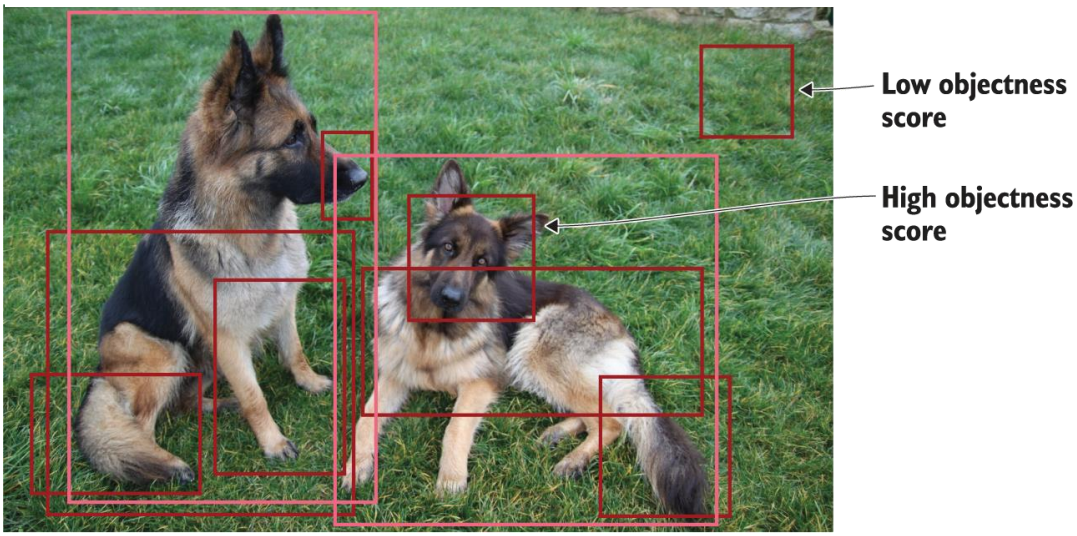
Elgendy (2021) cita que, geralmente, os *framewoks* de detecção de objetos possuem quatro componentes principais, que são descritos a seguir.

Region propoal (região proposta): Ainda nas palavras de Elgendy (2021), um modelo de *Deep learning* gera RoIs (*regions of interest*, regiões de interesse). Estas regiões são onde a rede acredita que existam objetos, para cada imagem um modelo gera inicialmente várias áreas (RoIs) e cada uma com um score. As áreas com maior score são então encaminhadas para processamento.

Feature extraction (extração de características): Elgendy (2021) explica que é nessa etapa onde as características visuais são extraídas de cada área encontrada. Essas características são então analisadas e é determinado se aquela área selecionada contém ou não o objeto a ser detectado.

Nom-maximum suppression (NMS): Imaginanado uma fotografia que possui uma pessoa, o modelo pode detectar várias áreas para o mesmo objeto em questão (a pessoa). Para que a rede não retorne vários resultados para o mesmo objeto, nesta etapa, o modelo selecionará a área mais ampla de todas as selecionadas e, também, combinará a intersecção de áreas menores a fim de pegar um único objeto que contemple toda a área classificada como contendo o objeto (Elgendy, 2021). A Figura 39 apresenta um exemplo de múltiplas áreas identificadas.

Figura 39 - Exemplo de resultado com múltiplas áreas identificadas

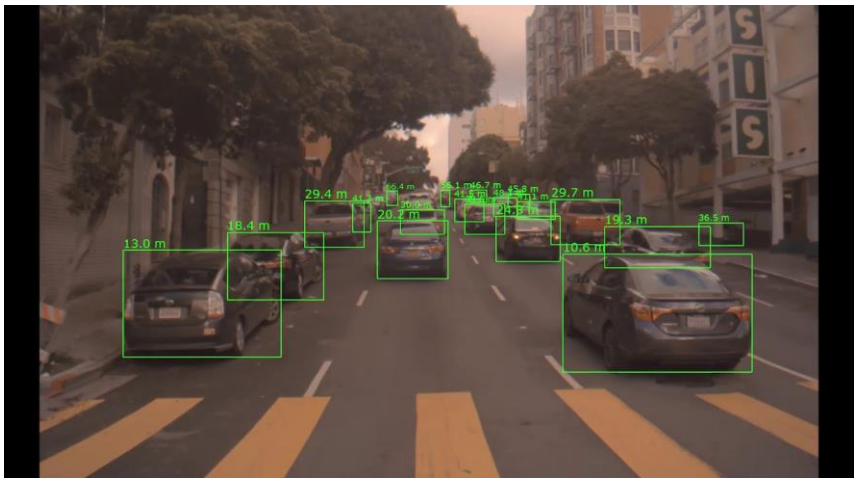


Fonte: Elgendy, 2021

Evaluation metrics (métricas de validação): Assim como em um modelo de classificação contém métricas para validar o desempenho da rede, os modelos voltados para detecção também possuem suas métricas. Entre elas estão mAP (*mean average precision*, média da precisão média), PR curve (*precision-recall curve*, curva de precisão-recal) e IoU (*intersection over union*, intersecção sobre união) (Elgendy, 2021).

A técnica de detecção de objetos tem sido empregada amplamente em projetos de *deep learning*, especialmente em projetos voltados para carros autômatos (Elgendy, 2021). A Figura 40 apresenta um exemplo.

Figura 40 - Exemplo do emprego de detecção de objetos

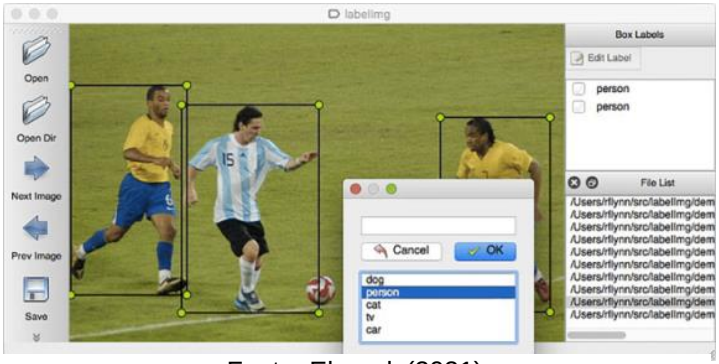


Fonte: site Nvidia (2022)

5.2.2 Limitações no uso de detecção de objetos

O que limita o uso de detecção de objetos no projeto proposto é que primeiramente seria necessário um conjunto de dados anotados para treinar e validar o modelo de rede proposto. Não foram encontrados nenhum conjunto de dados anotados para o projeto em questão, tentar fazer a anotação dos dados na mão consumiria muito tempo e mesmo para um time com várias pessoas seriam necessárias algumas horas de trabalho. Elgendy (2021) indica a utilização da ferramenta LabelIMG que facilita este trabalho. Tentou-se utilizar a mesma para anotar os dados fornecidos por Flah *et al*. (2021), mas o tempo para fazer todo o processo por uma pessoa seria considerável. Ao tentar realizar o processo de forma manual o processo para anotar cerca de 250 imagens levou mais de duas horas. A Figura 41 apresenta um exemplo de tela da ferramenta LabelIMG.

Figura 41 - Ferramenta LabelIMG

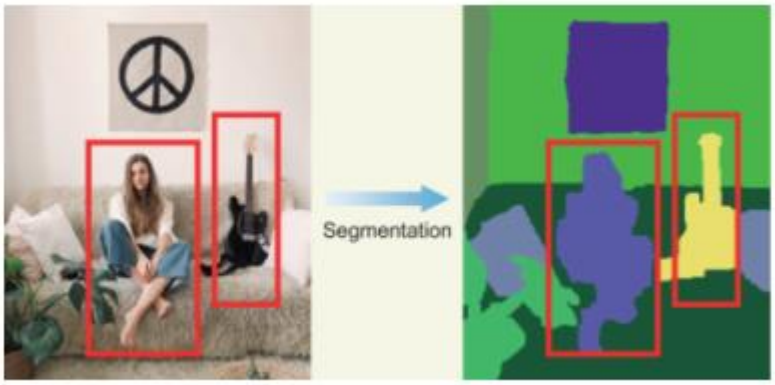


Fonte: Elgendy(2021)

5.2.3 Segmentação semântica

Segundo Markus (2021) uma tarefa de modelos de segmentação semântica consiste em classificar cada pixel da imagem em uma classe de objeto, pintando cada pixel da mesma cor mediante a categoria classificada. Ainda nas palavras de Markus (2021), um exemplo seria classificar em uma imagem gatos e cachorros, os gatos teriam todos os pixels que compõem a região em vermelho e os cães em azul (por exemplo). Uma característica de redes de segmentação é que essas têm como saída a imagem com sua largura de saída como de entrada, mas os canais de saída divergem dos canais de entrada (geralmente RGB), sendo os canais de saída relacionados a quantidade de classes a serem identificadas (Markus, 2021). UM exemplo de segmentação é apresentado na Figura 41.

Figura 42 - Exemplo de aplicação de segmentação



Fonte: Elgendy (2021)

5.2.4 Upsampling

O grande diferencial das redes voltadas para segmentação é que precisam retornar, como citado acima, a imagem com seus canais, só que no processo de convolução a imagem acaba sofrendo alterações dentro da rede, o que tornaria inviável somente utilizando uma rede neural tradicional. É preciso então aumentar a resolução da imagem de forma que volte ao formato de entrada (Markus, 2021). Esta técnica de “restaurar” a imagem para seu estado original é denominada *upsampling*. Markus (2021) destaca algumas das formas com que isto é feito: n*earest neighbor interpolation e bilinear interpolation*.

5.2.5 Definições de passos a seguir

O principal limitador do uso de detecção de objetos neste projeto é o fato de que simplesmente destacar com um retângulo onde estão localizadas as rachaduras em uma imagem não seria a melhor forma de apresentar os resultados. Também vale ressaltar que para possíveis melhorias futuras, detectar os locais de forma mais precisa seria necessário. Outra questão é a quantidade de dados com tais anotações, até o momento deste trabalho não foram localizadas bases dados anotadas para o uso em detecção de objetos.

A utilização de segmentação de objetos possibilita uma detecção visual mais direta, mostrando onde foram localizadas imagens e abre possibilidades para que os resultados possam ser utilizados em projetos futuros para coisas como calcular a espessura da uma rachadura utilizando-se dos pixels identificados em relação a distância da imagem, também pode-se tentar mensurar as áreas identificadas e controlar se com o passar o tempo a área tem se expandido ou não.

5.2.6 Desenvolvimento de aplicação com Segmentação Semântica

A seguir são descritos os passos utilizados para o desenvolvimento da nova rede utilizando Segmentação Semântica e os resultados obtidos.

5.2.7 Fluxo de atividades para nova rede

Diferentemente da primeira rede desenvolvida esta nova rede apresenta a necessidade de mudança de algumas etapas e a adição de novas etapas no desenvolvimento.

Obtenção dos dados: os dados obtidos devem possuir tanto imagens com rachaduras quanto imagens sem a presença de rachaduras. Mas é obrigatório a presença de uma máscara para cada imagem, tanto para os casos positivos quanto para os casos negativos.

Normalização dos dados: todas as imagens devem possuir o mesmo tamanho e resolução. Isso se faz necessário para que a rede tenha como entrada o mesmo conjunto de parâmetros. Validar o conjunto de dados

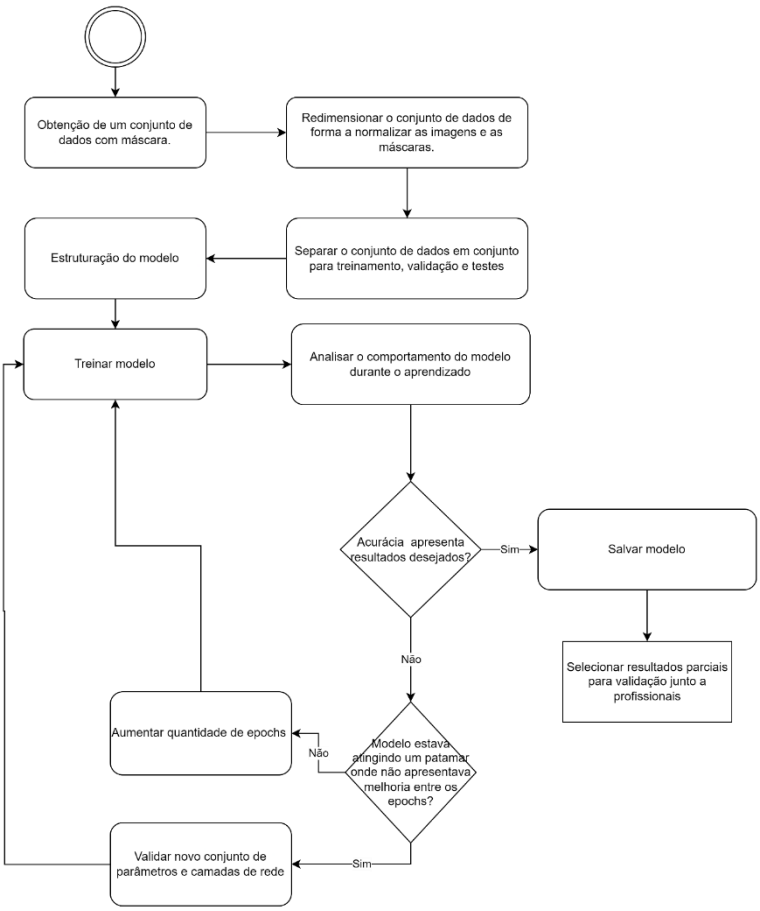
obtidos e realizar a normalização. Esta etapa não foi realizada no desenvolvimento da primeira rede pois se tratava de um conjunto de dados já normalizados, como apresentado por *Flah et al.* (2020).

Separação do conjunto de dados: a separação do conjunto de dados deve seguir o padrão apresentado para a primeira rede: 60% para treinamento, 20% para validação e 20% para testes.

Estruturação do modelo: nesta etapa será necessário estruturar a rede de forma a apresentar o resultado segmentado.

Validação: o objetivo da rede é ser capaz de segmentar as imagens de forma a refletir a verdade apresentada na máscara representado as áreas com rachaduras. Mas o indicador a ser considerado para analisar a performance da rede será a acurácia apresentada. A acurácia vai indicar qual a percentagem de pixels foi classificada corretamente em relação a máscara informada.

Figura 43 - Fluxo de desenvolvimento de Segmentação

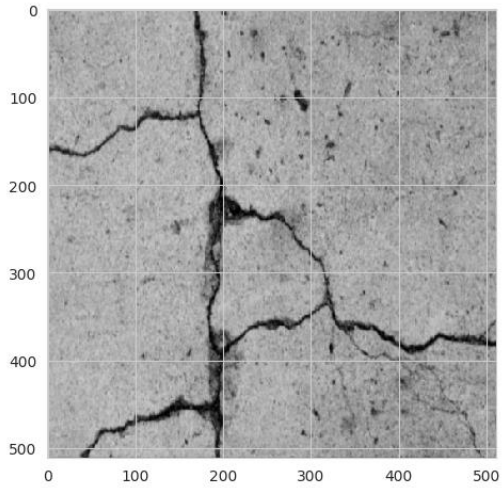


Fonte: do autor

5.2.8 Conjunto de dados

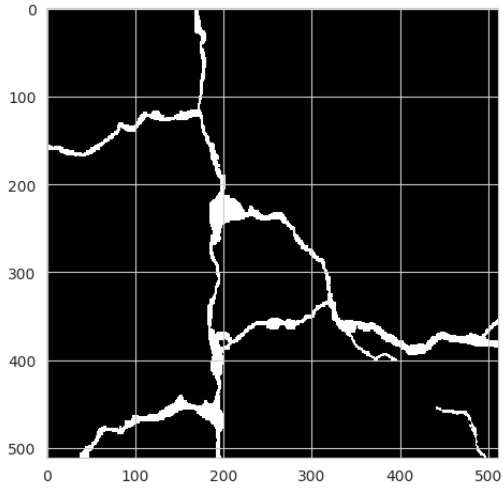
Outro conjunto de dados foi obtido (PlumX Metrics, 2022), a fim de trabalhar outras características desejadas da aplicação, a segmentação. Para isto foi necessário encontrar um conjunto de dados que continha uma máscara. O que seria essa máscara? Seria a representação binária da imagem, onde uma cor representa o *background* (ou qualquer outro elemento da imagem) e outra cor representando a parte que a categoria que rede deve identificar. A Figura 44 apresenta a imagem real, que se quer analisar. Já a Figura 45 é a máscara desta figura, ou seja, o resultado que se espera obter da rede neural. O conjunto de dados apresenta 400 imagens com tamanhos e resoluções variadas.

Figura 44 - Imagem do conjunto de dados após redimensionamento



Fonte: do autor

Figura 45 - Máscara da imagem



Fonte: do autor

5.2.9 Equalizando o conjunto de dados

Este conjunto de dados apresenta dois problemas: as imagens possuem tamanhos diferentes e estão em tons de cinza.
Como o conjunto de imagens possuía uma variedade bem grande de tamanho, optou-se por fazer o redimensionamento para uma qualidade não tão baixa, iniciando com um tamanho de 512x512 pixels. Acontece que o hardware utilizado não conseguia processar esse volume de dados, resultando em um *overflow* de memória. Seguiu-se então com imagens redimensionadas para 256x256 pixels de tamanho.

Outro detalhe é que o conjunto de imagens está em formato de tons de cinza, ou seja, somente 1 canal. Todas as redes pré-treinadas como ResNet50, VGG e outras são treinadas em imagens com canais RGB. Foi então alterado, ao fazer o redimensionamento, adicionando 3 canais (mantendo as informações do canal original em cada um dos demais). Seguir com a rede trabalhando com 3 canais, ainda que não sejam os 3 canais RGB de forma real, possibilita já ter os dados prontos para seguir com outra abordagem, caso necessário.

5.2.10 U-Net

Ao seguir com a estruturação da nova rede optou-se por usar o modelo de uma rede U-Net. Procurou-se manter uma estrutura similar da desenhada por Flah *et al*.(2020). Algumas alterações tiveram que ser feitas, além de novas camadas serem adicionadas. U-Net é uma estrutura de uma rede neural apresentada por Ronneberg, Fischer e Brox em 2015 (Markus, 2021). U-Net é representada visualmente em formato de um ‘U’, do lado esquerdo estão os chamados *encoders*, responsáveis por aplicar a convolução e aprender com os dados, do lado direito se encontram o *decoders*. Partindo do princípio de que a tarefa de segmentação trabalha com a ideia de aplicar a técnica de *upsampling,* nos dados de saída da rede neural, de modo a obter a imagem no seu formato original (Markus, 2021).

5.2.11 Codificação da rede

Ao estruturar a rede, procurou-se manter a mesma estrutura definida no modelo de Flah *et al*. (2020). A Figura 46 apresenta como era configurada no modelo dos autores e na Figura 47 como foi configurada a rede de forma a trabalhar com a segmentação. A estrutura de parâmetros (ativação, *padding*, tamanho de *kernel*) permaneceu a mesma. A taxa de *dropout* também foi preservada. O diferencial dos dois blocos se encontra na quantidade de parâmetros, e na ordem que a camada de *pooling* é carregada no modelo. A quantidade de parâmetros da primeira camada não pode iniciar em 64, como definido previamente, pois devido a nova rede possuir mais camadas do que a inicial a mesma apresenta um consumo de memória muito maior, o que ocasionava erros ao treinar o modelo por falta de recursos computacionais.

Figura 46 - Configuração modelo de Flah et al. (2020)

```
model.add(Conv2D(32, kernel_size=(3, 3), strides=1, padding='same', activation='relu', input_shape=(227,227,3)))
model.add(Conv2D(32, kernel_size=(3, 3), strides=1, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.3))
```

Fonte: do autor

Figura 47 - Bloco de código da nova configuração da rede

```
c1 = tf.keras.layers.Conv2D(InitialSize * 1, kernel_size=(3, 3), kernel_initializer='he_normal', padding='same', activation='relu')(s)
c1 = tf.keras.layers.Conv2D(InitialSize * 1, kernel_size=(3, 3), kernel_initializer='he_normal', padding='same', activation='relu')(c1)
c1 = tf.keras.layers.Dropout(rate=0.3)(c1)
pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(c1)
```

Fonte: do autor

A camada de *pooling* foi movida, pois ao realizar o fluxo de *upsampling* será necessário configurar a rede na mesma estrutura, só que para a rede configurada não é realizado o processo inverso de *pooling*, sendo assim, ela é carregada em um bloco separado ao que vai ser passado para os blocos de mesmo nível no processo inverso. Pois para redes U-Net deve ser ter uma mesma configuração no mesmo nível no processo inverso, conforme exemplificado na Figura 48.

Figura 48 - Bloco de mesmo nível na etapa de upsampling

```
c1 = tf.keras.layers.Conv2D(InitialSize * 1, kernel_size=(3, 3), kernel_initializer='he_normal', padding='same', activation='relu')(s)
c1 = tf.keras.layers.Conv2D(InitialSize * 1, kernel_size=(3, 3), kernel_initializer='he_normal', padding='same', activation='relu')(c1)
c1 = tf.keras.layers.Dropout(rate=0.3)(c1)
pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(c1)
```

Fonte: do autor

Foi então adicionado um segundo bloco de camadas de mesma estrutura do primeiro. E para o processo de *upsampling* (no mesmo nível) um outro bloco de deconvolução de mesma configuração.

Um importante detalhe é que Flah *et al*. (2020) definiam um bloco para “achatar” os parâmetros da rede. Essa etapa era necessária pois as camadas seguintes são de camadas densas, ou seja, totalmente conectadas. Para trabalhar com tais camadas precisa-se alterar a estrutura da saída da camada anterior. Elgendy (2021) cita como exemplo uma imagem de 28x28 pixels, onde aponta que a mesma é representada em uma estrutura de matriz, fora que imagens RGBs ainda possuem 3 canais, o que a torna um tensor. Usando a camada de *Flatten* (achatar) vai converter esses *outputs* em um *array*. Esse tipo de camada não pode ser aplicado para redes que tem como finalidade a segmentação, pois aplicar uma camada *Flatten* resultaria em perder as características espaciais da imagem (Elgendy, 2021), não sendo possível assim fazer a etapa de *upsampling*.

Ao seguir com o estudo para implementação de segmentação chegou-se ao material do professor Sreenivas (2022), apresentados em uma série de vídeos denominados “*Deep learning using keras in python*”. Toda a implementação da rede proposta foi baseada no material apresentado por Sreenivas (2022) e também cabe ressaltar que todo o código de redimensionamento de imagens foi baseado neste mesmo material. A rede ficou com 1.941.105 parâmetros, a Figura 49 apresenta o sumário da rede.

Figura 49 - Sumário da rede

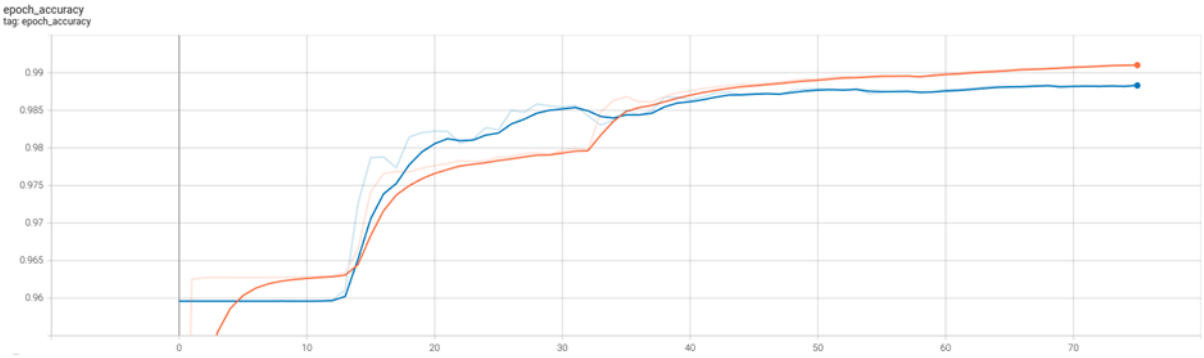
```
=================================
Total params: 1,941,105
Trainable params: 1,941,105
Non-trainable params: 0
_________________________________
```

Fonte: do autor

5.2.12 Resultados parciais

A rede apresentou resultados com uma acurácia alta, 98.8% no conjunto de validação. Levando em consideração a quantidade da base de dados ser pequena, apresentou uma precisão alta. Na Figura 50 é possível observar que tanto no conjunto de treinamento (linha laranja), quanto no conjunto de validação (linha azul), apresentaram um comportamento semelhante.

Figura 50 - Resultados da rede neural(acurácia)



Fonte: Do autor

Ao treinar o modelo foi utilizada uma funcionalidade denominada *early stop*, que trabalha de forma a controlar se o modelo está atingindo um platô. Quando o modelo começar a apresentar resultados muito próximos, em cada *epoch,* isso sinaliza que o mesmo não está mais aprendendo com os dados e não há necessidade de continuar treinando. Pode-se perceber na Figura 51 que dos 100 *epochs* configurados para a rede, o modelo parou no *epoch* 76.

A Figura 52 apresenta uma amostra do conjunto de dados usados para validação. Na primeira coluna está a imagem de entrada, na coluna do meio o resultado esperado e na terceira coluna o resultado obtido pela rede. As imagens da terceira coluna são obtidas a partir do resultado da rede. A rede vai ter como output um tensor representando a imagem, e para cada pixel uma probabilidade daquele pixel ser classificado ou não como compreendendo a categoria (que pode ser uma pessoa, cachorro, carro, avião etc.) ou não. No resultado do modelo, para carregar os

pixels classificados pela rede, foram carregados somente pixels que a rede classificou com uma probabilidade 90% ou mais.

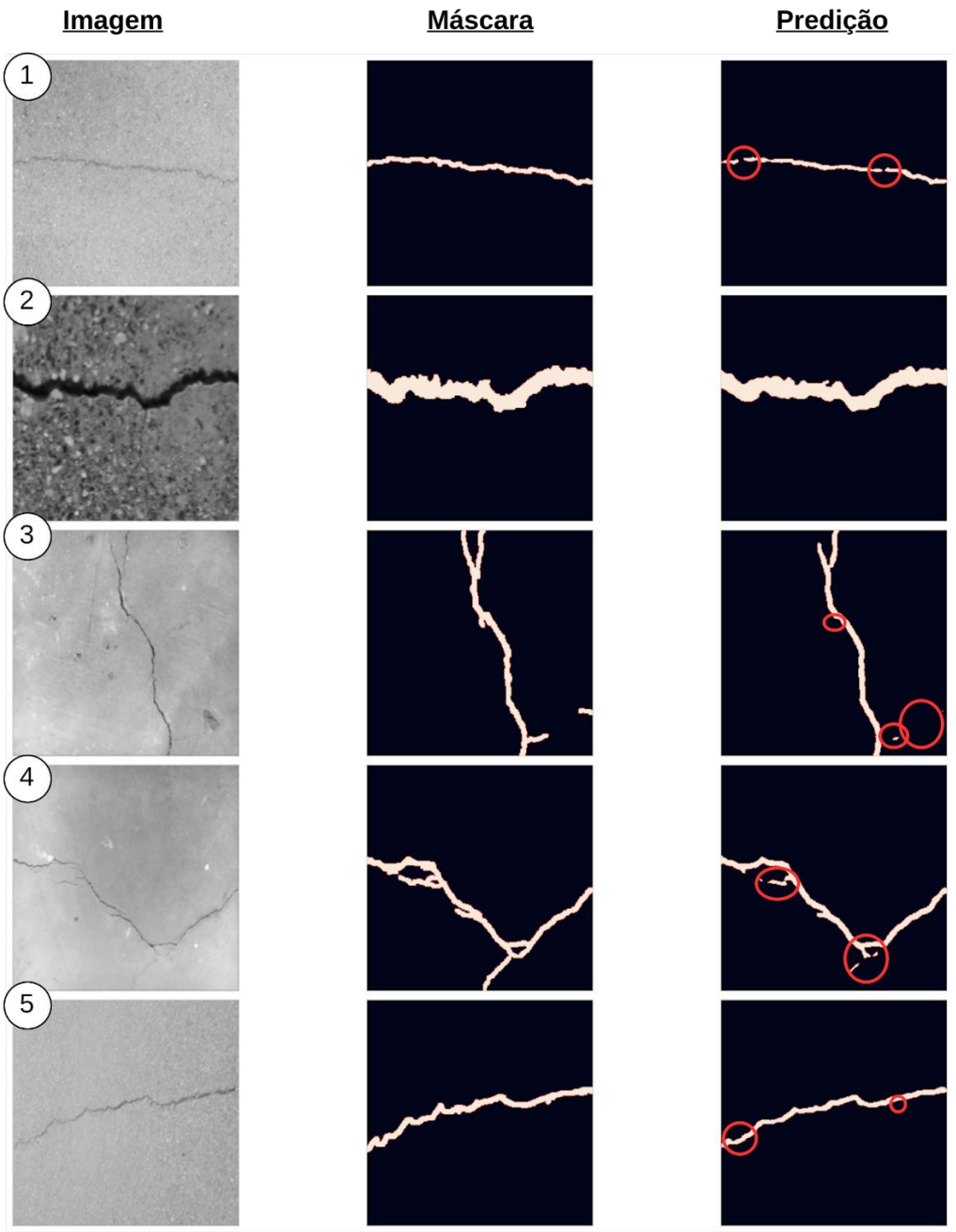
Figura 51 - Log de treinamento da rede

```
11/11 [==============================] - 5s 444ms/step - loss: 0.0253 - accuracy: 0.9906 - val_loss: 0.0362 - val_accuracy: 0.9882
Epoch 68/100
11/11 [==============================] - 5s 414ms/step - loss: 0.0253 - accuracy: 0.9905 - val_loss: 0.0363 - val_accuracy: 0.9883
Epoch 69/100
11/11 [==============================] - 5s 416ms/step - loss: 0.0252 - accuracy: 0.9906 - val_loss: 0.0366 - val_accuracy: 0.9883
Epoch 70/100
11/11 [==============================] - 5s 417ms/step - loss: 0.0246 - accuracy: 0.9907 - val_loss: 0.0371 - val_accuracy: 0.9880
Epoch 71/100
11/11 [==============================] - 5s 434ms/step - loss: 0.0242 - accuracy: 0.9908 - val_loss: 0.0371 - val_accuracy: 0.9882
Epoch 72/100
11/11 [==============================] - 5s 432ms/step - loss: 0.0242 - accuracy: 0.9909 - val_loss: 0.0371 - val_accuracy: 0.9882
Epoch 73/100
11/11 [==============================] - 5s 423ms/step - loss: 0.0238 - accuracy: 0.9910 - val_loss: 0.0372 - val_accuracy: 0.9882
Epoch 74/100
11/11 [==============================] - 5s 429ms/step - loss: 0.0235 - accuracy: 0.9911 - val_loss: 0.0376 - val_accuracy: 0.9883
Epoch 75/100
11/11 [==============================] - 5s 416ms/step - loss: 0.0235 - accuracy: 0.9910 - val_loss: 0.0395 - val_accuracy: 0.9881
Epoch 76/100
11/11 [==============================] - 5s 424ms/step - loss: 0.0234 - accuracy: 0.9911 - val_loss: 0.0382 - val_accuracy: 0.9885
```

Fonte: do autor

Os resultados parciais obtidos através da rede devem então ser submetidos a profissionais da área a fim de serem validados. Na Figura 52 foram adicionados, pelo autor, círculos vermelhos. Nestas regiões pode-se observar que a rede não conseguiu classificar todos os pixels corretamente, como apresentado pela máscara. Pode-se observar nos resultados da Figura 52 que os casos 1, 2 e 5 apresentam pouquíssimos detalhes de diferença, já para os casos 3 e 4 teve uma diferença um pouco maior. Obviamente é necessário a assessoria de um profissional da área para corretamente validar os resultados.

Figura 52 - Resultados parciais



Fonte: do autor

6 Conclusão

Este estudo buscou aplicar modelos de redes neurais já estudadas para detecção de rachaduras em paredes de concreto. Foram analisados diversos materiais de pesquisa a fim de escolher as melhores abordagens para o uso de visão computacional e *deep learning* a fim de auxiliar no processo de inspeção predial.

Inspeções prediais consomem tempo e requerem profissionais qualificados para realizarem as mesmas. Também existe o fator da dificuldade de acesso, que acaba dificultando ainda mais a etapa de vistoria. O prazo de periodicidade das vistorias ainda é algo que não é uma norma nacional, mas cada município acaba decretando suas próprias leis, em alguns como o caso da cidade de Capão da Canoa/RS, a cada 3 anos (Leis Municipais, 2022).

O processo de manutenção predial e de estruturas iniciou com o uso de sensores e tem evoluído com o decorrer dos anos. No decorrer deste estudo encontrou-se vários materiais que empregavam a utilização de visão computacional com *deep learning* a fim de reduzir custos e esforços na realização de inspeção. Alguns estudos apresentavam toda uma plataforma para monitoramento estrutural com o uso de tais tecnologias e ainda o emprego de drones.

O foco inicial do projeto era construir uma aplicação capaz de categorizar imagens e detectar visualmente a localização de rachaduras para fornecer um auxílio visual ao engenheiro, ou técnico, quando estivesse realizando a vistoria. Optou-se em seguir o estudo de Flah *et al*. (2020) como base do projeto. Percebeu-se que a abordagem dos autores não era a mais indicada quando pensado em uma aplicação que receberia uma imagem como entrada e deveria retornar uma outra imagem com a representação visual do que foi identificado. Isto porque os autores realizavam a etapa de identificação visual em outra ferramenta, o que demandaria tempo e acabaria restringindo o uso da aplicação.

Acabou-se seguindo com a aplicação de uma técnica de *deep learning* chamada segmentação semântica. Com o uso de segmentação obteve-se o resultado esperado. Dois problemas ao utilizar segmentação acabaram limitando os resultados, pois dado que para treinar uma rede é preciso de um conjunto de testes anotados, o que para segmentação se trata de uma imagem (geralmente com dois tons de cores, ou mais dependendo da quantidade de categorias) para cada imagem do conjunto de

testes. Apenas um conjunto de dados foi encontrado, e ainda possuía uma quantidade de imagens pequena, 500 imagens. Outro fator é o consumo de memória necessário para realizar o treinamento de tais redes. A simples tarefa de redimensionar o conjunto de imagens consumia cerca de 30 minutos no *setup* utilizado.

O projeto abre possibilidades de outros trabalhos mais complexos e que podem agregar um valor inestimável para o auxílio no controle de manutenções periódicas, entre eles utilizar o resultado obtido da rede neural para calcular a espessura da rachadura e o comprimento dela. Com isso será possível categorizar e mensurar os dados coletados a fim de utilizar para histórico em futuras comparações.

REFERÊNCIAS BIBLIOGRÁFICAS

GRANT SANDERSON. **Neural Networks**. Disponível em: https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa1ZvOVMyaHIzN2cxeUYwSTBfYmRpUlVOcy1sUXxBQ3Jtc0ttS3JtV1MzUklLUU54amJma05TbE0zYTRrYUVBb015UGF0dS1XZGtMNW4xZ1ZPVW0tTFBBZHE1YXdDRW9JeTQ4NUlRUEdqYzlzVnZoVS03SWVYVDlmaEtBRDJRRk1sbWtyVDJEaDdibzkzdENSVEdzbw&q=http%3A%2F%2F3b1b.co%2. Acesso em: 1 mai. 2022.

BAI, Y. *et al*. DEEP CASCADED NEURAL NETWORKS FOR AUTOMATIC DETECTION OF STRUCTURAL DAMAGE AND CRACKS FROM IMAGES. **ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences**, USA, v. 2, n. 2020, p. 411-417, ago./2020. Disponível em: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/V-2-2020/411/2020/. Acesso em: 1 jun. 2022.

BAI, Yongsheng; YILMAZ, Alper; SEZEN, H.. DETECTING CRACKS AND SPALLING AUTOMATICALLY IN EXTREME EVENTS BY END-TO-END DEEP LEARNING FRAMEWORKS. **ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences**, Canada, v. 2, n. 2021, p. 161-168, jun./2021. Disponível em: https://www.researchgate.net/publication/352495295_DETECTING_CRACKS_AND_SPALLING_AUTOMATICALLY_IN_EXTREME_EVENTS_BY_END-TO-END_DEEP_LEARNING_FRAMEWORKS. Acesso em: 1 jun. 2022.

BALAGEAS, D.; FRITZEN, C.; GÜEMES, A.. **Structural Health Monitoring**. 1. ed. USA: ISTE, 2006. p. 13-39.

BALAGEAS, Daniel; FRITZEN, Claus-peter; GÜEMES, Alfredo. **Structural Health Monitoring**. 1. ed. USA: ISTE, 2006. p. 13-39.

CIDADE DE SÃO PAULO. **Programa de Manutenção de Pontes e Viadutos recebe R$ 100 milhões em investimentos**. Disponível em:

https://www.capital.sp.gov.br/noticia/programa-de-manutencao-de-pontes-e-viadutos-recebe-r-100-milhoes-em-investimentos. Acesso em: 16 jun. 2022.

CORREIO BRAZILIENSE. **Prédio que desabou nunca teve Habite-se em 25 anos de fundação**. Disponível em: https://www.correiobraziliense.com.br/cidades-df/2022/01/4976028-predio-que-desabou-nunca-teve-habite-se-em-25-anos-de-fundacao.html. Acesso em: 17 mar. 2022.

CORREIO DO ESTADO. **Projeto de lei sobre vistorias em pontes é aprovado na Câmara**. Disponível em: https://correiodoestado.com.br/politica/projeto-de-lei-sobre-vistorias-em-pontes-e-aprovado-na-camara/359986. Acesso em: 17 mar. 2022.

CORREIO DO POVO. **Inquérito aponta material de baixa qualidade e reformas como causas para desabamento**. Disponível em: https://www.correiodopovo.com.br/not%C3%ADcias/geral/inquérito-aponta-material-de-baixa-qualidade-e-reformas-como-causas-para-desabamento-1.16373. Acesso em: 17 mar. 2022.

DENG, W. *et al*. Vision based pixel-level bridge structural damage detection using a link ASPP network. **Automation in Construction**, Tokyo, v. 110, n. 102973, p. 1-9, set./2019. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0926580519305783. Acesso em: 1 jun. 2022.

FLAH, M.; SULEIMAN, A. R.; NEHDI, M. L.. Classifcation and quantifcation of cracks in concrete structures using deep learning image-based techniques. **Cement and Concrete**, Canada, v. 114, n. 1, p. 1-19, ago./2020. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0958946520302870?via%3Dihub. Acesso em: 13 mar. 2022.

GLOBO G1. **Ponte desaba e quase 'engole' caminhões em Brusque**. Disponível em: https://g1.globo.com/sc/santa-catarina/noticia/2021/06/09/ponte-desaba-e-quase-engole-caminhoes-em-brusque-video.ghtml. Acesso em: 17 mar. 2022.

GLOBO G1. **VÍDEO mostra momento em que parte de prédio de 4 andares desaba no DF**. Disponível em: https://g1.globo.com/df/distrito-federal/noticia/2022/01/06/video-mostra-momento-em-que-parte-de-predio-com-24-apartamentos-desaba-no-df.ghtml. Acesso em: 17 mar. 2022.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A.. **Deep Learning**. 1. ed. London, England: MIT Press, 2016. p. 1-2.

HTTPS://LEISMUNICIPAIS.COM.BR/. **Legislação Municipal de Capão da Canoa/RS**. Disponível em: https://leismunicipais.com.br/a1/rs/c/capao-da-canoa/lei-ordinaria/2009/267/2678/lei-ordinaria-n-2678-2009-estabelece-a-obrigatoriedade-de-realizacao-de-vistorias-periodicas-nas-edificacoes-construidas-no-municipio-e-da-outras-providencias. Acesso em: 7 nov. 2022.

IBM. **O que é inteligência artificial?**. Disponível em: ibm.com/br-pt/cloud/learn/what-is-artificial-intelligence. Acesso em: 3 abr. 2022.

INTEL. **Visão Computacional**. Disponível em: https://www.intel.com.br/content/www/br/pt/internet-of-things/computer-vision/overview.html. Acesso em: 3 abr. 2022.

JENKINS, M. D. *et al*. A Deep Convolutional Neural Network for Semantic Pixel-Wise Segmentation of Road and Pavement Surface Cracks. **European Signal Processing Conference**, Italy, v. 26, n. 2018, p. 2120-2124, jul./2018. Disponível em: https://ieeexplore.ieee.org/document/8553280. Acesso em: 1 jun. 2022.

JOVEM PAN. **Falta de manutenção foi um dos motivos de queda de ponte na Itália, diz relatório**. Disponível em: https://jovempan.com.br/noticias/mundo/falta-de-manutencao-foi-um-dos-motivos-de-queda-de-ponte-na-italia-diz-relatorio.html. Acesso em: 17 mar. 2022.

JR, A. D. S. M. *et al*. INSPEÇÃO PREDIAL: DESCOMPASSO ENTRE LEGISLAÇÃO E PRATICA. **XIX COBREAP**, Fóz do Iguaçu, v. 1, n. 1, p. 1-28, ago./2017. Disponível em: https://ibape-nacional.com.br/biblioteca/wp-content/uploads/2017/08/092.pdf. Acesso em: 20 mar. 2022.

JR, B. F. S; HOSKERE, V.; NARAZAKI, Y.. Advances in Computer Vision-Based Civil Infrastructure Inspection and Monitoring. **Engineering**, USA, v. 5, n. 2, p. 199-222, mar./2019. Disponível em: https://www.sciencedirect.com/science/article/pii/S2095809918308130?via%3Dihub. Acesso em: 13 mar. 2022.

KAYA, Y.; SAFAK, E.. Real-time analysis and interpretation of continuous data from structural health monitoring (SHM) systems. **Bulletin of Earthquake Engineering**, London, v. 13, n. 1, p. 917-934, jun./2014. Disponível em: https://link.springer.com/article/10.1007/s10518-014-9642-9#article-info. Acesso em: 13 mar. 2022.

KAYA, Yavuz; SAFAK, Erdal. Real-time analysis and interpretation of continuous data from structural health monitoring (SHM) systems. **Bulletin of Earthquake Engineering**, Canada, v. 20, n. 7, p. 917-934, jun./2014.

MENDELEY DATA. **Asphalt Crack Dataset**. Disponível em: https://data.mendeley.com/datasets/xnzhj3x8v4. Acesso em: 9 set. 2022.

MENDELEY DATA. **Concrete Crack Segmentation Dataset**. Disponível em: https://data.mendeley.com/datasets/jwsn7tfbrp/1. Acesso em: 2 set. 2022.

MIT TECHNOLOGY REVIEW. **AlphaGo Zero Shows Machines Can Become Superhuman Without Any Help**. Disponível em: https://www.technologyreview.com/2017/10/18/148511/alphago-zero-shows-machines-can-become-superhuman-without-any-help/. Acesso em: 2 jun. 2022.

NVIDIA. **NumPy**. Disponível em: https://www.nvidia.com/en-us/glossary/data-science/numpy/. Acesso em: 16 set. 2022.

NVIDIA. **To Go the Distance, We Built Systems That Could Better Perceive It**. Disponível em: https://blogs.nvidia.com/blog/2019/06/19/drive-labs-distance-to-object-detection/. Acesso em: 22 out. 2022.

NY TIMES. **Google's AlphaGo Defeats Chinese Go Master in Win for A.I.**. Disponível em: https://www.nytimes.com/2017/05/23/business/google-deepmind-alphago-go-champion-defeat.html. Acesso em: 2 jun. 2022.

OZER, E.; Q.FENG, M.. Structural Health Monitoring. **Start-Up Creation**, United States, v. 1, n. 2, p. 345-367, mai./2020. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780128199466000138?via%3Dihub. Acesso em: 12 mar. 2022.

PLUMX METRICS. **Asphalt Crack Dataset**. Disponível em: https://plu.mx/plum/a?mendeley_data_id=xnzhj3x8v4&theme=plum-bigben-theme. Acesso em: 9 set. 2022.

TECH JURY. **How Much Data Is Created Every Day in 2022?**. Disponível em: https://techjury.net/blog/how-much-data-is-created-every-day/. Acesso em: 8 abr. 2022.

TECNOBLOG. **Além do joguinho chinês: o que muda com o AlphaGo, máquina de IA do Google**. Disponível em: https://tecnoblog.net/especiais/alphago-inteligencia-artificial/. Acesso em: 2 jun. 2022.

TOWARDS DATA SCIENCE. **What is a GPU and do you need one in Deep Learning?**. Disponível em: https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d. Acesso em: 7 mai. 2022.

TOWARDS DATA SCIENCE. **What is the best programming language for Machine Learning?**. Disponível em: https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7. Acesso em: 16 set. 2022.

UDACITY. **Deep Learning**. Disponível em: https://www.udacity.com/course/deep-learning-nanodegree--nd101. Acesso em: 1 jan. 2022.

WANG, N. *et al*. Automatic damage detection of historic masonry buildings based on mobile deep learning. **Automation in Construction**, United States of America, v. 103,

n. 1, p. 53-66, mar./2019. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0926580518307568?via%3Dihub. Acesso em: 12 mar. 2022.

YE, X. W.; JINA, T.; YUNB, C. B.. A review on deep learning-based structural health monitoring of civil infrastructures. **Smart Structures and Systems**, China, v. 24, n. 5, p. 567-586, ago./2019. Disponível em: https://doi.org/10.12989/sss.2019.24.5.567. Acesso em: 13 mar. 2022.

YOUTUBE. **DigitalSreeni**. Disponível em: https://www.youtube.com/c/DigitalSreeni. Acesso em: 2 ago. 2022.

YOUTUBE. **Playing Guitar in a Self-Driving Car**. Disponível em: https://youtu.be/fCLI6kxFFTE. Acesso em: 5 ago. 2022.