UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE LORENA

CARLOS ALBERTO CORTEZ JUNIOR

Estudo de modelos de *Machine Learning* para detecção de defeitos superficiais em placas de aço

Lorena
2020

CARLOS ALBERTO CORTEZ JUNIOR

Estudo de modelos de *Machine Learning* para detecção de defeitos superficiais em placas de aço

Trabalho de conclusão de curso apresentado à Escola de Engenharia de Lorena da Universidade de São Paulo como requisito parcial para conclusão da Graduação do curso de Engenharia Física.

Orientador: Prof. Dr. Durval Rodrigues Junior

Lorena
2020

Ficha catalográfica elaborada pelo Sistema Automatizado
da Escola de Engenharia de Lorena,
com os dados fornecidos pelo(a) autor(a)

Cortez Junior, Carlos Alberto
Estudo de modelos de Machine Learning para detecção de defeitos superficiais em placas de aço / Carlos Alberto Cortez Junior; orientador Durval Rodrigues Junior. - Lorena, 2020.
98 p.

Monografia apresentada como requisito parcial para a conclusão de Graduação do Curso de Engenharia Física - Escola de Engenharia de Lorena da Universidade de São Paulo. 2020

1. Machine learning. 2. Aprendizado de máquina. 3. Inteligência artificial. 4. Defeitos superficiais. 5. Placas de aço. I. Título. II. Rodrigues Junior, Durval, orient.

Dedico esse trabalho aos meus pais, a quem devo tudo o que conquistei.

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer aos meus pais, Carlos Cortez e Maria Geralda Cortez por todo o apoio, carinho e amor. Devido ao sacrifício deles, pude chegar onde estou. Todo o tempo de trabalho e orações que dedicaram a mim jamais serão esquecidos. Agradeço à minha mãe por me ensinar, com grande valentia, o imenso valor da mansidão, da lealdade, do silêncio, da oração e do amor. Ao meu pai, agradeço todo o investimento feito em minha formação.

Agradeço imensamente às minhas irmãs, Regiane Cortez e Elaine Cortez, a quem devo muito por todo o apoio e carinho que sempre nutriram por mim. Rê, obrigado por toda a ajuda nos meus afazeres quando criança; meus méritos acadêmicos são seus também. Nana, obrigado por ter cuidado de mim até a época da faculdade. Aproveito para agradecer a vocês duas e os meus cunhados, João e Jouber, pelos afilhados que me deram, Cauã e Emanuely, a quem amo como todo o coração.

Gostaria também de agradecer à minha sobrinha, Maria Eduarda, que é quase uma irmã mais nova, por todo carinho que tem por mim. Minha gratidão também ao meu sobrinho, Pedro, o mais novo da família, que nos enche de amor com seu jeitinho de ser.

Meu imenso agradecimento também aos meus amigos, Anderson José (*in memoriam*); Bruno Telles; Caíque Camargo; Caíque Benedito; Gregory Alexandre e Lucas Andreotti, pessoas que tive o prazer de conhecer ao longo da vida e que estarão comigo para sempre por tudo o que compartilhamos juntos.

Gostaria de reconhecer a Universidade de São Paulo e os professores que tive (e tenho) o imenso prazer de chamar de mestres por todo o comprometimento com a sociedade, combatendo por meio da ciência toda a espécie de obscurantismo, fanatismo e mentiras que tanto nos assombram nos tempos atuais. Deixo aqui meu agradecimento especial ao Prof. Dr. Juan Fernando Zapata, que foi o maior exemplo de profissional que tive durante minha vida acadêmica. Agradeço também ao Prof. Dr. Durval Rodrigues Jr pela orientação em meu trabalho de graduação e ao Prof. Dr. Luiz Eleno por todo o apoio que me deu nessa reta final de curso.

Por fim, gostaria de agradecer ao meu grande amigo, que tenho como irmão, Murilo Afonso Robiati Bigoto. Jamais conseguiria expressar em poucas palavras o tamanho da gratidão que tenho por ele. Agradeço por ser minha dupla em todas as matérias, por todos os momentos vividos, por se preocupar comigo e por me ouvir e estender a mão sempre que precisei.

RESUMO

Nas indústrias siderúrgicas e metalúrgicas no geral, um dos grandes desafios enfrentados é a detecção de defeitos nas peças produzidas com intuito de se prevenir erros de fornecimento e, assim, garantir a qualidade do produto, a segurança e a satisfação do cliente. Em se falando de placas de aço, expressiva parte das oportunidades de melhoria se encontram nos defeitos superficiais, dentre os quais se destacam: fissuras, inclusões, manchas, pites, carepas incrustadas e riscos. Nessa perspectiva, estabelecer mecanismos precisos e eficientes para detecção desses tipos de defeitos é uma excelente oportunidade de melhoria, podendo atuar como espécies de barreiras nas linhas de inspeção, impedindo que eventuais falhas humanas passem despercebidas e prejudiquem a qualidade do produto final. Desta forma, emerge o *Machine Learning* como alternativa para a identificação desses tipos de defeitos, caracterizando os tipos de problemas encontrados nas placas de aço por meio de treinamentos efetuados sobre um conjunto de metalografias. Neste trabalho, as imagens pré-categorizadas foram extraídas do banco de dados da *NEU* (*Northeastern University*) e, então, foram empregados métodos diretos e indiretos de aprendizado supervisionado para predizer os tipos de defeito encontrados em um conjunto de testes, com o qual o modelo não tem contato prévio. Nos métodos indiretos, o treinamento foi realizado sobre as características de textura da imagem e os modelos empregados foram: K Vizinhos mais Próximos, Máquinas de Vetores de Suporte, Árvores de Decisão, Florestas Aleatórias, *AdaBoost* e *Gradient Boosting*. Já no método direto, foram empregadas as redes neurais convolucionais. Em todos os modelos, os melhores hiperparâmetros foram obtidos via validação cruzada e o modelo que apresentou a melhor acurácia foram as Máquinas de Vetores de Suporte, com 95,83% de acurácia nos dados de teste.

Palavras-chave: Aprendizado de Máquina; Defeitos Superficiais; Placas de Aço.

ABSTRACT

One of the major challenges faced in metallurgical industries is the detection of defects in order to avoid supply problems and ensure the quality of products, the safety of users and the customer's satisfaction. Concerning steel plates, a significant part of the opportunities is found in surface defects, among which stand out: crazing, inclusions, patches, pitted surface, rolled-in-scale and scratches. Therefore, establishing precise and efficient detecting mechanisms for these types of defects is an enormous improvement opportunity and it can act as a barrier at inspection lines preventing human errors from going unnoticed, which could affect the quality of the final product. Thus, Machine Learning emerges as a good alternative for detecting these kinds of defects, characterizing them by training data extracted from a set of steel plates metallographies. In this work, the pre-categorized images were extracted from NEU's (Northeastern University) database and then, both direct and indirect supervised learning methods were used in order to predict the types of defects found in the testing set, which has no previous contact with the model's algorithm. In indirect methods, the training step was applied over images' texture features and the studied models were: K-Nearest Neighbors, Support Vector Machines, Decision Trees, Random Forests, AdaBoost and Gradient Boosting. In direct method, Convolutional Neural Networks was the chosen model. In all studied models, the best hyperparameters were obtained using cross-validation and the best accuracy in the test data, 95,83%, was found while using the Support Vector Machines method.

Keywords: Machine Learning; Surface Defect; Steel Plates.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A detecção de defeitos superficiais é uma técnica já bastante conhecida e empregada ao longo dos anos no estudo e fabricação de aços (BAPTÍSTA, SOARES e NASCIMENTO, 2020). Todavia, a análise das metalografias para caracterização desses defeitos é feita, em geral, pela interpretação humana das imagens disponibilizadas e, ao depender desse tipo de dispositivo, o processo de detecção fica sujeito a uma série de fatores que não podem ser objetivamente controlados. Condições físicas, psíquicas e expertise da pessoa que está trabalhando na caracterização (questões de saúde, estresse, ansiedades, tempo de trabalho, experiência, formação, dentre outros) podem influenciar diretamente na qualidade dos resultados obtidos. Esse conjunto de fatores eventualmente levará a uma falha na detecção e o produto liberado pode ter algum tipo de aplicação crítica no cliente final: as placas de aço têm aplicações estruturais; no revestimento de tanques; na indústria automobilística (fabricação de chassis, painéis, portas, dentre outros); na indústria aeronáutica (em hélices, motores, cilindros, por exemplos); na indústria naval (em revestimento de navios); dentre outros (HOEPPNER e ARRISCORRETA, 2012). Portanto, uma falha na inspeção do material pode levar – como já levou (vide seção 2.1) - a consequências ambientais (vazamento de óleos), humanas (acidentes com aviões) e financeiras (elevado índice de sucateamento e retrabalho) gravíssimas (HOEPPNER e ARRISCORRETA, 2012). Nessa perspectiva, propor métodos automatizados de detecção de defeitos emerge como uma potencial solução a esses tipos de problemas e uma das ferramentas que possibilita tal automatização é o Aprendizado de Máquina (AM). Aliás, nas denominadas indústrias 4.0 (conceito moderno de revolução industrial, baseado em automação, internet das coisas e ciência de dados), conhecimentos acerca de aprendizado de máquina já estão sendo amplamente aplicados na detecção de problemas na linha de produção, na previsão de demandas, em otimizações de logísticas, dentre outros. Isso reforça ainda mais o potencial desses mecanismos de reconhecimento de imagens para serem aplicados também em detecções de defeitos superficiais em metais (DATA SCIENCE ACADEMY, 2018).

Desta forma, este trabalho tem o propósito de empregar métodos de *Machine Learning* para treinar modelos supervisionados que sejam capazes de detectar defeitos superficiais em placas de aço. Esse procedimento será feito em duas etapas: uma direta, utilizando-se as redes neurais convolucionais, com extração de níveis de cinza da imagem, preprocessamento e teste do modelo; e outra indireta, extraindo características de textura para posterior aplicação dos modelos de AM (K-vizinhos mais próximos, Máquinas de Vetores de Suporte, Árvores de

Decisão, Florestas Aleatórias, *AdaBoost* e *Gradient Boosting*). Para ambas as metodologias, será aplicada a validação cruzada para reduzir o sobreajuste dos dados e, ao final, a eficácia do modelo será medida por meio do parâmetro denominado acurácia.

Com a metodologia empregada, espera-se obter resultados capazes de cumprir com o propósito de detecção de defeitos superficiais em placas de aço e, para isso, os modelos serão avaliados um a um, comparando-os entre si; entre os tipos de defeitos que conseguem rotular de maneira mais eficiente e com a literatura, a fim de garantir a robustez e completude dos resultados.

2 FUNDAMENTAÇÃO TEÓRICA

No presente trabalho, os resultados serão obtidos, no geral, de duas maneiras distintas: a primeira delas consiste, basicamente, na extração de características de textura da imagem por meio das funções *greycomatrix* (ver 2.2.1) e *greycoprops* (ver 2.2.4) e subsequente interpretação desses aspectos para classificação das imagens em grupos de defeitos. A segunda forma consistirá na interpretação direta dos pixels das imagens por meio das denominadas Redes Neurais Convolucionais (*Convolutional Neural Networks – CNN).* Note que, para essa última forma, a classificação em grupos de defeitos não passa pelo passo intermediário da extração de texturas da imagem, conforme será discutido nas seções a seguir.

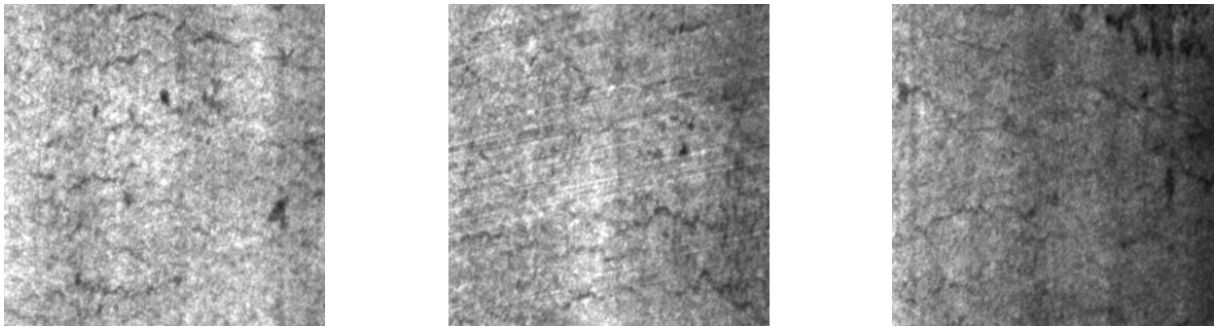
Portanto, para início do trabalho, convém a explanação sobre quais serão os tipos de defeitos a serem estudados e a importância dessa detecção. Posteriormente, serão estudados os modelos e toda a metodologia inerente às suas implementações.

2.1 Defeitos superficiais em placas de aço

2.1.1 Quebras (fissuras)

As fissuras (ou quebras) são espécies de redes formadas por finas rachaduras na superfície do material. Esses defeitos se propagam em regiões da superfície metálica que estão submetidas a altas tensões, levando à formação de microvazios e pequenas rachaduras. As quebras se formam em regiões geralmente associadas a riscos (marcas) superficiais (seção 2.1.6) ou inomogeneidades moleculares e, em metais, essa espécie de defeito se caracteriza também pela perda de resistência mecânica do material se comparado a um análogo em condições íntegras. Esses fatores podem levar a perdas de processos industriais (sucateamento de material) ou, em casos de falhas na inspeção final, os materiais podem sofrer rupturas nos clientes, causando graves transtornos em termos financeiros e de segurança (MENTOURI, MOUSSAOUI, *et al.*, 2018; HAO, LU, *et al.*, 2020; CORROSIONPEDIA, 2014). Na Figura 1, são apresentados exemplos de quebras presentes nas superfícies de placas de aço.

Figura 1 - Quebras (fissuras) na superfície de placas de aço



Fonte: (SONG e YAN, 2020)

Observe que as fissuras se caracterizam, como dito anteriormente, por finas rachaduras dispersas ao longo da matriz metálica. Na imagem, essas rachaduras estão em tom mais escuro se comparadas ao tom do restante do material.

2.1.2 Inclusões

As inclusões são pequenos sítios de impurezas que se manifestam ao longo da estrutura metálica, podendo ser classificadas, quanto à origem, em exógenas, quando resultantes de agentes externos (escórias, reações com o refratário do forno, dentre outros) ou endógenas, quando se originam de reações ou transformações internas no metal. Quanto à composição química, são classificadas em metálicas ou não metálicas (BAPTÍSTA, SOARES e NASCIMENTO, 2020).

As inclusões são, na indústria, um tipo de defeito que vem sendo frequentemente monitorado pois influenciam diretamente na qualidade do aço, possuindo relação direta com as propriedades mecânicas e estruturais do material fornecido. Isso acontece, pois, esse tipo de defeito interrompe a matriz metálica, quebrando a homogeneidade de distribuição de tensões, elevando a concentração destas em determinados pontos do material. Essa configuração implica em aumento de fragilidade e usinabilidade visto que o aumento na energia interna, o bloqueio dos movimentos atômicos intercristalinos e da propagação de discordâncias favorece o aumento de resistência e a redução da ductibilidade (BAPTÍSTA, SOARES e NASCIMENTO, 2020).

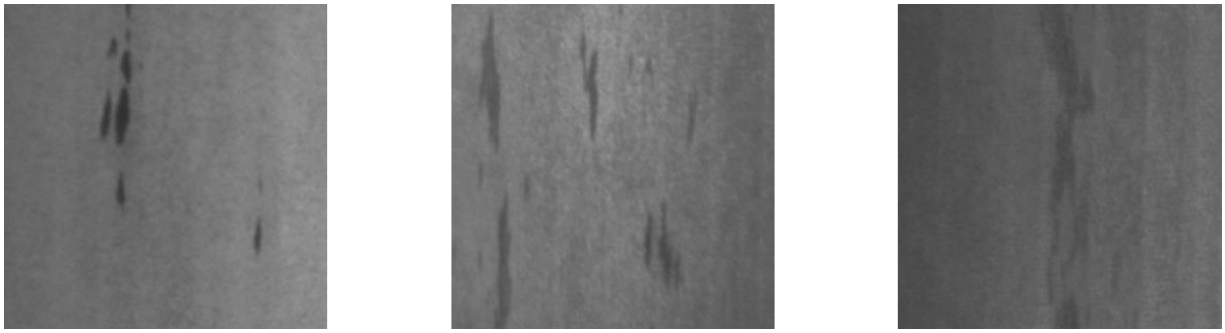
Outro fator agravado pelas inclusões são as corrosões dado que, pela diferença de composição química entre as inclusões e as matrizes, criam-se condições favoráveis ao surgimento de ações

corrosivas localizadas (inclusive as causadas por pite, como será visto na seção 2.1.4) (INFOMET, 2020).

Portanto, para determinadas aplicações, o parâmetro inclusão apresenta alto índice de criticidade. Isso pois a alteração de propriedades mecânicas - como a ductibilidade por exemplo – pode levar à fratura das peças já durante as aplicações, o que pode ocasionar acidentes fatais: imagine, por exemplo, que uma mola ou um eixo feito de aço rompa durante as atividades de um ônibus. O mesmo é aplicável às corrosões: se, por exemplo, a carcaça de um navio petroleiro corrói durante uma navegação, podem haver consequências ambientais irreparáveis.

Na Figura 2, são apresentados exemplos de inclusões presentes nas superfícies de placas de aço. Observe que as inclusões são as marcas alongadas mais escuras ao longo da matriz metálica.

Figura 2 - Inclusões na superfície de placas de aço



Fonte: (SONG e YAN, 2020)

2.1.3 Manchas

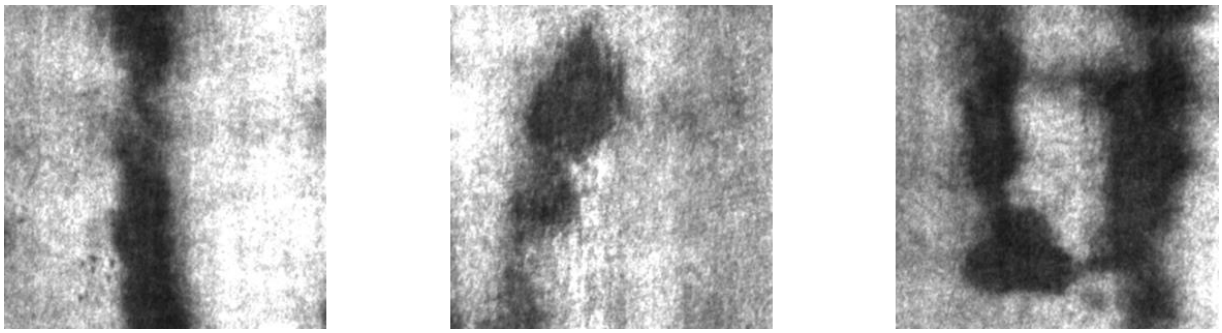
Os defeitos doravante denominados manchas consistem em óxidos presentes na superfície do material que não foram completamente removidos devido à ineficiência do processo de decapagem. As principais consequências do aparecimento desses óxidos na superfície dos materiais é a redução na resistência à corrosão e ao desgaste dos equipamentos de aço sob análise (MENTOURI, MOUSSAOUI, *et al.*, 2018; HAO, LU, *et al.*, 2020).

Ademais, as chapas de aço são amplamente utilizadas na indústria automobilística, com aplicação em chassis, painéis, portas, fechos de porta-malas, dentre outros. Um dos requisitos para esse tipo de aplicação é que a superfície seja livre desses óxidos, uma vez que elas precisam ser revestidas. O revestimento protege o aço contra corrosão, aumentando a vida útil, além de

propiciar também benefícios estéticos. Portanto, do ponto de vista industrial, as manchas por óxidos representam um grande problema de qualidade, sendo que as peças sujeitas a essas condições necessitam de passar por processos de retrabalho (repolimentos ou esmerilhamentos) para que atendam às especificações requeridas e não necessitem ser sucateadas, levando a enormes prejuízos (PAWAR, CHAKRABORTY, *et al.*, 2020).

Na Figura 3, são apresentados exemplos de manchas em superfícies de placas de aço. Como o próprio nome diz, o que caracteriza esse defeito são as manchas mais escuras observadas sobre a matriz metálica nas metalografias.

Figura 3 - Manchas na superfície de placas de aço



Fonte: (SONG e YAN, 2020)

2.1.4 *Pitting*

Pitting (ou, como denominado no trabalho, corrosão superficial por pite) é uma espécie de corrosão que ocorre em superfícies de materiais feitos de aço. Esse tipo de defeito se caracteriza por um ataque corrosivo extremamente localizado, nos quais as regiões de corroídas são relativamente pequenas se comparadas à toda a região exposta ao ataque. Ainda assim, é considerada uma forma muito mais perigosa do que as corrosões uniformes, com uma taxa de corrosão cerca de 10 – 100 vezes superior (DAIDOLA, PARENTE, *et al.*, 1995; KOPELIOVICH, 2020).

Normalmente, as corrosões por pite são categorizadas como processos autocatalíticos, nos quais ocorrem oxidações metálicas resultantes de focos localizados de acidez. O início dessas reações geralmente se dá em regiões superficiais da placa que sofrem danos mecânicos, como riscos (marcas superficiais) por exemplo, ou em regiões internas nas quais existem inclusões não-metálicas ou intermetálicas. Esses locais formam espécies de catodos e anodos, que iniciam o

processo de corrosão autocatalítica, gerando os desgastes conhecidos por *pitting* (DAIDOLA, PARENTE, *et al.*, 1995; KOPELIOVICH, 2020). Na Figura 4, são apresentadas ilustrações de como ocorre o *pitting* (Figura 4-a); um exemplo desse tipo de corrosão em uma placa de aço (Figura 4-b) e a consequência da ocorrência de *pitting* (Figura 4-c), que é o desgaste total da estrutura.

Figura 4 - Processos e consequências do *pitting*

a) Reações autocatalíticas

b) *Pitting* em uma placa de metal

c) Consequências de *pitting* na ponte de ferro do rio Nandu, na China

Fonte: (DAIDOLA, PARENTE, *et al.*, 1995; KOPELIOVICH, 2020)

As corrosões por pite são responsáveis por diversos casos de vazamentos em navios tanques e, também, por diversos incidentes aeronáuticos (DAIDOLA, PARENTE, *et al.*, 1995). A Tabela 1 por exemplo ilustra diversos casos que ocorreram na década de 90, incluindo 3 casos com aviões da Embraer, 1 deles fatal. Além desse, houveram outros dois casos fatais. Ocorreu também um caso com a aeronave C-141 que não culminou em acidente fatal pois foi detectado o defeito durante a inspeção. Isso reforça a relevância de se estudar a detecção de defeitos superficiais em placas de aço e, principalmente, a importância de serem estabelecidos métodos automatizados e confiáveis para auxiliarem os processos de inspeção e análise de amostras.

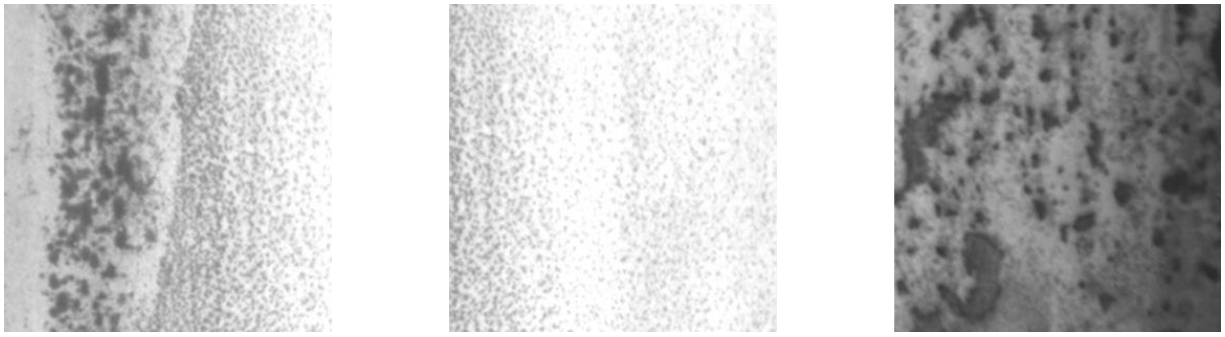
Tabela 1 - incidentes aeronáuticos relacionados aos *pittings*

Aircraft	Location of failure	Cause	Incident severity	Place	Year	From
Bell Helicopter	Fuselage, longeron	Fatigue, corrosion and pitting present	Serious	AR	1997	NTSB
DC-6	Engine, master connecting rod	Corrosion pitting	Fatal	AK	1996	NTSB
Piper PA-23	Engine, cylinder	Corrosion pitting	Fatal	AL	1996	NTSB
Boeing 75	Rudder control	Corrosion pitting	Substantial damage to plane	WI	1996	NTSB
Embraer 120	Propeller blade	Corrosion pitting	Fatal and serious, loss of plane	GA	1995	NTSB
Gulfstream GA-681	Hydraulic line	Corrosion pitting	Loss of plane, no injuries	AZ	1994	NTSB
L-1011	Engine, compressor assembly disk	Corrosion pitting	Loss of plane, no injuries	AK	1994	NTSB
Embraer 120	Propeller blade	Corrosion pitting	Damage to plane, no injuries	Canada	1994	NTSB
Embraer 120	Propeller blade	Corrosion pitting	Damage to plane, no injuries	Brazil	1994	NTSB
Mooney Mooney 20	Engine, interior	Corrosion pitting, improper approach	Minor injuries	TX	1993	NTSB
C-130	Bulkhead "Pork chop" fitting	Fatigue, corrosion pitting	Pressurization leaks	—	1995	LMAS
C-141	FS998 main frame	Corrosion pitting, stress corrosion cracking	Found crack during inspection	—	1991	LMAS

Fonte: (HOEPPNER e ARRISCORRETA, 2012)

Na Figura 5 são apresentadas metalografias de placas de aço com o defeito estudado. Note que os pites são de fácil visualização: são as manchas mais escuras imersas na matriz acinzentada da superfície do aço.

Figura 5 – Metalografias de placas de aço com pitting na superfície



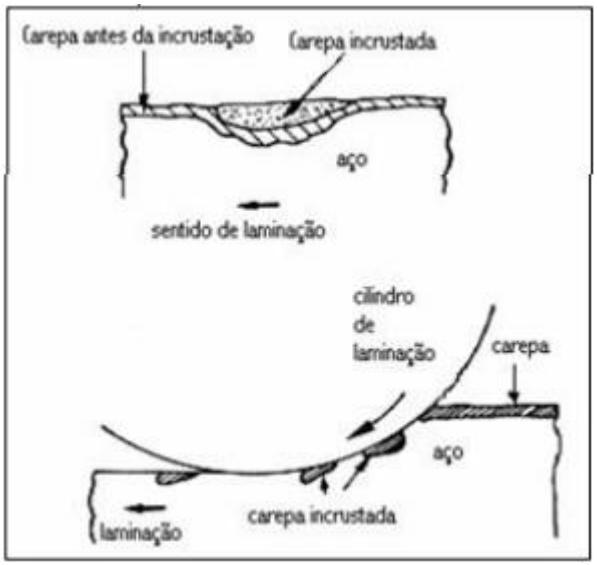
Fonte: (SONG e YAN, 2020)

2.1.5 Carepas incrustadas

As carepas são películas ou resíduos que se formam na superfície de placas metálicas resultantes dos processos de oxidação superficial do aço laminado a quente. Essa formação pode se dar durante tratamentos térmicos, superficiais ou mesmo pela ação do tempo. Os óxidos formados são ferrosos, geralmente nas formas FeO (wüstita), Fe_2O_3 (hematita) e Fe_3O_4 (magnetita) (RIBEIRO JUNIOR, 2019).

O incrustamento de carepas, por sua vez, refere-se às marcas superficiais deixadas nas bobinas (de aço) como consequências da adesão de carepas ao longo das etapas de laminação. Esses defeitos se distribuem aleatoriamente pelo material e são compostos por pequenos desníveis com óxidos em seu interior conforme ilustra a Figura 6 (FÉLIX, MARTINS, *et al.*, 2019).

Figura 6 - Formação de carepas incrustadas durante o processo de laminação a quente



Fonte: (FÉLIX, MARTINS, *et al.*, 2019)

Observe que, quando os cilindros de laminação passam pela superfície do material, as carepas podem aderir ao cilindro, sendo repassadas aos demais materiais laminados de forma subsequente (FÉLIX, MARTINS, *et al.*, 2019).

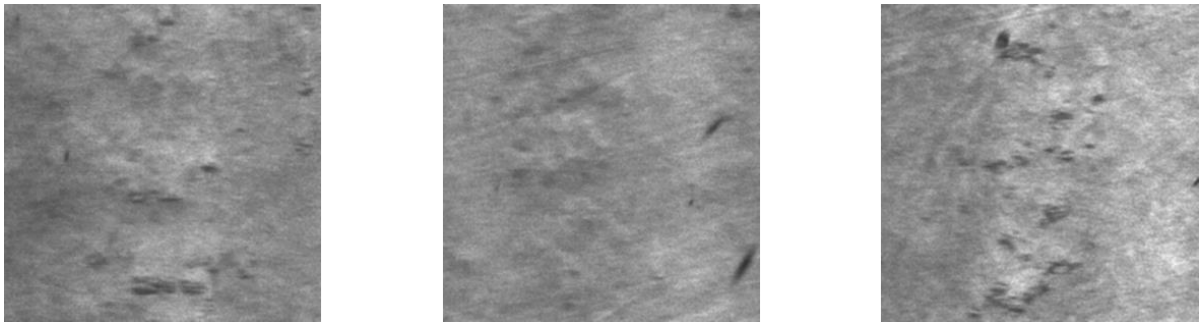
As carepas incrustadas constituem um problema de qualidade muito grande para as indústrias que produzem metais. A formação desses defeitos pode ocasionar uma série de consequências, como por exemplos:

- Perda ao fogo causada pelos óxidos formados;
- Prejuízo à superfície do aço situado abaixo das carepas ocasionado pela oxidação de determinados elementos do material;
- Falhas na planicidade e surgimento de pontas altas/baixas; fatores que podem desfavorecer a entrada do material no laminador, levando ao sucateamento;
- Choque do material laminado com os elementos da linha de laminação, danificando os equipamentos.

(CASTRO, 2005; GORNI, FREITAS, *et al.*, 2008).

Na Figura 7 são apresentadas metalografias de placas de aço com encrustamento de carepas. Na imagem, as carepas são os pontos em relevo distribuídos ao longo da placa metálica.

Figura 7 - Metalografias de placas de aço com carepas na superfície



Fonte: (SONG e YAN, 2020)

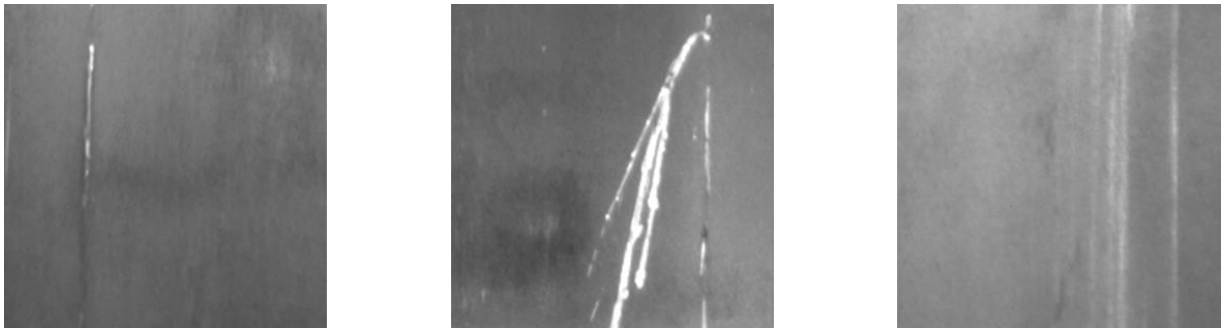
2.1.6 Marcas superficiais

As marcas ou riscos superficiais são entalhes acentuados presentes na superfície do material ocasionados, geralmente, por componentes dos maquinários envolvidos nos processos de laminação. Pode ser gerado também pelo movimento relativo entre as placas de aço e os rolos, resultantes de uma alteração na velocidade de rolamento (TIAN e XU, 2017; MENTOURI, MOUSSAOUI, *et al.*, 2018).

Os riscos também são problemas intimamente associados à qualidade do produto final e, assim como no caso das manchas, podem vir a prejudicar eventuais tratamentos estéticos a serem realizados *a posteriori* nas placas de aço, como as pinturas por exemplo. Outro efeito comprovadamente presente em materiais com marcas superficiais de laminação é uma concentração de tensão nesses pontos, que acabam se tornando os mais frágeis da estrutura. Por fim, os riscos potencializam também a possibilidade de corrosão nos metais e, pelo conjunto de fatores apresentados, entende-se o elevado potencial desse tipo de defeito de causar graves perdas industriais sob o ponto de vista produtivo e também financeiro (ZERBST, MADIA, *et al.*, 2019).

Na Figura 8 são apresentadas metalografias de placas de aço com riscos na superfície, que são as fibras mais claras alongadas ao longo da matriz metálica acinzentada.

Figura 8 - Metalografias de placas de aço com marcas (riscos) superficiais



Fonte: (SONG e YAN, 2020)

Note que, com exceção da figura central, existe um baixo contraste entre as regiões das imagens de riscos em placas de aço. Por isso, as marcas costumam ser defeitos que apresentam certo grau de dificuldade na detecção via métodos automatizados, reforçando a importância de serem obtidos bons resultados neste trabalho.

2.2 Extração de características da imagem – Matriz GLCM

Para o início das análises nos métodos indiretos, a exemplo do trabalho desenvolvido por GHATNEKAR (2018), serão extraídas das imagens suas características de textura, como por exemplos homogeneidade, momento angular secundário, contraste, dissimilaridade e energia. Essa extração será feita por meio da matriz GLCM (*Grey-Level Co-Occurrence Matrix*[1]).

A matriz GLCM é uma importante ferramenta para extração de informações de imagem, permitindo inclusive a obtenção de dados sobre a forma do elemento sob análise, ou seja, é um método que permite ao usuário inferir conclusões acerca das relações espaciais projetadas pelos pixels examinados (GHATNEKAR, 2018; THE MATHWORKS, INC., 2020b; THE MATHWORKS, INC., 2020c).

2.2.1 A função *greycomatrix*

Todavia, para que a GLCM seja compreendida, é importante o conhecimento sobre uma função denominada *greycomatrix*. Essa função gera a GLCM ao calcular a frequência que um pixel

[1] Em tradução direta para o português: “Matriz de Co-Ocorrência em Níveis de Cinza”

com valor de intensidade (ou nível de cinza) i aparece em uma relação espacial específica com um pixel de intensidade j (THE MATHWORKS, INC., 2020a; THE MATHWORKS, INC., 2020c). Essa relação específica é, por definição, o pixel localizado adjacentemente à direita do pixel de interesse, ou seja, o pixel de intensidade j definido anteriormente seria o vizinho imediato à direita daquele com intensidade i conforme a ilustração na Figura 9 (THE MATHWORKS, INC., 2020c).

Figura 9 - Relação específica padrão entre pixels de intensidades i e j

Fonte: o autor

É possível, por outro lado, que o usuário especifique outros tipos de relação caso seja de interesse para a análise a ser realizada. Isso pode ser feito utilizando-se parâmetros denominados *Offsets.* Ao final, cada elemento (i, j) na matriz GLCM resultante será a soma do número de vezes que a relação espacial especificada ocorre entre o pixel de interesse (de intensidade i) e um outro de intensidade j (THE MATHWORKS, INC., 2020a; THE MATHWORKS, INC., 2020c). Mais à frente, na seção 2.2.2, serão apresentados exemplos das funções *greycomatrix*, da formação das GLCMs, bem como variações das relações especificadas.

A principal vantagem de se usar a *greycomatrix* é a otimização do processamento durante análise de uma determinada imagem. Por certo, calcular a GLCM para toda a extensão de uma imagem seria uma tarefa com exigências exorbitantes de capacidade de processamento; assim, a função *greycomatrix* toma conta de escalonar as escalas de cinza da imagem sob análise. Esse escalonamento advém da combinação de dois parâmetros: *NumLevels* e *GreyLimits* (THE MATHWORKS, INC., 2020a; THE MATHWORKS, INC., 2020c).

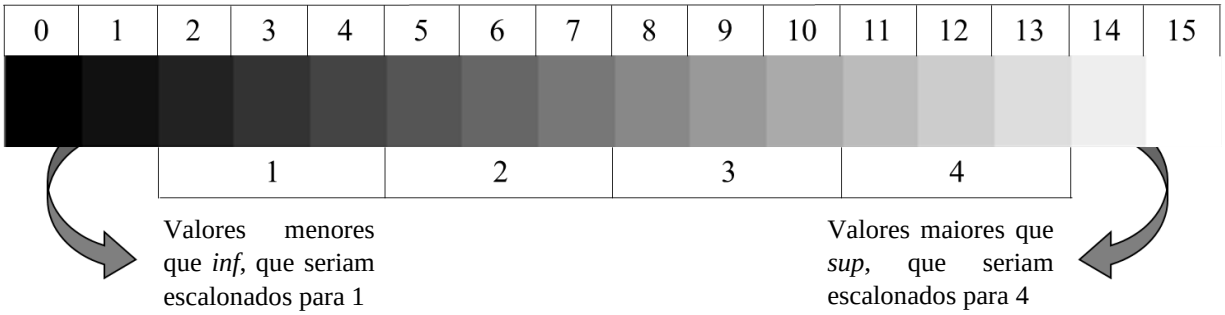
O parâmetro denominado *NumLevels* especifica o número de níveis de cinza que estarão presentes na GLCM. Se, por exemplo, for considerado que 2 níveis de cinza (ou seja, os pixels podem ter uma de duas cores disponíveis, tal como preto e branco) são suficientes para as análises a serem feitas, o usuário pode definir o parâmetro *NumLevels* como sendo igual a 2

(THE MATHWORKS, INC., 2020a). Conforme será visto em 2.2.2, esse parâmetro determina o tamanho da matriz GLCM.

Já o parâmetro *GreyLimits* é um vetor de duas coordenadas: [$inf\ sup$], sendo a primeira o limite inferior e a segunda, o superior dos níveis de cinza escalonados linearmente da imagem sob análise. Valores de cinza menores que inf são escalonados para 1. Valores superiores a *sup* são escalonados para $NumLevels$ (THE MATHWORKS, INC., 2020a).

A Figura 10 ilustra uma aplicação dos parâmetros discutidos acima. Nessa ilustração, utilizou-se o intervalo [2 13][2] para *GreyLimits*. *NumLevels*, por sua vez foi definido como igual a 4.

Figura 10 - GreyLimits e NumLevels em uma imagem de 4 bits



Fonte: o autor

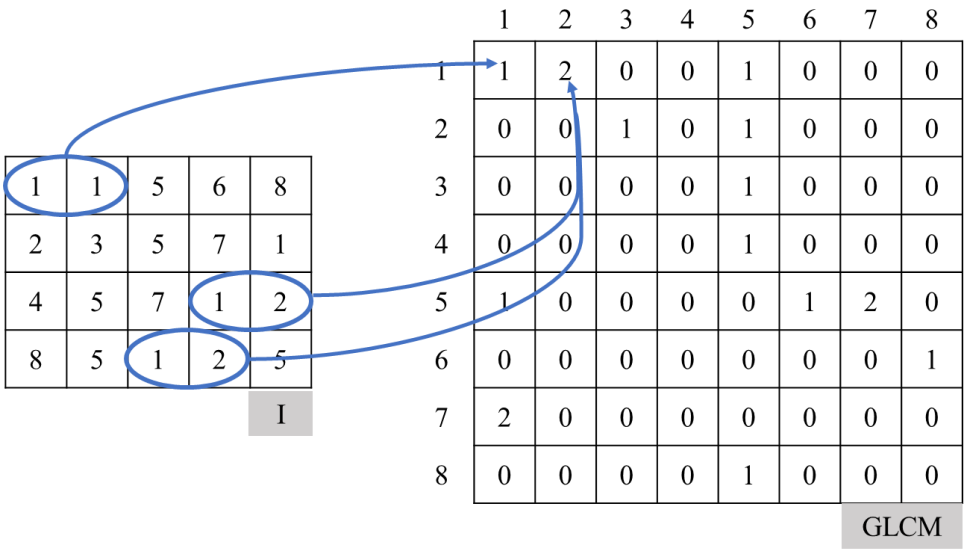
Na Figura 10, observa-se uma escala de cinzas para uma imagem de 4 bits. Isso significa que cada pixel é representado por um conjunto de 4 bits, ou seja, essa imagem pode apresentar $2^4 = 16$ valores, que vão de 0 (preto) até 15 (branco). Note que, para esse exemplo, $NumLevels = 4$.

2.2.2 Formação da Matriz GLCM

Na sequência, será apresentado na Figura 11 um exemplo de formação de uma matriz GLCM de uma imagem I por meio da função *greycomatrix* (THE MATHWORKS, INC., 2020a). Portanto, a *greycomatrix* irá interpretar a imagem I e transformá-la, de acordo com os limites [$inf\ sup$] estabelecidos e o número de níveis de cinza (*NumLevels*), gerando as coordenadas da GLCM.

[2] uint(x) é um comando utilizado para converter x bits para a representação de inteiros sem sinal. Nesse caso, utilizando-se uint(4), obtém se o intervalo [0 1 2 3 ... 15] apresentado na Figura 10. Utilizando-se o comando int(x), obtém-se a representação com sinal, que seria o intervalo [-8 -7 ... 6 7].

Figura 11 - Geração da GLCM a partir de uma imagem I



Fonte: Adaptado de (THE MATHWORKS, INC., 2020a)

Na imagem, é possível observar de forma clara como o parâmetro *NumLevels* determina o tamanho da GLCM: ao escolher $NumLevels = 8$, obtém-se uma matriz GLCM de ordem 8. Desta forma, é trivial inferir que, para dada uma matriz *M*, tem-se:

$$Ordem(M) = NumLevels \qquad (I)$$

Observe também na Figura 11 como é feita a correspondência entre I e GLCM via *greycomatrix*: o elemento (1,1) da GLCM tem o valor 1 pois só há uma ocorrência na imagem de dois pixels horizontalmente adjacentes com intensidades de cinza iguais a 1. Da mesma forma, o elemento (1,2) da matriz é igual a 2 pois existem duas ocorrências nas quais os pixels horizontalmente adjacentes têm intensidades 1 e 2. É importante ressaltar que está sendo utilizada a relação espacial específica padrão (adjacentemente à direita). Caso seja de interesse a especificação de outros tipos de relações espaciais, deve-se utilizar o parâmetro de *Offset* na fórmula da GLCMS:

$$glcms = greycomatrix(I, param1, val1, param2, val2, \dots) \qquad (II)$$

Tal que $param_n$ é o n-ésimo parâmetro e val_n é o valor associado ao n-ésimo parâmetro (THE MATHWORKS, INC., 2020a).

Outro aspecto interessante de se notar é que a matriz GLCM fornece informações sobre a imagem real. Por exemplo, se a maioria dos valores se concentrarem na diagonal da matriz, a textura da imagem é grosseira com relação ao *Offset* pré-determinado, ou seja, há uma

concentração elevada de pixels de intensidades similares ao longo da direção escolhida para análise (THE MATHWORKS, INC., 2020c). Na seção 2.2.4, serão apresentadas interpretações estatísticas de uma GLCM.

2.2.3 O parâmetro *Offset*

Conforme dito anteriormente, esse parâmetro permite a avaliação da imagem I sob perspectivas espaciais distintas da padrão. Sua sintaxe consiste de dois números inteiros que mapeiam a distância[3] entre o pixel analisado e o seu adjacente: $[Offset_{Linha}; Offset_{Coluna}]$. Nessa estrutura, $Offset_{Linha}$ e $Offset_{Coluna}$ representam, respectivamente, o número de linhas e colunas entre o pixel de interesse e seu vizinho (THE MATHWORKS, INC., 2020a; THE MATHWORKS, INC., 2020c). Em geral, os *Offsets* são apresentados como ângulos, conforme a tabela a seguir:

Tabela 2 - Ângulos mais comuns e sua representação em *Offsets*

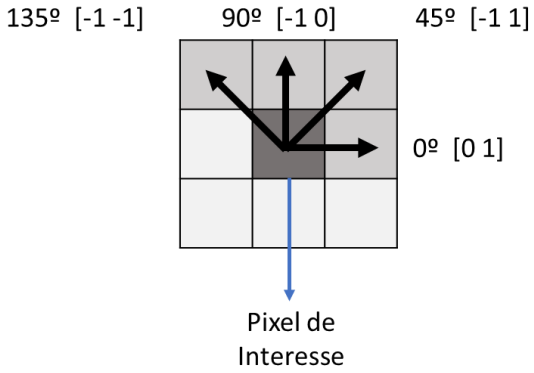
Ângulo	*Offset*
0º	$[0 \quad D]$
45º	$[-D \quad D]$
90º	$[-D \quad 0]$
135º	$[-D \quad -D]$

Fonte: (THE MATHWORKS, INC., 2020a)

Tal que D é a distância entre o pixel analisado e o adjacente ($[Offset_{Linha}; Offset_{Coluna}]$ mencionados anteriormente). A Figura 12 ilustra um exemplo desses arranjos para $D = 1$:

[3] Ou "deslocamento" em tradução direta de *Offset*

Figura 12 – Representação gráfica dos *Offsets* para os ângulos mais comuns



Fonte: Adaptado de (THE MATHWORKS, INC., 2020a)

Assim, utilizando-se de um arranjo de *Offsets* que especificam 1 distância ($D = 1$) e 4 direções (duas diagonais, horizontal e vertical), a imagem de entrada (I) será representada por 4 GLCMs. Se, por exemplo, fossem tomadas 4 distâncias ($D = 4$) e as mesmas 4 direções, I seria representada por 16 GLCMs.

Dito isso, é possível interpretar de maneira mais clara a equação generalizada encontrada em THE MATHWORKS, INC. (2020a), que expressa a formação, via função *greycomatrix* atuante em uma imagem real I, da imagem escalonada SI, a partir da qual serão calculadas as GLCMs:

$$[glcms, SI] = graycomatrix(\dots) \qquad (III)$$

As GLCMs formadas podem fornecer diversas informações estatísticas, que por sua vez trarão dados importantes sobre a textura da imagem I. Essas informações dizem respeito a contraste, correlação, intensidade (energia) e homogeneidade. A função que permite especificar qual das estatísticas será analisada é denominada *greycoprops* (THE MATHWORKS, INC., 2020c).

2.2.4 A função *greycoprops*

Conforme dito na seção anterior, essa função é capaz de extrair propriedades estatísticas (*stats*) da GLCM. Para isso, a função normaliza a GLCM, de tal forma que a soma de seus elementos seja 1. Ao final, cada elemento (i_N, j_N) da GLCM normalizada representa a probabilidade conjunta de ocorrência dos pares de pixel que possuem níveis de cinza i_N e j_N, sujeitos a uma relação espacial definida pela função *Offset* (THE MATHWORKS, INC., 2020b).

Em suma, a função greycoprops calcula propriedades de textura da imagem I por meio da GLCM normalizada. Sua sintaxe, de acordo com THE MATHWORKS, INC. (2020b) é dada por:

$$stats = greycoprops(glcm, properties) \qquad (IV)$$

Em (IV), se o parâmetro *GLCM* consistir de uma combinação de GLCMs, por consequência, *stats* será um arranjo de estatísticas para cada GLCM (THE MATHWORKS, INC., 2020b). P*roperties* diz respeito à propriedade estatística a ser estudada, que neste trabalho serão: ASM (*Angular Second Moment* [4]), contraste, dissimilaridade, energia (ou uniformidade) e homogeneidade.

2.2.4.1 Contraste

O contraste é a propriedade de uma imagem que traz informações a respeito das variações locais existentes entre as regiões mais claras ou mais escuras dessa imagem. Se o contraste for aumentado, as regiões de sombra ou mais escuras tornam-se mais marcadas, intensificando as diferenças locais entre essas regiões e as mais claras. Se, por outro lado, o contraste for diminuído, a transição entre as regiões mais escuras e mais claras tornam-se mais suaves, reduzindo as diferenças entre elas. Desta forma, é possível inferir-se que um elevado valor de contraste pode representar bordas, ruídos ou rugas na imagem (OLIVEIRA, 2010). O contraste também recebe a denominação de soma do quadrado da variância; números situados na diagonal da GLCM simbolizam pouco contraste, que vai se elevando à medida que os pixels se distanciam da diagonal (VASCONCELOS, 2020).

A equação (V) fornece a expressão que será utilizada nesse trabalho para o cálculo do contraste. Nessa equação, $p(i,j)$ indica a probabilidade de ocorrência do par de pixels (i,j) sob a relação espacial especificada pelo *Offset* ao longo da imagem *I*, de $(i,j) = 0$ até N, sendo $N = NumLevels$.

$$C = \sum_{i,j=0}^{N} P_{i,j}(i-j)^2 \qquad (V)$$

[4] Em português, Momento Angular Secundário

(SCIKITS-IMAGE, 2020).

2.2.4.2 Dissimilaridade

Diversos algoritmos têm por base inferências relacionadas a distâncias entre determinadas características dos dados de entrada (como os pixels no caso do presente trabalho). Tais distâncias podem vir a resultar em alterações importantes nos dados de saída de um algoritmo, sendo usualmente atreladas a medidas denominadas dissimilaridade (CARVALHO, 2018).

A dissimilaridade pode ser entendida, de acordo com CARVALHO (2018), como sendo uma medida utilizada para definição do nível de similaridade entre as instâncias, agrupando-as com base nas distinções entre seus atributos. No presente trabalho, a fórmula utilizada no cálculo da dissimilaridade, assim como das outras propriedades, será aquela presente na biblioteca importada para extração das propriedades via *greycoprops* (vide seção 2.2.4). Portanto, segundo (SCIKITS-IMAGE, 2020), tal equação é dada por:

$$D = \sum_{i,j\,=\,0}^{N-1} P_{i,j}|i-j| \qquad (VI)$$

2.2.4.3 ASM e Energia

A propriedade denominada Momento Angular Secundário (ASM) tem esse nome pela sua semelhança com a fórmula da Física utilizada para o cálculo de Momentos Angulares. Todavia, de acordo com OLIVEIRA (2010), no contexto de análise de textura de imagens, essa medida, assim como aquela denominada Energia, diz respeito à uniformidade ou suavidade da distribuição de níveis de cinza na imagem. Isso significa que, se a variabilidade local dos tons de cinza for suave, os valores de ASM e de Energia tendem a ser altos e vice-versa.

Estatisticamente, as propriedades ASM e Energia podem ser descritas pelas equações (VII) e $(VIII)$ abaixo:

$$ASM = \sum_{i,j=0}^{N-1} P^2(i,j) \qquad (VII)$$

$$E = \sqrt{ASM} = \sqrt{\sum_{i,j=0}^{N-1} P^2(i,j)} \qquad (VIII)$$

(OLIVEIRA, 2010; SCIKITS-IMAGE, 2020).

2.2.4.4 Homogeneidade

Homogeneidade é uma propriedade que traz informações a respeito das similaridades na imagem I. Esses dados são extraídos com base na proximidade da distribuição dos elementos na GLCM com relação à sua diagonal. De acordo com TEIXEIRA e FERNANDES (2015), essa característica apresenta altos valores para variações menores nos níveis de cinza ao longo da textura da imagem e, segundo VASCONCELOS (2020), essa característica tem comportamento inversamente proporcional ao contraste quando analisados na matriz GLCM. Conforme diz a autora, a homogeneidade apresenta valores que reduzem de forma exponencial aos se afastarem da diagonal. A expressão fornecida por SCIKITS-IMAGE (2020) que permite o cálculo dessa propriedade é dada pela equação (IX):

$$H = \sum_{i,j=0}^{N-1} \frac{P_{i,j}}{1 + (i-j)^2} \qquad (IX)$$

Discutidas as propriedades que serão exploradas no trabalho, é possível, a título de simplificação, resumi-las, conforme apresentado na Tabela 3.

Tabela 3 - Propriedades: Descrições e Fórmulas

Propriedade	Descrição	Fórmula
Contraste	Traz informações a respeito das variações locais existentes entre os níveis de cinza da imagem.	$C = \sum_{i,j=0}^{N} P_{i,j}(i-j)^2 \qquad (V)$
Dissimilaridade	Medida utilizada para definição do nível de similaridade entre os pixels,	$D = \sum_{i,j=0}^{N-1} P_{i,j}\lvert i-j\rvert \qquad (VI)$

	agrupando-os com base nas distinções entre seus níveis de cinza.	
ASM / Energia	Dizem respeito à uniformidade ou suavidade da distribuição de níveis de cinza na imagem. Apresentam altos valores para baixas variabilidades.	$ASM = \sum_{i,j=0}^{N-1} P^2(i,j) \quad (VII);$ $E = \sqrt{ASM} \quad (VIII)$
Homogeneidade	Traz informações a respeito das similaridades na imagem I. Esses dados são extraídos com base na proximidade da distribuição dos elementos na GLCM com relação à sua diagonal. Tem comportamento inversamente proporcional ao contraste.	$H = \sum_{i,j=0}^{N-1} \frac{P_{i,j}}{1+(i-j)^2} \quad (IX)$

Fonte: O autor

Desta forma, fica consolidada a forma de extração dos dados: por meio da análise dos pixels das imagens, será formada, via função *greycomatrix*, uma matriz denominada GLCM. Dessa matriz, serão extraídas, via função *greycoprops*, as características de textura da imagem descritas na Tabela 3, que serão os dados de entradas para treinarem os modelos (indiretos) de *Machine Learning*, responsáveis por prever futuros defeitos em novas imagens por meio do aprendizado obtido por esses modelos.

Porém, antes de serem estudados os modelos de predição, é necessário elucidar o que é o *Machine Learning* (Aprendizado de Máquina), que é o cerne deste trabalho.

2.3 O Aprendizado de Máquina

O Aprendizado de Máquina (AM) ou *Machine Learning* (ML) pode ser definido de diversas formas distintas, mas que convergem para o mesmo fundamento. Por exemplo, de acordo com GERÓN (2017), é possível descrever o AM como sendo a ciência de programar os computadores para que eles consigam aprender ou tirar conclusões a partir de um conjunto de dados. FACELI, LORENA, *et al* (2011), por outro lado, definem o AM como "a indução de uma hipótese (ou aproximação de função) a partir da experiência passada".

O fato é que, independente da definição analisada, o principal fundamento por detrás do AM é o aprendizado por meio da experiência. E é essa forma de aprendizado que faz do ML um grande diferencial quando comparado aos métodos tradicionais (ou anteriores) de programação. Para elucidar essa diferença, considere o seguinte exemplo: suponha que uma rede de *e-commerce* esteja interessada em estudar padrões de consumo de seus clientes para que possa divulgar suas melhores ofertas em anúncios nas redes sociais. É possível que seja feito um estudo estatístico convencional desses padrões de consumo para que eles possam ser, então, inseridos em uma determinada rotina tradicional de programação que gere como saída uma oferta a esses consumidores. Porém, se forem analisadas as variáveis envolvidas, tais como: perfis de consumidores, localização geográfica do consumidor, sazonalidade nas vendas dos produtos, dentre outros, é possível perceber que existe uma gama imensa de variáveis a serem levadas em conta na programação para que seja atingida uma acurácia minimamente aceitável nas ofertas que serão feitas aos clientes. Além disso, no mundo atual, o perfil de consumo muda muito rápido ao longo do tempo e um algoritmo tradicional seria incapaz de detectar tais alterações por ele mesmo, ou seja, ele necessitaria de uma reprogramação todas as vezes em que ocorressem alterações *a priori* imprevistas no conjunto de dados de entrada.

Agora, imagine um método de programação e análise no qual seja possível retroalimentar e "ensinar" o algoritmo com os dados fornecidos em tempo real pelos consumidores da rede de *e-commerce*, ou seja, codificar um algoritmo que seja capaz de aprender com base em sua experiência obtida dos dados que entram constantemente em fluxo. Isso simplificaria muito o código e sua *performance*, exigindo menos recursos de memória, além de garantir uma adaptabilidade das predições aos *inputs* fornecidos em tempo real pelos clientes. E são exatamente esses pontos que o ML oferece como principais vantagens com relação aos códigos convencionais (GÉRON, 2017). Todavia, para que esses ganhos sejam alcançados, é necessária uma série de cuidados com os tipos de dados a serem analisados, suas tratativas, a quantidade e qualidade deles, dentre outros, que serão discutidos mais detalhadamente na sequência.

2.3.1 Analisando os dados

Segundo GERÓN (2017), existem dois principais desafios em se tratando de ML: algoritmos e dados problemáticos. Com relação aos dados, o autor afirma que as grandes dificuldades se concentram em:

- Quantidade insuficiente de dados de treinamento;
- Dados de treino não representativos;
- Dados de baixa qualidade;
- Características ou propriedades irrelevantes;

FACELI, LORENA, *et al.,* (2011) também menciona possíveis dificuldades a serem encontradas nos dados de entrada que estão diretamente relacionados aos mencionados acima, como por exemplos redundâncias, inconsistências, falta de uniformidade, incompletudes e ruídos. Desta forma, os autores ressaltam que é importante realizar análises e tratamentos prévios dos dados que serão inseridos nos modelos para treinamento de tal forma a mitigar os problemas mencionados acima. De maneira geral, quando tais problemas não são tratados, são observados dois tipos de consequências nos dados de saída dos modelos: o sobreajuste (*Overfitting*) e o subajuste (*Underfitting*).

2.3.1.1 Sobreajuste (*Overfitting*)

Quando se fala em sobreajuste, a ideia é de que o modelo proposto está generalizando mais do que deveria os dados de treinamento. Isso se torna perceptível quando tais dados apresentam elevada precisão nos treinos, mas isso acaba não se refletindo nos conjuntos de teste e, por isso, costuma-se dizer que, quando ocorre o *Overfitting*, o algoritmo utilizado decorou ou tornou-se especializado nos dados de treino. O *Overfitting* é comumente associado a um modelo muito complexo atrelado a um conjunto de dados de complexidade inferior ou com pouca riqueza de dados. Por isso, as possíveis soluções para modelos com sobreajuste envolvem a utilização de modelos mais simples (ou a redução dos parâmetros utilizados no modelo sobreajustado), a obtenção de mais dados para treino ou até mesmo a redução de ruídos nos dados de entrada via preprocessamento (FACELI, LORENA, *et al.*, 2011; GÉRON, 2017).

2.3.1.2 Subajuste (*Underfitting*)

Por outro lado, em se tratando de subajuste, o conceito é que o algoritmo analisado não é poderoso o suficiente na generalização dos dados de treinamento. Isso acaba gerando uma baixa precisão nos dados de treino e, por consequência, também nos dados de teste. Portanto, para contornar problemas de *Underfitting*, é comum que sejam tomadas estratégias de aumentar o número de parâmetros do modelo (ou até mesmo utilizar um modelo mais complexo), substituir as características que estão sendo analisadas para outras mais relevantes ou até mesmo diminuir as restrições impostas no modelo (GÉRON, 2017).

Nessa perspectiva, torna-se notória a importância de uma análise prévia dos dados, bem como o preprocessamento cauteloso (quando aplicável) deles. Feito isso, é possível seguir para o treinamento dos modelos e, caso seja necessário, retorna-se à etapa de preprocessamento e análise dos dados de entrada. Nesse trabalho, foram utilizados alguns mecanismos para redução de *Overfitting* e *Underfitting*. Eles serão discutidos de forma mais aprofundada no capítulo 3.

2.3.2 Estudo dos Modelos

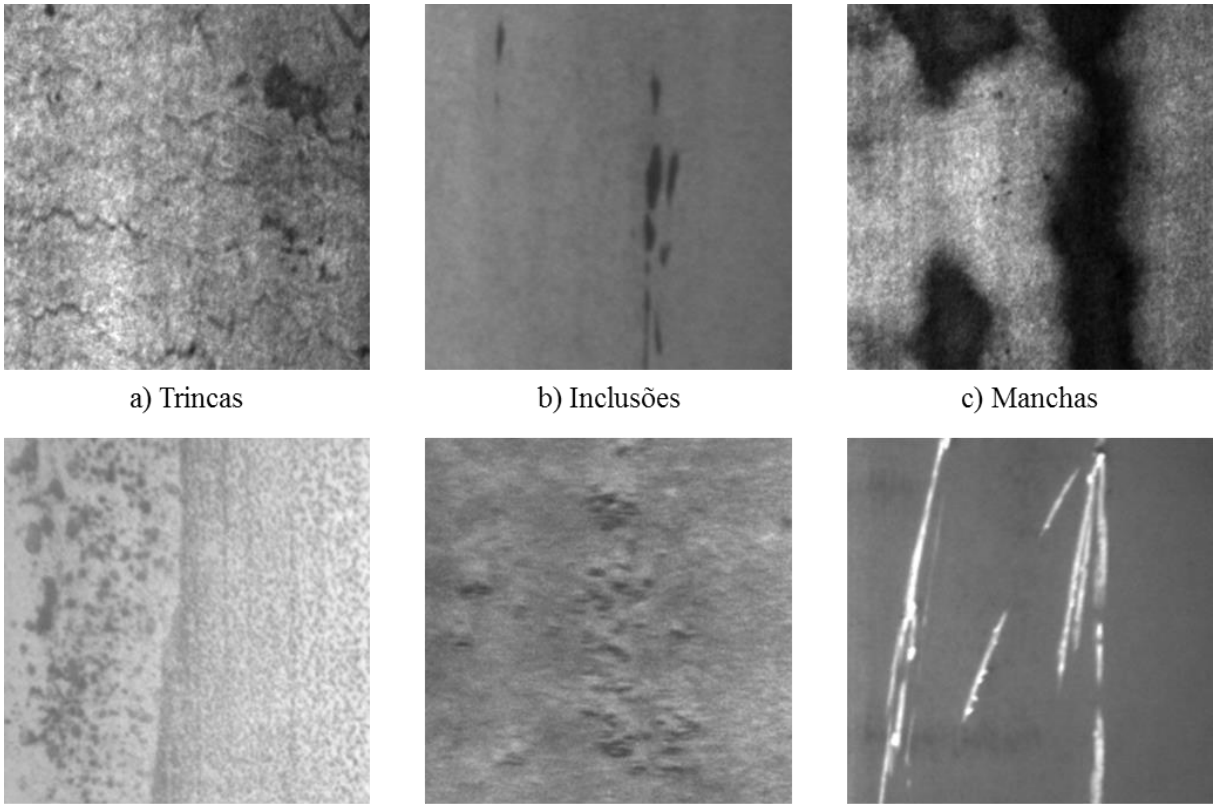
Tendo os dados tratados da forma correta, a próxima etapa consiste em estudar os modelos que serão empregados para fazer as predições desejadas. Existem duas formas principais de aprendizado, que estão correlacionadas com o tipo de supervisão que os dados recebem durante o treinamento: os aprendizados supervisionados e os não supervisionados (GÉRON, 2017).

Nos aprendizados supervisionados, os dados inseridos no algoritmo já estão rotulados, ou seja, eles possuem uma pré-classificação que é utilizada para "ensinar" o modelo a realizar as previsões. Já nos aprendizados não supervisionados, os dados inseridos não vêm rotulados e o algoritmo irá "aprender" e rotular os dados sem uma classificação prévia fornecida (GÉRON, 2017).

No presente trabalho, será utilizada apenas a forma supervisionada de aprendizado. Isso se dará dessa forma pois o banco de dados utilizado, extraído de uma plataforma da *Northeastern University* (NEU), já está rotulado: são 1800 imagens no formato .bmp, divididas igualmente em 6 tipos de defeitos encontrados em placas de aço: fissuras, inclusões, manchas, superfície com pite, carepa incrustada e riscos. Cada uma das imagens tem tamanho de 200x200 pixels e

40,1 kB (SONG e YAN, 2020). Os rótulos estão indicados no nome do arquivo .bmp e não na imagem propriamente dita. Abaixo seguem alguns exemplos das imagens presentes no banco de dados da NEU:

Figura 13 - Defeitos superficiais em placas de aço



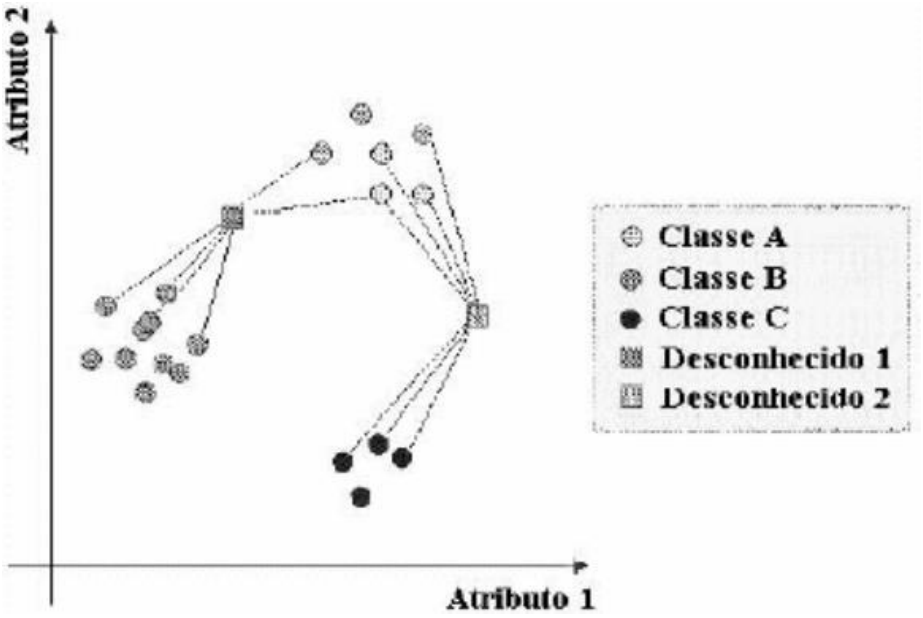
Fonte: (SONG e YAN, 2020)

2.3.2.1 KNN

O primeiro modelo a ser estudado é comumente denominado KNN. Essa sigla advém do inglês para os K-vizinhos mais próximos (*K-Nearest Neighbors*). A nomenclatura KNN está diretamente relacionada com o *modus operandi* do modelo: de forma simplificada, quando se trata de aprendizado supervisionado, o algoritmo do KNN toma a amostra que se deseja classificar e compara a distância dessa amostra com relação aos pontos cujos rótulos já são conhecidos (dados de treino). A classificação atribuída à amostra será a que apresentar maior frequência entre os K-vizinhos mais próximos (PUC-RIO, 2020a; SCIKIT-LEARN, 2020a).

A Figura 14 ilustra como é feita essa classificação: nesse exemplo, tomou-se $K = 7$. Desta forma, a amostra desconhecida 1 apresenta 5 vizinhos mais próximos de classe B e 2 de classe A; portanto, quando $K = 7$, a amostra 1 será classificada como pertencente à classe B. Seguindo-se a mesma linha de raciocínio, é possível inferir que a amostra desconhecida 2 será atribuída à classe A.

Figura 14 - Exemplo de classificação pelo modelo KNN



Fonte: (PUC-RIO, 2020a)

2.3.2.1.1 Distância entre K-vizinhos

A distância entre a amostra e os K-vizinhos pode ser calculada de diversas formas diferentes, que variam de acordo com a métrica escolhida. As métricas mais utilizadas são as distâncias Euclidiana, Manhattan e Minkowski. Nessa perspectiva, tomando-se os pontos: $(X, Y) \in \mathbb{R}^n \mid X = (x_1, x_2, \dots, x_n)\,; Y = (y_1, y_2, \dots, y_n)$, definem-se essas métricas de acordo com as equações (X), (XI) e (XII) sucessivamente:

$$d_{Euclidiana}(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \qquad (X)$$

$$d_{Manhattan}(X,Y) = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n| \qquad (XI)$$

$$d_{Minkowski}(X,Y) = (|x_1 - y_1|^q + |x_2 - y_2|^q + \cdots + |x_n - y_n|^q)^{\frac{1}{q}}, q \in \mathbb{N} \qquad (XII)$$

(PUC-RIO, 2020a).

A essas métricas de distâncias, é possível também a atribuição de pesos aos dados: ao definir-se o parâmetro de peso como “uniforme”, isso significa que a mesma importância será dada a todos os K-vizinhos da amostra. Por outro lado, se tal parâmetro for definido como “distância”, isso significa que haverá pesos inversamente proporcionais à distância dos K-vizinhos, o que significa que os pontos mais próximos à amostra terão maior importância no processo de classificação do modelo (SCIKIT-LEARN, 2020a).

2.3.2.1.2 Utilização de algoritmos

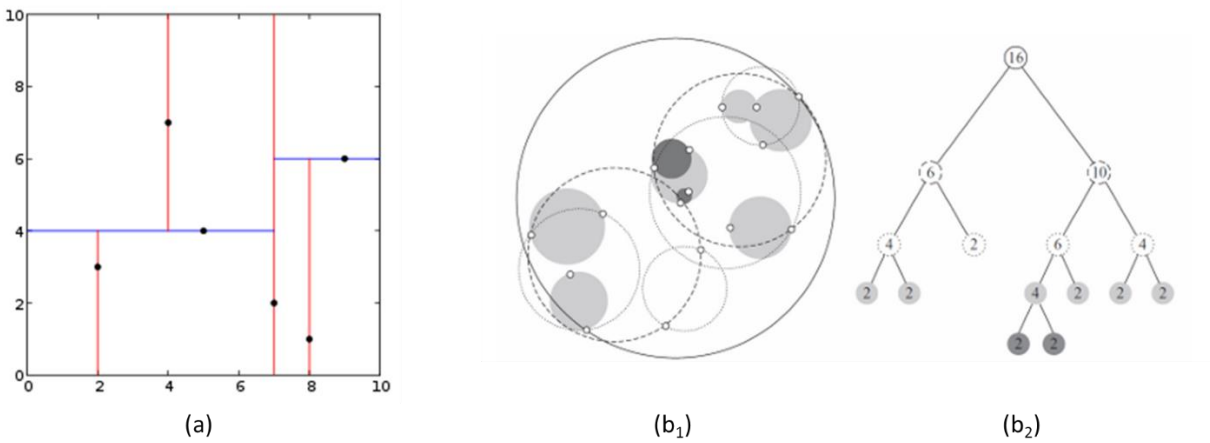
Por padrão, o modelo KNN utiliza na classificação um algoritmo denominado “força bruta”, que basicamente classifica a amostra da forma que foi explicado na Figura 14, ou seja, computa as distâncias, uma a uma, entre a amostra e os vizinhos para inferir acerca da classificação dessa amostra. Esse mecanismo, por outro lado, exige um custo de processamento deveras elevado à medida em que se eleva o número de dados envolvidos no cálculo. Nessa perspectiva, existem dois outros algoritmos que são comumente utilizados para otimizar tal custo de processamento: *K-D tree* e *Ball Tree* (SCIKIT-LEARN, 2020a).

De forma resumida, o algoritmo *K-D tree* consegue economizar tempo de processamento ao comparar distâncias já calculadas entre pontos para decidir se é necessário ou não efetuar novos cálculos. Portanto, se por exemplo um ponto A está muito distante de um ponto B, que por sua vez se encontra perto do ponto C, o algoritmo consegue entender, sem necessidade de cálculos, que o ponto A é também distante do ponto C, gerando uma redução no custo computacional que, de acordo com SCIKIT-LEARN (2020a) , é significativa para um elevado número de dados. O autor afirma, no entanto, que esse algoritmo perde eficiência quando se aumenta o número de dimensões do problema analisado devido ao elevado espalhamento entre os dados. Nesse contexto, emerge o outro algoritmo mencionado anteriormente: *Ball Tree*.

Enquanto no *K-D tree* o algoritmo é particionado ao longo dos eixos cartesianos (Figura 15 - a), no *Ball Tree* (Figura 15-b), essa partição é feita em hiperesferas, com nós definidos por um centroide C e raio r. Desta forma, cada ponto de um determinado nó (Figura 15-b_2) se encontra dentro da hiperesfera definida por C e r (Figura 15 – b_1). Essa configuração permite que o cálculo da distância entre a amostra e C seja suficiente para determinar os limites superiores e

inferiores de distância para todos os pontos pertencentes ao nó (Figura 15 – b_1) (SCIKIT-LEARN, 2020a).

Figura 15 - Partição dos nós nos algoritmos: (a) - K-D Trees; (b) - Ball Trees



Fonte: (WITTEN, FRANK, *et al.*, 2016)

2.3.2.2 SVM

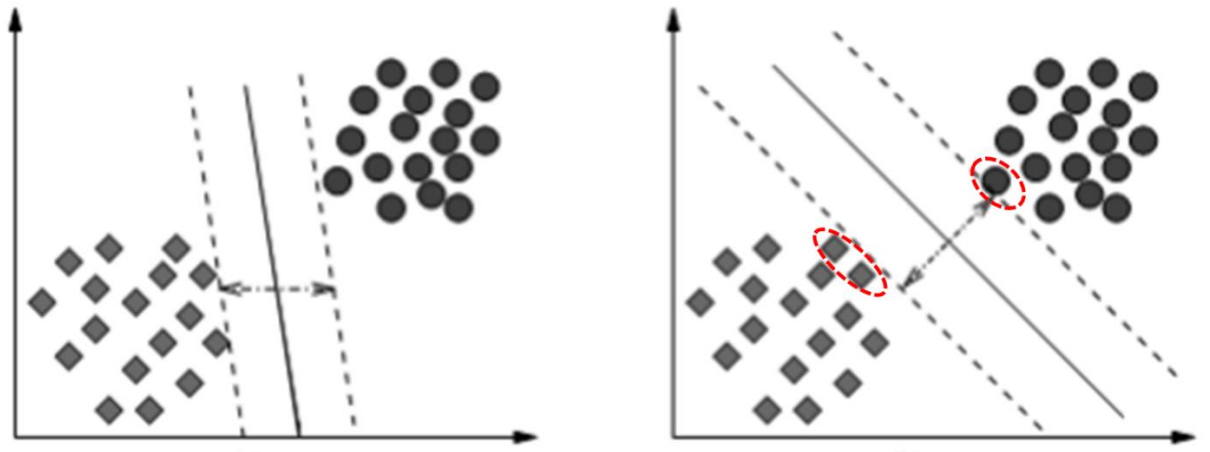
As *Support Vector Machines* (SVM) ou Máquinas de Vetores Suporte (MVS) consistem num método de classificação[5] que se baseia na Teoria de Aprendizado Estatístico, propondo mecanismos de ML que visam otimizar o potencial de generalização e o princípio de Minimização do Risco Estrutural. Em linhas gerais, melhorar o potencial de generalização significa tornar mais eficiente a classificação em face dos dados de treino; a Minimização do Risco Estrutural, por sua vez, refere-se à possibilidade de classificação incorreta de amostras ainda não introduzidas à máquina (TAKAHASHI, 2012).

Uma SVM funciona, simplificadamente, da seguinte forma: supondo-se a existência de dois rótulos de classificação e um dado arranjo de pontos pertencentes a essas classes, uma SVM tem a função de estabelecer o hiperplano que separa tais pontos de maneira a isolar o máximo possível de pontos rótulos iguais de um mesmo lado, maximizando também a distância de cada uma das classes até o hiperplano. Os pontos que determinam o hiperplano gerados pelas MVS são denominados vetores suporte (PUC-RIO, 2020b). A Figura 16 ilustra os processos citados anteriormente para estabelecimento de um hiperplano: na parte a) é exemplificado um

[5] pode ser também utilizado para regressão, mas está fora do escopo desse trabalho.

hiperplano com margem estreita (não otimizada); na parte b), a margem é otimizada e os vetores suporte se encontram destacados em vermelho.

Figura 16 - Hiperplanos nas SVMs



Fonte: Adaptado de (OLIVEIRA JUNIOR, 2010)

De acordo com os tipos de dados que estão sendo trabalhados, as SVMs podem ser classificadas em lineares (de margens rígidas ou suaves) e não lineares.

2.3.2.2.1 SVMs Lineares

Seja um conjunto de dados de treino com n pontos na forma $(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n)$, sendo y_i igual a ± 1 a depender da classe à qual pertence o ponto $\boldsymbol{x}_i$. De acordo com o explicado anteriormente, deseja-se encontrar o hiperplano com margem máxima que divide o grupo de pontos $\boldsymbol{x}_i$ para os quais $y_i = 1$ daqueles nos quais $y_i = -1$. É possível definir os hiperplanos como o conjunto de pontos $\boldsymbol{x}$ que satisfaz à condição:

$$\boldsymbol{w} \,.\, \boldsymbol{x} - b = 0 \qquad (XIII)$$

Sendo $\boldsymbol{w}$ um vetor normal ao hiperplano. Desta forma, o parâmetro $\frac{b}{||\boldsymbol{w}||}$ determina o deslocamento do hiperplano a partir da origem ao longo do vetor normal $\boldsymbol{w}$ (BISHOP, 2006; FACELI, LORENA, *et al.*, 2011).

Agora, considerando-se o conjunto de treino como sendo linearmente separável, é possível escolher dois hiperplanos paralelos que separam as duas classes de dados de tal maneira que a

distância entre elas seja tão ampla quanto possível. A região delimitada por esses dois hiperplanos é denominada "margem" e, nesse caso, o hiperplano com margem máxima é aquele que está equidistante a esses dois hiperplanos. Considerando-se dados normalizados, esses hiperplanos podem ser descritos pelas equações (XIV), na qual os dados acima ou sobre essa borda pertence à classe com rótulo 1, e (XV), na qual os dados sobre ou abaixo da borda pertence à classe com rótulo -1:

$$\boldsymbol{w}.\boldsymbol{x} - b = 1 \qquad (XIV)$$

$$\boldsymbol{w}.\boldsymbol{x} - b = -1 \qquad (XV)$$

(BISHOP, 2006; FACELI, LORENA, *et al.*, 2011).

Esses hiperplanos distam um do outro por um valor de $\frac{2}{||\boldsymbol{w}||}$ (valor correspondente à distância indicada na Figura 16 - b), ou seja, para maximizar a distância entre eles, o objetivo é minimizar $\left|\left|\boldsymbol{w}\right|\right|$. Portanto, calculando-se a distância dos pontos ao plano, conclui-se que o objetivo é minimizar os valores de $\left|\left|\boldsymbol{w}\right|\right|$ sujeitos a:

$$y_i(\boldsymbol{w}.\boldsymbol{x}_i - b) \geq 1, \qquad \forall\, 1 \leq i \leq n \qquad (XVI)$$

(BISHOP, 2006; FACELI, LORENA, *et al.*, 2011).

Logo, os valores de $\boldsymbol{w}$ e b que solucionam (XVI) minimizando $\left|\left|\boldsymbol{w}\right|\right|$ determinam o classificador:

$$\boldsymbol{x} \mapsto sgn(\boldsymbol{w}.\boldsymbol{b} - b) \qquad (XVII)$$

Sendo $sgn(.)$ a função sinal. A consequência geométrica mais importante das descrições acima é que o hiperplano com margem máxima pode ser determinado pelos valores de $\boldsymbol{x}_i$ mais próximos a ele, ou seja, os vetores de suporte (BISHOP, 2006; FACELI, LORENA, *et al.*, 2011).

O problema das SVMs lineares descrito acima, conforme já citado, trata com dados linearmente separáveis, e por isso, as margens são denominadas "rígidas". Caso o problema sob análise envolva dados não linearmente separáveis, as margens não conseguirão separar perfeitamente os dados de diferentes rótulos e então será necessária a introdução da função denominada "perda de articulação" ou *hinge loss*[6]:

[6] Para maiores informações, consultar BISHOP (2006) e FACELI *et al* (2011).

$$\max\left(0{,}1 - y_i(\boldsymbol{w}.\boldsymbol{x_i} - b)\right) \qquad (XVIII)$$

Por meio da qual é possível encontrar a nova equação a ser minimizada:

$$\left[\left(\frac{1}{n}\right)\sum_{i=1}^{n}\max\left(0{,}1 - y_i(\boldsymbol{w}.\boldsymbol{x_i} - b)\right)\right] + \lambda\left|\left|\boldsymbol{w}\right|\right|^2 \qquad (XIX)$$

Sendo λ o parâmetro relacionado à escolha entre o aumento do tamanho da margem e a garantia de que o ponto $\boldsymbol{x_i}$ esteja do lado correto da margem. Isso significa que, para pequenos valores de λ, o segundo termo na função de perda se torna ínfimo e o problema se reduz ao de margens rígidas (BISHOP, 2006; FACELI, LORENA, *et al.*, 2011).

2.3.2.2.2 SVMs não lineares

Para finalizar as MVS, restam as classificações não lineares, que são obtidas por meio da aplicação de funções kernel às formulações anteriores[7]. Esses kernels são aplicados onde existem os produtos escalares e, ao fim, possibilitam que o algoritmo consiga ajustar o hiperplano de margem máxima a um espaço de características transformado. Essa transformação pode ser não-linear e as dimensões do novo espaço podem ser maiores, o que pode aumentar o erro de generalização das SVMs, que, todavia, continuam sendo satisfatórias se o número de amostras for grande o suficiente. Na *Tabela 4* são apresentados os tipos mais comuns de kernel que são utilizados (RBF – *Radial Basis Function* ou, em português, função de base radial).

Tabela 4 - Tipos mais comuns de Kernel

Tipo de Kernel	Equação
Polinomial	$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i}.\boldsymbol{x_j})^d$
RBF	$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\gamma \parallel \boldsymbol{x_i} - \boldsymbol{x_j} \parallel^2)$

Fonte: o autor

Nas equações da tabela, d é o grau do polinômio a ser analisado e $\gamma > 0$ é um parâmetro livre (BISHOP, 2006; FACELI, LORENA, *et al.*, 2011).

[7] Para maiores informações, consultar BISHOP (2006) e FACELI *et al* (2011).

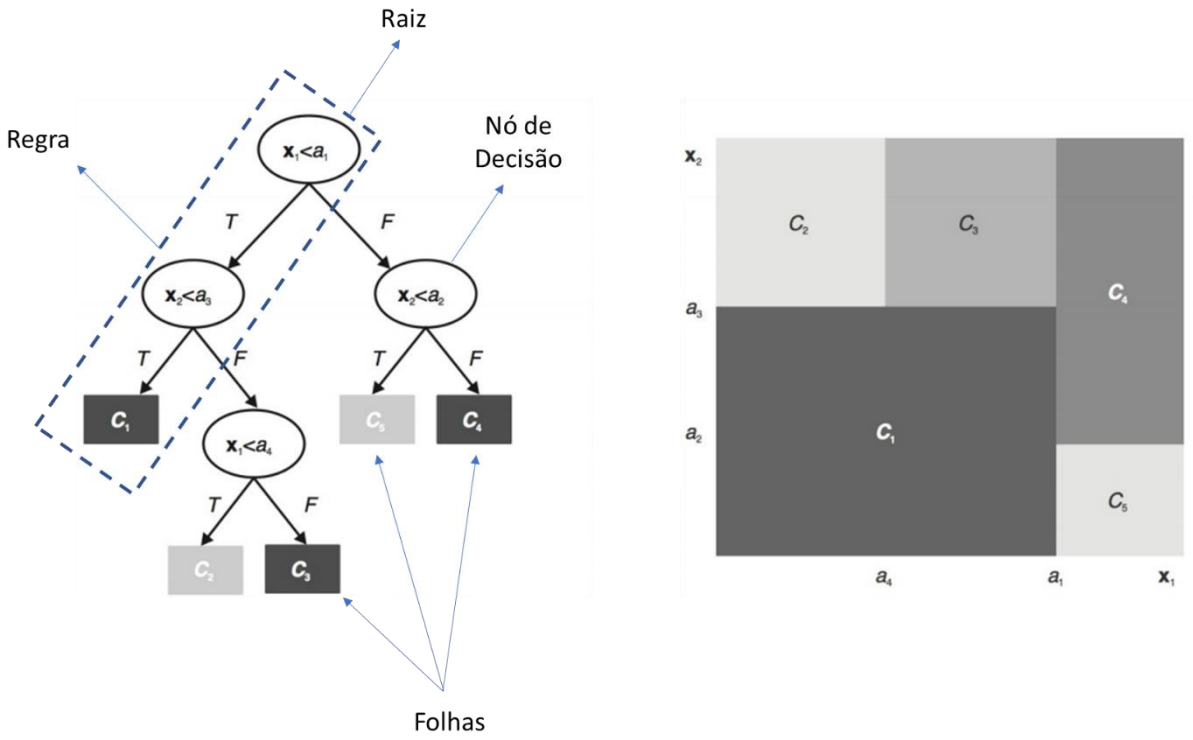
Para solucionar os problemas de minimização mencionados nesta seção, os algoritmos costumam valer-se da formulação lagrangeana. Todavia, esse nível de detalhe está fora do escopo deste trabalho, mas pode ser visto em BISHOP (2006) e FACELI *et al* (2011).

2.3.2.3 Árvores de Decisão

As árvores de decisão consistem em um modelo de classificação conhecido por se utilizar da técnica de dividir para conquistar durante o processo de decisão. Basicamente, um problema de complexidade elevada é dissolvido em situações menos complexas e isso é feito de forma iterada. No final, as resoluções encontradas para as subdivisões dos problemas se juntam para fornecerem uma resposta ao problema inicial (FACELI, LORENA, *et al.*, 2011).

Na Figura 17, está representada, à esquerda, uma árvore de decisão e, à direita, sua representação no espaço de objetos. Nessa figura, os nós de decisão contêm validações para características que estão sendo testadas: na raiz (primeiro dos nós), por exemplo, o teste genérico que está sendo feito é se $x_1 < a_1$, sendo x_1 e a_1 atributos de teste quaisquer. Na sequência, os ramos descendentes equivalem a valores prováveis para esses atributos. Por fim, as folhas estão associadas aos rótulos e o caminho que vai da raiz até a folha é uma regra de classificação (PUC-RIO, 2020c).

Figura 17 - Árvores de decisão e sua representação no espaço de objetos



Fonte: Adaptado de (FACELI, LORENA, *et al.*, 2011)

Para serem feitas as divisões observadas na Figura 17, são implementados critérios que estão relacionados a ganhos de informação por atributos. Para as árvores de decisão, os critérios mais utilizados são a Entropia e o índice Gini (PUC-RIO, 2020c). Na seção 4.3.3 é apresentado um bom exemplo de árvores de decisão aplicadas aos dados de treino deste trabalho.

2.3.2.3.1 Entropia

De forma simplificada, a entropia diz respeito à impureza dos dados, ou seja, é uma maneira de se medir, por meio do ganho de informação, a ausência de homogeneidade nas informações de entrada em comparação aos rótulos recebidos ao final. Desta forma, a entropia é mínima quando o conjunto de dados é homogêneo e vice-versa; portanto, o foco de uma árvore de decisão munida desse tipo de critério é a diminuição da entropia. Matematicamente, a entropia é descrita pela equação (XX), na qual S denota o conjunto de entrada; c são as diferentes classes desse conjunto e p_i é a parcela de dados em S pertencentes à classe i:

$$Entropia(S) = \sum_{i=1}^{c} -\log_2 p_i \qquad (XX)$$

(PUC-RIO, 2020c).

Por fim, a medida do ganho de informação relacionado à medida de entropia é dada pela equação $(XXII)$:

$$E(A) = \sum_{x \in P(A)} \frac{|S_x|}{|S|} Entropia(S_x) \qquad (XXI)$$

$$ganho(S, A) = Entropia(S) - E(A) \qquad (XXII)$$

Sendo A um atributo qualquer pertencente ao conjunto S; $P(A)$ o conjunto dos possíveis valores que A possa ter e $S_x \subset S$, sendo S_x composto pelos valores de A tais que $A = x$ (PUC-RIO, 2020c).

2.3.2.3.2 Índice Gini

Esse tipo de critério mensura o nível de heterogeneidade dos dados, sendo comumente empregado na medida de impurezas em um determinado nó. Desta forma, analisando-se um nó qualquer pertencente à árvore, tem-se o índice Gini dado pela equação $(XXIII)$:

$$Índice\ Gini = 1 - \sum_{i=1}^{c} p_i^2 \qquad (XXIII)$$

Sendo c, novamente, o conjunto das classes e p_i a medida da frequência de uma dada classe em cada um dos nós (PUC-RIO, 2020c).

Analisando-se a equação $(XXIII)$, é possível interpretar o significado desse índice: à medida em que ele se aproxima de 0, aumenta-se o grau de pureza do nó e, à medida em que se aproxima de 1, aumenta-se a impureza devido ao elevado número de classes distribuídas de forma uniforme ao longo do nó sob análise (PUC-RIO, 2020c).

2.3.2.4 Florestas Aleatórias

A partir deste momento, serão comentados os modelos denominados *ensemble*, começando pelas florestas aleatórias. A palavra *ensemble* vem do inglês, podendo ser traduzida como "conjunto" ou "grupo". Essa tradução por si só traz informações a respeito do funcionamento de mecanismos desse tipo: são algoritmos que compilam predições de um conjunto de outros classificadores de forma a obter uma predição mais apurada do que os outros preditores utilizados de forma individual (GÉRON, 2017).

Nesse sentido, emergem as florestas aleatórias: basicamente, esse modelo de classificação é um arranjo de árvores de decisão (por isso o nome "floresta"), utilizando para essas árvores subconjuntos aleatórios dos dados de treino. As previsões finais são então obtidas das individuais de cada árvore, escolhendo o rótulo que apareceu mais vezes (GÉRON, 2017).

As florestas aleatórias, diferentemente das árvores de decisão, escolhem características aleatórias na divisão dos nós durante a criação de suas árvores individuais. Isso faz com que a diversificação (e também o número de erros de predição) entre as árvores sejam maiores, o que não é um problema, uma vez que o elevado número de amostras garante que o resultado final tenha uma boa acurácia (GÉRON, 2017).

2.3.2.5 AdaBoost

O *Adaboost*, abreviação de *Adaptive Boosting* (algo como "impulso adaptável" em português), pertence a um conjunto de métodos *ensemble* denominados *Boosting*. Esses métodos atuam combinando diversos classificadores fracos para formar um que seja mais robusto. O conceito por detrás desses modelos é treinar classificadores em sequência, cada um deles tentando corrigir seu predecessor. Dentre os modelos de *Boosting*, os mais conhecidos são os que serão vistos nessa seção e na próxima (*Gradient Boosting*) (GÉRON, 2017).

De forma resumida, esse algoritmo funciona da seguinte maneira: dado um conjunto de vetores de treinamento $(\boldsymbol{x_t}, y_t)$, geram-se, em cada rodada de aprendizagem, hipóteses h_t baseadas em cada uma das amostras. Essas hipóteses buscam garantir, em sucessivas iterações, a classificação correta das amostras atreladas aos maiores pesos D_t, buscando para isso a minimização do erro de treinamento:

$$e_t = \sum_{i:h_t(\boldsymbol{x_t}) \neq y_t} D_t(i) \qquad (XXIV)$$

(CHAVES, 2012).

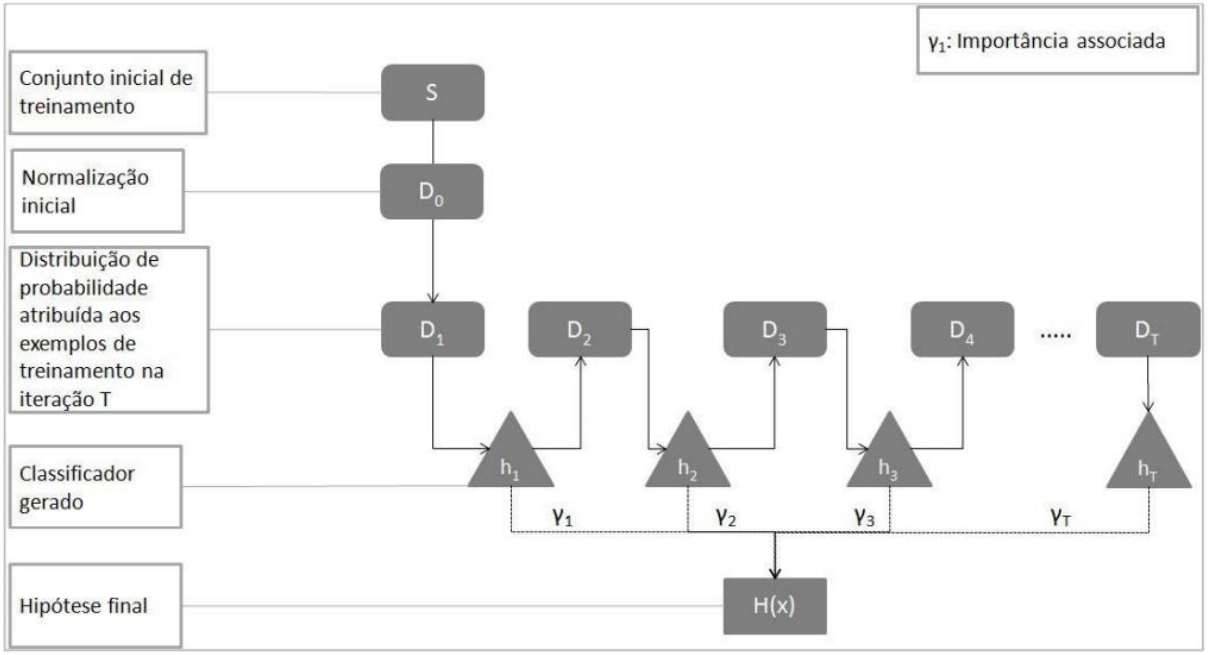
Em cada uma das rodadas, os pesos são redefinidos, tornando maiores aqueles que dizem respeito aos dados rotulados de forma incorreta. Ao final das T interações, o algoritmo combina as hipóteses testadas para formular uma hipótese final mais generalizada e robusta:

$$H(x) = sign\left(\sum_{t=1}^{T} \gamma_t h_t(x)\right) \qquad (XXV)$$

Sendo γ_t importância atrelada à h_t (CHAVES, 2012).

A Figura 18 sintetiza o funcionamento do algoritmo do *Adaptive Boosting*, indo desde o conjunto inicial de treinamento, S, até a hipótese final, $H(x)$.

Figura 18 - esquema de funcionamento do AdaBoost



Fonte: (CHAVES, 2012)

2.3.2.6 Gradient Boosting

Conforme já mencionado na seção 2.3.2.5, o *Gradient Boosting* é um método *ensemble* que agrupa classificadores menos robustos para gerar um que seja mais eficiente. Tipicamente, o *Gradient Boosting* combina árvores de decisão, que atuam de forma iterada corrigindo suas

predecessoras por meio da análise dos erros residuais cometidos por elas (diferentemente do *AdaBoost*, que utiliza uma sistemática de atribuição de pesos). Para isso, o método se inicia com uma previsão constante, $f_0(\boldsymbol{x}) = \bar{y}$, correspondente à média das respostas nos dados de treino. A cada rodada, são inseridos termos adicionais, que objetivam a redução dos erros de previsão. Isso faz com que os cálculos das novas iterações ocorram em sentido inverso com relação ao gradiente da função objetivo, $\Psi\big(y_i, f(\boldsymbol{x_i})\big)$, no tocante às estimativas correntes, $f(\boldsymbol{xi})$. A iteração segue até atingir um critério de parada, como por exemplo um limite M de iterações, conforme descreve a equação $(XXVI)$:

$$\hat{y}_i^M = \sum_{m=0}^{M} f_m(\boldsymbol{x}) = \hat{y}_i^{M-1} + \eta f_M(\boldsymbol{x}) \qquad (XXVI)$$

Sendo $\hat{y}_i^m$ a resposta estimada para a i-ésima análise dos dados de treinamento, após m iterações do algoritmo. η é denominado *shrinkage* ou passo de aprendizado e é responsável por controlar o quão rápido o algoritmo converge (MAYRINK, 2016).

Assim, a cada rodada, conforme mostra a equação $(XXVII)$, o algoritmo treina as árvores de decisões de tal forma que elas possam utilizar as previsões do modelo mais recente para regular os gradientes da função objetivo.

$$r_i^m = -\frac{\partial}{\partial \mathrm{f}(\mathbf{x_i})} \Psi\big(y_i, f(\boldsymbol{x_i})\big)\Big|_{\boldsymbol{f}(x_i)=\hat{f}_m(\boldsymbol{x}_i)} \qquad (XXVII)$$

Sendo r_i^m a m-ésima observação dos dados de treino (MAYRINK, 2016).

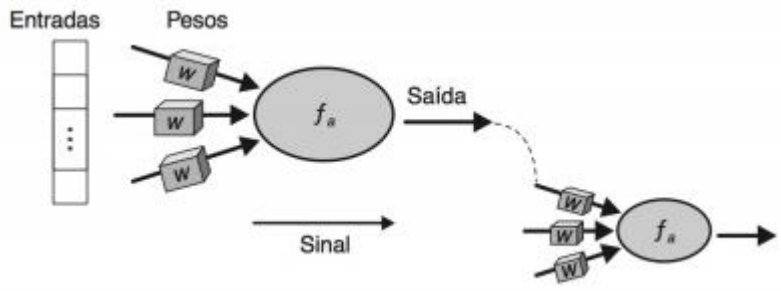
2.3.2.7 Análise direta: as Redes Neurais Convolucionais

Conforme dito anteriormente, todos os métodos discutidos até aqui abordam métodos indiretos de classificação, ou seja, na metodologia do trabalho, as características de textura discutidas na seção 2.2.4 serão extraídas das imagens para posterior análise de dados e classificação. Por outro lado, com as redes neurais convolucionais, a etapa de extração de características não será utilizada: esse algoritmo irá analisar as imagens de forma direta, buscando entender os padrões dos níveis de cinza encontrados para cada um dos tipos de defeito para então inferir conclusões, ou seja, rotular, os dados no conjunto de validação.

As redes neurais artificiais têm esse nome pois espelham o comportamento do sistema nervoso de um ser humano, sendo compostas por extensos conjuntos de neurônios artificiais que formam interconexões entre si. Essas conexões executam cálculos sobre funções matemáticas de maneira similar ao que é feito nas sinapses em seres vivos sendo que, nesse processo, são atribuídos pesos às conexões de tal maneira a controlar o comportamento de cada neurônio dentro da rede. Durante o aprendizado, esses pesos são adequados em três tipos de comportamentos: excitatório (para pesos positivos), inibitórios (para pesos negativos) e inativos (quando o peso é nulo) (FACELI, LORENA, *et al.*, 2011).

Na Figura 19 é apresentado um esquema de funcionamento para um neurônio artificial dentro da rede. Note que, aos dados de entrada, são atribuídos pesos w; então, eles passam por uma função de ativação f_a, sendo que a saída medida no neurônio é consequência da resposta de f_a aos dados de entrada.

Figura 19 - Funcionamento de um neurônio artificial



Fonte: (FACELI, LORENA, *et al.*, 2011)

Matematicamente, sendo $\boldsymbol{x}$ um vetor composto por d características ($\boldsymbol{x} = [x_1, x_2, \dots, x_d]$) e $\boldsymbol{w}$ um vetor com d pesos associados às d entradas ($\boldsymbol{w} = [w_1, w_2, \dots, w_d]$), então, a entrada em um determinado neurônio pode ser representada pela seguinte equação:

$$u(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=1}^{d} x_j w_j \qquad (XXVIII)$$

A função de ativação, f_a, como dito anteriormente, é responsável por regular a saída do neurônio. Existem diversas possibilidades de comportamentos para essa função, mas, para esse trabalho, será utilizada a função conhecida como ReLU (*Rectified Linear Units*) (FACELI, LORENA, *et al.*, 2011).

Partindo para as redes neurais convolucionais, pode-se definir as CNNs (*Convolutional Neural Networks*) como algoritmos de IA estruturados em redes multi-camadas que aprendem a

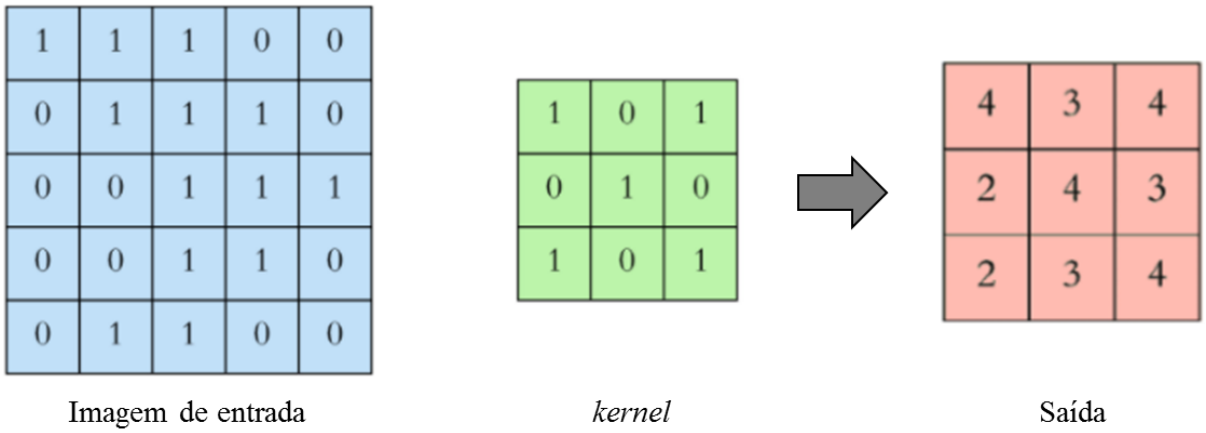
classificar imagens por meio de suas características relevantes. A ideia por detrás das CNNs fundamenta-se na capacidade cerebral de reconhecer objetos e pessoas partindo de aspectos básicos, tais como reconhecer cabeças e pernas de uma determinada pessoa e, a partir daí, estender o conceito para aspectos mais profundos. Sob o ponto de vista técnico, as CNNs trabalham com operações matemáticas denominadas convoluções, que dão nome ao modelo. Portanto, essas redes neurais são compostas pela parte convolucional (de extração de características) e pela rede neural tradicional, elucidada nos parágrafos anteriores (CLAPPIS, 2019).

O conceito moderno das CNNs advém de um trabalho de Yann LeCun, que data de 1998, no qual o autor propôs uma rede denominada LeNet para reconhecimento de manuscritos. Basicamente, uma rede neural convolucional é composta por 4 tipos de camadas: *Convolutional*, *Padding*, *Rectified Linear Units* (*Relu*) e *Pooling* (que traduzidas ao português ficam: Convolucional, de Preenchimento, Unidades Lineares Retificadas e de Agrupamento). Após essas camadas, são comumente empregadas outras duas: *Dropout* (“jogar fora”) e *Flatten* (“achatar” ou “nivelar”) para então partir-se para as redes tradicionais elucidadas nos parágrafos anteriores.

2.3.2.7.1 – Camada Convolucional

As camadas de convoluções são compostas por um conjunto de filtros (*Kernels*) que transformam as imagens de entrada por meio da aplicação de operações lineares que convertem duas funções em uma terceira (denominada mapa de características ou *feature map*). Os *Kernels*, por sua vez, são matrizes aplicadas de maneira iterada nas diversas localidades de uma determinada imagem, alterando esse local por um parâmetro denominado *stride* (CLAPPIS, 2019). A Figura 20 ilustra a aplicação de um *kernel* em uma imagem de entrada qualquer.

Figura 20 - exemplo de aplicação de *Kernel* em uma imagem



Fonte: (CLAPPIS, 2019)

Observe que, na transformação apresentada na Figura 20, o kernel, com coordenadas $b_{m,n}$, aplica uma transformação na imagem, com coordenadas $a_{i,j}$, gerando a matriz de saída com coordenadas c_{i_0,j_0} dadas por:

$$c_{i_0,j_0} = \sum_{\substack{i,j \\ m,n}} a_{i,j} b_{m,n} \qquad (XXIX)$$

Sendo que os índices (m,n) vão de $(1,1)$ à $(3,3)$ enquanto que (i,j) vai de (i_0,j_0) até (i_0+2, j_0+2) para cada (i_0,j_0) de c_{i_0,j_0}.

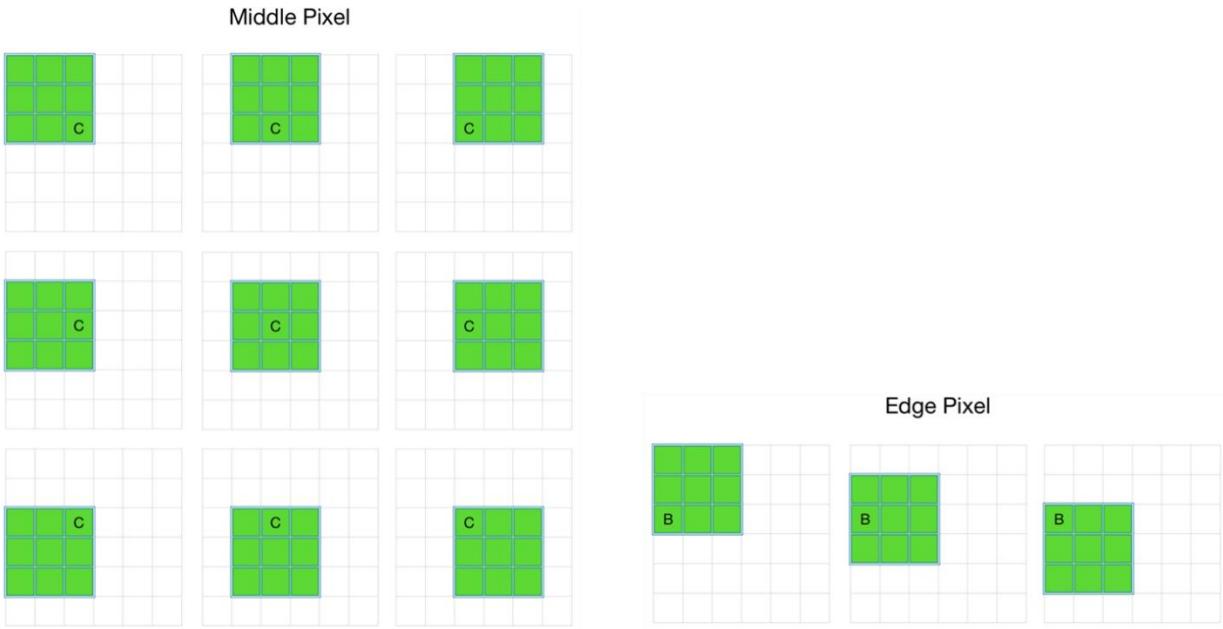
Ao final, a matriz resultante tem dimensões muito inferiores se comparadas à de origem, ou seja, as convoluções afetam diretamente a acurácia do modelo se a dimensionalidade da imagem for muito reduzida. Uma forma de contornar esse problema é a utilização dos *Paddings* (CLAPPIS, 2019).

2.3.2.7.2 – *Padding*

Padding ou camadas de preenchimento são mecanismos nos quais são inseridos pixels em volta da imagem, mantendo assim a dimensionalidade na matriz de saída durante a subsequente operação de convolução. Isso é feito porque, durante a aplicação de funções, como por exemplo a exemplificada na Figura 20, ocorrem perdas nas bordas da imagem (CLAPPIS, 2019).

Observe na Figura 21 que a função atua mais vezes nos índices centrais do que nos periféricos: por exemplo, no elemento $a_{1,1}$, a função passa apenas uma vez - durante o cálculo de $c_{1,1}$; por outro lado, no elemento central, $a_{3,3}$, a função passa 9 vezes durante os cálculos de $c_{1,1}, c_{1,2}, \dots, c_{3,2}, c_{3,3}$.

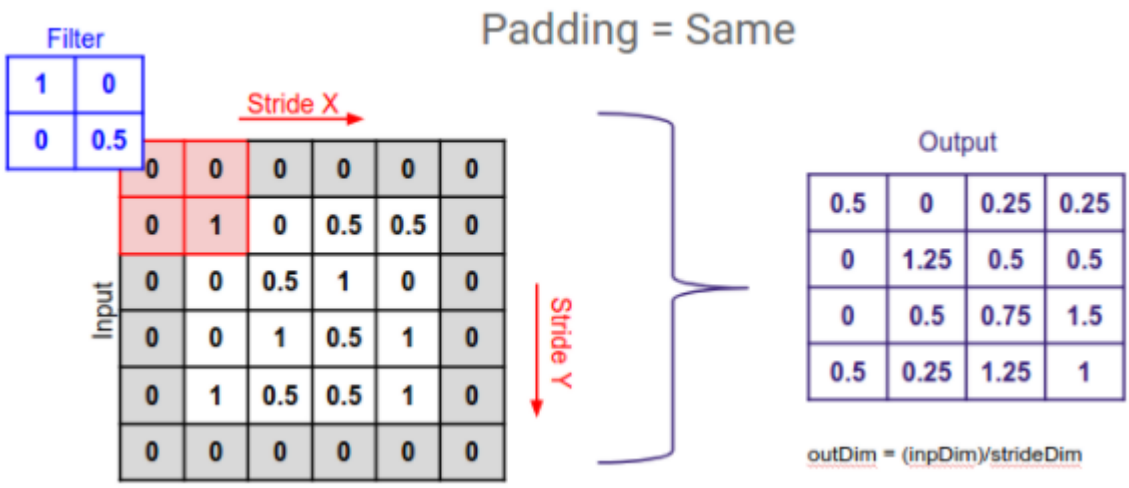
Figura 21 - Perdas de informações nas bordas das imagens



Fonte: (KHOSLA, 2019)

Sendo assim, para contornar os problemas mencionados, o processo de *Padding* propõe adicionar camadas de zeros ao redor das matrizes mencionadas, de tal maneira que, ao final, as convoluções passem pelos pixels dos cantos em frequências iguais às que atuam nos centrais (KHOSLA, 2019). Quando isso ocorre, garantindo às imagens de entrada e saída as mesmas dimensões, o processo de *Padding* é denominado "*Same*", conforme ilustra a Figura 22.

Figura 22 - Funcionamento do Padding Same



Fonte: (KHOSLA, 2019)

2.3.2.7.3 – *Relu* (*Rectified Linear Units*)

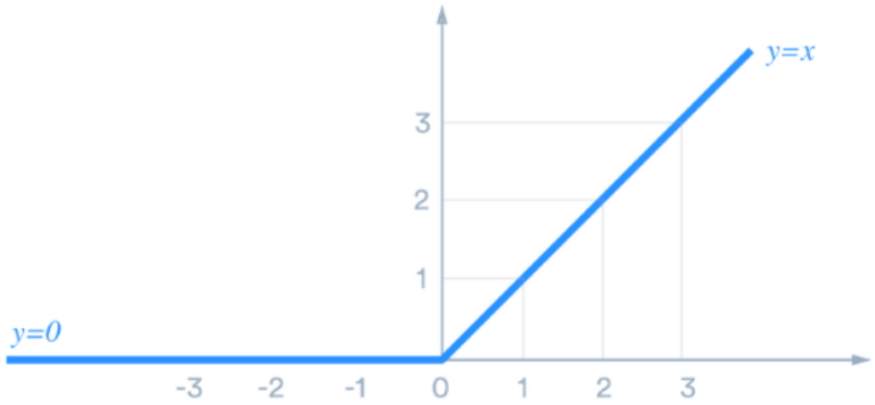
Conforme discutido no início da seção 2.3.2.7, nos modelos de redes neurais artificiais, existem as denominadas funções de ativação. Essas funções conferem às redes o caráter não-linear do problema e, para o caso de análise de imagens, a função mais utilizada é conhecida como *Relu*. As *Relu's* são definidas pela seguinte expressão:

$$y = \max(0, x) \qquad (XXX)$$

(CLAPPIS, 2019).

Graficamente, as Unidades Lineares Retificadas se comportam de acordo com o ilustrado na Figura 23:

Figura 23 - Gráfico da função Relu



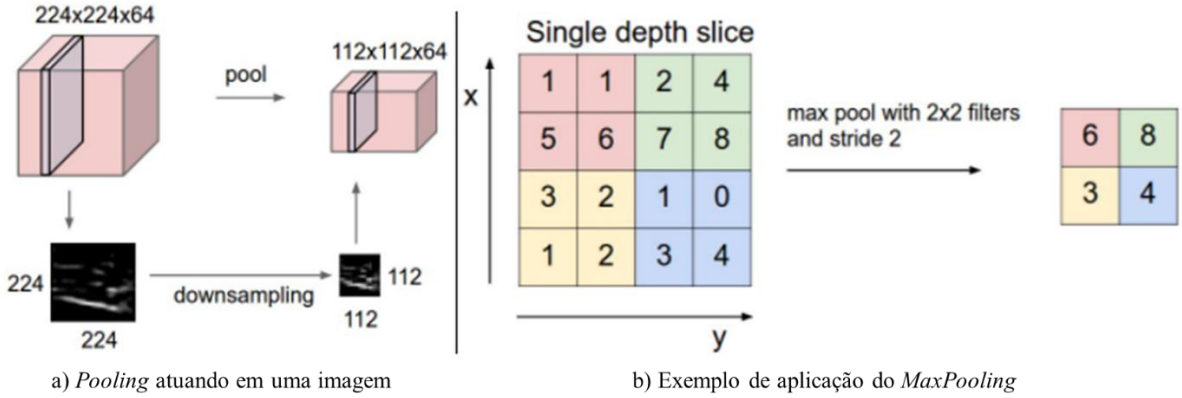
Fonte: (LIU, 2017)

As maiores vantagens desse tipo de função são o baixo custo computacional e a alta velocidade de convergência (LIU, 2017).

2.3.2.7.4 – *Pooling*

O processo de *Pooling* funciona, basicamente, como uma diminuição no tamanho da imagem que visa reduzir a variância do modelo sob pequenos ruídos, minorando também o número de variáveis a serem treinadas pelo algoritmo, conforme ilustra a Figura 24 (a). Existem três formas diferentes de serem feitas essas reduções: *MaxPooling*, *SumPooling* e *AveragePooling*. Nesse trabalho, será utilizado o *MaxPooling*, que opera de acordo com o ilustrado na Figura 24 (b) (CLAPPIS, 2019).

Figura 24 – Modelo de funcionamento do *Pooling*



a) *Pooling* atuando em uma imagem

b) Exemplo de aplicação do *MaxPooling*

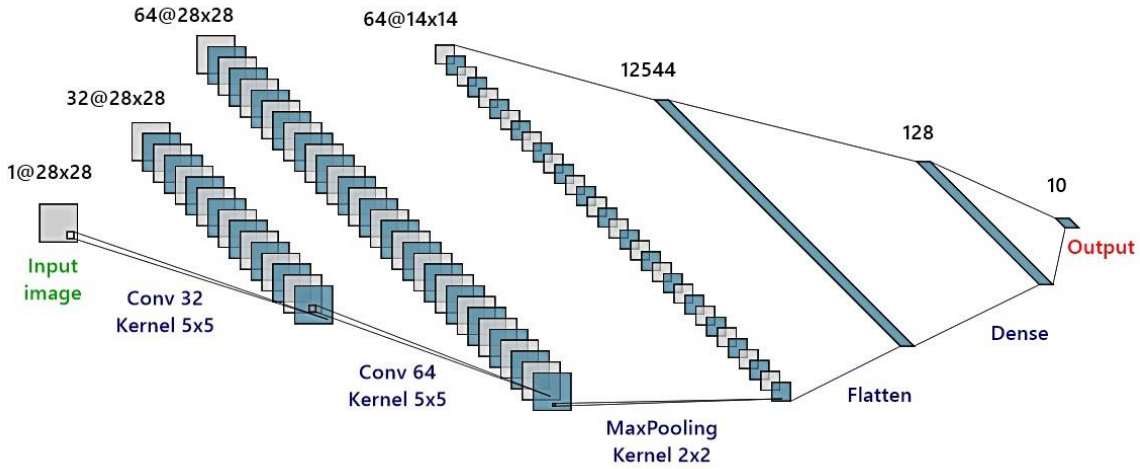
Fonte: (CLAPPIS, 2019)

Note que, no *MaxPooling*, o procedimento consiste em retirar o índice de maior valor para um determinado local da matriz, que depende das dimensões do *pool* (agrupador) aplicado (CLAPPIS, 2019). Na Figura 24 (b), o *pool* tem dimensões $2x2$ e, então, ele subdivide a matriz $4x4$ em 4 partes, gerando um dado de saída com dimensões também $2x2$.

Outras camadas comumente empregadas são:

- *Dropouts*: camadas utilizadas para balancear o peso das atribuições nas diversas localidades da rede, impedindo sensibilidade elevada a alterações de ordens menores:
- *Flatten*: camadas utilizadas para transformar a matriz nxn em um arranjo $n * n\ x\ 1$, facilitando o emprego das *dense layers*, que são as redes neurais tradicionais explicadas no início da seção 2.3.2.7 (CLAPPIS, 2019).

Figura 25 - Resumo das Camadas em uma CNN



Fonte: (CLAPPIS, 2019)

Note que, na imagem, ocorrem duas convoluções: na primeira, são geradas 32 novas imagens a partir da entrada e, na segunda, são geradas 64. Na sequência é aplicado o *MaxPooling*, que reduz o tamanho das imagens (seguindo as operações descritas em 2.3.2.7.4) de $28x28$ para $14x14$. A partir daí, as imagens são compiladas em um arranjo por meio da camada *Flatten* e então passam para a rede tradicional, a partir de onde emerge o *output* (CLAPPIS, 2019).

3 METODOLOGIA

De forma resumida, conforme mencionado anteriormente, o trabalho foi desenvolvido em duas frentes: a primeira delas por meio da extração de características de textura das imagens e consequente classificação e a segunda de maneira direta, com as análises feitas diretamente sobre os níveis de cinza das imagens de entrada.

Aqui, vale destacar as máquinas utilizadas durante a execução do projeto: para a maioria dos algoritmos de metodologia direta, foi utilizado um *notebook* Dell, modelo *inspiron 3421*, sistema operacional *Windows 8.1* 64 bits, com processador Intel(R) Core(TM) i5-3337U, CPU @ 1.80GHz, 1801 Mhz, 2 Núcleos e 4 Processadores Lógicos. Em alguns casos, como no *GradientBoosting* e também durante a implementação das *CNNs*, utilizou-se a plataforma *online* do *Google*, o *Google Colab*, que é um ambiente virtual onde podem ser treinados *online* algoritmos de aprendizado de máquina. Nesse local, o usuário pode ter acesso a GPUs, CPUs com velocidades dependentes da sobrecarga do servidor e uma memória RAM de 13GB, que é muito superior à velocidade do dispositivo físico citado e que, portanto, garante uma taxa de processamento muito superior.

3.1 Método indireto

Nesse método, estão os classificadores KNN, SVM, Árvores de Decisão, Florestas Aleatórias, *AdaBoost* e *Gradient Boosting*. Nessa lista, os algoritmos estão organizados de acordo com o nível de complexidade de cada um deles. A ideia de implementar nessa sequência é observar o comportamento dos resultados obtidos à medida em que são empregados modelos mais robustos sob o ponto de vista matemático e computacional. Não necessariamente os modelos mais complexos serão aqueles que entregarão os melhores resultados, uma vez que essa entrega depende das características dos dados de entrada, bem como do tipo de problema que está sendo avaliado (GÉRON, 2017; FACELI, LORENA, *et al.*, 2011).

Os algoritmos, conforme estudado na seção 2.3.2, são distintos, porém o fluxo da metodologia é similar para tais classificadores: As imagens serão importadas, conforme dito anteriormente, do banco de dados da *Northeastern University*. A partir de então, será realizada uma avaliação dos níveis de cinza das imagens por categoria, ou seja, para cada tipo de defeito, serão plotados

histogramas de $frequência\ x\ intensidade$ dos níveis de cinza, que permitirão uma análise prévia do comportamento dos dados, fornecendo uma previsão da possibilidade (ou não) de rotular as imagens com base no estudo desses níveis e também se a textura das imagens podem ser uma boa fonte de dados a ser inserida nos algoritmos. Nessas primeiras etapas serão utilizadas no *Python* as bibliotecas *pandas* (para análise e manipulação de dados), *numpy* (também para manipulação de dados, especialmente para os casos de arranjos matriciais), *OS* (utilizada para interações com o Sistema Operacional, como por exemplo importações de pastas e arquivos) e *matplotlib* (utilizada para plotar e manipular gráficos).

Após essa etapa, nos algoritmos de método direto, serão extraídas as características de textura das imagens discutidas na seção 2.2.4 (contraste, dissimilaridade, *ASM*, energia e homogeneidade) e, conforme já mencionado nesse mesmo item, serão utilizadas para isso as funções *greycomatrix* e *greycoprops*. Essas funções pertencem à biblioteca *skimage*, do s*ci-kit learn* que, assim como a biblioteca *CV2*, são comumente empregadas em análises de imagens.

O próximo passo consistirá na análise dessas características em gráficos de dispersão. Esses gráficos permitirão uma análise mais detalhada desses aspectos, fornecendo informações importantes acerca de eventuais tratamentos (como normalização, por exemplo) a serem realizados nos dados para incrementar o desempenho dos algoritmos. Nessa etapa, serão utilizadas também as bibliotecas *sklearn* (especificamente a função *preprocessing*, empregada em processamento, transformação e tratamento de dados) e *seaborn* (utilizada para plotar e manipular dados para análises estatísticas).

Feito isso, os dados serão inseridos na etapa de treinamento e validação. Nessa etapa os algoritmos de *Machine Learning* são empregados para treinar e rotular as imagens em suas respectivas classes de defeitos. A métrica de validação a ser utilizada no trabalho é a acurácia, que é dada pela fórmula:

$$acurácia = \frac{imagens\ classificadas\ corretamente}{total\ de\ imagens\ de\ entrada} \qquad (XXXI)$$

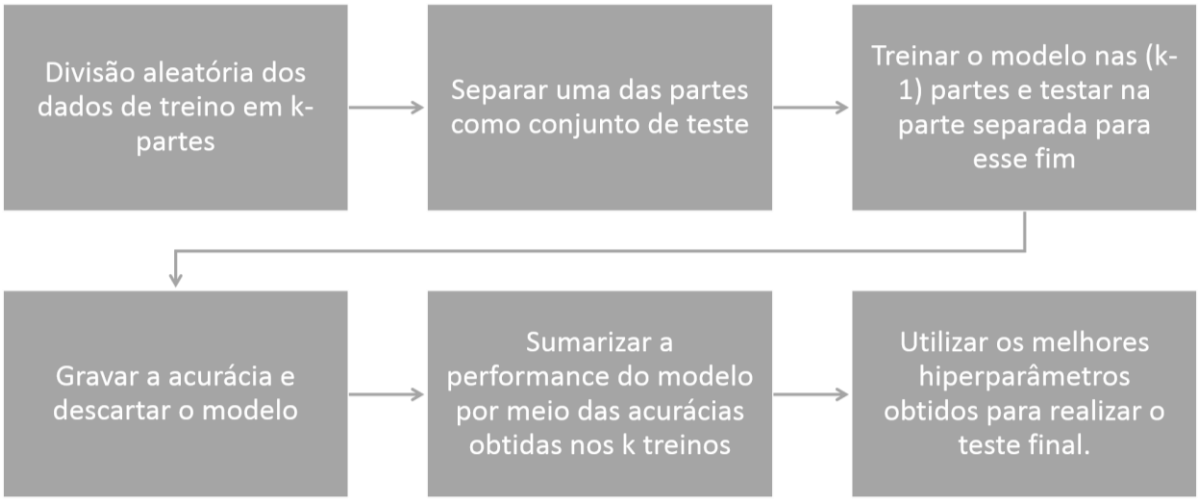
Essa métrica é eficiente para o problema que está sendo analisado, uma vez que a quantidade de dados de entrada está balanceada (são 300 imagens de cada tipo de defeito), além de que o problema sob análise é de classificação (GHATNEKAR, 2018). Caso fosse um problema envolvendo regressão, haveriam métricas mais adequadas a serem utilizadas, como por exemplo o erro médio quadrático e o coeficiente de correlação.

Para otimizar a acurácia e garantir que os modelos tenham o melhor desempenho possível além de garantir que não sofram com o *overfitting*, será aplicado um processo denominado *Cross Validation* (validação cruzada). Esse processo será utilizado também nas redes neurais convolucionais e, por isso, convém elucidar melhor seu funcionamento.

3.1.1 Validação Cruzada

Um dos grandes problemas nos algoritmos de classificação é o *overfitting*. Naturalmente, se o aprendizado dos parâmetros for realizado sobre os mesmos dados que serão testados, o modelo simplesmente irá repetir os rótulos que aprendeu e terá uma precisão enorme nos dados de treino, mas se eventualmente forem oferecidos novos dados para teste, a acurácia não é a mesma. Nessa perspectiva, emerge a validação cruzada como importante ferramenta a ser utilizada para combater esse sobreajuste de dados. Em linhas gerais, o processo de validação cruzada segue os passos apresentados no fluxograma da Figura 26.

Figura 26 - Funcionamento do processo de Validação Cruzada



Fonte: o autor

Devido a essa divisão em *k*-partes, a validação cruzada é também denominada *k-fold cross-validation* (validação cruzada em *k*-dobras) e, usualmente, esse processo é feito junto à função denominada *gridsearchCV* que, além de realizar a validação cruzada, efetua também uma busca pelos melhores hiperparâmetros de cada modelo.

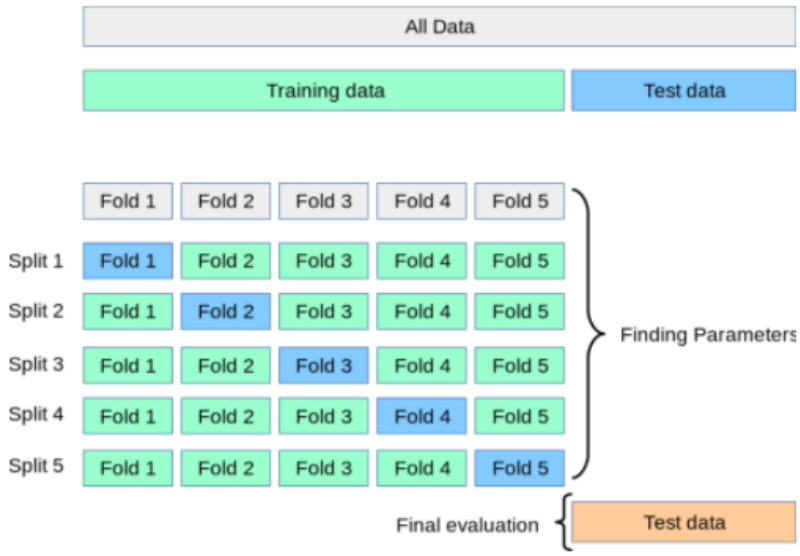
Basicamente, os hiperparâmetros são as variáveis a serem definidas de cada modelo. A título de exemplo, no modelo KNN, os hiperparâmetros seriam, por exemplo, o número de vizinhos

e a métrica a ser utilizada; nas Árvores de Decisão, eles poderiam ser a profundidade da árvore, o número mínimo de amostras para dividir um nó, dentre outros.

Assim sendo, tomando o caso do KNN como exemplo, se fosse desejada uma validação cruzada com $k = 5$ e fossem escolhidos 3 possíveis valores para os 2 hiperparâmetros mencionados, a função *gridsearchCV* realizaria um total de $5 \,.\, 3 \,.\, 3 = 45$ simulações para encontrar o *score* final do modelo. A Figura 27 ilustra o funcionamento da função citada. Note que o conjunto de dados de treino foi dividido em 5 conjuntos, sendo um deles para teste e o restante para treino. Em cada uma das rodadas, um conjunto diferente é utilizado para teste e, ao final, o resultado obtido é aquele que apresenta a melhor acurácia obtida nos testes para todos os hiperparâmetros testados. Esses valores (dos melhores hiperparâmetros) serão utilizados para o teste final do modelo na etapa *test data*, destacada em laranja no canto inferior direito da imagem (SCIKIT-LEARN, 2020b).

Naturalmente, serão utilizados números maiores do que os mencionados acima, tanto de hiperparâmetros quanto de valores que eles podem assumir, o que implica em um número elevado de simulações, gerando um custo computacional muito elevado (SCIKIT-LEARN, 2020b). Desta forma, em alguns momentos, para modelos mais caros do ponto de vista computacional, serão escolhidas quantias restritas de hiperparâmetros devido às limitações da máquina utilizada para computar os algoritmos.

Figura 27 - Cross Validation e obtenção de melhores hiperparâmetros



Fonte: (SCIKIT-LEARN, 2020b)

Feita a validação cruzada para encontrar os melhores hiperparâmetros e otimizar o *score* do modelo ao passar pelo algoritmo, a próxima etapa consiste em visualizar como foi feita a classificação dos dados para que a acurácia final fosse encontrada. Essa visualização é feita por meio de uma matriz denominada “matriz de confusão”.

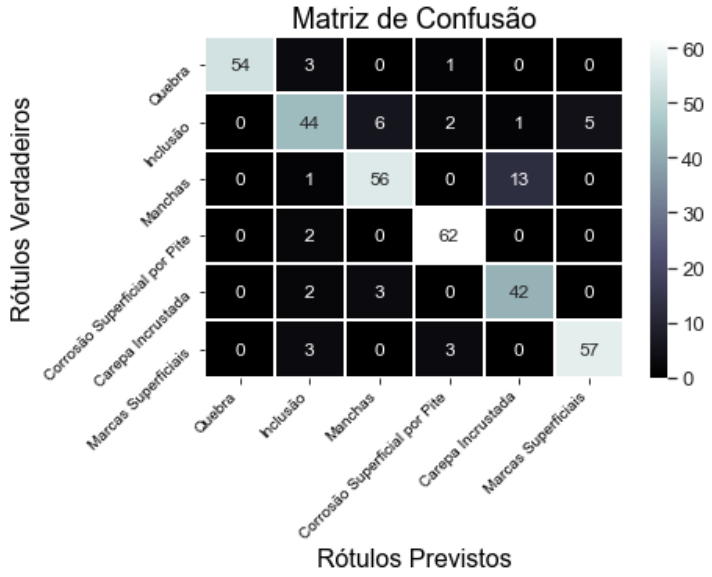
3.1.2 Matriz de Confusão

A matriz de confusão é uma representação em forma de tabela de como os dados foram classificados no modelo sob análise. Nessa matriz, os rótulos dos eixos são as possíveis classes que os dados podem assumir. No presente trabalho, esses eixos são compostos pelos 6 tipos de defeitos mencionados na seção 2.3.2, sendo que, no eixo horizontal, estão os rótulos que foram previstos pelo modelo e, no eixo vertical, as categorias verdadeiras, que deveriam ter sido encontradas. Essa configuração faz com que as previsões feitas de maneira correta fiquem na diagonal principal da matriz. A acurácia do modelo, por consequência, é dada pela soma dos valores dessa diagonal dividida pela soma de todos os elementos da matriz:

$$acurácia = \frac{\sum_{i=j} a_{i,j}}{\sum_{i,j} a_{i,j}} \qquad (XXXII)$$

A Figura 28 é um exemplo de matriz de confusão (que será novamente apresentada nos resultados com o devido contexto).

Figura 28 - Exemplo de Matriz de Confusão



Fonte: o autor

Note que a escala de cor à direita da matriz informa a quantidade de itens presentes em cada célula sendo que, quanto mais clara a célula, mais valores foram previstos para essa categoria. Portanto, o fato de a diagonal principal estar visualmente mais clara que o restante da imagem é um indício de que o modelo teve uma boa acurácia. De fato, ao se calcular esse *score* do modelo, encontra-se:

$$acurácia = \frac{54 + 44 + 56 + 62 + 42 + 57}{360} = \frac{315}{360} = 87{,}5\%$$

Ademais, outro ponto importante de ser destacado é que o maior erro de classificação ocorreu nas manchas, que por 13 vezes foram identificadas como sendo carepa incrustada.

3.2 Método direto

Esse método contempla as redes neurais convolucionais. Conforme visto na seção 2.3.2, as *CNNs* são os modelos mais complexos para estudo de imagens e isso acarreta em análises diferentes a serem feitas nesse tipo de modelo com relação àquelas feitas pelos outros, estudados na seção 3.1.

O início das análises é similar: os estudos dos histogramas $frequência\ x\ intensidade$ dos níveis de cinza também serão utilizados justamente para avaliar se a extração direta desses níveis pode ser uma boa alternativa de classificação. Na sequência, parte-se para a extração propriamente dita. Nessa etapa, bibliotecas e funções utilizadas são as mesmas apresentadas em 3.1: *pandas*, *numpy*, *OS* e *CV2*. Feito isso, parte-se para um preprocessamento rápido dos dados: será feita uma verificação para conferir se os níveis de cinza estão dentro da faixa RGB esperada: de 0 a 255; serão inseridas nas matrizes os rótulos das imagens e, por fim, são plotadas (utilizando-se a biblioteca *matplotlib*) imagens aleatórias para validar se as matrizes de fato reproduzem a realidade das imagens obtidas do banco de dados da *NEU*.

Finalizado o preprocessamento, inicia-se a implementação do algoritmo das *CNNs* propriamente dito. Nessa etapa, utiliza-se o *TensorFlow*, biblioteca comumente empregada no treinamento de redes neurais por permitir processamento paralelo em GPU. Conforme mencionado na introdução do capítulo 3, o *Google Colab* foi o ambiente escolhido para execução das *CNNs* devido a sua melhor capacidade de processamento. Após o treinamento do modelo, a acurácia será avaliada e, com intuito de corrigir eventuais problemas de *Overfitting*

ou até mesmo incrementar um bom resultado já obtido, serão empregadas duas técnicas computacionais tipicamente atreladas a otimização de resultados: a validação cruzada e o *Data Augmentation*.

O processo de validação cruzada é o mesmo já discutido na seção 3.1.1: divide-se os dados de treino em *k*-partes, treinam-se esses dados com $(k-1)$ partes e valida-se na parcela restante. Isso é feito iterativamente durante k vezes e, ao final, os hiperparâmetros que obtiverem melhores resultados na acurácia são utilizados para implementação final do modelo. A metodologia denominada *Data Augmentation*, por sua vez, será melhor discutida na próxima seção.

3.2.1 *Data Augmentation*

Conforme estudado na seção 2.3.1.1, o *overfitting* pode estar associado a alguns fatores, tais como quantia reduzida de dados e dados com baixa qualidade por exemplos. No capítulo 3.1.1 foi vista a validação cruzada como forma de contornar esses problemas. Para o caso das redes neurais convolucionais, existe uma outra técnica comumente empregada denominada *Data Augmentation* (ou acréscimo de dados em português).

Essa técnica consiste num método de regularização que visa diminuir os erros de generalização do algoritmo por meio da introdução de novos dados ao conjunto original, aumentando a diversidade de informações nesse conjunto (TENSORFLOW, 2020). Esses dados introduzidos são obtidos dos próprios dados originais por meio de operações feitas neles, tais como por exemplos:

- Introdução de ruídos e distorções;
- Modificação de contraste, brilho e nitidez;
- Remoção de alguns pixels da imagem;
- Dar *zoom* em porções aleatórias da imagem;
- Transladar ou rotacionar as imagens em diversos ângulos
- Espelhar as imagens

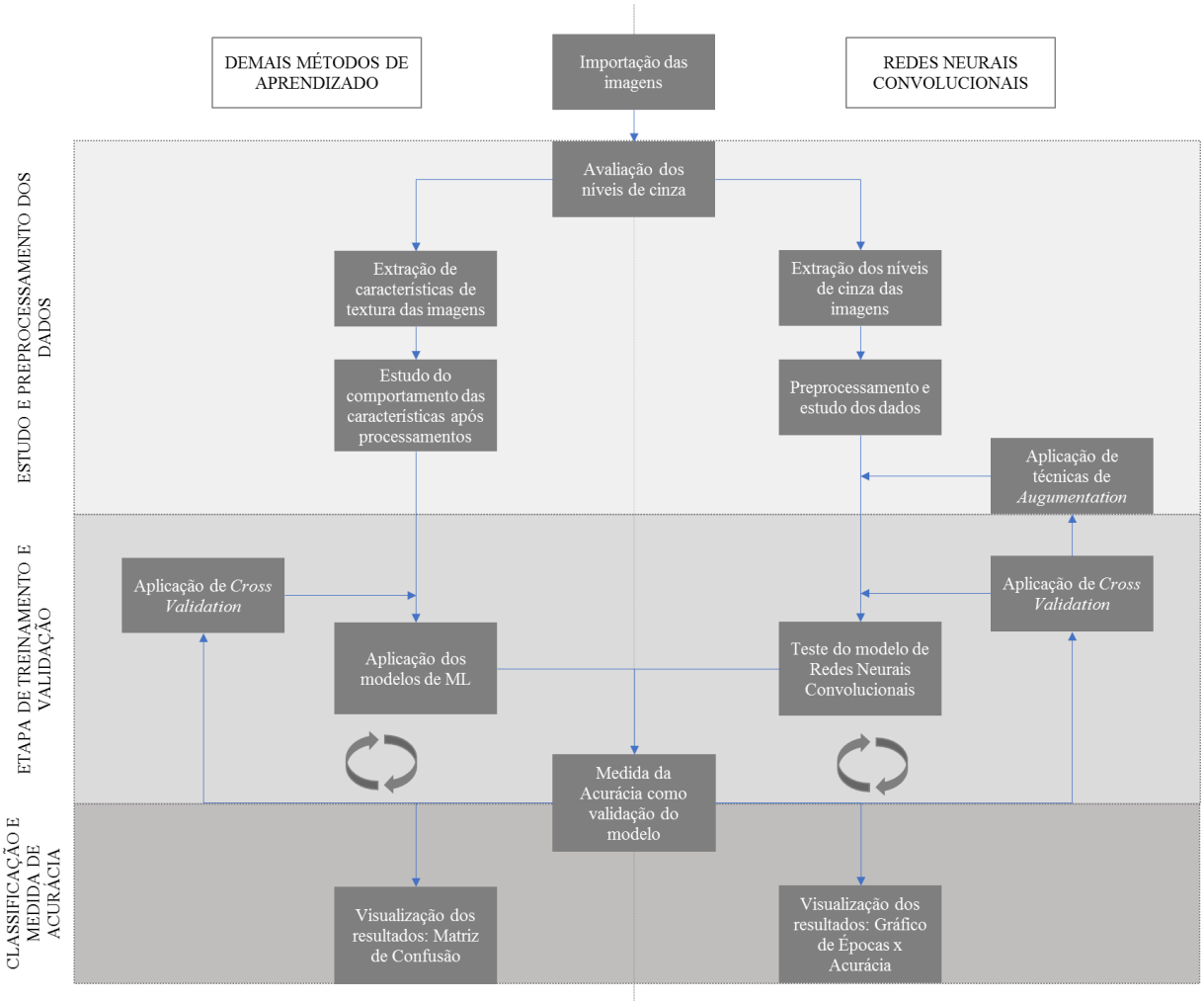
Certamente nem todos os casos mencionados acima são aplicáveis para o caso desse trabalho, porém, sob um ponto de vista geral, introduzir corretamente esses tipos de operações nos dados

de treino causa aumento na variedade desse conjunto, reduzindo assim a possibilidade de o algoritmo se tornar “viciado” nos dados originais importados do banco. Se, por exemplo, o modelo estiver correlacionando uma parcela das bordas da imagem que apresentam sempre intensidades similares de níveis de cinza a um determinado modelo, introduzir ruídos, *zoom* e remover alguns desses pixels pode ajudar o algoritmo no processo de treinamento, reduzindo o *Overfitting* e garantindo, portanto, um resultado melhor na acurácia dos testes.

A aplicação do acréscimo de dados pode causar uma queda na acurácia dos dados de treino, mas isso não é ruim. Esse fato geralmente corrobora a redução do *Overfitting*, que pode ser notada *a posteriori* no aumento da acurácia do conjunto de teste. Essa técnica pode ajudar a melhorar esse *score* até mesmo para modelos eficientes na classificação e, por isso, é importante que o *Augmentation* seja sempre levado em consideração em se tratando das *CNNs* (TENSORFLOW, 2020).

A Figura 29 ilustra um resumo de como se dará a metodologia do trabalho. Na porção direita do eixo horizontal estão os passos aplicados nas redes neurais convolucionais, enquanto que na parte esquerda se encontram os passos para os demais modelos. O eixo vertical, por sua vez é comum aos dois modelos e diz respeito às etapas de preprocessamento, treinamento e classificação. Na etapa de treinamento, as setas cíclicas indicam que o processo é feito de forma iterada para que seja encontrado o melhor valor de eficácia.

Figura 29 - Fluxo da metodologia empregada



Fonte: o autor

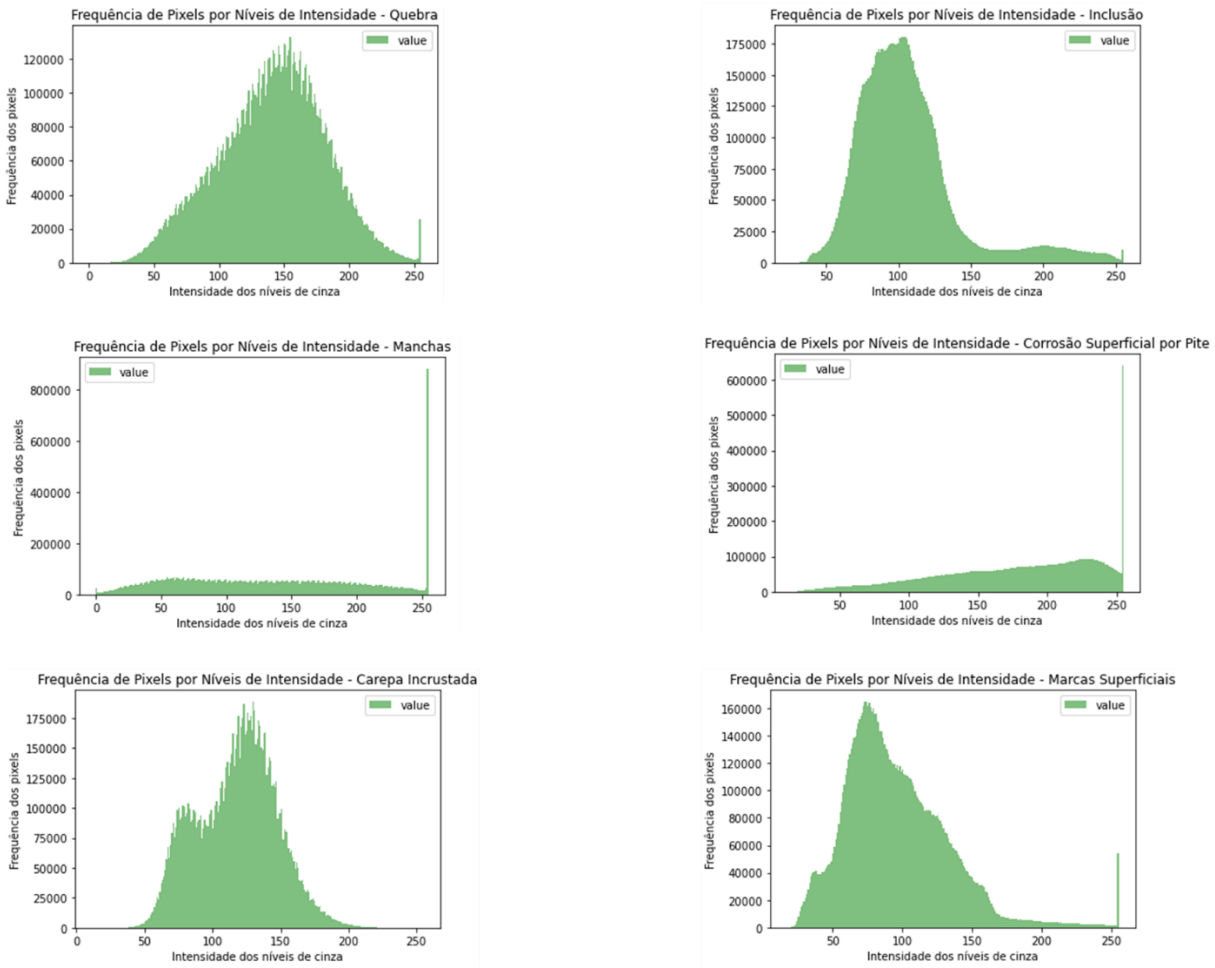
4 RESULTADOS

Na sequência serão apresentados os resultados para cada um dos modelos estudados. Os códigos utilizados ficam disponíveis para consulta em Cortez Junior (2020).

4.1 Avaliação dos níveis de cinza

O primeiro passo na obtenção dos resultados do trabalho foi a extração dos níveis de cinza para cada uma das categorias de defeito estudadas. Os resultados são apresentados nos histogramas da Figura 30.

Figura 30 - Frequências de pixels por Níveis de intensidade para cada defeito



Fonte: o autor

Analisando-se as imagens acima, é possível perceber que as distribuições de níveis de cinza são específicas para cada um dos tipos de defeito. Por exemplo, no caso das quebras, observa-se uma curva com pico em nível de intensidade de aproximadamente 150, enquanto que para as inclusões, esse pico ocorre ao redor do valor 110 e, ao chegar no mesmo valor de 150, observa-se uma estabilização em frequências menores.

Essas distinções na distribuição são de suma importância para o início do trabalho pois, a partir delas, é possível inferir-se que as propriedades de textura discutidas na seção 2.2.4 são, de fato, métodos plausíveis para se treinar um algoritmo de classificação. Tal conclusão pode ser obtida uma vez que, conforme descrito na Tabela 3, essas propriedades se correlacionam com os níveis de cinza de maneira que distribuições diferentes desses níveis geram valores distintos de texturas. Assim sendo, uma vez que os gráficos da Figura 30 são visivelmente distintos para cada um dos tipos de defeitos, espera-se que as texturas sejam igualmente variáveis e que, desta forma, sejam um bom parâmetro de classificação.

A Tabela 5 ilustra as médias de cada uma das propriedades sob análise por tipo de defeito. As colunas de valores estão organizadas em classes de cor, que vão das cores mais claras (menores valores) às mais escuras (maiores valores).

Tabela 5 - Média das propriedades por defeito

	contrast	*dissimilarity*	*homogeneity*	*ASM*	*energy*
Quebra	11379011	520695	3139	358565	595
Inclusão	381820	91543	14077	4476131	2067
Manchas	9701245	449448	5721	15763508	2404
Corrosão por pite	2296732	210095	9283	12944727	2223
Carepa incr.	2210379	231534	6394	1366001	1151
Marcas superf.	1535895	124080	13962	4671984	2100

Fonte: o autor

Essa tabela é mais um argumento que fortalece as conclusões ditas acima acerca da capacidade das propriedades funcionarem como boa fonte de dados para os modelos de *Machine Learning*. Isso pode ser dito pois, conforme ilustrado, cada uma das linhas apresenta comportamentos distintos com relação às outras quando comparadas coluna a coluna, ou seja, defeitos diferentes apresentam texturas diferentes.

Por fim, é importante ressaltar também que a Figura 30 é igualmente um bom fundamento para a aplicação das redes neurais nos dados de entrada: se os níveis de cinza estão distribuídos de maneiras distintas ao longo da imagem e as CNNs analisam esses níveis de maneira direta, esses algoritmos possivelmente serão capazes de detectar essas variações e, por consequência,

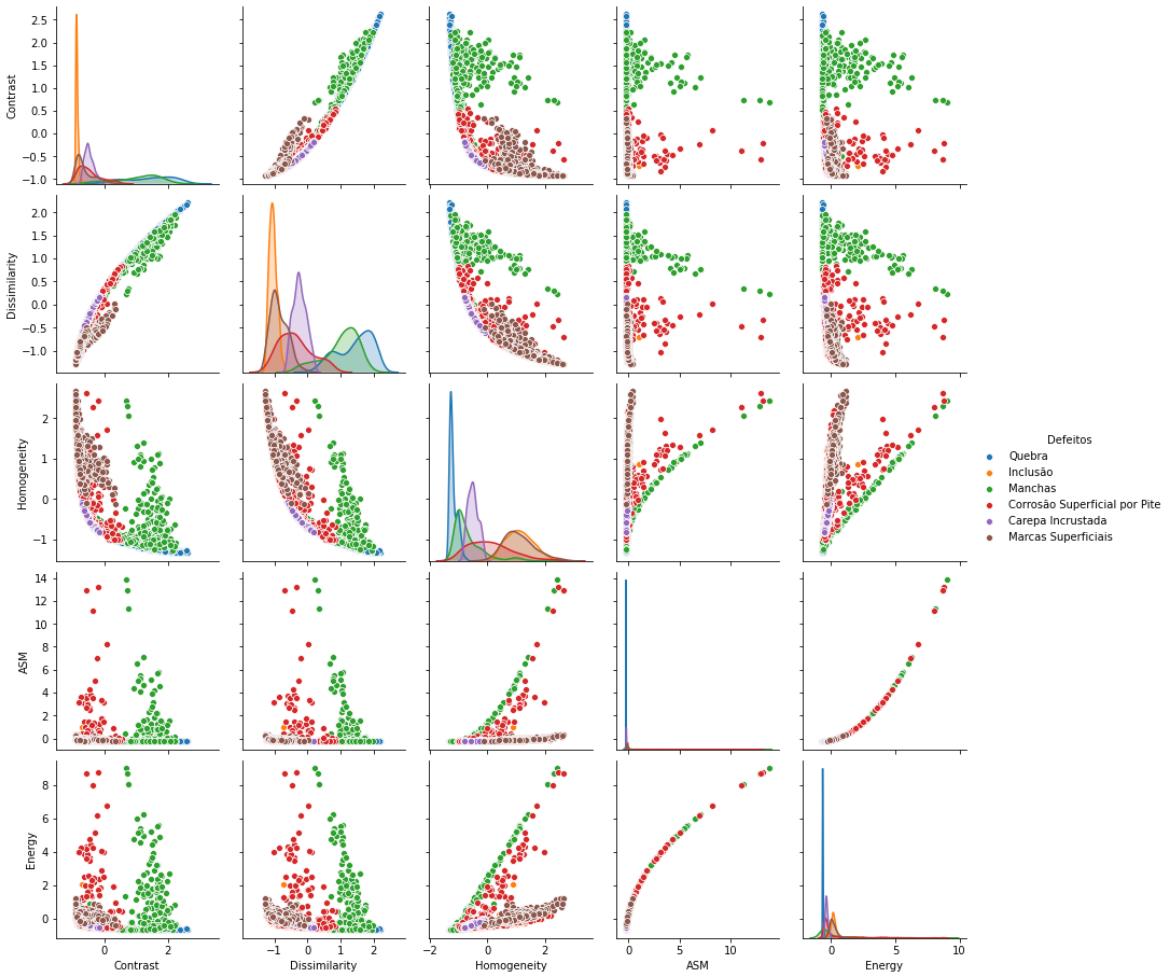
classificar os modelos de forma correta. Portanto, esse modelo foi também aplicado aos dados de entrada das redes neurais e, na próxima seção, os resultados dessa aplicação serão apresentados, tanto para o modelo direto quanto para os modelos indiretos.

4.2 Preprocessamento e estudo dos dados manipulados

Agora que existe uma importante compreensão de que as características de textura podem ser boas fontes de dados aos modelos de *ML*, parte-se para a implementação desses algoritmos para averiguar, por meio da acurácia, se as conclusões da seção 4.1 estão corretas ou, caso não estejam, discutir os resultados. Antes disso, é importante averiguar como os dados de entrada se comportam e se um eventual processamento (ou ajuste) prévio deles pode ou não ser uma boa alternativa para o modelo. Essa análise foi feita por meio de plotagens dos dados, dois a dois em gráficos de dispersão conforme ilustra a Figura 31 e a Figura 32.

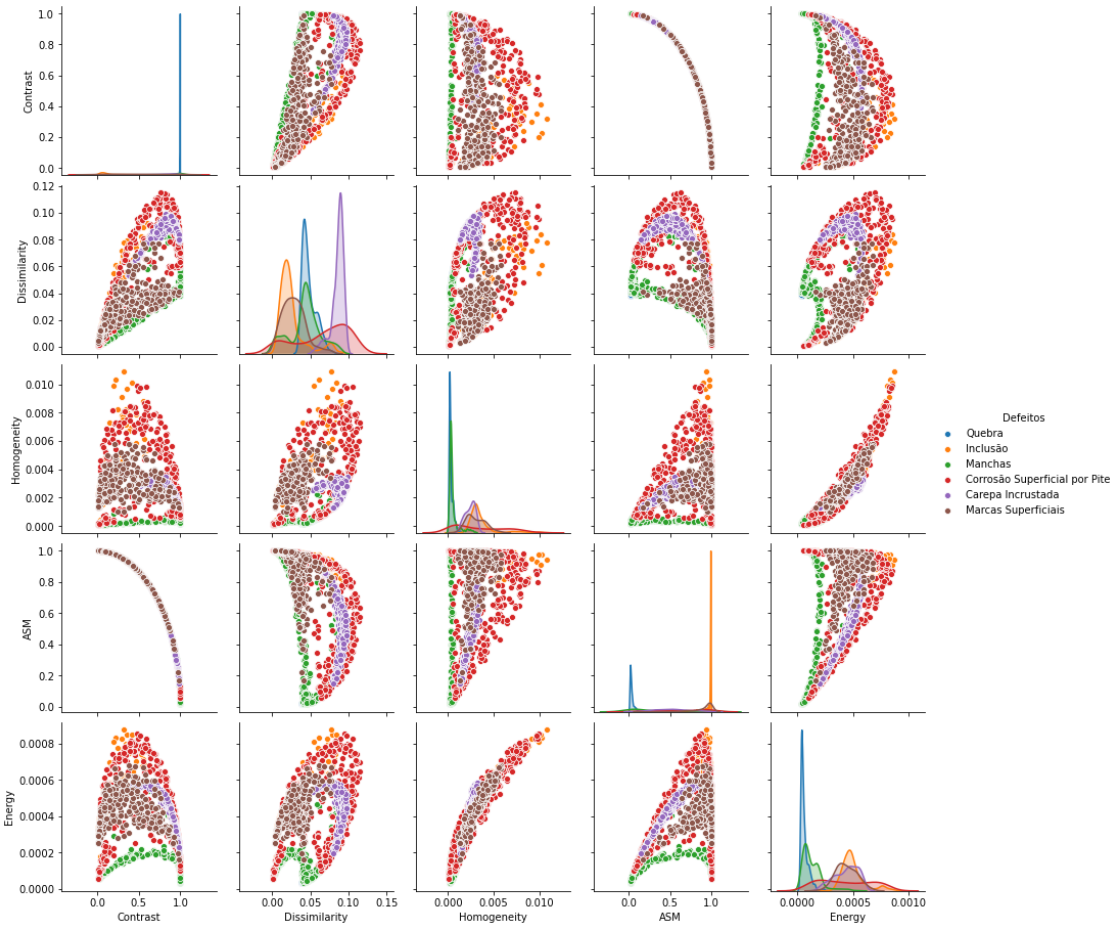
Na Figura 31 o padrão de comportamento dos dados é aquele apresentado quando não são aplicados preprocessamentos. Eles foram apenas colocados em escala para plotagem. Já na Figura 32, os dados foram normalizados antes de serem inseridos nos gráficos.

Figura 31 - Comportamento dos dados (plotados em escala)



Fonte: o autor

Figura 32 - Comportamento dos dados (normalizados)



Fonte: o autor

Analisando-se as figuras acima, observa-se uma notória diferença de comportamentos para os dados originais e os dados normalizados. Veja que existem casos nos quais os dados são mais facilmente separáveis nos gráficos da Figura 31 do que na Figura 32 e vice-versa. Por exemplo, as plotagens de $ASM\ x\ Energia$ na Figura 31 indicam que as dificuldades de separação dos dados para rotulação seriam maiores se comparadas aos mesmos gráficos da Figura 32. Para modelos nos quais as configurações da distribuição de dados apresentam grande relevância, como por exemplo o *KNN* e as *SVMs* (vide seções 2.3.2.1 e 2.3.2.2), aplicar a normalização nos dados iniciais pode, portanto, ser uma importante ferramenta na otimização da classificação do modelo, impactando diretamente na acurácia final. Desta forma, durante todas as classificações os dados serão normalizados para observação do impacto desse preprocessamento no *score* obtido.

4.3 Resultados para os classificadores de método indireto

4.3.1 *KNN*

Na Tabela 6 são apresentados os hiperparâmetros utilizados para treinamento dos dados no modelo dos K-vizinhos mais próximos.

Tabela 6 - hiperparâmetros testados no *KNN*

K-Nearest Neighbors	
weights	*uniform; distance*
metric	*euclidean; manhattan; minkowski*
algorithm	*auto; ball_tree; kd_tree; brute*
n_neighbors	2; 3; 4; 5; 6; 10; 12; 15; 20
Cross Validation (folds)	4
nº de treinos	864

Fonte: o autor

Foram utilizados um total de 18 hiperparâmetros que, combinados com 4 *folds* na validação cruzada, resultaram em 864 treinos para obtenção da eficácia do modelo. Na Tabela 7 são apresentados os melhores parâmetros obtidos bem como a acurácia resultante da utilização desses valores para o modelo *KNN*.

Tabela 7 – Melhores hiperparâmetros – *KNN*

Melhores hiperparâmetros obtidos	
weights	*distance*
metric	*manhattan*
algorithm	*auto*
n_neighbors	3
acurácia	88,89%

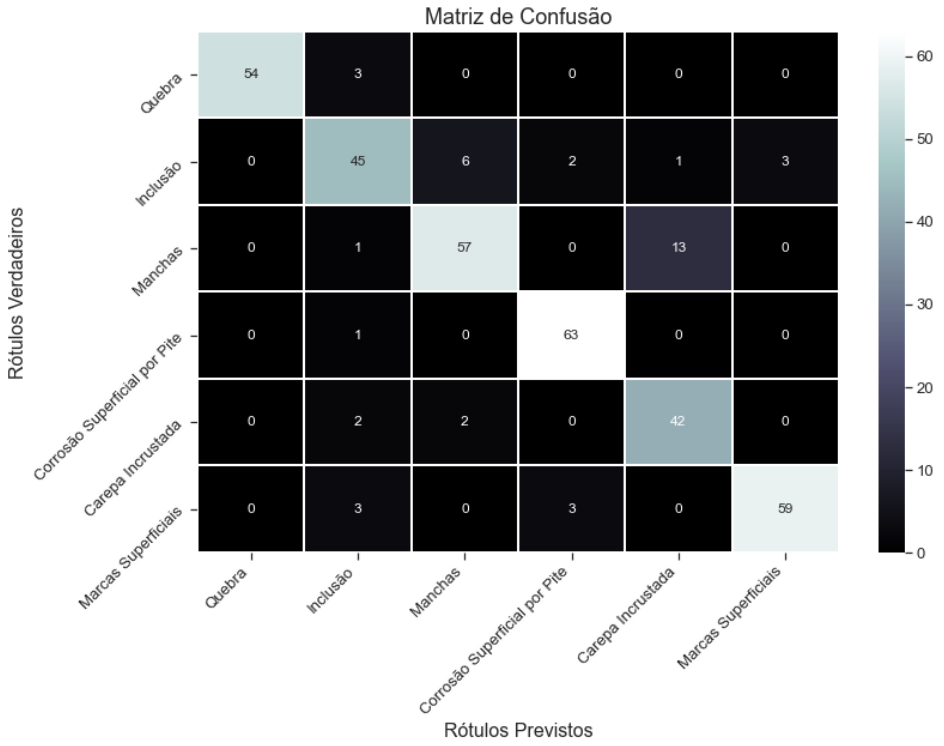
Fonte: o autor

Analisando-se os hiperparâmetros escolhidos, é possível constatar que, para o banco de dados que está sob análise, o modelo funciona melhor quando são atribuídos pesos maiores aos vizinhos mais próximos das amostras de teste. Ademais, um número reduzido de vizinhos funciona melhor no modelo, o que é esperado. O aumento demasiado do número de vizinhos necessários para classificação é uma fonte potencial de *underfitting* devido aos ruídos que emergem da análise de muitos dados para classificar a amostra.

Por fim, na Figura 33 é apresentada a matriz de confusão para os K-vizinhos mais próximos. Na imagem, é possível notar pela diagonal principal que foram previstos 320 valores corretos para um total de 360 caracterizações realizadas no conjunto de teste. Sem a utilização do

gridsearchcv para obtenção dos melhores hiperparâmetros, o total de acertos foram de 312, com acurácia de 86,6%. A normalização não produziu efeitos significativos no *score* final.

Figura 33 - Matriz de confusão para o modelo *KNN*



Fonte: o autor

Para esse modelo, as corrosões foram classificadas corretamente 98% das vezes, apresentando apenas um erro em 64 tentativas. Destaca-se também que nenhum defeito foi rotulado erroneamente como quebra. As inclusões, por outro lado, foram classificadas de forma equivocada por 12 vezes em 57 tentativas, acarretando em um percentual de acerto de aproximadamente 79%. Outro ponto de atenção fica por conta das manchas que, por 13 vezes, foram rotuladas como carepa incrustada; o maior número de erros por pares de defeitos.

4.3.2 *SVM*

Na Tabela 8*Tabela 6* são apresentados os hiperparâmetros utilizados para treinamento dos dados nas máquinas de vetores de suporte.

Tabela 8 - hiperparâmetros testados no SVM

Support Vector Machines	
C	*1; 10; 50; 100; 250; 500; 1000; 2000; 3000*
kernel	*linear; poly; rbf*
degree	*2; 3; 4; 5*
Cross Validation	4
nº de treinos	432

Fonte: o autor

Foram utilizados um total de 16 hiperparâmetros que, combinados com 4 *folds* na validação cruzada, resultaram em 432 treinos para obtenção da eficácia do modelo. Esse número foi inferior ao modelo anterior pois o custo de processamento das *SVMs* é muito maior e, portanto, a velocidade de processamento é inferior, necessitando de menos parâmetros para não sobrecarregar os processadores. Na Tabela 9 são apresentados os melhores parâmetros obtidos bem como a acurácia resultante da utilização desses valores para o modelo *SVM*.

Tabela 9 - Melhores hiperparâmetros – SVM

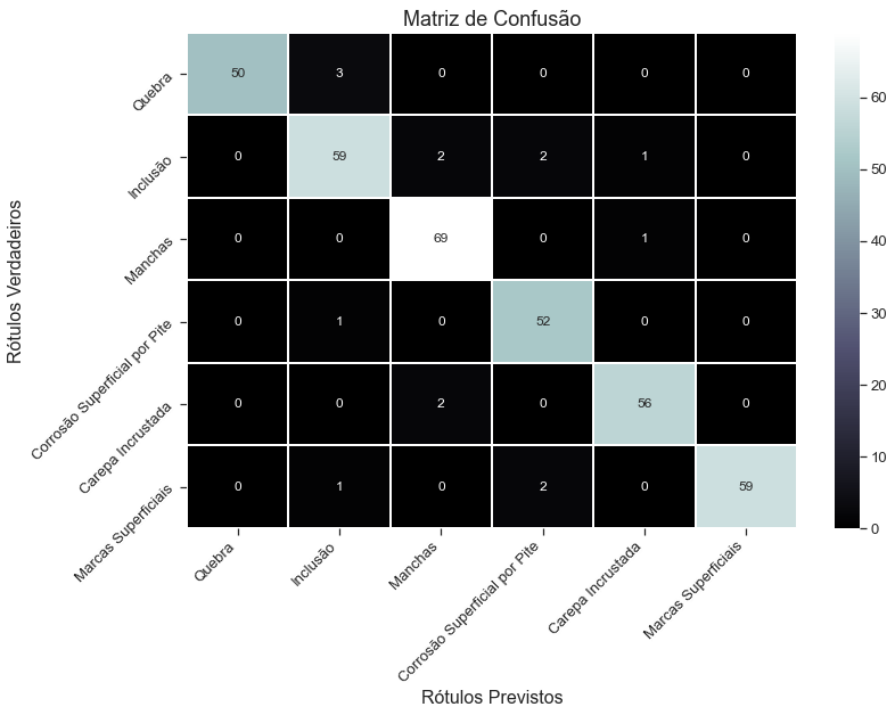
Melhores Hiperparâmetros obtidos	
C	*1*
kernel	*rbf*
degree	N/A
acurácia	95,83%

Fonte: o autor

Aqui, foram utilizados, na prática, 2 hiperparâmetros. O parâmetro de grau (*degree*) foi, sim, levado em conta durante os processamentos, mas esse parâmetro é aplicável apenas ao *kernel* do tipo polinomial e, conforme pode ser visto na Tabela 9, o *kernel* que otimiza o desempenho do modelo é o *rbf*. O parâmetro C é um fator de correção, que penaliza o modelo todas as vezes em que uma amostra é classificada de forma errada (FACELI, LORENA, *et al.*, 2011) e o valor padrão do modelo ($C = 1$) foi o que mais se encaixou para os dados de entrada inseridos. A acurácia obtida foi muito boa, o que indica que esse modelo é uma boa forma de se interpretar imagens para a determinação de defeitos superficiais em placas de aço.

Por fim, na Figura 34 é apresentada a matriz de confusão para as *Support Vector Machines*. Na imagem, é possível notar pela diagonal principal que foram previstos 345 valores corretos para um total de 360 caracterizações realizadas no conjunto de teste.

Figura 34 - Matriz de confusão para o modelo SVM



Fonte: o autor

Para as SVMs, as manchas apresentaram o maior percentual de acertos, sendo classificadas corretamente em 69 oportunidades de 70, cerca de 99% das vezes. As inclusões são os defeitos que apresentaram maior percentual de classificação incorreta: foram 5 em 64, ou seja, aproximadamente 8% de erro. As quebras foram 3 vezes classificadas como inclusões, o maior número de classificações incorretas por pares de defeitos.

Nesse classificador, o preprocessamento tem papel fundamental: se os dados não receberem nenhum tipo de tratamento antes da classificação, ocorre *Overfitting*: no conjunto de treino, a acurácia passa a ser de 100%, enquanto que, no conjunto de testes, o *score* cai para 15%. Por outro lado, normalizar os dados leva a uma acurácia de 80,35% nos dados de treino e 77,50% nos testes. Colocar os dados em escala, por outro lado é o que garante a enorme precisão observada nos dados apresentados: o *score* no conjunto de treino sobe para 96,53% e, no teste, para 95,83%. A Tabela 10 apresenta um resumo da acurácia por tipo de preprocessamento dos dados de entrada.

Tabela 10 - Acurácia por tipo de processamento dos dados - SVM

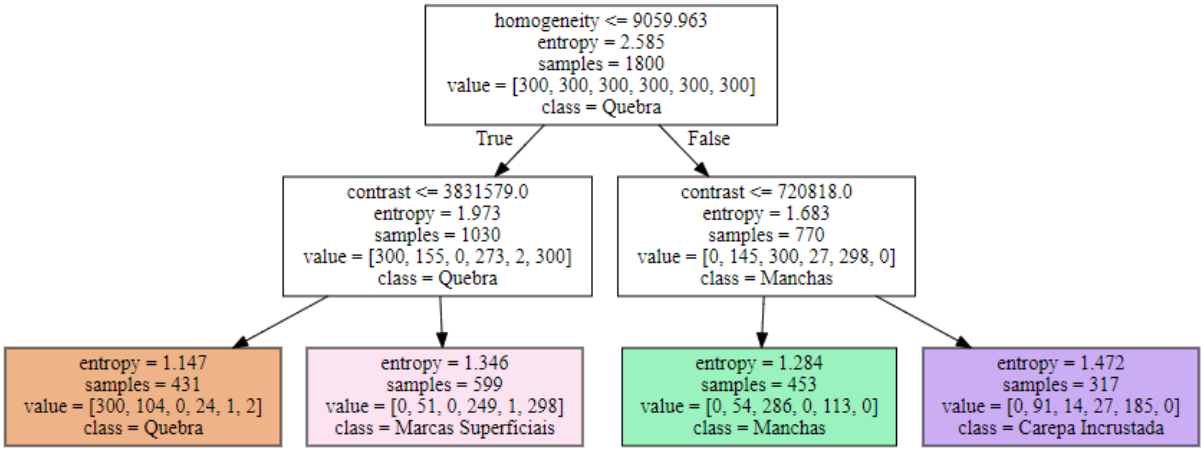
	Treino	Teste
Sem preprocessamento	100%	15%
Dados normalizados	80,35%	77,50%
Dados em escala	96,53%	95,83%

Fonte: o autor

4.3.3 Árvores de Decisão

Nessa seção, primeiramente será ilustrada uma árvore de decisão com profundidade menor obtida nos resultados (Figura 35) e, depois, serão apresentados os melhores hiperparâmetros. O procedimento se dará dessa forma pois, nos melhores resultados obtidos, as árvores apresentam profundidade elevada, o que impede que a ilustração seja anexada de maneira visível ao longo do texto.

Figura 35 - exemplo dos resultados de classificação obtidos com Árvores de Decisão



Fonte: o autor

Na Figura 35, utilizou-se uma profundidade igual a 2 e o critério associado aos ganhos de informação foi a entropia. Para se ter uma ideia melhor de como é feita a classificação, suponha que seja inserida no modelo uma amostra com valor de homogeneidade igual a 9000 e contraste igual a 4000000. Portanto, partindo-se da raiz, como o valor da homogeneidade é inferior ao critério estabelecido, a condição é verdadeira e a árvore passa ao primeiro nível de profundidade, no qual a condição subjacente é igualmente correta. Assim sendo, a árvore passa ao último nó, classificando a amostra como “quebra”. Note que, nessa regra, de acordo com o campo *value* do último nó, são 300 amostras pertencentes ao conjunto de quebra; 104 das inclusões; 0 das manchas; 24 de corrosão por pite; 1 de carepa incrustada e 2 de marcas superficiais, totalizando 431 amostras. Sendo assim, são 300 amostras classificadas corretamente em um universo de 431, o que garante uma acurácia de 69,6% na regra em questão.

No exemplo da Figura 35 são 4 regras que totalizam, no último nó, 1800 amostras, das quais 1069 foram classificadas corretamente implicando em uma acurácia final do modelo de 59,4%. É importante notar que o funcionamento do modelo permite inferir quais são as características que mais influem no processo de decisão.

Elucidado melhor o funcionamento das árvores de decisão para os dados do presente trabalho, é possível partir para a apresentação dos hiperparâmetros utilizados para o treinamento de dados no modelo. Eles são apresentados na Tabela 11.

Tabela 11 – hiperparâmetros testados nas Árvores de Decisão

Árvores de Decisão	
criterion	*entropy; gini*
splitter	*best; random*
max_depth	21; 23; 25; 27; 29; 30; 31; 33; 35; 40; 45; 50; 55; 60; 65; 70; 75; 80; 90; 100; 150; 200; 250; 300; 350; 400
min_samples_leaf	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60
min_samples_split	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 50, 100, 200
Cross Validation	4
nº de treinos	87360

Fonte: o autor

Na tabela acima, os critérios “gini” e “entropia” são os mesmos discutidos na seção 2.3.2.3. *Splitter* ou “divisor” são as estratégias para divisão em cada nó, sendo que “*best*” denota a escolha da melhor característica para dividir os nós e “*random*” denota a escolha de características aleatórias para realizar a divisão. *max_depth* é o parâmetro que denota a profundidade máxima da árvore; *min_samples_leaf* é o mínimo de amostra por nó e *min_samples_split* é o mínimo de amostras para dividir um nó (SCIKIT-LEARN, 2020c).

Foram utilizados um total de 59 hiperparâmetros que, combinados com 4 *folds* na validação cruzada, resultaram em 87360 treinos para obtenção da eficácia do modelo. O número de treinos é muito superior aos modelos anteriores devido à necessidade de menor custo de processamento. Na Tabela 12 são apresentados os melhores parâmetros obtidos bem como a acurácia resultante da utilização desses valores para o modelo das árvores de decisão.

Tabela 12 - Melhores hiperparâmetros – Árvores de Decisão

Melhores Hiperparâmetros obtidos	
criterion	*entropy*
splitter	*best*
max_depth	10
min_samples_leaf	3
min_samples_split	8
acurácia	86,39%

Fonte: o autor

É importante ressaltar que os valores observados na Tabela 12 dizem respeito aos dados preprocessados (normalizados nesse caso). Para os dados sem processamento ou colocados em escala, a acurácia, conforme ilustra a Tabela 13, foi ligeiramente inferior.

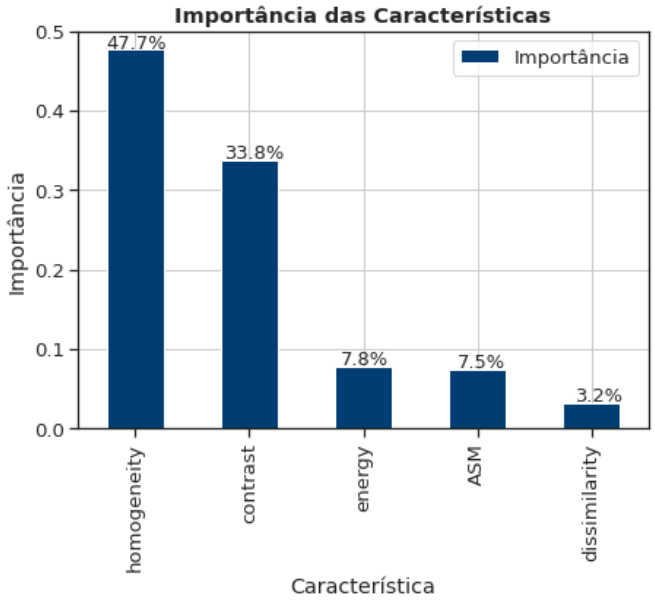
Tabela 13 - Acurácia por tipo de processamento dos dados - Árvores de Decisão

	Treino	Teste
Sem preprocessamento	93,26%	85,00%
Dados normalizados	94,58%	86,39%
Dados em escala	95,21%	86,11%

Fonte: o autor

Conforme dito anteriormente, nas árvores de decisão, o algoritmo possibilita o cálculo das características utilizadas que foram mais relevantes nos ganhos de informação e, por consequência, na obtenção dos resultados apresentados. Conforme pode ser visto na Figura 36, a homogeneidade e o contraste tiveram a maior importância, tendo apresentado 47,7% e 33,8% de relevância respectivamente, totalizando um percentual de 81,5% do resultado final.

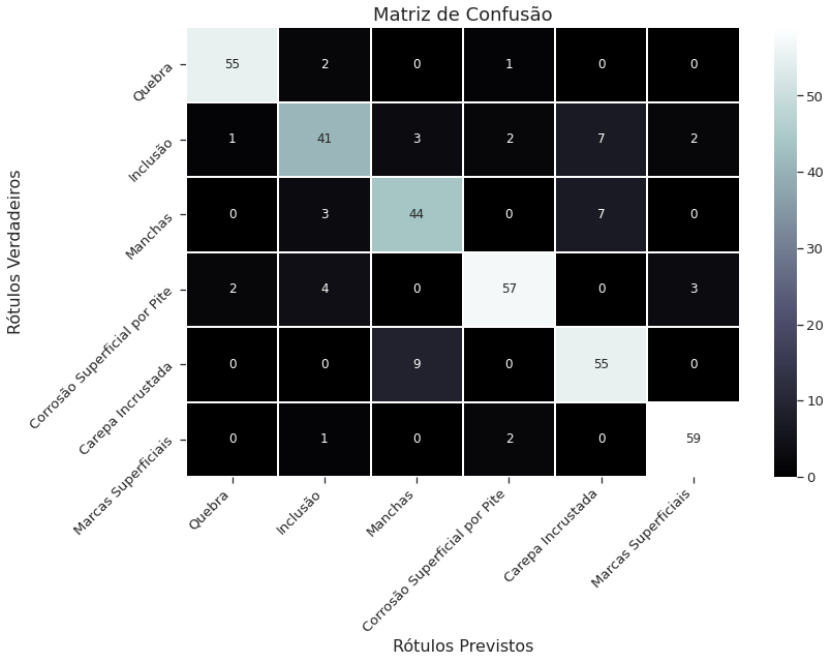
Figura 36 - Características mais importantes - Árvores de Decisão



Fonte: o autor

Por fim, na Figura 37 é apresentada a matriz de confusão para as árvores de decisão obtidas com os hiperparâmetros da Tabela 12. Na imagem, é possível notar pela diagonal principal que foram previstos 311 valores corretos para um total de 360 caracterizações realizadas no conjunto de teste.

Figura 37 - Matriz de confusão para o modelo de Árvores de Decisão



Fonte: o autor

Para as árvores de decisão, as marcas superficiais foram os defeitos que foram rotulados de forma correta com maior frequência: foram 59 acertos em 62 tentativas, um percentual de 95%. Por outro lado, as inclusões foram classificadas de forma incorreta o maior número de vezes: 15 de 56 vezes, ou seja, 27% de erros. As carepas incrustadas foram classificadas como manchas erroneamente por 9 vezes, caracterizando o maior número de erros por pares de defeitos.

4.3.4 Florestas Aleatórias

Na Tabela 14 são apresentados os hiperparâmetros utilizados para treinamento dos dados nas florestas aleatórias.

Tabela 14 - hiperparâmetros testados nas Florestas Aleatórias

Florestas Aleatórias	
criterion	*entropy; gini*
n_estimators	50; 100; 150; 200; 250; 300; 350; 400
max_depth	50, 60, 70, 80, 90, 100, 200
min_samples_leaf	1, 3, 5, 6, 7, 9, 10
min_samples_split	3, 4, 5, 6, 7, 8, 9, 10
Cross Validation	4
nº de treinos	25088

Fonte: o autor

Note que, por ser um modelo *ensemble* baseado nas árvores de decisão, os hiperparâmetros testados são basicamente os mesmos, diferindo apenas pelo parâmetro *n_estimators*, que representa o número de árvores nas florestas.

Foram utilizados um total de 32 hiperparâmetros que, combinados com 4 *folds* na validação cruzada, resultaram em 25088 treinos para obtenção da eficácia do modelo. O número de treinos é inferior ao das árvores de decisão pois, por ser do tipo *ensemble*, naturalmente exige maior capacidade de processamento. Na Tabela 15 são apresentados os melhores parâmetros obtidos bem como a acurácia resultante da utilização desses valores para o modelo das florestas aleatórias.

Tabela 15 - Melhores hiperparâmetros – Florestas Aleatórias

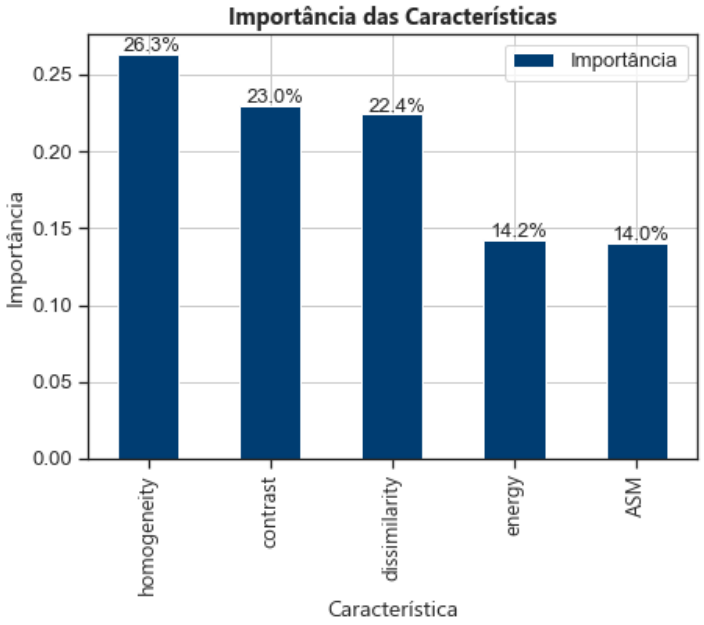
Melhores Hiperparâmetros obtidos	
criterion	*gini*
n_estimators	*100*
max_depth	100
min_samples_leaf	1
min_samples_split	4
acurácia	93,89%

Fonte: o autor

No modelo das florestas aleatórias, colocar os dados em escala ou normalizá-los não gerou melhoras significativas nos *scores* observados. A acurácia obtida foi muito boa, a segunda melhor até o momento. Isso indica que as *Random* Forests são um bom modelo para interpretação de micrografias e detecção de defeitos superficiais em placas de aço.

Sendo as florestas aleatórias um modelo *ensemble* baseado nas árvores de decisão, elas também permitem o cálculo das características utilizadas que foram mais relevantes nos ganhos de informação. Conforme pode ser visto na Figura 38, a homogeneidade, o contraste e a dissimilaridade tiveram a maior importância, tendo apresentado 26,3%, 23,0% e 22,4% de relevância respectivamente. Os resultados estão melhores distribuídos se comparados com as árvores de decisão, o que contribuiu na obtenção de um resultado melhor para o presente modelo se comparado ao anterior.

Figura 38 - Características mais importantes - Florestas Aleatórias



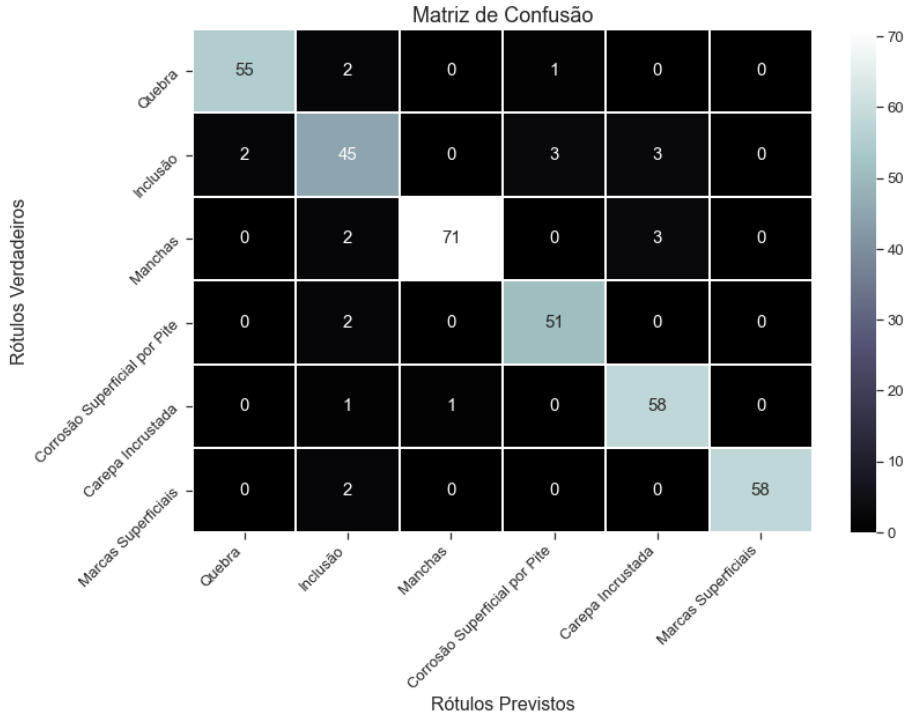
Fonte: o autor

Por fim, na Figura 39 é apresentada a matriz de confusão para as florestas aleatórias. Na imagem, é possível notar pela diagonal principal que foram previstos 338 valores corretos para um total de 360 caracterizações realizadas no conjunto de teste.

Para as *Random Forests*, as manchas apresentaram o maior número de classificações corretas (71). Percentualmente, carepas incrustadas e marcas superficiais ficam a frente, com 97% de índice de acerto cada uma. As inclusões são os defeitos que apresentaram maior percentual de classificações incorretas: foram 8 em 53, ou seja, aproximadamente 15% de erro. Destaca-se também que as inclusões foram os defeitos que mais apareceram de forma incorreta nas tentativas de classificar os outros defeitos: foram 9 classificações equivocadas, sendo 2 para cada tipo de defeito, com exceção das carepas, rotuladas como inclusão apenas uma vez.

Um último ponto de atenção com relação às inclusões é que elas aparecem por duas vezes nos pares de defeitos com maiores números de erros de classificação: uma vez formando par com as carepas e, na outra, com as marcas superficiais. A outra dupla de defeitos com maior erro de previsão em número são as manchas e as carepas.

Figura 39 - Matriz de confusão para o modelo de Florestas Aleatórias



Fonte: o autor

Devido aos fatos mencionados nos parágrafos anteriores, é importante notar-se que, embora as florestas aleatórias sejam, em geral, um excelente método de caracterização para os defeitos em

estudo, para aplicações nas quais o foco seja detecções de inclusão em placas de aço, talvez a utilização desse modelo atrelado às metodologias empregadas no trabalho não sejam a melhor escolha a ser feita.

4.3.5 *AdaBoost*

Na Tabela 16 são apresentados os hiperparâmetros utilizados para treinamento dos dados no modelo *AdaBoost*.

Tabela 16 - hiperparâmetros testados no *Adaptive Boosting*

	AdaBoost
n_estimators	15; 16; 17; 18; 19; 20; 21; 22; 23; 24; 25; 30; 40; 45; 50; 60; 70; 80; 90; 100; 150; 200; 300
learning_rate	1; 0.1; 0.01; 0.02; 0.03; 0.04; 0.005
Cross Validation	4
nº de treinos	644

Fonte: o autor

Para o *AdaBoost*, foram utilizados apenas dois grupos de hiperparâmetros, o número de estimadores e a taxa de aprendizado. No total, foram utilizados 30 hiperparâmetros que, combinados com 4 *folds* na validação cruzada, resultaram em 644 treinos para obtenção da eficácia do modelo. Na Tabela 17 são apresentados os melhores parâmetros obtidos bem como a acurácia resultante da utilização desses valores para o modelo.

Tabela 17 - Melhores hiperparâmetros – *AdaBoost*

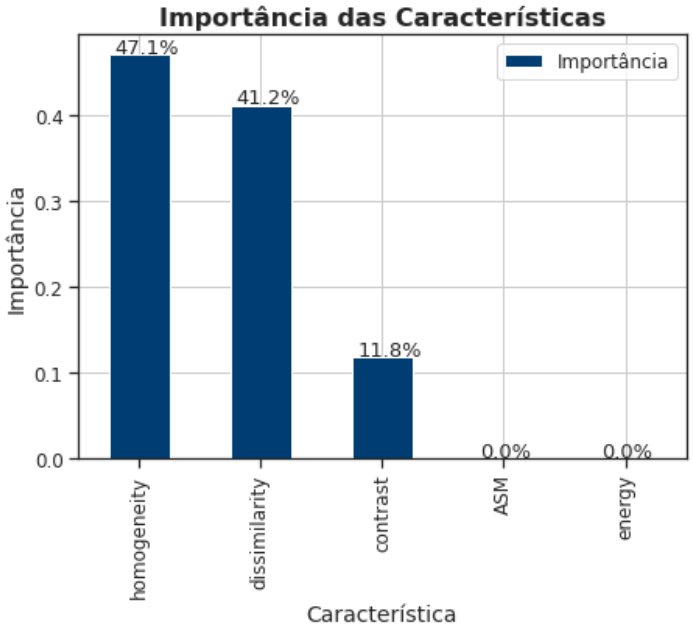
Melhores Hiperparâmetros obtidos	
n_estimators	*17*
min_samples_split	0,01
acurácia	51,11%

Fonte: o autor

No *Adaptive Boosting*, colocar os dados em escala ou normalizá-los não gerou melhoras significativas nos *scores* observados. A acurácia obtida foi muito ruim, a pior até o momento. Por serem um modelo *ensemble,* é possível também calcular as características utilizadas que foram mais relevantes nos ganhos de informação deste modelo. Conforme pode ser visto na Figura 40, a homogeneidade, a dissimilaridade e o contraste tiveram a maior importância, apresentando 47,1%, 41,2% e 11,8% de relevância respectivamente. A distribuição não é boa: homogeneidade e dissimilaridade, juntas, representam quase 90% de importância, enquanto que

ASM e energia não foram, sequer, levadas em consideração. Isso reflete a baixa acurácia do modelo, sugerindo a ocorrência de *underfitting*; hipótese que se confirma quando se analisa a acurácia no conjunto de treino: 49,72%.

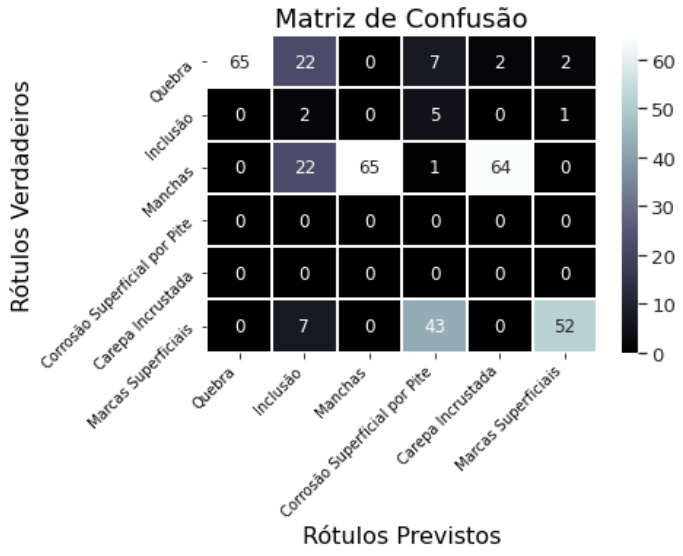
Figura 40 - Características mais importantes - AdaBoost



Fonte: o autor

Por fim, na Figura 41 é apresentada a matriz de confusão para o *Adaptive Boosting*. Na imagem, é possível notar pela diagonal principal que foram previstos 184 valores corretos para um total de 360 caracterizações realizadas no conjunto de teste.

Figura 41 - Matriz de confusão para o modelo *AdaBoost*



Fonte: o autor

Na matriz de confusão, podemos ver os claros efeitos do *underfitting*: o algoritmo não foi capaz de aprender como classificar as carepas e corrosões e, por isso, não rotulou nenhuma amostra com esses defeitos. As inclusões também aparecem pouquíssimas vezes: 7 no total. Os demais tipos de defeitos apresentam erros elevados, sendo que, para as manchas, a ocorrência de classificações incorretas superou a metade dos dados: foram cerca de 57% de um total de 152 rótulos propostos.

As classificações com maiores índices de erros foram as inclusões e as manchas, com 75% e 57% de rótulos incorretos, respectivamente. Em número, os pares de defeitos que foram rotulados de forma equivocada por mais vezes foram as manchas e carepas além das marcas e corrosões. Os resultados apresentados indicam que esse modelo não é uma boa escolha para caracterizar defeitos superficiais em aço dada a metodologia empregada na coleta dos dados.

4.3.6 *Gradient Boosting*

Na Tabela 18 são apresentados os hiperparâmetros utilizados para treinamento dos dados no modelo *Gradient Boosting*.

Tabela 18- hiperparâmetros testados no *Gradient Boosting*

Gradient Boosting	
n_estimators	*100; 300; 450; 850; 1000*
learning_rate	*0,001; 0,01; 0,5*
max_depth	*10, 30, 50, 70*
min_samples_leaf	*1; 3; 5; 6; 7*
min_samples_split	1, 5, 9, 13, 15
Cross Validation	4
nº de treinos	6000

Fonte: o autor

No total, foram testados um total de 22 hiperparâmetros que, combinados com 4 *folds* na validação cruzada, resultaram em 6000 treinos para obtenção da eficácia do modelo. O número de rodadas não é elevado pois o custo de processamento para esse modelo é muito alto, reduzindo a velocidade devido às limitações das máquinas utilizadas. Na Tabela 19 são apresentados os melhores parâmetros obtidos bem como a acurácia resultante da utilização desses valores para o modelo de *Gradient Boosting*.

Tabela 19 - Melhores hiperparâmetros - *Gradient Boosting*

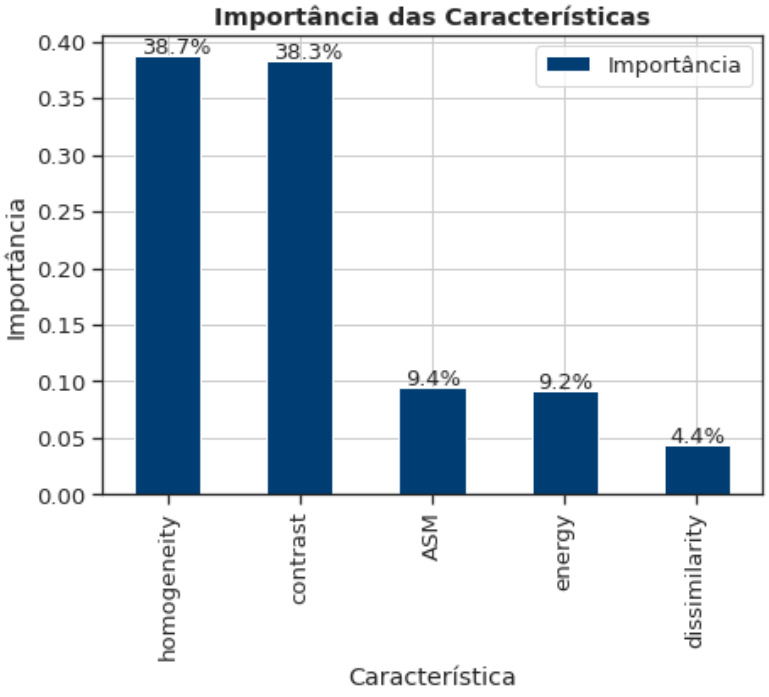
Melhores Hiperparâmetros obtidos	
n_estimators	*100*
learning_rate	*0,01*
max_depth	*10*
min_samples_leaf	*7*
min_samples_split	*7*
acurácia	90,83%

Fonte: o autor

Escalonar ou normalizar os dados não acarretou em melhorias representativas na acurácia do modelo. O *score* obtido, todavia, foi muito bom, acima de 90% - o 3º melhor dos modelos observados até agora – o que indica que o atual modelo é uma boa forma de detectar defeitos superficiais em placas de aço por meio da metodologia corrente empregada.

O *Gradient* Boosting, sendo ensemble, comporta também a apresentação das características mais relevantes. Nessa perspectiva, destacam-se, na Figura 42, a homogeneidade, com 38,7% de importância e o contraste, com 38,3%, como sendo as características mais importantes para o ganho de informação, totalizando 77% do ganho total. Os resultados estão bem concentrados nessas duas características, o que prejudicou a obtenção de uma acurácia final ainda melhor.

Figura 42 - Características mais importantes - *Gradient Boosting*

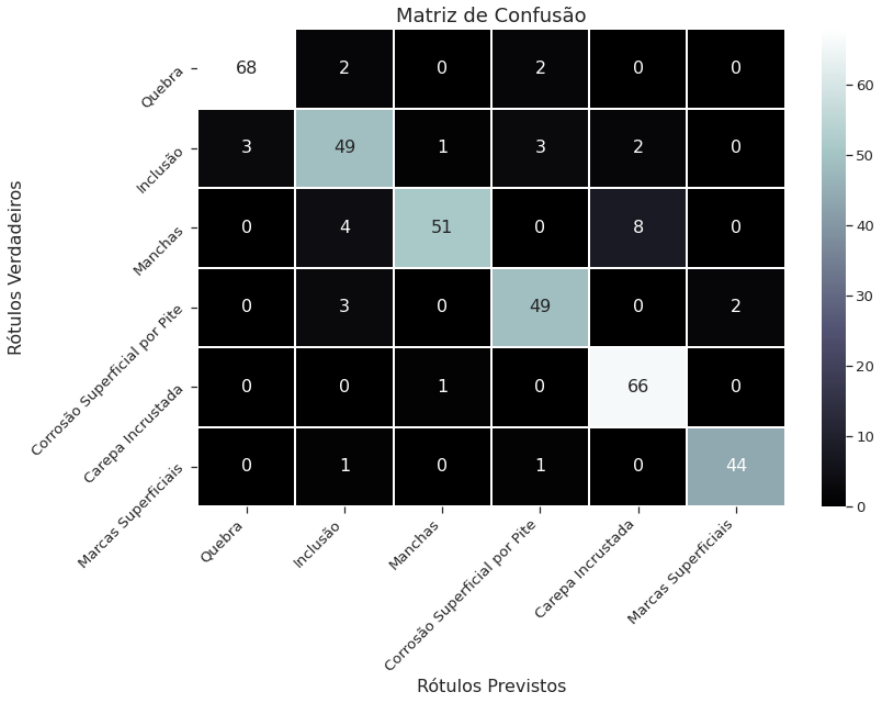


Fonte: o autor

Finalmente, na Figura 43, apresenta-se a matriz de confusão para o *Gradient Boosting*. Nota-se, na diagonal principal, que foram acertados 327 testes de 360 amostras disponíveis. Nesse modelo, as quebras apresentaram o maior número de classificações corretas (68). Percentualmente, as carepas e marcas superficiais ficam a frente, com 99% e 96% de acertos cada uma respectivamente. As classificações incorretas são lideradas, percentualmente, pelas manchas, seguidas das inclusões: no caso das manchas, foram 12 classificações incorretas em 63 tentativas (19% de erro) e, para as inclusões, foram 9 erros em um total de 58 rotulagens (16% de erro).

Com relação aos pares de defeitos com maior número de erros em classificação, destacam-se as manchas e carepas, sendo que o primeiro tipo foi classificado 8 vezes de forma equivocada como sendo o segundo tipo de defeito. A outra dupla de defeito também envolve as manchas, que foram classificadas erroneamente por 4 vezes como sendo inclusões.

Figura 43 - Matriz de confusão para o modelo *Gradient Boosting*



Fonte: o autor

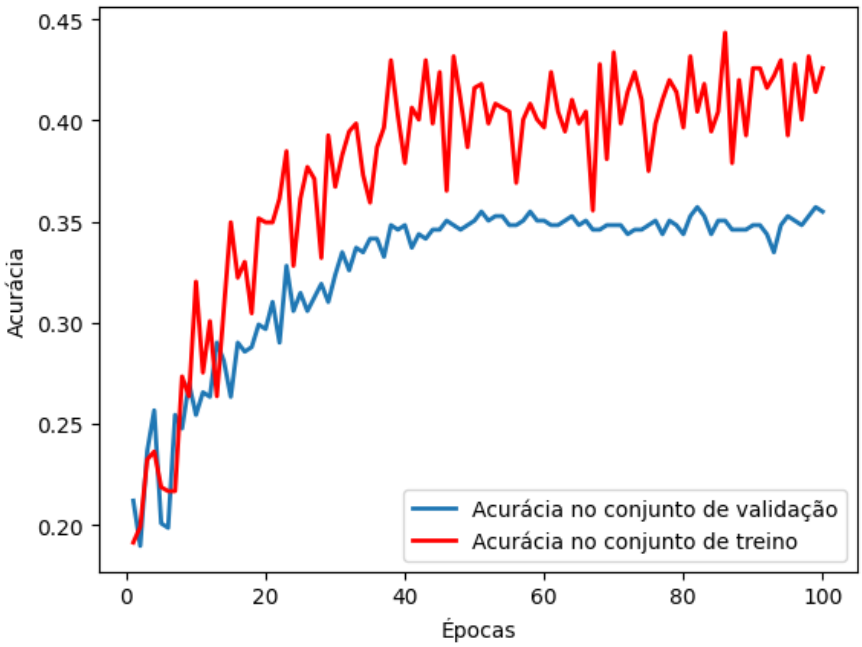
Desta forma, é prudente destacar que o presente modelo não apresenta bom desempenho na rotulação de manchas e, se a caracterização desse defeito for ponto crítico em algum projeto no qual se deseja aplicar modelos de ML, uma vez utilizada a mesma metodologia empregada neste trabalho, o *Gradient Boosting* não é o melhor modelo a ser utilizado.

4.3.7 Redes Neurais Convolucionais

Por fim, o último método empregado foram as redes neurais convolucionais, que obtêm os níveis de cinza de forma direta. Conforme mencionado na seção 3.2, esse método parte do preprocessamento de dados, passando por aplicações de validação cruzada e *Augmentation*. Em todas as etapas, os dados passaram pelas camadas mencionadas na seção 2.3.2.7 e, ao final, foram obtidos 3 resultados: um para o modelo puro, sem aplicações de validação cruzada ou *Augmentation*; outro para o modelo com validação cruzada e o terceiro para o modelo com ambos os tratamentos.

Na sequência é apresentada a Figura 44, referente ao modelo CNN aplicado aos dados sem aplicações de validação cruzada ou *data Augmentation*. No eixo y são apresentadas as acurácias obtidas para o modelo em função das épocas (que são o número de vezes que o modelo é passado, em sua totalidade, em sentido direto e inverso pela *CNN*).

Figura 44 - Acurácia por épocas - sem validação cruzada ou *Augmentation*

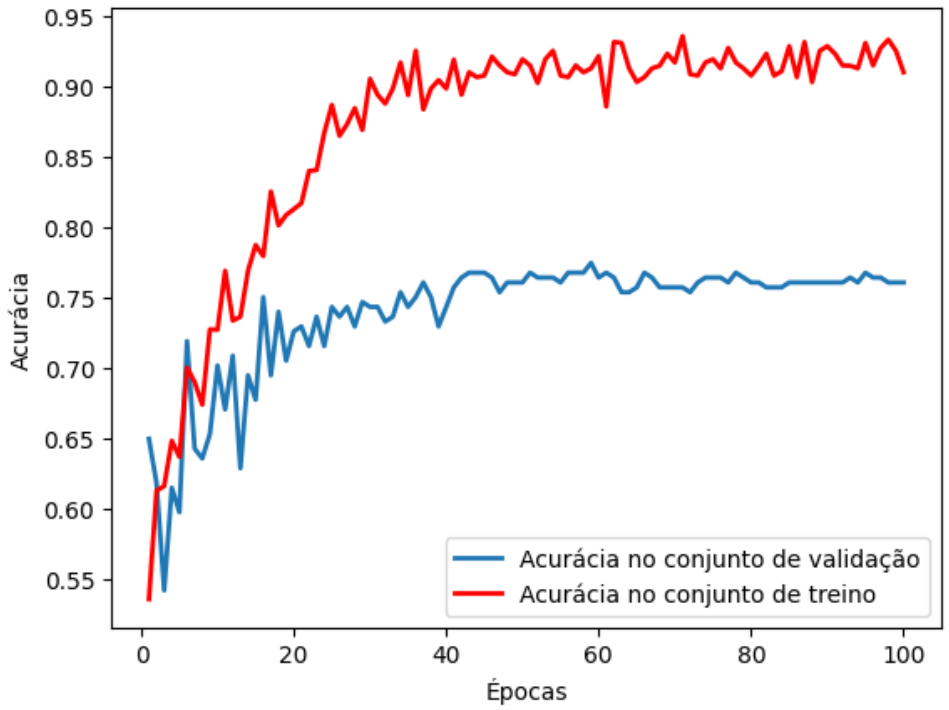


Fonte: o autor

Observe na imagem que a acurácia, tanto no conjunto de treino quanto no de teste, não é satisfatória, ficando em cerca de 40,5% para o primeiro e 35% para o segundo. Esse fato pode ser explicado pela grande oscilação constatada no conjunto de treino ao longo das épocas. Esse comportamento indica que está havendo *overfitting* nos dados e, portanto, sugere que sejam aplicadas técnicas que visem reduzir esse tipo de ocorrência. Nessa perspectiva, a validação

cruzada pode ser uma boa alternativa para a mitigação dessa falha. A técnica foi empregada, com 5 *folds*, e o resultado da aplicação desse método é apresentado na Figura 45.

Figura 45 - Acurácia por épocas - com aplicação de validação cruzada



Fonte: o autor

Note que, de fato, a validação cruzada causou bom impacto no modelo: reduziu a amplitude das oscilações observadas no conjunto de treino e aumentou sua acurácia, que chegou a 92%. Como consequência, houve também aumento no *score* do conjunto de validação que, no achatamento da curva, atingiu o patamar de 76%. Por outro lado, as oscilações no conjunto de treino ainda persistiram, ainda que em patamares menos elevados. O custo computacional para se aumentarem o número de épocas ou até mesmo de *folds* nas validações cruzadas é muito elevado e, como ainda não haviam sido aplicadas técnicas de *data Augmentation* na tratativa do problema, esse foi o próximo passo dado na obtenção dos resultados.

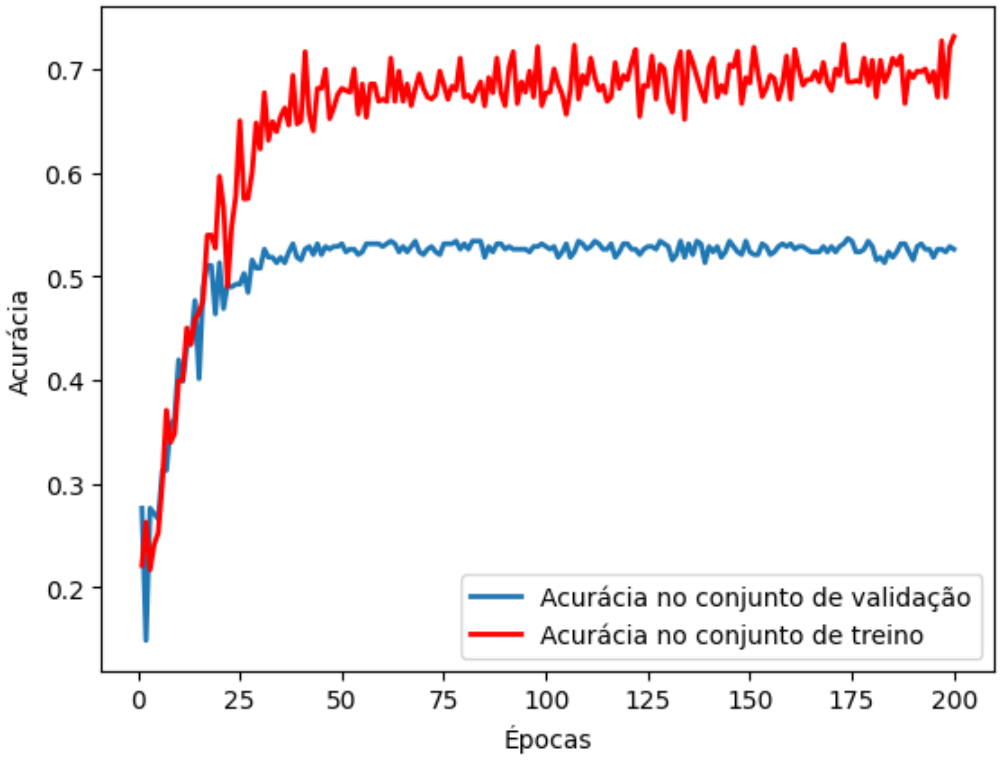
O *Data Augmentation* utilizado gerou 1000 novas amostras com base nas 1800 imagens originalmente presentes no banco de dados. Essas novas imagens foram geradas por meio da aplicação de combinações aleatórias das seguintes operações:

- Adição de ruídos nos pixels de 10% das imagens;
- Modificações aleatórias no contraste de 20% das imagens;
- Alterações randômicas no brilho de 10% das imagens;

- Aleatoriamente, remover 5% dos pixels em 1% das imagens;
- Aplicar aleatoriamente de 0 a 3 três dos procedimentos às imagens:
    - Variação randômica da nitidez em 1%;
    - Inverter horizontalmente 10% das imagens;
    - Rotacionar, de -45º à 45º, 60% das imagens;
    - Dar *zoom* de 5 a 50% em 40% das imagens.

Ao final desse processamento, os dados foram novamente treinados e o resultado é apresentado na Figura 46.

Figura 46 - Acurácia por épocas - com aplicação de validação cruzada e *Data Augmentation*



Fonte: o autor

Conforme observado, a curva manteve as oscilações no conjunto de treino e, além disso, no achatamento, observa-se uma redução da acurácia medida nesse conjunto ($\approx 70\%$) se comparada à do modelo anterior (Figura 45). No que tange ao *score* do modelo, o mesmo ocorreu com o conjunto de validação: observa-se uma redução no valor médio obtido durante a estabilização da curva ($\approx 52\%$). A redução da acurácia observada no conjunto de treino indica que, possivelmente, as operações aplicadas durante o processo de *Augmentation* podem ter dificultado a classificação das imagens pelo modelo, o que se refletiu no conjunto de teste.

Sendo assim, para o modelo das redes neurais convolucionais, o melhor valor de acurácia obtido foi encontrado nas seguintes circunstâncias:

Figura 47 - Parâmetros utilizados nas CNNs

Convolutional Neural Networks	
Camadas aplicadas	Convolucional *Padding* *Relu* *Pooling* *Dropout* *Flatten*
Cross Validation	5
Nº de Épocas	100
Acurácia	76%

Fonte: o autor

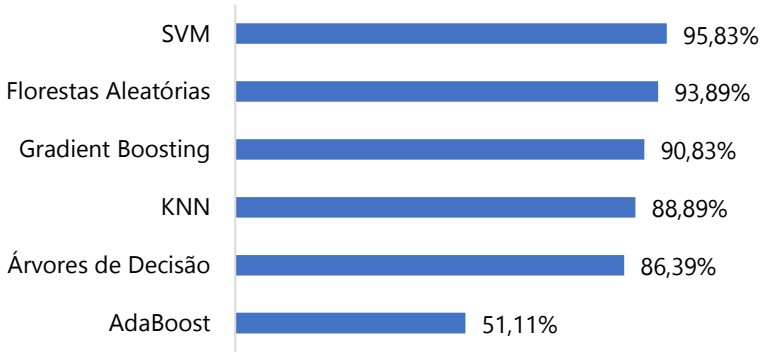
4.3.8 Fechamento dos resultados

Nessa seção, são apresentadas as consolidações dos resultados obtidos ao longo de todo o capítulo 4 do trabalho, bem como as conclusões inferidas dos resultados obtidos ao longo dele.

4.3.9 Métodos diretos

Na Figura 48 são apresentadas as acurácias obtidas para cada um dos modelos diretos estudados. Analisando-se sobre essa perspectiva geral, é possível inferir-se que o modelo que mais apresentou capacidade de generalização e rotulação dentre todos os estudados foram as *SVMs*, que alcançaram acurácia de 95,83%.

Figura 48 - Resumo das acurácias por modelos



Fonte: o autor

Na sequência das *SVMs*, estão as florestas aleatórias e o *Grandient Boosting*, que também apresentaram *scores* superiores a 90%. O *AdaBoost* foi o modelo mais falho na rotulação dos defeitos superficiais em placas de aço, com apenas 51,11% de acurácia. Os resultados da Figura 48 são também uma evidência de que os modelos com algoritmos mais complexos nem sempre são os que apresentam os melhores resultados, muito embora, no geral, os resultados tenham sido muito bons (acima de 85%) para a maioria dos modelos. Os dados podem ser comparados com a literatura por meio de uma tabela encontrada em Ghatnekar (2018) para a previsão dos mesmos defeitos através do mesmo banco de dados:

Figura 49 - Acurácia encontrada na literatura

Classifier	Accuracy (%)
KNN	75.27
AdaBoost	51.11
SVC	14.72
Decision Tree	88.33
Random Forest	89.44
Gradient Boosting	92.50

Fonte: (GHATNEKAR, 2018)

Os resultados são parecidos, muito embora os *SVMs* (ou *SVCs – Support Vector Classifiers*) tenham apresentado uma diferença muito grande, sendo o mais preciso neste trabalho e o menos robusto no de Ghatnekar (2018). Isso pode ter ocorrido devido ao preprocessamento aqui efetuado que, conforme visto na seção 4.3.2, aumentou bastante o desempenho do modelo.

Agora, detalhando-se um pouco melhor os resultados obtidos, na Tabela 20 são apresentados resumos do número de acertos por quantidade total de classificações efetuadas para cada tipo

de defeito. Desta forma, por exemplo, no modelo *KNN*, os dados de testes foram classificados 57 vezes como quebras, sendo 54 dessas vezes rotulados de forma correta.

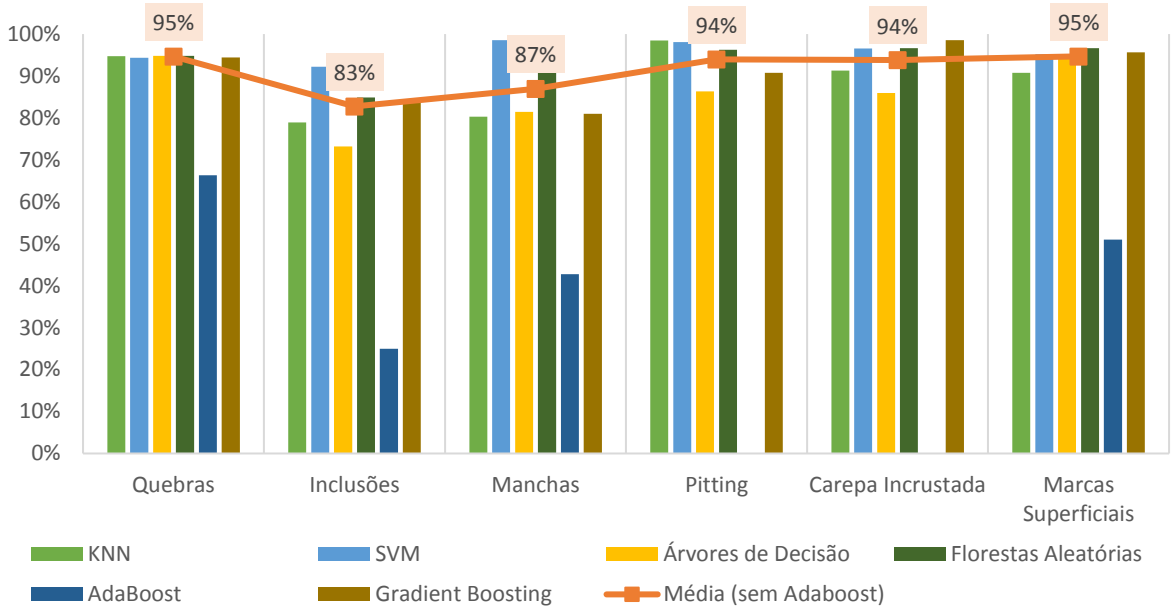
Tabela 20 - Número de Acertos (A) e total de classificações (T) por tipo de defeito

	KNN		SVM		Árvores de Decisão		Florestas Aleatórias		*AdaBoost*		*Gradient Boosting*	
	A	T	A	T	A	T	A	T	A	T	A	T
Quebras	54	57	50	53	55	58	55	58	65	98	68	72
Inclusões	45	57	59	64	41	56	45	53	2	8	49	58
Manchas	57	71	69	70	44	54	71	76	65	152	51	63
Pitting	63	64	52	53	57	66	51	53	0	0	49	54
Carepa Incrustada	42	46	56	58	55	64	58	60	0	0	66	67
Marcas Superficiais	59	65	59	62	59	62	58	60	52	102	44	46

Fonte: o autor

Na Figura 50 é apresentada uma consolidação dos resultados da Tabela 20, agora em percentuais de acertos (A/T). As barras dizem respeito a cada um dos modelos e, a linha é a precisão média do conjunto de classificadores por classe de defeitos (*AdaBoost* não incluso)[8].

Figura 50 - Resumo de acurácia por defeitos



Fonte: o autor

Essa figura corrobora os ótimos números observados para as *SVMs* anteriormente. De fato, para quase todos os tipos de defeito, esse modelo foi o que melhor se aproximou da capacidade de generalização ideal. Apenas nas carepas incrustadas, na qual o *Gradient Boosting* performou em 99% e nas marcas superficiais, que foram previstas mais precisamente pelas frorestas aleatórias (97%), as *SVMs* desempenharam um pouco abaixo. Ainda assim a diferença é muito pequena: 2% em cada caso. Logo, para um projeto que objetiva a detecção de diversos defeitos

[8] Aqui, para o cálculo da média, os resultados obtidos no *AdaBoost* não foram considerados pois o resultado foi muito discrepante se comparado aos demais, conforme pode ser visto na Tabela 20.

superficiais em placas de aço, as *SVMs* se apresentam como fortes candidatas; se por outro lado o objetivo do projeto for, por exemplo, a detecção de carepas, então seria interessante levar em consideração a utilização do *Gradient Boosting*.

Outro ponto a se destacar na Figura 50 é que as inclusões e as manchas foram os defeitos que mais foram classificados de forma equivocada mas, se analisarmos a os rótulos feitos pelas *SVMs*, foram de 92% e 99% de precisão, respectivamente – números muito bons. Isso sugere que alguns tipos de defeitos são mais facilmente detectados por algoritmos específicos e, por isso, é conveniente que diversos modelos sejam testados antes da iniciação de um projeto, por exemplo.

4.3.10 *CNN*

Nas redes neurais convolucionais, a melhor acurácia obtida foi inferior aos resultados vistos nos outros modelos. O valor mais alto encontrado foi de 76%, com acurácia de 92% no conjunto de treino, porém, conforme visto na seção 4.3.2, a curva de acurácia x épocas dos dados de treino apresentou alguns ruídos que podem ter prejudicado os resultados constatados no conjunto de validação. Para contornar esse tipo de problema, tentou-se a aplicação do *Augmentation*, que não foi eficiente. Haveriam outras abordagens a serem testadas, como por exemplo o aumento do númedo de *folds* na validação cruzada; o aumento de épocas ou manter a taxa de aprendizado um pouco mais lenta. Porém, essas propostas exigem um custo computacional muito elevado e, conforme especificado na seção 3, as máquinas utilizadas possuem limitações de processamento que devem ser levadas em conta.

Existem evidências, no entanto de que o modelo pode ser melhorado. Por exemplo, no trabalho de Zhou *et al* (2016), ao se utilizar *Data Augmentation* combinada a um treino com 1000 características e 500 épocas, alcançou-se uma acurácia no teste de 99%. Logo, com maior poder de processamento disponível e diferentes abordagens no processamento dos dados, é possível treinar uma rede neural convolucional para classificar os defeitos em placas superficiais de aço de forma mais efetiva do que o atingido neste trabalho.

5 CONCLUSÃO

Com base nos estudos realizados e nos dados apresentados, conclui-se que os resultados obtidos neste trabalho foram eficientes na classificação de defeitos superficiais em placas de aço, especialmente para os algoritmos de classificação por método indireto. As máquinas de vetores de suporte foram os modelos que apresentaram a maior acurácia, 95,83%, seguidas das florestas aleatórias (93,89%), *Gradient Boosting* (90,83%), *KNN* (88,89%) e árvores de decisão (86,39%). No geral, os modelos indiretos, com exceção do *AdaBoost*, conseguiram rotular de forma satisfatória os defeitos estudados, sendo que as maiores dificuldades ficaram por conta das inclusões e manchas, com 83% e 87% de acertos por tentativas de classificação respectivamente. Ainda assim, mesmo para esses defeitos as *SVMs* desempenharam bem, com 92% de acertos para as inclusões e 99% nas manchas. Comparando-se os resultados dos métodos diretos com aqueles obtidos por Ghatnekar (2018), observa-se grande similaridade, todavia, por não utilizar técnicas de preprocessamento, alguns dos resultados do autor ficaram abaixo, especialmente as *SVMs* (nas quais Ghatnekar (2018) obteve acurácia de apenas 14,72%). Portanto, na média, este trabalho apresentou desempenho superior, com rendimento de 84,49% contra 68,56% do autor.

Com relação ao método indireto, a acurácia obtida foi de 76%, que não é um valor satisfatório sob o ponto de vista da aplicabilidade do modelo. Todavia, em outros trabalhos, como o desenvolvido por Zhou *et al* (2016), o mesmo modelo (as redes neurais convolucionais) apresentou desempenho de 99% no conjunto de validação. Desta forma, é possível concluir-se que o tratamento dos dados, as camadas e parâmetros utilizados neste trabalho não foram os melhores e que fica a possibilidade de melhorias nesses aspectos em trabalhos futuros para que sejam obtidos resultados melhores, ainda que com custo computacional mais elevado.

Por fim, é possível inferir-se que os trabalhos desenvolvidos com a utilização de técnicas de ML vêm ganhando cada vez mais força no mercado (Seção 1) e detectar defeitos superficiais em placas de aço pode ser uma das diversas aplicações que esse conceito eventualmente está e estará desempenhando dentro das indústrias e universidades. A utilização de mecanismos automatizados para esse propósito é de extrema conveniência pois mitiga a possibilidade de falhas humanas e controla estatisticamente os processos por meio do conhecimento acerca de erros cometidos em simulações realizadas previamente e, por isso, o conhecimento em técnicas de ML ganha cada vez mais espaço dentro da academia e também do mercado de trabalho.

REFERÊNCIAS

ALBREGTSEN, Fritz. **Statistical Texture Measures Computed from Gray Level Coocurrence Matrices**, 2008. Disponivel em: https://www.uio.no/studier/emner/matnat/ifi/INF4300/h08/undervisningsmateriale/glcm.pdf. Acesso em: 17 ago. 2020.

AZEVEDO, Paulo Roberto Medeiros de. **Introdução à Estatística**. 3. ed. Natal: edufrn, 2016.

BAPTÍSTA, André Luís de Brito; SOARES, Ângelo Rosestolato; NASCIMENTO, Ivaldo Assis do. **O Ensaio Metalográfico no Controle da Qualidade**. [S.l.]: Spectru Ltda, 2020. Disponivel em: http://www.spectru.com.br/ensaio_metal.pdf. Acesso em: 27 nov. 2020.

BISHOP, Cristopher M. **Pattern Recognition and Machine Learning**. 1ª. ed. New York: Springer, v. I, 2006.

CARVALHO, Taiane Viana de. **Segmentação de Imagens por Texturas Através de um Método de Agrupamento de Dados**. 76f. Dissertação (Mestrado) - Curso de Engenharia Elétrica e de Computação, Universidade Federal do Rio Grande do Norte, Natal, 2018. Disponível em: https://repositorio.ufrn.br/jspui/bitstream/123456789/25963/1/Segmenta%C3%A7%C3%A3oimagenstexturas_Carvalho_2018.pdf. Acesso em: 22 out. 2020.

CASTRO, Geovane Martins. **Estudo da Oxidação a Quente no Aço Inoxidável Ferrítico ABNT 430**. 139 f. Dissertação (Mestrado) - Curso de Engenharia Metalúrgica e de Minas, Universidade Federal de Minas Gerais, Belo Horizonte, 2005. Disponível em: https://ppgem.eng.ufmg.br/defesas/722M.PDF. Acesso em: 29 nov. 2020.

CHAVES, Bruno Butilhão. **Estudo do Algoritmo AdaBoost de Aprendizagem de Máquina Aplicado a Sensores e Sistemas Embarcados**. 137f. Dissertação (Mestrado) - Curso de Engenharia Mecânica, Universidade de São Paulo, São Paulo, 2012. Disponível em: https://teses.usp.br/teses/disponiveis/3/3152/tde-12062012-163740/publico/Bruno_Chaves_Diss_vfrevisada.pdf. Acesso em: 09 nov. 2020.

CLAPPIS, A. M. Uma introdução as redes neurais convolucionais utilizando o Keras. **Data Hackers**, 2019. Disponivel em: https://medium.com/data-hackers/uma-introdu%C3%A7%C3%A3o-as-redes-neurais-convolucionais-utilizando-o-keras-41ee8dcc033e. Acesso em: 01 dez. 2020.

CORROSIONPEDIA. Crazing. **Site da CorrosionPedia**, 2014. Disponivel em: https://www.corrosionpedia.com/definition/343/crazing. Acesso em: 29 nov. 2020.

CORTEZ JUNIOR, Carlos Alberto. **Estudo de modelos de Machine Learning para detecção de defeitos superficiais em placas de aço**. 2020. Disponível em: https://github.com/ccortezjunior/TG_ML_SteelSurfaceDefects. Acesso em: 15 dez. 2020.

DAIDOLA, J. C. *et al*. **Residual Strength Assessment of Pitted Plate Panels**. Washington: Ship Structure Committee, 1995. 91 p. Disponível em: http://www.shipstructure.org/pdf/394.pdf. Acesso em: 25 nov. 2020.

DATA SCIENCE ACADEMY. Casos de Uso de Machine Learning. **Data Science Academy**, 2018. Disponivel em: http://datascienceacademy.com.br/blog/17-casos-de-uso-de-machine-learning/. Acesso em: 14 dez. 2020.

FACELI, K. *et al*. **Inteligência Artificial:** Uma Abordagem de Aprendizado de Máquina. 1ª. ed. Rio de Janeiro: LTC, v. I, 2011.

FÉLIX, Glaudistoni da Silva *et al*. CARACTERIZAÇÃO DO DEFEITO DE SUJEIRA SUPERFICIAL (0SUJ) EM CHAPAS METALICAS DE AÇO - PLTCM. **Abm Proceedings**, [S.L.], v. 74, n. 74, p. 2586-2596, out. 2019. Editora Blucher. http://dx.doi.org/10.5151/2594-5327-33732. Disponível em: https://abmproceedings.com.br/ptbr/article/caracterizao-do-defeito-de-sujeira-superficial-0suj-em-chapas-metalicas-de-ao-pltcm. Acesso em: 17 nov. 2020.

GÉRON, A. **Hands-On Machine Learning with Scikit-Learn and TensorFlow**. 1ª ed. Sebastopol: O'Reilly Media, v. I, 2017.

GHATNEKAR, S. **Use Machine Learning to Detect Defects on the Steel Surface**, 27 jun. 2018. Disponivel em: https://software.intel.com/content/www/us/en/develop/articles/use-machine-learning-to-detect-defects-on-the-steel-surface.html. Acesso em: 17 ago. 2020.

GORNI, Antonio Augusto *et al*. Laminação de Chapas Grossas de Aço ao Níquel. In: CONGRESSO ANUAL DA ABM, 63., 2008, Santos. **Anais [...] .** Santos: Abm, 2008. p. 19-25. Disponível em: http://www.gorni.eng.br/Gorni_CongABM_2008.pdf. Acesso em: 29 nov. 2020.

HAO, Ruiyang. *et al*. A steel surface defect inspection approach towards smart industrial monitoring. **Journal Of Intelligent Manufacturing**, [S.L.], 20 set. 2020. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s10845-020-01670-2.

HOEPPNER, David W.; ARRISCORRETA, Carlos A.. Exfoliation Corrosion and Pitting Corrosion and Their Role in Fatigue Predictive Modeling: state-of-the-art review. **International Journal Of Aerospace Engineering**, [S.L.], v. 2012, p. 1-29, 2012. Disponível em: https://www.hindawi.com/journals/ijae/2012/191879/. Acesso em: 27 nov. 2020.

INFOMET. Aços & Ligas | Aços e Ferros Fundidos | Inclusões Não Metálicas em Aços. **Site da Infomet**, 2020. Disponivel em: https://www.infomet.com.br/site/acos-e-ligas-conteudo-ler.php?codConteudo=228. Acesso em: 27 nov. 2020.

KHOSLA, S. CNN | Introduction to Padding. **GeeksforGeeks**, 2019. Disponivel em: https://www.geeksforgeeks.org/cnn-introduction-to-padding/. Acesso em: 01 dez. 2020.

KOPELIOVICH, D. Pitting Corrosion. **SubsTech - Substances & Technologies**, 2020. Disponivel em: https://www.substech.com/dokuwiki/doku.php?id=pitting_corrosion#:~:text=Pitting%20corrosion%20is%20an%20electrochemical,coated%20with%20a%20passive%20film.&text=Pitting%20corrosion%20is%20highly%20accelerated,present%20in%20the%20electrolyte%20solution. Acesso em: 25 nov. 2020.

LIU, D. A Practical Guide to ReLU. **medium**, 2017. Disponivel em: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7. Acesso em: 02 dez. 2020.

MAYRINK, Vitor Teixeira de Melo. **Avaliação do Algoritmo Gradient Boosting em Aplicações de Previsão de Carga Elétrica a Curto Prazo**. 2016. 91 f. Dissertação (Mestrado) - Curso de Modelagem Computacional, Universidade Federal de Juiz de Fora, Juiz de Fora, 2016. Disponível em: https://www.ufjf.br/pgmc/files/2009/08/Dissertacao_Victor.pdf. Acesso em: 11 nov. 2020.

MENTOURI, Z. et al. Steel strip surface defect identification based on binarized statistical features. **UPB Scientific Bulletin**, Gelma. n. 2. 13 f. Janeiro 2018.

O'BYRNE, M. et al. Texture Analysis based Detection and Classification of Surface Features on Ageing Infrastructure Elements. **Bridge & Concrete Research in Ireland**, Cork, 22 maio 2018. Disponivel em: https://hal.archives-ouvertes.fr/hal-01009012/document. Acesso em: 17 ago. 2020.

OLIVEIRA JUNIOR, Gilson Medeiros de. **Máquina de Vetores Suporte: estudo e análise de parâmetros para otimização de resultado**. 2010. 46 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal de Pernambuco, Pernambuco, 2020. Disponível em: https://www.cin.ufpe.br/~tg/2010-2/gmoj.pdf. Acesso em: 02 nov. 2020.

OLIVEIRA, Márcia Silva de. **Análise de Textura em Imagens Cerebrais: aplicações em Acidente Vascular Cerebral Isquêmico, Epilepsia Mioclônica Juvenil, Doença de Machado-Joseph, Déficit Cognitivo Leve e Doença de Alzheimer**. 2010. 145 f. Tese (Doutorado) - Curso de Física, Universidade Estadual de Campinas, Campinas, 2010. Disponível em: http://repositorio.unicamp.br/jspui/handle/REPOSIP/277836. Acesso em: 23 nov. 2020.

PAWAR, S. et al. Study of White Patch Defect in Automotive Grade Interstitial Free Steel. **Journal of Failure Analysis and Prevention**, Jamshedpur, v. 19, p. 6, set. 2020.

PUC-RIO. KNN (K – Nearest Neighbors). **Site da PUC-Rio**, 2020a. Disponivel em: <https://www.maxwell.vrac.puc-rio.br/7587/7587_6.PDF>. Acesso em: 27 out. 2020.

PUC-RIO. Máquinas de Vetor Suporte. **Site da PUC-Rio**, 2020b. Disponivel em: https://www.maxwell.vrac.puc-rio.br/9191/9191_3.PDF. Acesso em: 31 out. 2020.

PUC-RIO. Árvore de Decisão. **Site da PUC-Rio**, 2020c. Disponivel em: https://www.maxwell.vrac.puc-rio.br/7587/7587_4.PDF. Acesso em: 04 nov. 2020.

RIBEIRO JUNIOR, João Rangel. **Reciclagem em Cerâmica Vermelha de Pó de Carepa da Indústria Siderúrgica**. 2019. 98 f. Dissertação (Mestrado) - Curso de Engenharia e Ciência dos Materiais, Universidade Estadual do Norte Fluminense, Campos dos Goytacazes, 2019. Disponível em: http://uenf.br/posgraduacao/engenharia-de-materiais/wp-content/uploads/sites/2/2019/10/Disserta%C3%A7%C3%A3o-Final.pdf. Acesso em: 29 nov. 2020.

SCIKIT-LEARN. Nearest Neighbors. **Site do scikit-learn**, 2020a. Disponivel em: https://scikit-learn.org/stable/modules/neighbors.html#classification. Acesso em: 27 out. 2020.

SCIKIT-LEARN. cross-validation: evaluating estimator performance. **site do sci-kit learn**, 2020b. Disponivel em: https://scikit-learn.org/stable/modules/cross_validation.html. Acesso em: 16 nov. 2020.

SCIKIT-LEARN. sklearn.tree.DecisionTreeClassifier. **Site do scki-kit learn**, 2020c. Disponivel em: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. Acesso em: 21 nov. 2020.

SCIKITS-IMAGE. **Module- feature.texture (greycomatrix)**, 2020. Disponivel em: https://scikit-image.org/docs/0.7.0/api/skimage.feature.texture.html. Acesso em: 21 out. 2020.

SONG, K.; YAN, Y. NEU surface defect database. **Site da Northeastern University**, 2020. Disponivel em: http://faculty.neu.edu.cn/yunhyan/NEU_surface_defect_database.html. Acesso em: 26 out. 2020.

TAKAHASHI, Adriana. **Máquinas de Vetores Suporte Intervalar**. 2012. 72 f. Tese (Doutorado) - Curso de Engenharia Elétrica, Universidade Federal do Rio Grande do Norte, Natal, 2012. Disponível em: https://repositorio.ufrn.br/jspui/bitstream/123456789/15225/1/AdrianaT_TESE.pdf. Acesso em: 31 out. 2020.

TEIXEIRA, Everton Franco; FERNANDES, Sandro Roberto. Desenvolvimento de uma ferramenta computacional para classificação de padrões em imagens. **Seminários de Trabalho de Conclusão de Curso do Bacharelado em Sistemas de Informação**, Juiz de Fora, v. 1, n. 1, p. 1-15, 2015. Disponível em: http://periodicos.jf.ifsudestemg.edu.br/revistabsi/article/view/27/24. Acesso em: 22 out. 2020.

TENSORFLOW. Data Augumentation. **Site do TensorFlow**, 2020. Disponivel em: https://www.tensorflow.org/tutorials/images/data_augmentation. Acesso em: 17 nov. 2020.

THE MATHWORKS, INC. Graycomatrix (Image Processing Toolbox User's Guide). **Site do Matlab**, 2020a. Disponivel em: http://matlab.izmiran.ru/help/toolbox/images/graycomatrix.html. Acesso em: 17 ago. 2020.

THE MATHWORKS, INC. Graycoprops (Image Processing Toolbox User's Guide). **Site do Matlab**, 2020b. Disponivel em: http://matlab.izmiran.ru/help/toolbox/images/graycoprops.html. Acesso em: 17 ago. 2020.

THE MATHWORKS, INC. Using a Gray-Level Co-Occurrence Matrix (GLCM): Analysing and Enhancing Images (Image Processing Toolbox User's Guide). **Site do Matlab**, 2020c. Disponivel em: http://matlab.izmiran.ru/help/toolbox/images/enhanc15.html. Acesso em: 17 ago. 2020.

TIAN, S.; XU, K. An Algorithm for Surface Defect Identification of Steel Plates Based on Genetic Algorithm and Extreme Learning Machine. **Metals**, Pequim, v. 7, p. 11, Julho 2017.

VASCONCELOS, Simone. Insituto de Computação - UFF. **Matrizes de Co-Ocorrência**, 2020. Disponivel em: www.ic.uff.br/~aconci/co-ocorrencia.pdf. Acesso em: 21 out. 2020.

WITTEN, I. H. et al. **Data Mining:** Practical Machine Learning Tools and Techniques. 4ª. ed. Cambridge: Elsevier, v. I, 2016.

ZERBST, U. et al. Defects as a root cause of fatigue failure of metallic components. III: Cavities, dents, corrosion pits, scratches. **Elsevier**, Fukuoka, v. 97. 18 f. Janeiro 2019.

ZHOU, S. et al. Classification of Surface Defects on Steel Sheet Using Convolutional Neural Networks. **Materials and Technology**, Wuhan, v. LI. 9 f. Jan. 2016. Disponivel em: http://mit.imt.si/Revija/izvodi/mit171/zhou.pdf. Acesso em: 05 dez. 2020.