UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

BRUNO HENRIQUE NASCIMENTO DE ANDRADE

FERRAMENTAS DE TESTE PARA SISTEMAS BASEADOS EM APRENDIZADO DE MÁQUINA: UM ESTUDO EMPÍRICO

CAMPINA GRANDE - PB
2020

BRUNO HENRIQUE NASCIMENTO DE ANDRADE

FERRAMENTAS DE TESTE PARA SISTEMAS BASEADOS EM APRENDIZADO DE MÁQUINA: UM ESTUDO EMPÍRICO

Trabalho de Conclusão de Curso apresentado a Universidade Estadual da Paraíba, como requisito parcial à obtenção do título em bacharel em Computação.

Área de concentração: Teste de Software.

Orientadora: Profa. Dra. Sabrina de Figueirêdo Souto

CAMPINA GRANDE - PB
2020

A553f Andrade, Bruno Henrique Nascimento de.
Ferramentas de teste para sistemas baseados em aprendizado de máquina [manuscrito] : um estudo empírico / Bruno Henrique Nascimento de Andrade. - 2020.
83 p. : il. colorido.
Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2020.
"Orientação : Profa. Dra. Sabrina de Figueirêdo Souto , Coordenação do Curso de Computação - CCT."
1. Inteligência artificial. 2. Aprendizado profundo. 3.Deeplearning. 4. Redes neurais. I. Título
21. ed. CDD 006.3

Elaborada por Giulianne M. Pereira-CRB-15/714 BC/UEPB

BRUNO HENRIQUE NASCIMENTO DE ANDRADE

FERRAMENTAS DE TESTE PARA SISTEMAS BASEADOS EM APRENDIZADO DE MÁQUINA: UM ESTUDO EMPÍRICO

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 27 de Fevereiro de 2020.

Sabrina de F. Souto.
Profa. Dra. Sabrina de Figueirêdo Souto (DC - UEPB)
Orientador(a)

Kézia de Vasconcelos Oliveira Dantas
Profa. Dra. Kézia de Vasconcelos Oliveira Dantas (DC - UEPB)
Examinador(a)

Luciana de Queiroz Leal Gomes
Profa. MSc. Luciana Queiroz (DC - UEPB)
Examinador(a)

“Se eu fosse contar pra vocês tudo que aconteceu na minha história até eu chegar nesse momento, talvez vocês nem acreditassem, as vezes nem eu acredito. E de verdade hoje eu só quero agradecer a mim, [...] porque eu não desisti!”

Anitta, Rock in Rio 2019

AGRADECIMENTOS

Não sei nem por onde começar. Essa frase ficou presa aqui durante semanas enquanto a ideia e sua execução eram feitas. Com esse tempo refleti sobre muita coisa que aconteceu durante esses quatro anos que convive dentro da universidade. Eu era outra pessoa, tímida e com medo do mundo, toda essa experiência vivida transformaram toda a minha essência e abriram meus olhos para um mundo muito maior do que eu jamais havia imagino dentro do meu quarto lá no sertão.

Eu agradeço primeiramente a toda a minha família que me deu apoio todo esse tempo, mesmo durante as dificuldades e quando eu sei que não podia sempre davam um jeito. Minha mão que entre trancos e barrancos sempre colocou o bem de seus filhos ao invés do seu próprio dedico todo meu amor e esforço. Ao meu pai que durante minha empreitada na universidade não foi tão presente por conta da saúde, mas do jeito que pode esteve sempre comigo. A minha irmã e eterna primeira amiga, que mesmo diante a distância sempre foi aquela disposta a escutar todos os meus problemas e dar o ombro para me apoiar.

Eu agradeço aos professores, não vou mentir que pelos primeiros períodos o choque de realidade com a experiência que eu tinha foi grande. Mas no decorrer do caminho eu fui encontrando aqueles que não só queria que eu crescesse, mas pegavam a minha mão e me impulsionavam e acreditavam no meu potencial. Não quero especificar nada, mas sempre existem aqueles que marcam o nosso caminho. Agradeço a Edson pela extrema paciência e comprometimento com os alunos. Agradeço a Luciana pelas aulas incríveis e por tamanha empatia por seus alunos. Agradeço a Scherer por me ensinar que o caminho é árduo sim, mas com esforço você consegue vencer todo desafio. Agradeço a Janderson por ser tão comprometido e conosco enquanto alunos. Agradeço a Kézia que apesar de conhecer a relativamente pouco tempo, com sua didática me cativou e incentivou a crescer. Agradeço a Fred por reavivar em mim o interesse que me fez ingressar no curso para começo de história, a inteligência artificial. Para não me alongar, agradeço a Sabrina, minha orientadora que aguentou minhas loucuras durante todo esse tempo, ela que me abriu os olhos no decorrer do curso para outras áreas que eu nem conhecia e acabei me apaixonando, obrigado por estar sempre me ajudando e acreditando no potencial de seus alunos, a senhora fez toda a diferença acredite.

Chegando aos amigos, eu nunca pensei que iria firmar os laços que eu firmei quando entrei no começo do curso. Desde o começo eu sabia que por ser um curso difícil, não eram

todos que estavam ali naquela sala que iriam até o final, nem mesmo eu sabia se conseguiria. A surpresa foi as amizades firmadas no decorrer dessa jornada, eu já falei para essas pessoas e retorno a dizer aqui, se não fossem por elas como suporte durante todo esse tempo, eu não teria conseguido. Agradeço a manu, que apesar de nem sempre sermos tão unidos, é aquele tipo de pessoa que eu sei que posso contar e sempre vai estar lá pra me escutar. Agradeço a vitor, uma pessoa que apareceu de surpresa na minha vida e que agradeço até hoje por nossa amizade, obrigado por sempre escutar minhas paranoias e dar um norte nos meus pensamentos. Para fechar nosso grupo, falta falar um pouco de carol, eu agradeço por ela ser minha companhia nessa jornada, minha outra metade que descobri durante o curso, minha eterna dupla. Agradeço a todos vocês que de alguma parte fizeram parte da minha jornada e me ajudaram a crescer como pessoa, cada uma tem um espaço no meu coração, tenham total certeza disso.

E que venham novos desafios, novas pessoas e novas experiências. Afinal, qual seria a graça de crescer se não tivéssemos todas essas coisas para percorrer não é mesmo? A você que está lendo isso dou um conselho, acredite em você e que você pode fazer tudo que você se propor a fazer. Os outros não podem fazer isso no seu lugar, cabe a você se levantar e correr atrás do que você deseja, eu aprendi isso da maneira mais difícil. Sonhe grande e saiba que um dia os resultados chegam.

RESUMO

O *Deep Learning* (DL) define um novo paradigma de programação orientado a dados, em que a lógica interna do sistema é amplamente moldada pelos dados de treinamento. A maneira padrão de avaliar modelos de DL é examinar seu desempenho em um conjunto de dados de teste. A qualidade do conjunto de dados de teste é de grande importância para obter confiança nos modelos treinados. Usando um conjunto de dados de teste inadequado, os modelos de DL que atingiram alta precisão de teste ainda podem não ter generalidade e robustez. No entanto, devido à diferença fundamental entre software tradicional e software baseado em *deep learning*, as técnicas tradicionais de teste de software não podem ser aplicadas diretamente a esses sistemas. Dito isto, com a disponibilidade de várias ferramentas e bibliotecas de código aberto que usam o conceito de DL, os desenvolvedores hoje em dia podem programar facilmente suas aplicações apenas fazendo uso dessas interfaces de programação de aplicativos (APIs) de aprendizado sem conhecer os detalhes do algoritmo. Modelos de DL são notoriamente difíceis de interpretar e depurar. No entanto, os proprietários de ferramentas e bibliotecas de DL geralmente possuem mais atenção à correção e funcionalidade de seu algoritmo, gastando muito menos esforço em manter seu código livre de bugs e com um nível de alta qualidade. Considerando a popularidade do aprendizado de máquina no mundo de hoje, as ferramentas e bibliotecas de DL podem ter um enorme impacto em produtos que usam algoritmos dessa tecnologia. Portanto, nesta monografia, seu objetivo é mostrar que diferentes abordagens de se testar essas ferramentas são importantes para garantir um produto final mais seguro. Como resultado desse estudo, criamos uma rotina de testes que se mostrou viável com resultados confiáveis. Uma comparação de técnicas de teste White e Black box também foi feita para mostrar os pontos positivos e negativos do uso em sistema *deep learning*. O desenvolvimento da abordagem ainda permite que seja aplicada e utilizada em qualquer sistema que se assemelhe ao que foi trabalho nesta pesquisa, contribuindo positivamente para a inteligência artificial sob a ótica de testes focados para sistemas de aprendizado profundo.

Palavras-Chave: Inteligência Artificial. Aprendizado Profundo, Testes de Software.

ABSTRACT

Deep Learning (DL) defines a new data-oriented programming paradigm, in which the internal logic of the system is largely shaped by training data. The standard way to evaluate DL models is to examine their performance against a set of test data. The quality of the test data set is of great importance to obtain confidence in the trained models. Using an inadequate test data set, DL models that have achieved high test accuracy may still lack generality and robustness. However, due to the fundamental difference between traditional software and software based on deep learning, traditional software testing techniques cannot be applied directly to these systems. That said, with the availability of several open source tools and libraries that use the DL concept, developers today can easily program their applications just by using these learning application programming interfaces (APIs) without knowing the details of the algorithm. DL models are notoriously difficult to interpret and debug. However, owners of DL tools and libraries generally pay more attention to the correctness and functionality of their algorithm, spending much less effort on keeping their code bug-free and at a high-quality level. Considering the popularity of machine learning in today's world, DL tools and libraries can have a huge impact on products that use algorithms for this technology. Therefore, in this monograph, your objective is to show that different approaches to testing these tools are important to ensure a safer final product. As a result of this study, we created a test routine that proved to be feasible with reliable results. A comparison of White and Black box test techniques was also made to show the positive and negative points of using in deep learning system. The development of the approach still allows it to be applied and used in any system that resembles what was done in this research, contributing positively to artificial intelligence from the perspective of tests focused on deep learning systems.

Keywords: Artificial Intelligence, Deep Learning, Software testing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Desde o surgimento da inteligência artificial (IA) em meados dos anos 50 ela segue numa evolução constante, suas grandes áreas seguindo uma linha do tempo são: IA, *Machine Learning* e *Deep Learning*. E cada área originou diversas subáreas cada uma com seu próprio objetivo e finalidade. O objetivo principal da IA é fazer com que nossos computadores consigam se comportar como pessoas, adquirindo a habilidade de pensar por si próprio e fazer decisões sem auxilio externo, a partir disso o próximo passo é fazer com que ele possa aprender sozinho (RUSSEL et al., 2013).

É sabido então, que ao desenvolver um programa voltado para IA é necessário ter um conhecimento em algoritmos de classificação, saber qual método de busca é o melhor para determinado problema, além de uma boa base matemática. O aprendizado de máquina (*Machine Learning* - ML) é cada vez mais implantado em sistemas críticos e de larga escala, graças às inovações recentes na área. Agora, cada vez mais aplicativos de softwares desenvolvidos por ML em aspectos críticos de nossas vidas diárias são usados nas áreas de: finanças, energia, saúde e transporte. No entanto, detectar e corrigir falhas nos programas de ML ainda é mais desafiador do que sistemas tradicionais, conforme evidenciado pelo incidente de carro do Uber que resultou na morte de um pedestre (THEGUARDIAN, 2018). A principal razão por trás da dificuldade de testar programas de ML é a mudança no paradigma de desenvolvimento induzido por ML e IA.

Tradicionalmente, os sistemas de software são construídos dedutivamente, anotando as regras que governam o comportamento do sistema como código de programa. No entanto, com o ML, essas regras são inferidas a partir de dados de treinamento (ou seja, são gerados indutivamente). Essa mudança de paradigma no desenvolvimento de aplicativos dificulta o raciocínio sobre o comportamento de sistemas de software com componentes ML, resultando em sistemas intrinsecamente difíceis de testar e verificar, uma vez que eles não possuem especificações (completas) ou mesmo código-fonte correspondente a alguns de seus comportamentos críticos.

De suas áreas citadas, o aprendizado profundo (*Deep Learning* – DL), vem ganhando muito espaço por possibilitar o desenvolvimento de programas muito mais inteligentes e flexíveis no sentido de aprendizado do que nas outras áreas. Essa área aborda diretamente os problemas lidar com um banco muito grande de dados, como fazer o melhor uso deles de forma que o resultado seja confiável com um menor gasto computacional possível.

Substituindo muitas vezes o trabalho manual, fazendo com que a ferramenta possa lidar com os problemas futuros e possa responder a novas situações. Em DL, a preocupação maior é saber montar o cenário da sua aplicação, pois a partir do seu modelo e da rede neural ele sozinho gera as saídas do programa.

O processo de se testar um software consiste em um conjunto de atividades que irão garantir qualidade de um produto, ou seja, aumentando a confiabilidade no mesmo. Embora na atualidade existem diversos tipos de teste, as atividades que os gerenciam são basicamente as mesmas, com foco em: planejar, monitorar, controlar, analisar, modelar, implementar, executar e concluir o teste (BSTQB, 2019). Com IA não é diferente, muitas vezes seguimos esse mesmo processo, mas com o comportamento singular do sistema, o resultado desses tipos de teste muitas vezes não pode ser validado, visto que o comportamento não pode ser estimado como ocorre no teste tradicional.

Portanto, para sistemas DL críticos, assim como o software tradicional, devem ser testados sistematicamente em diferentes casos para detectar e corrigir idealmente possíveis falhas ou comportamentos indesejados. Este cenário pode ficar mais complexo para o teste automatizado e sistemático de sistemas DL, de larga escala, com milhares de neurônios e milhões de parâmetros para todos os possíveis casos de teste, o que é extremamente desafiador e custoso.

Nesse contexto, testar esse tipo de sistema se torna mais difícil por sua especificidade em resolver o que ele se propõe, seja classificar uma imagem, um texto ou mesmo um arquivo de áudio. Tendo em vista que o modelo possui várias subpartes, o estudo sobre o teste nessa área é importante porque existem várias abordagens como: Dados (SEKHON et al., 2019), Implementação (HINTON et al., 2012), (XIE et al., 2018), White-Box (CHEN et al., 2015), (MA et al., 2018), Black-Box (GULSHAN et al., 2016), propondo diferentes formas de comprovar o modelo.

Garantir que os sistemas de aprendizagem de máquina sejam seguros e confiáveis é de suma importância, pois esses tipos de sistemas geralmente são críticos para a segurança (GOODFELLOW et al., 2014). Ao mesmo tempo, testar esses tipos de sistema é extremamente difícil, visto que o comportamento não é 100% garantido. De acordo com o estado da arte e com o estado da pratica, existem algumas abordagens que tentam tratar alguns dos aspectos citados anteriormente (BRAIEK et al., 2018).

Porém, ainda existe uma grande escassez de técnicas e ferramentas de teste para sistemas de aprendizagem de máquina. Considerando os desafios de testar sistemas de

aprendizagem de máquina, essa pesquisa tem o objetivo de aumentar a confiabilidade nesses sistemas, propondo e desenvolvendo abordagens de teste para atacar os problemas apontados.

Dada a grande necessidade de garantir que esses sistemas são confiáveis para uso, diversas técnicas de teste foram desenvolvidas para melhorar a acurácia do sistema e garantir para o usuário que ele é seguro para uso (ZHANG et al., 2019). Sendo assim, foram surgindo muitas abordagens diferentes para testar esses modelos complexos que utilizam de DL para trabalhar. A partir do momento que mais atenção foi dada a esse ambiente e que os devidos testes não estavam sendo suficientes devido à complexidade do sistema, os processos de testes no decorrer da evolução das ferramentas foram melhorando e mudando seu foco de validação no sistema. Com a pesquisa bibliográfica foi possível dividir esses testes em quatro abordagens distintas: o teste nos dados, na implementação, teste utilizando uma abordagem black-box e após isso surgiu o teste white-box com as métricas de teste tudo focado para ML.

Visto que durante esse desenvolvimento novas formas de teste iam surgindo e o progresso da IA sempre aumentando, questionamentos foram surgindo se as novas abordagens faziam as devidas verificações que eram realizadas no início do processo ou seguiam o processo a partir do avanço na área.

A partir disso, surgiu a indagação que se as ferramentas seguissem uma rotina de testes em todas as subpartes do sistema, seria modificado o resultado final e se melhoraria sua confiabilidade ou não. Para tentar verificar essa ideia, temos as seguintes questões de pesquisa referente a essa monografia:

QP1 - *Quais são as principais estratégias utilizadas para testar sistemas baseados em ML?*

QP2 - *Quais são as principais características das ferramentas de teste para sistemas baseados em ML?*

QP3 - *Quão eficientes são as ferramentas de teste de sistemas baseados em ML?*

QP4 - *Quão eficazes são as ferramentas de teste de sistemas baseados em ML?*

O objetivo geral deste trabalho é analisar ferramentas de teste para DL a fim de apresentar alternativas que realizem testes de maneira mais efetiva para esse tipo de software. Para isso, foi proposta uma abordagem que faz uso de uma sequência de testes que deve ser seguida no processo de validação para garantir uma maior acurácia do sistema.

Com base no objetivo geral, os objetivos específicos são:

- Realizar pesquisa bibliográfica para mapear quais as principais abordagens de testes para ML/DL;
- Propor um estudo para testar sistemas DL com ferramentas de teste selecionadas;
- Verificar a eficiência e eficácia das ferramentas White e Black Box através de um estudo comparativo.

1.1 Estrutura da monografia

A monografia segue a estrutura descrita pelos capítulos citados a seguir:

- Capítulo 2: fornece embasamento teórico para compreensão do problema contextualizado neste trabalho;
- Capítulo 3: descreve a revisão bibliográfica feita para o estudo dessa pesquisa;
- Capítulo 4: mostra o estudo empírico realizado em duas ferramentas de teste em DL;
- Capítulo 5: apresenta as conclusões assumidas após a problematização, seu contexto e a aplicação da proposta como solução e suas contribuições para o meio de teste de software.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção tem como objetivo estabelecer uma base teórica para a compreensão de conceitos que são fundamentais para o entendimento do contexto a ser abordado na pesquisa realizada. Inicialmente serão apresentados os conceitos relacionados à Inteligência Artificial; *Machine Learning*; *Deep Learning*; Redes Neurais e por fim, Testes de software, na qualidade do processo.

2.1 Inteligência Artificial

A IA oficialmente nasceu na conferência de verão de 1956 em Dartmouth College, NH, USA. Na proposta dessa conferência, escrita por McCarthy (Dartmouth), Marvin Minsky (Hardward), Nathaniel Rochester (IBM) e Claude Shannon (Bell Laboratories) e submetida à fundação Rockfeller, consta a intenção dos autores de realizar "um estudo durante dois meses, por dez homens, sobre o tópico inteligência artificial". Ao que tudo indica, esta foi a primeira menção oficial à expressão "Inteligência Artificial" (INFINITOMAIZUM, 2009).

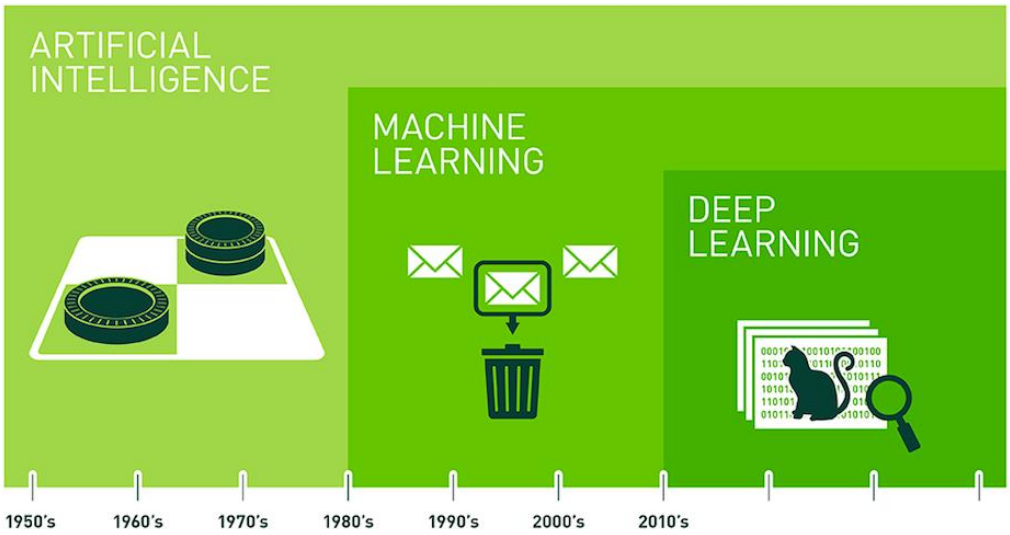
Na conferência de 1956 o sonho dos pioneiros da IA era de construir máquinas complexas — possibilitadas por computadores que emergiam na época — que possuíssem as mesmas características da inteligência humana. Esse é o conceito definido como "IA genérica" — máquinas que tem todos os nossos sentidos, toda a nossa razão e pensam como nós pensamos. O que foi feito se encaixa no conceito de "IA limitada". Tecnologias que são capazes de executar tarefas específicas tão bem quanto, ou até melhor, que nós humanos conseguimos. Exemplos de IA limitadas são tecnologias como: classificação de imagens em um serviço como o Pinterest ou reconhecimento de rostos no Facebook. (MEDIUM, 2016).

Nos últimos anos a popularidade de IA cresceu exponencialmente, especialmente desde 2015. Ela é o novo fator de produção e tem potencial de iniciar novas fontes de crescimento. Isso muda a maneira de trabalhar e reforça o papel das pessoas para impulsionar o crescimento comercial. Uma pesquisa da Accenture (ACCENTURE, 2016) sobre o impacto da IA em 12 economias, revela que ela poderá duplicar as taxas de crescimento econômico anual até 2035. Isso altera a natureza do trabalho e cria uma nova relação entre o homem e a máquina. A previsão é que o impacto das tecnologias de IA sobre o setor empresarial

aumentará a produtividade da força de trabalho em até 40% e permitirá a otimização do tempo por parte das pessoas.

Na Figura 1 é mostrada uma forma de visualização simples para entender as áreas da IA como círculos concêntricos, tendo a IA como a mais à esquerda (a ideia que veio primeiro), *machine learning* (que floresceu depois), e finalmente *deep learning* (que está liderando o mercado de IA hoje) contendo parte das duas.

Figura 1 – Evolução da Inteligência Artificial



Fonte: https://medium.com/data-science-brigade/a-diferen%C3%A7a-entre-intelig%C3%AAncia-artificial-machine-learning-e-deep-learning-930b5cc2aa42.

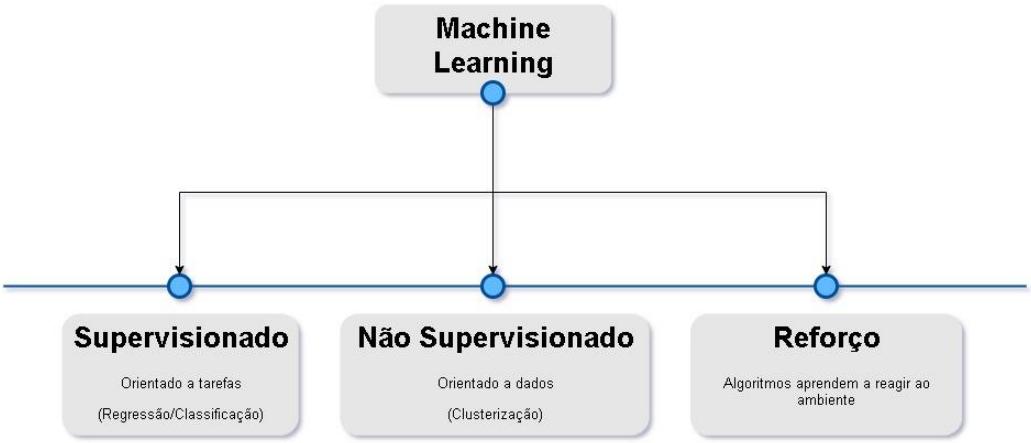
2.1.1 *Machine Learning*

O termo surgiu pela década dos anos 60, com o pioneiro da IA Arthur Samuel do MIT. Ele descreveu o conceito como o campo que ia possibilitar aos computadores a habilidade de aprender sem terem sido programados para tal (CANALTECH, 2017).

De acordo com Naqa e Murphy (2015), ML é um ramo em evolução de algoritmos computacionais projetados para emular a inteligência humana, aprendendo com o ambiente circundante. Eles são considerados o futuro na nova era do chamado *big data*. Técnicas baseadas no aprendizado de máquina foram aplicadas com sucesso em diversos campos, desde reconhecimento de padrões, visão computacional, engenharia de espaçonaves, finanças, entretenimento e biologia computacional a aplicações biomédicas e médicas.

Nesse contexto, o ambiente de ML cresceu muito e foi se subdividindo. De acordo com a Figura 2, vemos seus principais subcampos: Aprendizado Supervisionado, Não Supervisionado e por Reforço.

Figura 2 – Subdivisão de *Machine Learning*



Fonte: Elaborado pelo autor (2019).

2.1.1.1 Aprendizado Supervisionado

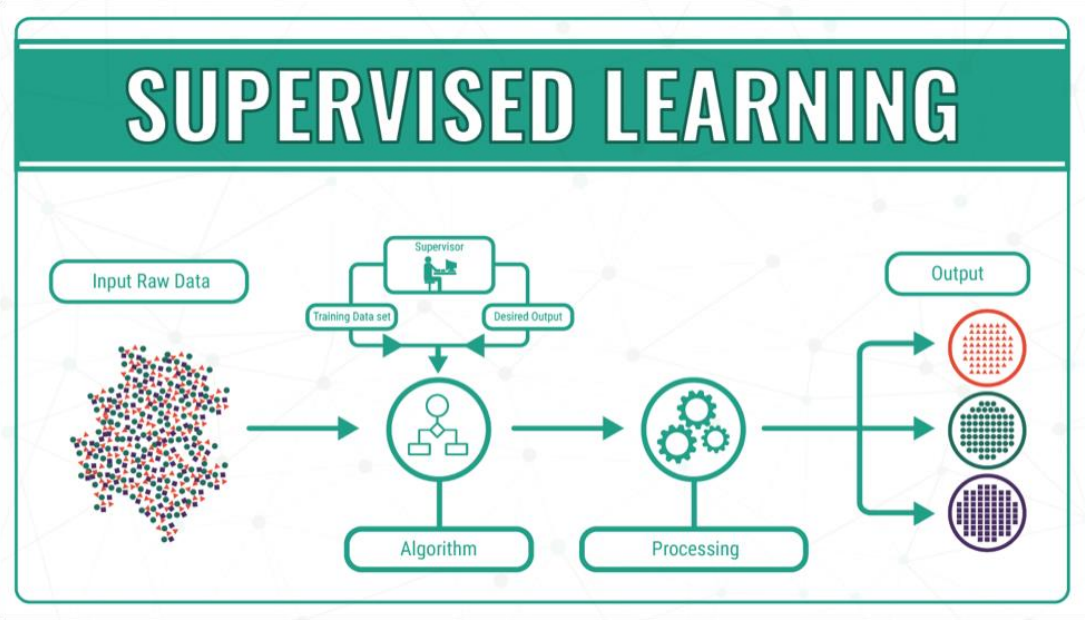
Essa área consiste basicamente da seguinte ideia: dado um conjunto de dados rotulados que já sabemos qual é a saída correta e que este deve ser semelhante ao conjunto, sabendo que existe uma relação entre a entrada e a saída, podemos predizer se uma classificação está correta ou errada.

Problemas de aprendizagem supervisionados são classificados em problemas de "regressão" e "classificação". Em um problema de regressão, estamos tentando prever os resultados em uma saída contínua, o que significa que estamos a tentando mapear variáveis de entrada para alguma função contínua. Em um problema de classificação, estamos tentando prever os resultados em uma saída discreta. Em outras palavras, estamos tentando mapear variáveis de entrada em categorias distintas (MEDIUM, 2016).

Sua execução é dividida em duas fases. A primeira fase (denominada fase de aprendizado) analisa um conjunto de dados de treinamento, que consiste em vários exemplos, cada um com vários valores de atributos e um rótulo. O resultado dessa análise é um modelo que tenta fazer generalizações sobre como os atributos se relacionam com o rótulo. Na segunda fase, o modelo é aplicado a outro conjunto de dados não visto anteriormente (os

dados de teste) em que os rótulos são desconhecidos. Na Figura 3 é exemplificado como funcionam essas duas fases para classificação de um dado, na parte superior onde vemos o supervisor com os dados de treino e o que se espera da saída, a partir de sua junção é gerado o modelo (algoritmo). Os dados de entrada mais à esquerda, nossos dados de teste, vão fazer contato com esse algoritmo e é a partir desse processo que conseguimos gerar nossa saída como mostra a figura.

Figura 3 – Fluxo de processos do Aprendizado Supervisionado



Fonte:https://datafloq.com/read/machine-learning-explained-understanding-learning/4478.

Em um algoritmo de classificação, o sistema tenta prever o rótulo de cada exemplo individual; em um algoritmo de ranking, a saída dessa fase é uma classificação tal que, quando os rótulos se tornam conhecidos, pretende-se que os rótulos de maior valor estejam no topo ou perto do topo do ranking, com os rótulos de menor valor na parte inferior ou perto dele (MURPHY et al., 2007).

2.1.1.2 Aprendizado Não Supervisionado

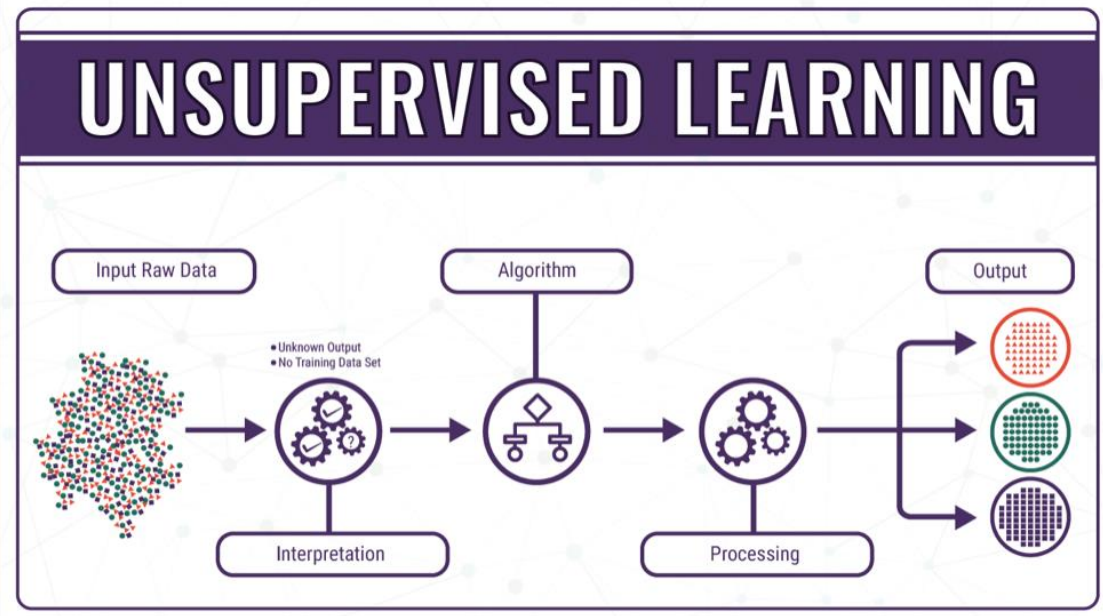
O aprendizado não supervisionado é o treinamento de um algoritmo de IA usando informações que não são classificadas nem rotuladas, com o objetivo de modelar a estrutura ou distribuição subjacente em um conjunto de dados. Esse tipo de aprendizado não requer conhecimento prévio e pode ser amplamente utilizado em uma ampla variedade de aplicações,

como segmentação de mercado para clientes-alvo, detecção de anomalias ou fraudes no setor bancário, *clustering* de atributos para agrupar e classificar genes, derivando índices climáticos de dados de ciências da terra e agrupamento de documentos com base no conteúdo. A eficácia do ML não supervisionado depende em grande parte do uso de um algoritmo de *clustering* não supervisionado apropriado e / ou de seu programa correspondente (XIE et al, 2018).

Seguindo o pensamento de Xie et al. (2018), no aprendizado de máquina não supervisionado, o *clustering* é uma técnica para dividir um grupo de objetos de dados em *clusters*, de modo que os objetos de dados no mesmo *cluster* sejam "semelhantes" uns aos outros; enquanto objetos de dados de diferentes *clusters* mostram recursos "distintos" um do outro. Como o *cluster* tenta descobrir padrões ocultos nos dados sem conhecimento prévio, é difícil avaliar a correção ou a qualidade dos resultados do *cluster*.

Na Figura 4 vemos como funciona o fluxo de uma classificação não supervisionada. Não temos controle da saída nem possuímos um *dataset* de treino. Seguindo os passos, o algoritmo irá ter que classificar sem nenhuma base de comparação e gerar a saída com o que ele acha mais correto. Apesar de ser um método que demore um pouco para gerar resultados satisfatórios, ele é muito menos custoso computacionalmente que o supervisionado visto anteriormente.

Figura 4 – Fluxo de processos do Aprendizado Não Supervisionado



Fonte:https://datafloq.com/read/machine-learning-explained-understanding-learning/4478.

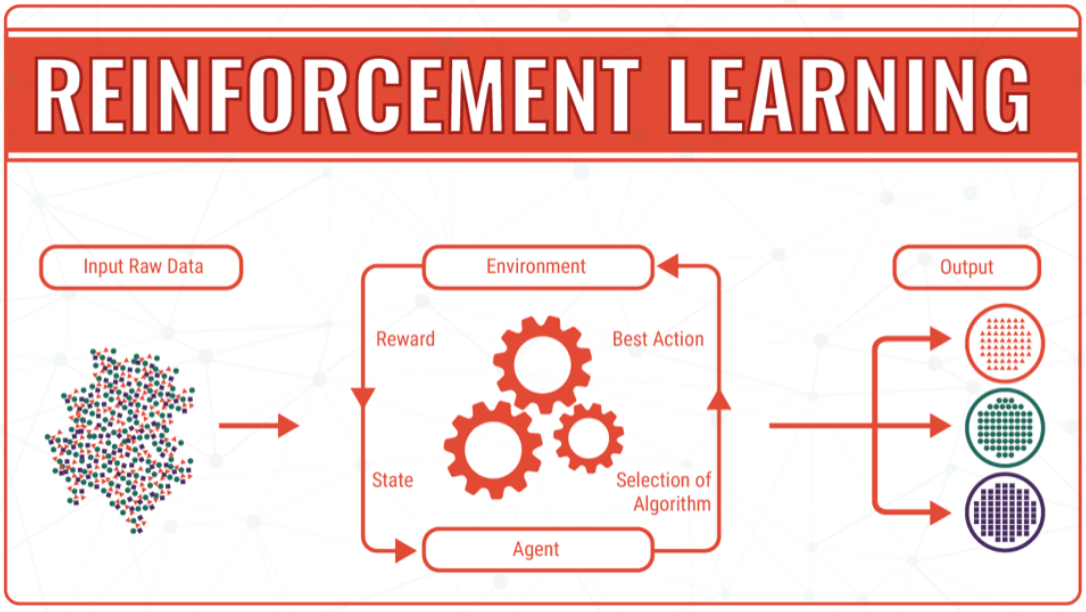
2.1.1.3 Aprendizado por Reforço

Aprendizagem por reforço é diferente das abordagens anteriores porque não há conjunto de treinamento, rotulado ou não. Essa é uma área de aprendizagem de máquina que investiga como agentes de software devem agir em determinados ambientes de modo a maximizar alguma noção de recompensa cumulativa (STATSBOT, 2017).

Podemos exemplificar: imagine um robô em um lugar desconhecido. Ele pode realizar atividades e receber recompensas do ambiente pelas atividades realizadas. Após cada ação seu comportamento ficará mais complexo e inteligente, ou seja, ele está treinando para se comportar de modo mais efetivo a cada instante. Em biologia, isto é chamado de adaptação ao ambiente natural.

Na Figura 5 podemos ver claramente como isso funciona tendo como base o exemplo acima, o agente na imagem seria o nosso robô, a partir do algoritmo e os dados de entrada ele vai selecionar qual será a melhor ação a se tomar dado o ambiente em que se encontra. Dependendo da ação tomada, ele pode ganhar uma recompensa e vai aprendendo com seus acertos e erros.

Figura 5 – Fluxo de processos do Aprendizado por Reforço



Fonte:https://datafloq.com/read/machine-learning-explained-understanding-learning/4478.

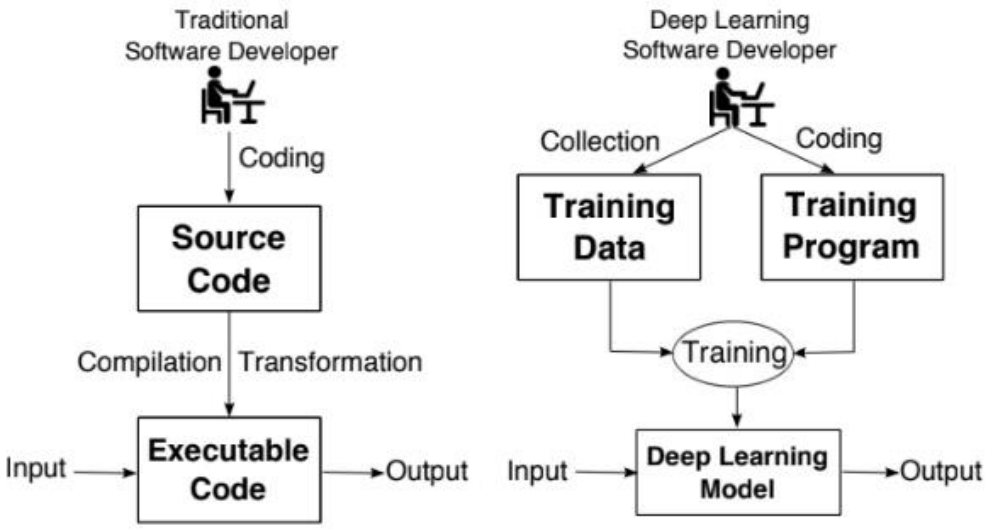
2.2 Deep Learning

Ma et al. (2018) nos diz que o DL define um novo paradigma de programação orientado a dados que constrói a lógica interna do sistema de uma rede de neurônios criada por meio de um conjunto de dados de treinamento.

Os sistemas DL (LECUN et al, 2015) alcançaram um progresso significativo em muitos domínios, incluindo reconhecimento de imagem (FARABET et al., 2013), (KRIZHEVSKY et al., 2012), (SZEGEDY et al., 2015), reconhecimento de fala (HINTON et al., 2012) e tradução automática (JEAN et al., 2015), (SUTSKEVER et al., 2014). Com base em sua capacidade de igualar ou superar o desempenho humano, os sistemas de DL estão sendo cada vez mais adotados como parte de sistemas maiores em domínios críticos de segurança e proteção, como condução autônoma (BOJARSKI et al., 2016), (CHEN et al., 2015) e detecção de malware(CUI et al., 2018).

Diferentemente dos sistemas de software tradicionais, dos quais a lógica de decisão é frequentemente implementada pelos desenvolvedores de software na forma de código, o comportamento de um sistema DL é determinado principalmente pela estrutura das redes neurais profundas (*Deep Neural Networks* - DNNs) e pelos pesos de conexão na rede. Especificamente, pelos seus pesos que são obtidos através da execução do programa de treinamento no conjunto de dados de treinamento, onde as estruturas DNN são frequentemente definidas por fragmentos de código do programa de treinamento em linguagens de alto nível (por exemplo, Python (ABADI et al., 2016), (CHOLLET et al., 2015) e Java (GIBSON et al, 2016)). Portanto, o conjunto de dados de treinamento e o programa de treinamento são duas fontes principais dos sistemas de DL (MA et al., 2018). Na Figura 6 temos uma comparação simples do software tradicional e o DL em relação ao seu desenvolvimento.

Figura 6 – Comparação entre o desenvolvimento tradicional e com DL



Fonte: MA et al. (2018).

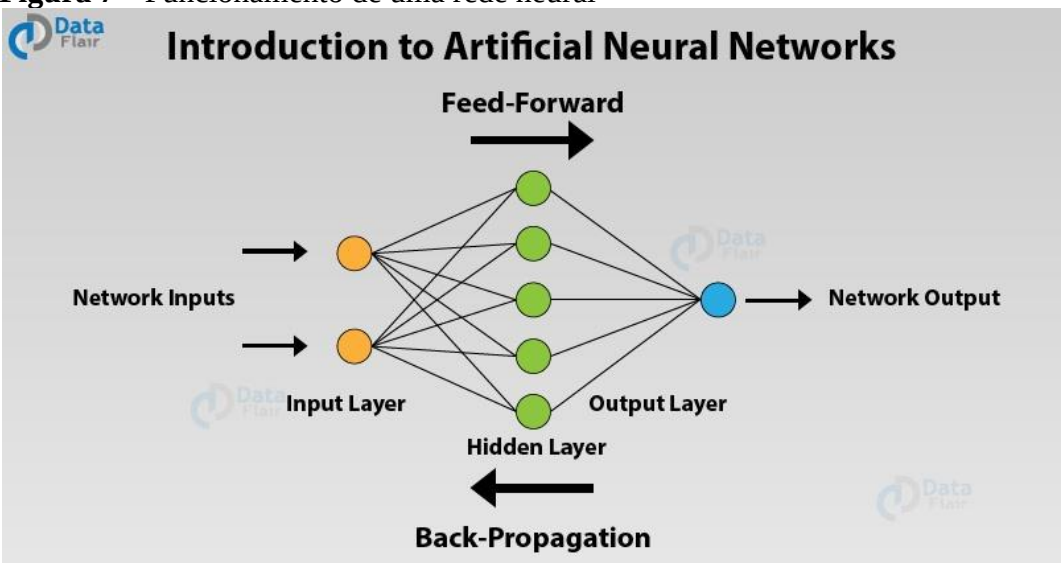
2.2.1 *Redes Neurais*

As redes neurais estão gradualmente sendo usadas em mais contextos que afetam vidas humanas, inclusive para diagnóstico médico (GULSHAN et al., 2016), em veículos autônomos (ANGELOVA et al., 2015), (BOJARSKI et al., 2016), (HUVAL et al., 2015), como entradas nos processos corporativos e judiciais (BERK et al., 2017), (SCARBOROUGH et al., 2006), no controle do tráfego aéreo (KATZ et al., 2017) e no controle da rede elétrica (SIANO et al., 2012). As redes neurais têm o potencial de transformar essas aplicações, salvando vidas e oferecendo benefícios a mais pessoas do que seria possível com o trabalho humano. No entanto, antes que isso possa ser alcançado, é essencial garantir que as redes neurais sejam confiáveis quando usadas nesses contextos.

Redes neurais são implementadas como uma sequência de multiplicações de matrizes seguida por operações elementares. A implementação de software subjacente dessas operações pode conter muitas instruções de ramificação, mas muitas delas são baseadas no tamanho da matriz e, portanto, na arquitetura da rede neural, de modo que o comportamento de ramificação é principalmente independente dos valores específicos da entrada da rede neural. Uma rede neural é executada em várias entradas diferentes, portanto, geralmente executa as mesmas linhas de código e assume as mesmas ramificações, mas produz variações interessantes de comportamento devido a alterações nos valores de entrada e saída (ODENA et al., 2018). De forma simples, a Figura 7 nos mostra o funcionamento de uma rede neural,

no qual mais à esquerda é onde ela recebe nossos dados de entrada que passa para a camada de entrada (círculos amarelos), dela passa para a camada oculta (círculos verdes) e o cálculo das matrizes é feito para nos passar a última camada, a de saída (círculo azul).

Figura 7 – Funcionamento de uma rede neural



Fonte:https://data-flair.training/blogs/artificial-neural-networks-for-machine-learning/.

2.2.1.1 Tipos de Redes Neurais

As redes neurais são modelos computacionais que funcionam de maneira semelhante ao funcionamento do sistema nervoso humano. Esse tipo de rede é implementado com base nas operações matemáticas e em um conjunto de parâmetros necessários para determinar a saída.

Existem diferentes tipos de redes neurais, cada tipo usa princípios diferentes para determinar suas próprias regras, cada uma com suas singularidades. Alguns tipos são mais recorrentes e serão explanados a seguir:

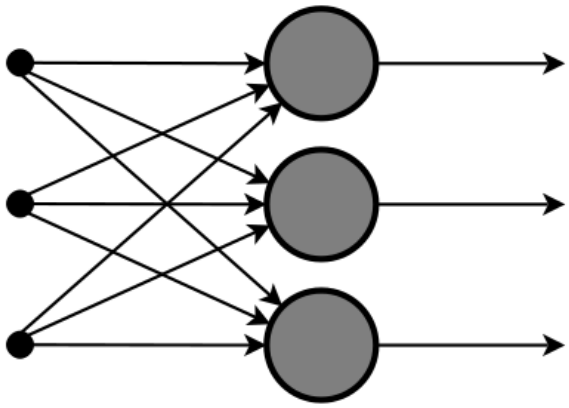
- ***Feedforward***

Este tipo é um dos mais simples de redes neurais artificiais. Em uma rede neural de *feedforward*, os dados passam pelos diferentes nós de entrada até atingir o nó de saída. Em outras palavras, os dados se movem em apenas uma direção a partir da primeira camada até atingir o nó de saída. Isso também é conhecido como onda propagada frontal, que geralmente é alcançada usando uma função de ativação classificadora. Ao contrário dos tipos mais

complexos de redes neurais, não há retropropagação e os dados se movem em apenas uma direção. Uma rede neural *feedforward* pode ter uma única camada ou camadas ocultas (DIGITALVYDIA, 2018).

Em uma rede neural *feedforward*, a soma dos produtos das entradas e seus pesos são calculados. Este é então alimentado para a saída. Na Figura 8 podemos ver um exemplo de uma rede neural *feedforward*. A figura 8 nos mostra os ligamentos dos neurônios em uma rede *feedforward*, eles sempre possuem um fluxo seguindo da camada de entrada para a camada de saída para gerar nosso resultado.

Figura 8 – Rede Neural Feedforward



Fonte:https://www.digitalvidya.com/blog/types-of-neural-networks/.

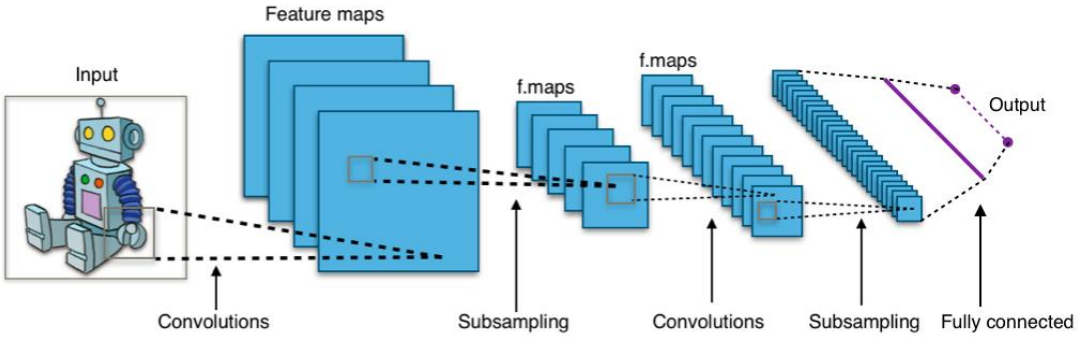
Esse tipo de rede é caracterizado para lidar com dados que contêm muito ruído. As redes neurais *feedforward* também são relativamente simples de manter.

- ***Convolutional***

Uma rede neural convolucional (CNN) usa uma variação em multicamadas, podendo ser completamente interconectadas ou agrupadas. Antes de passar o resultado para a próxima camada, a camada convolucional usa uma operação na entrada. Devido a esta operação, a rede pode ser muito mais profunda, mas com muito menos parâmetros. Com essa capacidade, as CNNs mostram resultados muito eficazes no reconhecimento de imagens e vídeos, processamento de linguagem natural e sistemas de recomendação. Elas também apresentam ótimos resultados na análise semântica e na detecção de paráfrase, sendo aplicados no processamento de sinais e classificação de imagens (DIGITALVYDIA, 2018).

As CNNs também estão sendo usadas na análise e reconhecimento de imagens na agricultura, onde as características climáticas são extraídas de satélites como o LSAT para prever o crescimento e o rendimento de um pedaço de terra. Na Figura 9 temos uma imagem do fluxo de uma CNN.

Figura 9 – Rede Neural Convolucional

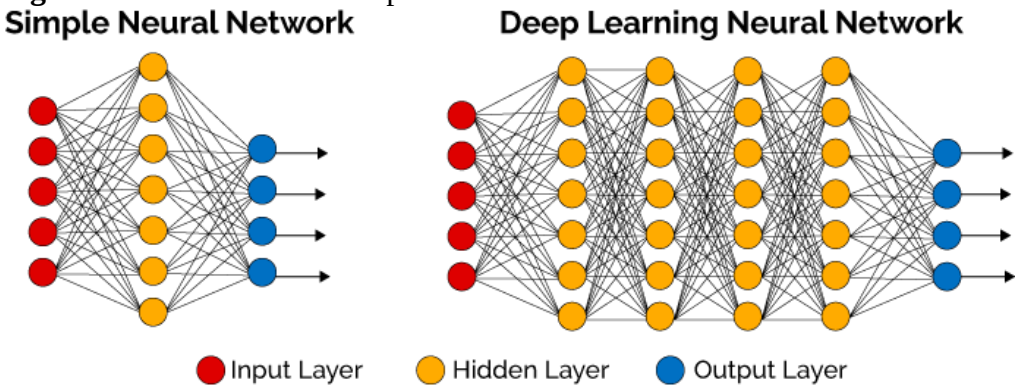


Fonte:https://towardsdatascience.com/introducing-convolutional-neural-networks-in-deep-learning-400f9c3ad5e9

2.2.2 *Deep Neural Network*

Uma DNN é uma rede com várias camadas de neurônios artificiais entre as camadas de entrada e saída (SCHMIDHUBER, 2015). Ela codifica o mapeamento matemático das entradas às saídas, o que aproxima a distribuição oculta nos dados de treinamento, com uma composição em cascata de funções simples implementadas pelos neurônios. Devido à sua natureza aproximada, aos possíveis dados tendenciosos do treinamento e ao excesso ou mal encaixe no processo de treinamento, uma DNN pode se comportar mal em algumas entradas. Portanto, ela precisa ser suficientemente testada antes de ser colocado em uso. Um comparativo simples é mostrando na Figura 10 entre uma rede neural simples e uma profunda, no qual sua principal diferença é o número de camadas ocultas, se a rede possuir mais de uma camada oculta, ela é classificada com profunda.

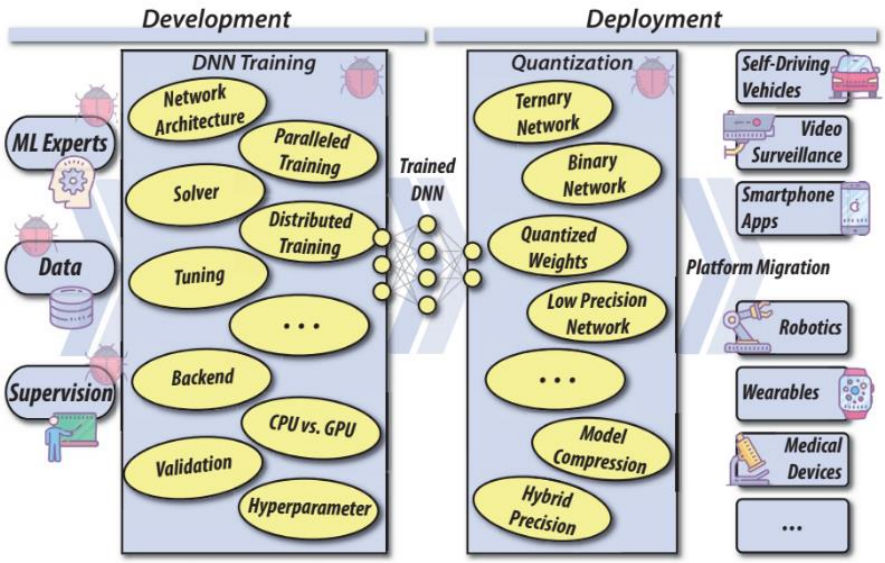
Figura 10 – Rede Neural Simples e Rede Neural Profunda



Fonte:http://deeplearningbook.com.br/o-que-sao-redes-neurais-artificiais-profundas/.

Diferentemente da simples, uma rede DNN possui um custo computacional bem maior para executar as operações internas. Tudo vai depender de como a rede se comporta e de quantas camadas possui. Algumas ferramentas que utilizam essa tecnologia funcionam apenas com utilização de GPUs e processamento na nuvem, se não demoraria muito tempo para executar seus processos.

Figura 11 – Processo de Desenvolvimento e Implantação de um software baseado em DNN



Fonte: XIE et al. (2018).

De acordo com a Figura 11, podemos acompanhar o processo completo para desenvolver uma DNN. É preciso primeiramente os especialistas que irão lidar com o sistema e o montante de dados que irá ser utilizado para treinamento e teste. Tendo em mão esses dados, é começado o processo de treinamento da rede que envolve esses vários processos citados no lado esquerdo da imagem. Após treinar entra a parte de quantizar a rede, fazendo

os vários processos para implantar e verificar o modelo criado. No fim o sistema é migrado para seu objetivo final.

2.3 Teste de Software

Podemos definir como teste de software o ato de verificar através de uma execução controlada se o desempenho equivale ao especificado, com intenção de revelar o máximo de falhas com o mínimo de esforço (IEEE, 1990). Um teste é eficaz quando este tem uma elevada probabilidade de revelar um erro ainda não descoberto, eles são utilizados para garantir a qualidade do sistema, aumentando por consequência a sua confiança.

Entretanto, apesar de revelarem falhas no software, as atividades de teste não possuem a capacidade de mostrar a ausência delas, só podem certificar se erros existem e estão presentes (DIJKSTRA, 1972). Seria impraticável verificar todas as possibilidades de combinações de testes em um software e validar que ele está em estado perfeito, além do que as atividades de desenvolvimento são executadas por pessoas, e pessoas são passíveis de falhas.

Nesta atividade, o principal objetivo é projetar testes que descubram sistematicamente diferentes classes de erros e que sejam executados em uma quantidade de tempo e esforços mínimos. Se a atividade de teste for conduzida com sucesso, ela descobrirá erros no software, que são imprescindíveis (TOMELIN, 2001).

Ao longo da história da computação, é bastante comum que softwares e sistemas sejam entregues à operação e, devido à presença de defeitos, subsequentemente causar falhas ou não atender às necessidades dos *stakeholders*. No entanto, com o uso de técnicas de teste apropriadas pode-se reduzir a frequência de tais entregas problemáticas, quando essas técnicas são aplicadas com o nível apropriado de experiência em testes, e nos pontos certos do ciclo de vida de desenvolvimento de software (BSTQB, 2019).

2.3.1 *Black-box*

As técnicas de teste caixa-preta (também chamadas de técnicas comportamentais ou baseadas no comportamento) são fundamentadas em uma análise da base de teste apropriada (p.e., documentos de requisitos formais, especificações, casos de uso, estórias de usuários ou processos de negócios). Essas técnicas são aplicáveis a testes funcionais e não funcionais. As

técnicas de teste caixa-preta se concentram nas entradas e saídas do objeto de teste sem referência a sua estrutura interna (BSTQB, 2019).

As características comuns das técnicas de teste caixa-preta incluem o seguinte:

• As condições de teste, os casos de teste e os dados de teste são derivados de uma base de teste que pode incluir requisitos de software, especificações, casos de uso e estórias de usuários;

• Os casos de teste podem ser usados para detectar lacunas entre os requisitos e as suas implementações, bem como desvios nos requisitos;

• A cobertura é medida com base nos itens testados na base de teste e na técnica aplicada nesta base;

2.3.2 *White-box*

As técnicas de teste caixa-branca (também chamadas de técnicas estruturais ou baseadas na estrutura) são baseadas em uma análise da arquitetura, do detalhamento do projeto, da estrutura interna ou o código do objeto de teste. Ao contrário das técnicas de teste caixa-preta, as técnicas de teste caixa-branca concentram-se na estrutura e no processamento dentro do objeto de teste (BSTQB, 2019).

As características comuns das técnicas de teste caixa-branca incluem o seguinte:

• As condições de teste, os casos de teste e os dados de teste são derivados de uma base de teste que pode incluir código, arquitetura de software, o detalhamento do projeto ou qualquer outra fonte de informação relacionada à estrutura do software;

• A cobertura é medida com base nos itens testados em uma estrutura selecionada (p.ex., o código ou interfaces) e a técnica aplicada à base de teste.

2.4 Teste em *Machine Learning*

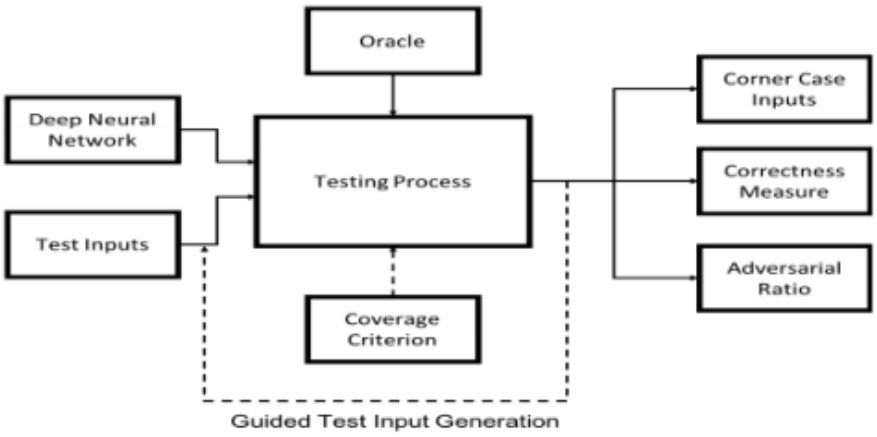
Os aspectos do problema de teste para sistemas ML são compartilhados com soluções já conhecidas dos testes tradicional, amplamente estudadas na literatura de engenharia de software. No entanto, a natureza estatística dos sistemas de aprendizado de máquina e sua capacidade de tomar decisões autônomas levantam questões de pesquisa adicionais e desafiadoras para testes de software (PACULA, 2011), (RAMANATHAN et al, 2016).

Os testes em ML apresentam desafios que surgem da natureza e construção fundamentalmente diferentes dos sistemas de aprendizado de máquina, em comparação com

os sistemas de software tradicionais (relativamente mais determinísticos e menos orientados estatisticamente). Por exemplo, um sistema ML segue inerentemente um paradigma de programação orientado a dados, em que a lógica de decisão é obtida por meio de um procedimento de treinamento a partir de dados de treinamento sob a arquitetura do algoritmo de aprendizado de máquina (AMERSHI et al., 2019). O comportamento do modelo pode evoluir ao longo do tempo, em resposta ao fornecimento frequente de novos dados (AMERSHI et al., 2019). Embora isso também ocorra nos sistemas de software tradicionais, o comportamento subjacente principal de um sistema tradicional normalmente não muda em resposta a novos dados, da maneira que um sistema ML pode.

Testar ML também sofre com um problema bem dificultoso: O Problema do Oráculo (BARR et al., 2015). Os sistemas de aprendizado de máquina são difíceis de testar porque são projetados para fornecer uma resposta a uma pergunta para a qual não existe resposta anterior (MURPHY et al., 2007). Como Davis e Weyuker (1981) nos dizem, não teríamos necessidade de escrever programas ML, se a resposta correta já fosse conhecida. Grande parte da literatura sobre teste em sistemas ML procura encontrar técnicas que possam solucionar o problema do oráculo (*oracle problem*), geralmente recorrendo às abordagens tradicionais de teste de software e tentando adaptá-las.

Figura 12 – Representação alto-nível de como funciona a maioria dos testes em DNN



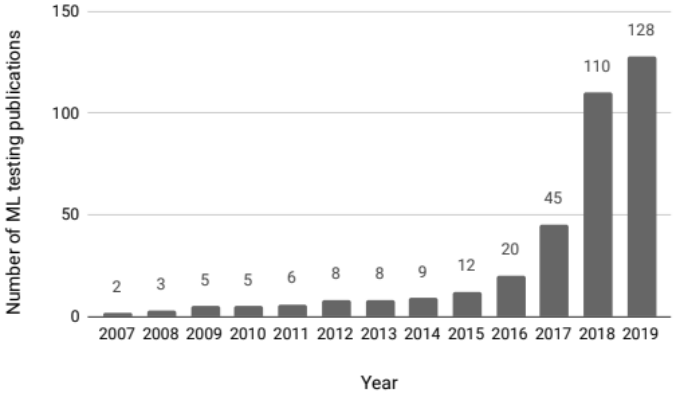
Fonte: SEKHON et al. (2019).

De acordo com a imagem da Figura 12, vemos que o processo de teste em DNN é complicado. Nesses sistemas precisamos de duas coisas para conseguir validar sua confiabilidade: os dados de teste e os critérios de cobertura. Os dados de teste (*Test Set*) irão nos servir como *Oracle* (Oráculo) no sistema, após gerar o modelo com os dados de

treinamento. O *Test Set* funcionara como *oracle* e auxiliara no *output* da rede neural. Os critérios de cobertura foram criados para garantir uma eficácia maior da rede neural que está sendo utilizada, com eles é possível fazer uma verificação das camadas internas da rede para sabermos quais neurônios estão sendo ativados dado o *dataset* que executamos com o algoritmo. Isso nos permite mapear onde, quando acontece algum comportamento errôneo, em qual parte da rede isso aconteceu e aumentando assim a confiabilidade do sistema.

Dado esse contexto, os sistemas de aprendizado de máquina são às vezes considerados como software não testável. Com esses desafios, foi notado um progresso considerável e um notável aumento no interesse e na atividade. A Figura 13 mostra o número acumulado de publicações sobre o tópico de testar sistemas ML entre 2007 e junho de 2019. A partir dessa figura, podemos ver que 85% dos artigos foram publicados desde 2016, testemunhando o surgimento de um novo domínio de interesse de teste de software: teste de aprendizado de máquina.

Figura 13 – Publicações de teste de aprendizado de máquina durante 2007-2019



Fonte: ZHANG et al. (2019).

2.4.1 *Técnicas de Teste para ML*

Como já foi dito anteriormente, existe uma diferença clara de testar um software tradicional e testar um software ML. Além do problema do oráculo já citado, surgem diversos outros problemas quando diz respeito a teste em ML. Após pesquisa em mais de quarenta artigos relacionados a teste em ML, quatro áreas são encontradas em que as técnicas de teste têm uma maior diferença uma da outra para sanar esses bugs: *Data* (Dados), *Implementation*

(Implementação), Black-box (Testes de caixa-preta / funcionais) e White-box (Testes de caixa-branca / estrutura interna).

***Data* (Dados)**: a qualidade dos dados se tornou um problema sério nas comunidades de mineração de dados e aprendizado de máquina. Chamamos os dados com problemas de qualidade como dados sujos (*dirty data*). Como os dados sujos afetam a precisão de uma tarefa de mineração de dados ou aprendizado de máquina (por exemplo, classificação, clustering ou regressão), é preciso conhecer a relação entre a qualidade do conjunto de dados de entrada e a precisão dos resultados. Com base nesse relacionamento, podemos selecionar um algoritmo apropriado com a consideração dos problemas de qualidade dos dados e determinar o compartilhamento de dados a serem limpos (QI et al., 2018).

***Implementation* (Implementação)**: dado os problemas tidos com relação ao oráculo para se testar algoritmos de aprendizado de máquina, surgiram duas abordagens para mitigar essa falha que os desenvolvedores podem implementar sem depender de resultado nenhum da rede neural. A primeira dela é testar as bibliotecas baixo nível que irão ser usadas pela ferramenta em questão (ZHANG et al., 2018). Podemos ver um exemplo disso com o TensorFuzz (ODENA e GOODFELLOW, 2018), que nos mostra o uso da ferramenta em bibliotecas como Tensorflow e Keras, no qual encontram erros numéricos e comportamentos irregulares da rede neural. A segunda é mais alto nível, no qual testes são feitos no training program (Figura 6). As abordagens mais utilizadas são: Mutation Testing (MA et al., 2018), (XIE et al., 2018), (GUO et al., 2018) e Metamorphic Testing (XIE et al., 2018), (MURPHY et al., 2008), (DWARAKANATH et al., 2018).

Black-box (Teste Funcional): essa técnica funciona sem que o *tester* precise se preocupar com os componentes da implementação. Uma ferramenta DL que utiliza de abordagem *black* para testes não é muito diferente. O comum para as abordagens de teste de caixa preta é a geração de um conjunto de dados contraditórios que é usado para testar os modelos de DL. Essas abordagens utilizam técnicas de análise estatística para criar um processo aleatório multidimensional que pode gerar dados com as mesmas características estatísticas dos dados de entrada do modelo, os chamados exemplos contraditórios. Mais especificamente, eles constroem modelos que podem se ajustar a uma distribuição de probabilidade que melhor descreve os dados de entrada. Esses modelos permitem amostrar a distribuição de probabilidade dos dados de entrada e gerar quantos pontos de dados forem necessários para testar os modelos de ML (BRAIEK et al., 2018).

A vantagem dessa abordagem é que os dados sintéticos (contraditórios) usados para testar o modelo são independentes do modelo DL, mas estatisticamente próximos aos dados

de entrada. O aprendizado de máquina adverso é uma técnica que visa avaliar a robustez dos modelos ML com base na geração de exemplos adversos.

Entendemos então que, os modelos de DL são projetados para identificar conceitos latentes do treinamento, a fim de aprender a prever o valor alvo em relação aos dados de teste não vistos. No entanto, o fato de presumir que o conjunto de dados de treinamento e teste é gerado a partir da mesma distribuição torna o sistema DL vulnerável em relação a adversários maliciosos que manipulam os dados de entrada e violam algumas de suas hipóteses anteriores. Portanto, é importante testar a robustez dos modelos DL para essas variações nos dados de entrada. Existem vários mecanismos para a criação de exemplos contraditórios, como: fazer pequenas modificações nos pixels de entrada (GOODFELLOW et al., 2014), aplicar transformações espaciais (ENGSTROM et al., 2019) ou simples adivinhação e verificação para encontrar erros de classificação (GILMER et al., 2018).

Uma grande limitação dessa técnica é a representatividade dos exemplos contraditórios gerados. De fato, muitos modelos adversos que geram imagens sintéticas geralmente aplicam apenas perturbações minúsculas, indetectáveis e imperceptíveis, já que qualquer alteração visível exigiria inspeção manual para garantir a correção da decisão do modelo. Isso pode resultar em transformações estranhas ou representações simplificadas em conjuntos de dados sintéticos, que por sua vez podem ter efeitos ocultos no desempenho de um modelo de DL quando desencadeados em um cenário do mundo real.

White-box (Teste de Estrutura Interna): a técnica de caixa branca surgiu com uma ideia completamente diferente do que vimos anteriormente com a caixa preta. Com o surgimento de técnicas desse tipo, com ela surgiram novas métricas de teste para essas aplicações. Como explicado anteriormente, um teste caixa branca é aquele onde o desenvolvedor terá acesso à estrutura interna do sistema. Esse acesso nos possibilita fazer verificações em tempo de execução no modelo para conseguirmos dados mais acurados do que está acontecendo em seu backend.

Sabemos então que, as abordagens de teste de caixa branca levam em consideração a lógica de implementação interna do modelo. O objetivo dessas abordagens é cobrir um máximo de pontos específicos (por exemplo, neurônios) nos modelos. Em 2017, PEI et al. publicaram o primeiro documento de teste de caixa branca em sistemas de aprendizagem profunda. Esse se tornou o marco dos testes de aprendizado de máquina, pioneiros na proposta de critérios de cobertura para DNN. A partir desse artigo, surgiram várias técnicas de teste de aprendizado de máquina, falaremos de algumas delas a seguir.

2.5 Considerações Finais

Neste capítulo foi apresentado o embasamento teórico para o contexto da pesquisa elaborada, envolvendo os conceitos de inteligência artificial, ML (tipos de aprendizado, DL e redes neurais), e por fim o entendimento sobre testes de software e teste em ML. Esses conceitos serão utilizados no próximo capítulo para dar um melhor entendimento nas tecnologias que serão revisadas na revisão exploratória realizada.

3 Revisão Bibliográfica

Nesta seção nosso objetivo será apresentar as técnicas e ferramentas encontradas para elaboração deste trabalho. Faz-se entender que diferente do método tradicional, esses métodos desenvolvidos para ambientes DL possuem um comportamento diferente do habitual, sendo necessário um estudo específico para entender como ele procede.

Foi feita uma pesquisa bibliográfica em quarenta e cinco artigos que foram lidos e estudados, os quais possuem ferramentas e técnicas disponibilizadas publicamente para reprodução de seus resultados.

3.1 Metodologia

A revisão bibliográfica conduzida nesse trabalho consiste em uma pesquisa exploratória, sem um critério rígido como numa revisão sistemática, mais com o objetivo de mapear o conhecimento mais recente na área. Por isso, tomamos como base os dois artigos mais recentes de pesquisa nessa área (BRAIEK et al., 2018), (ZHANG et al., 2019). Esses dois artigos foram selecionados por serem os atualmente mais completos em relatar os avanços nessa área de maneira mais ampla para um estudo. O datado de 2018 possui essa visão da área de teste em DL dividida em quatro partes distintas, e o de 2019 já possui uma visão mais ampla, possuindo o mesmo critério porém com uma visão diferente.

Esses dois artigos foram encontrados utilizando uma *string* de busca bem simples, composta das palavras: *Deep Learning*, *Testing Deep Learning*, e algumas variações disso em português como: teste em aprendizado profundo, técnicas de teste para aprendizado profundo. A partir desses materiais, definimos as questões de pesquisa, depois buscamos e selecionamos os artigos mais recentes (a partir do ano 2007) com base tanto nas referências encontradas nesses dois primeiros artigos quanto os que resultavam da *string* utilizada.

O objetivo dessa revisão é identificar e analisar as principais estratégias de teste para sistemas baseados em ML, bem como as ferramentas desenvolvidas para cada uma delas. Acreditamos que os testadores podem se beneficiar da sumarização e comparação dessas estratégias e ferramentas. Nesse contexto, propomos duas questões de pesquisa para investigar e entender o estado da arte relacionado as estratégias e ferramentas de teste para sistemas baseados em ML, são elas:

QP1 - *Quais são as principais estratégias utilizadas para testar sistemas baseados em ML?*

QP2 - *Quais são as principais características das ferramentas de teste para sistemas baseados em ML?*

Como o intuito da revisão é ter uma ampla sumarização de várias estratégias existentes nesta área, não foi definido um critério de busca específico para a procura dos artigos nesta pesquisa. Os únicos critérios que foram avaliados para verificar se determinado artigo seria incluído ou não eram a verificação de seus títulos e de seu resumo, se nesses dois elementos fosse identificado que ele não condizia com o foco da pesquisa para teste em ML, ele era descartado. Através desse filtro, foram selecionados quarenta e cinco artigos para estudo, sendo eles analisados e resumidos para responder às questões da pesquisa. Listamos todos os documentos usados nas etapas propostas da pesquisa como referência aos mesmos.

3.2 Análise dos Dados

Através da análise dos dados coletados dos artigos lidos, investigamos cada pergunta de pesquisa. Para responder à QP1, a Tabela 1 apresenta uma relação de todos os artigos usados nesta pesquisa que encontramos durante a revisão exploratória com suas estratégias de teste. Alguns trabalhos utilizam de estratégias parecidas, portanto, alguns itens da tabela são repetidos.

QP1 - *Quais são as principais estratégias utilizadas para testar sistemas baseados em ML?*

Com base nos dois artigos de referência, identificamos trabalhos nos quatro tipos de estratégias/técnicas principais de teste para esses tipos de sistema. As tabelas abaixo nos mostram quais estratégias de teste cada artigo apresenta. Os critérios de teste representam como e a partir de que algoritmos ou equações o artigo irá utilizar para poder gerar os testes da ideia que se propõe. Uma lista[1] com todos os critérios de teste se encontra no Apêndice B desta pesquisa. As técnicas de Geração de teste nos mostram as formas específicas de como cada abordagem irá gerar seus testes dado sua ambientação.

[1] O Apêndice B se encontra no final dessa pesquisa com um a explicação de todos os critérios de testes encontrados.

A Tabela 1 nos mostra os resultados encontrados para os artigos que foram classificados como baseado em dados. Sua abordagem mais encontrada era uma forma de corrigir erros nos *dirty data*.

Tabela 1 – Estratégia de teste para categoria dados

Estratégia	Artigo	Critério de Teste	Técnica de Geração de Teste
Baseada em Dados	ActiveClean: Interactive Data Cleaning for Statistical Modeling (KRISHNAM et al., 2016)	Data Cleaning	*dirty data*
	BoostClean: Automated Error Detection and Repair for Machine Learning (KRISHNAM et al., 2017)	IsoDetectCleaning (Library)	*Automated cleaning*
	Impacts of Dirty Data: An Experimental Evaluation (QI et al., 2018)	Data Cleaning and Algorithm Selection	*dirty data*
	SampleClean: Fast and Reliable Analytics on Dirty Data (KRISHNAM et al., 2015)	Approximate Query Processing	*Sampling accuracy*
	The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets (HYNES et al., 2017)	Miscoding of data, Outliers and Data Cleaning	*data lint*

Fonte: Elaborada pelo autor (2020).

Na Tabela 2 abaixo, são mostrados os artigos que abordagem alguma técnica para lidar com a implementação, seja o modelo da DNN ou suas bibliotecas de uso. Suas técnicas que foram mais encontradas possuem relação com as redes neurais utilizadas e algoritmos de clusterização específicos, como o *kmeans*.

Tabela 2 – Estratégia de teste para categoria implementação

Estratégia	Artigo	Critério de Teste	Técnica de Geração de Teste
Implementação	METTLE: A Metamorphic Testing Approach to Validating Unsupervised Machine Learning Methods (XIE et al., 2018)	Kmeans, xmeans, expectation-maximization	*Metamorphic testing*
	An Approach to Software Testing of Machine Learning Applications (MURPHY et al., 2007)	SVM and MartiRank	*ranking propensity failure*
	DeepMutation: Mutation Testing of Deep Learning Systems (MA et al., 2018)	LeNet, AdvGAN, Defensive Distillation	*Mutation testing*
	Developing Bug-Free Machine Learning Systems with Formal Mathematics (SELSAM et al., 2017)	Interactive proof assistant	*formal mathematic proff*
	Properties of Machine	VM, MartiRank and PAYL	*Metamorphic*

	Learning Applications for Use in Metamorphic Testing (MURPHY et al., 2008)		*testing*
	TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing (ODENA et al., 2018)	Coverage-GuidedFuzzing	*TensorFlow graph (tensorfuzz)*
	Testing and Validating Machine Learning Classifiers by Metamorphic Testing (XIE et al., 2011)	Clustering Weka Algorithm	*Metamorphic testing*
	Identifying Implementation Bugs in Machine Learning Based Image Classifiers using Metamorphic Testing (DWARAKANATH et al., 2018)	SVM, RestNet	*Metamorphict esting*
	An Empirical Study on TensorFlow Program Bugs (ZHANG et al., 2018)	Bugs in StackOverflow and Github on TensorFlow porgrams	*Verification of causes of the bugs, where are they located and how to detect them*
	CRADLE: Cross-Backend Validation to Detect and Localize Bugs in Deep Learning Libraries (PHAM et al., 2019)	Backend DL library	*Devianation between models*
	Testing MCMC code (GROOSE et al., 2014)	Unit and Integration	*Geweke test*
	Multiple-Implementation Testing of Supervised Learning Software (SRISAKAOKUL et al., 2016)	kNN, NaiveBayes algorithm	*Devianation of test*
	DeepHunter: Hunting Deep Neural Network Defects via Coverage-Guided Fuzzing (XIE et al., 2018)	Coverage-Guided Fuzzing, Neuron Coverage, Neuron Boundage Coverage, K-Multisec Neuron Coverage, Strong Neuron Activation Coverage, Top-k Neuron Ccoverage, Bottom-k Neuron Coverage	*Metamorphic mutation*
	DLFuzz: Differential Fuzzing Testing of Deep Learning Systems (GUO et al., 2018)	Coverage-Guided Fuzzing, Neuron Coverage	*Mutation testing*
	Test Selection for Deep Learning Systems (MA et al., 2019)	trigger misclassification with real and adversarial data	*retraining with data that cause miss behaviour*

Fonte: Elaborada pelo autor (2020).

A Tabela 3 nos mostra os resultados encontrados para os artigos que foram classificados como black-box. Sua abordagem mais encontrada é ligada ao lidar com os dados e gerar seus exemplos adversos como forma de treinar o modelo de forma mais abrangente.

Tabela 3 – Estratégia de teste para categoria black-box

Estratégia	Artigo	Critério de Teste	Técnica de Geração de Teste
Black-Box	DeepFool: a simple and accurate method to fool deep neural networks (MOOSAVI-DEZFOOLI et al., 2015)	Adversarial examples, Multiclass Classifiers	*Smallest perturbation*
	Explaining and Harnessing adversarial examples (GOODFELLOW et al., 2014)	Fast Gradient Sign Method	*linear perturbation*
	Motivating the Rules of the game for Adversarial Example Research (GILMER et al., 2018)	Adversarial Robustness	*Adversarial perturbation*
	Towards deep neural network architectures robust to adversarial examples (GU et al., 2014)	Gausian Noise, Adversarial	*Adversarial perturbation*
	"Why Should I Trust You?" Explaining the Predictions of Any Classifier (RIBEIRO et al.,2016)	Submodular pick	*Prediction features perturbation*
	Understanding Black-box Predictions via Influence Functions (KOH et al., 2017)	Conjugate Gradients, Stochastic Estimation	*Influence functions*
	Automated Directed Fairness Testing (UDESHI et al., 2018)	Robustness for Adversarialinputs	*Discriminatory discover*

Fonte: Elaborada pelo autor (2020).

A Tabela 4 nos mostra os resultados encontrados para os artigos que foram classificados como black-box. Os artigos que possuem essa estratégia são mais focados em abordar novas formas de cobertura dos modelos usados, de forma a verificar a ativação da rede utilizada.

Tabela 4 – Estratégia de teste para categoria white-box

Estratégia	Artigo	Critério de Teste	Técnica de Geração de Teste
White-Box	Combinatorial Testing for Deep Learning Systems (MA et al., 2018)	Neuron-Activation Configuration, TWSCov, TWDCov	*search based testing, guided random testing, symbolic execution, constraint-based testing*
	Concolic Testing for Deep Neural Networks (SUN et al., 2018)	Lipschitz Continuity, Neuron Coverage, Neuron Boundary Coverage, Sign-Sign Coverage	*Concolic Testing*
	DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems (MA et al., 2018)	K-Multisection Neuron Coverage, Neuron Boundary Coverage, Strong Neuron Activation Coverage, TKNC, TKNP	*gradientdescentmethods*
	DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems (ZHANG et al., 2018)	Metamorphic Testing, Input Validation	*GAN-based Metamorphic*
	DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars (TIAN et	Neuron Coverage	*Greedy search*

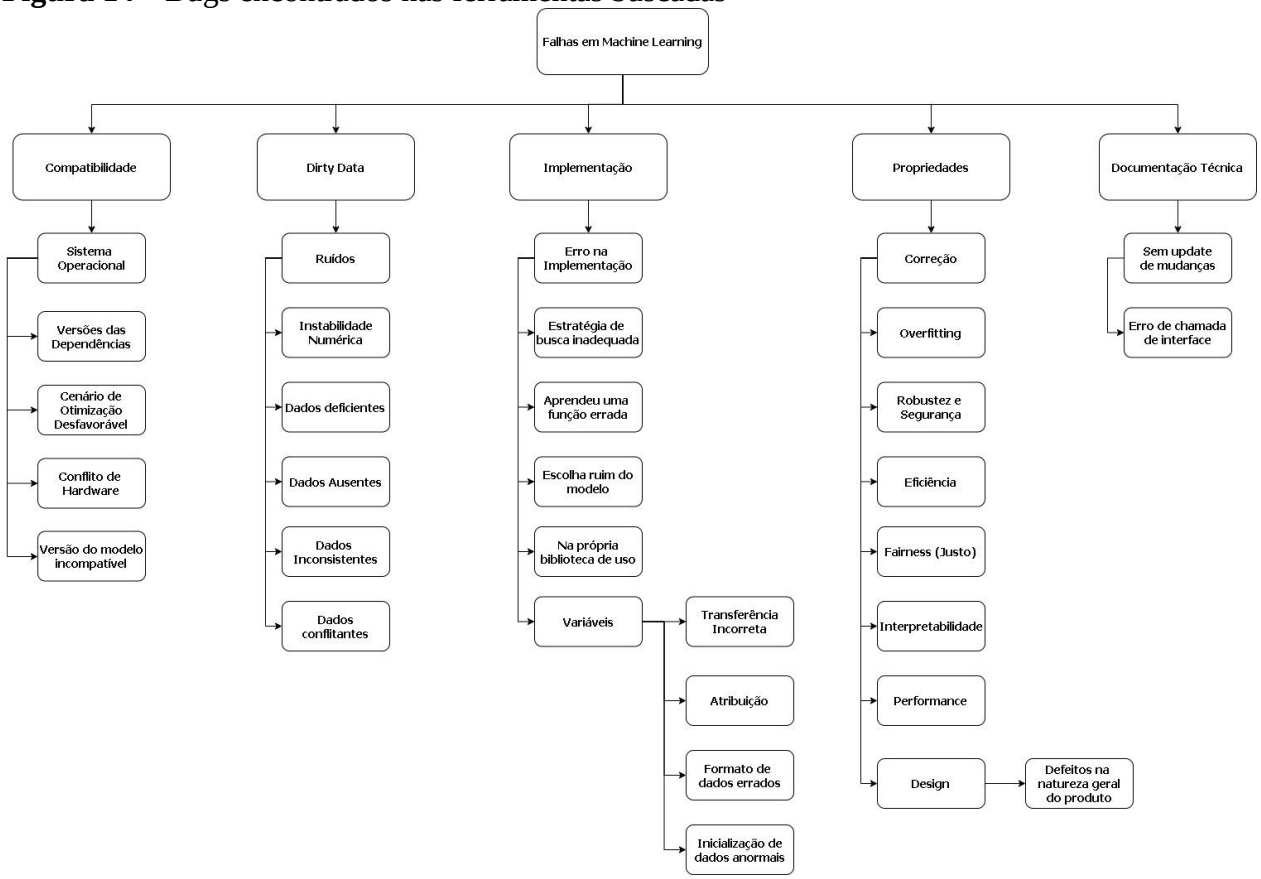
	al., 2018)		
	DeepXplore: Automated Whitebox Testing of Deep Learning Systems (PEI et al., 2017)	Neuron Coverage, Gradient Ascendent	*dual-optimization*
	Testing Deep Neural Networks (SUN et al., 2018)	MC/DC, Sign-Sign Coverage, Value-Sign Coverage, Sign-Value Coverage, Value-Value Coverage	*Symbolic execution*
	Towards Improved Testing for Deep Learning (SEKHON et al., 2019)	Neuron Coverage, 2-way Coverage	*joint optimization*
	Structural Coverage Criteria for Neural Networks Could Be Misleading (LI et al., 2019)	Neuron Coverage, SS Coverage, TWSCov, TWDCov	*search based testing, guided random testing*
	Robustness of Neural Networks: A Probabilistic and Practical Approach (MANGAL et al., 2019)	Lipschitz Continuity	*Probabilistic robustness*
	Guiding Deep Learning System Testing using Surprise Adequacy (KIM et al., 2018)	Activation Trace and Surprise Adequacy, Likelihood-based Surprise Adequacy, Distance-based Surprise Adequacy, Surprise Coverage	*Surprise optimization input*
	Testing Machine Learning Algorithms for Balanced Data Usage (SHARMA et al., 2019)	Balanced data usage	*balanced data*
	Input Prioritization for Testing Neural Networks (BYUN et al., 2019)	Softmax Output as Confidence Predictions, Bayesian Uncertainty, Input Surprise	*Surprise optimization input*
	Boosting Operational DNN Testing Efficiency through Conditioning (LI et al., 2019)	Operational Testing	*operational testing com SRS estimator, confidence-based stratified sampling estimator, cross entropy-based sampling estimator*
	DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems (MA et al., 2019)	Neuron-Activation Configuration, TWSCov, TWDCov	*Combinatorial testing*

Fonte: Elaborada pelo autor (2020).

Identificamos algumas estratégias de teste de acordo com os trabalhos que descrevem cada técnica de teste. Assim, coletamos aproximadamente 70% dos artigos na área de Dados que utiliza de um critério de *Data Cleaning* para gerar os testes. Para a Implementação, foi um total de 60% que utiliza de critério como um algoritmo de clusterização. No Black-Box foi visto uma média de 45% usa como critério a geração de exemplos adversos para teste. Para o White-Box, coletamos um total de 40% de estudos que utilizam o critério *Neuron Coverage* para validação.

Após feita a leitura dos artigos selecionados, conseguimos agrupar uma lista de *bugs* que podem acontecer ao desenvolver utilizando técnicas ML e cada uma possui uma abordagem diferente de correção e verificação da confiança do sistema.

Figura 14 – Bugs encontrados nas ferramentas buscadas



Fonte: Elaborado pelo autor (2019).

Exemplificando, os erros que foram sendo encontradas foram incorporando grupos maiores, como os mostrados na Figura 14. Erro na compatibilidade engloba um grupo de *bugs* como: problema no sistema operacional, versionamento de dependências necessárias para executar o modelo, cenário de otimização desfavorável para execução, conflito de *hardware* e versão do modelo disponibilizada incompatível com o mostrado em artigos.

O erro nos dados irá englobar: ruídos, instabilidade numérica, dados deficientes, ausente e/ou inconsistentes e conflitante. Erros na implementação mais aparentes: erro na mesma, estratégia de busca usada é inadequada, aprendeu uma função errada, escolha ruim do modelo para uso, erro na própria biblioteca de uso, erro nas variáveis que irá gerar uma má atribuição, formatos e transferências erradas e possíveis inicializações anormais.

Por fim entramos em um grupo mais alto nível, onde erros são encontrados nas propriedades e documentação técnica do algoritmo, envolvendo: correção, acurácia, robustez,

segurança, eficácia, quão justo o modelo é, sua interpretabilidade, desempenho, *design*, *update* de documentação e até erros de interface.

Para responder à QP2, a Tabela 2 apresenta as 22 ferramentas de teste para ML que encontramos nos artigos da revisão bibliográfica. Essas 22 ferramentas foram encontradas em 45 artigos selecionados. Alguns trabalhos não disponibilizaram a ferramenta para reprodução de resultados e, portanto, nossa análise foi realizada com 22 ferramentas.

QP2-*Quais são as principais características das ferramentas de teste para sistemas baseados em ML?*

Dentre os quarenta e cinco artigos reportados nas Tabela 1, 2, 3, 4 percebemos que vinte e dois deles implementam ferramentas para esses tipos de sistema. A Tabela 5 apresenta as principais características observadas nessas ferramentas.

Tabela 5 – Características das Ferramentas

Categoria	Ferramenta	Tecnologia	Entradas DNN	Datasets	Configuração de Processamento	Link
Baseada em Dados	Dirty-dataImpacts (QI et al., 2018)		single and multiple	UCI public datasets: Iris, Ecoli, Car, Chess, Adult, Seeds, Abalone, HTRU, Activity, Servo, Housing, Concrete, Solar flare	two Intel(R) Xeon(R) E5-2609 v3@1.90GHz CPUsand 32GB memory, under CentOS7	(QI et al., 2018)
	SampleClean (KRISHNAM et al., 2015)		Multiple	TPCH dataset		(KRISHNAM et al., 2015)
	Data Linter (HYNES et al., 2017)	Python, Apache, TensorFlow, Facets	Single	600 datasets públicos da Kaggle		(HYNES et al., 2017)
Implementação	Certigrad (SELSAM et al., 2017)	Lean Theorem Prover	Single	MNIST		(SELSAM et al., 2017)
	TensorFuzz (ODENA et al., 2018)	TensorFuzz Library	Multiple	MNIST	GPU	(ODENA et al., 2018)
	VerifyML (DWARAKANATH et al., 2018)	TensorFlow 1.7	Single	MNIST, CIFAR-10, Kaggle Fruits,	Intel i5 CPU	(DWARAKANATH et al., 2018)

				SVHN		
	TensorFlow-Program-Bugs (ZHANG et al., 2018)	TensorFlow				(ZHANG et al., 2018)
	DLFuzz (GUO et al., 2018)	Tensorflow 1.2.1 and Keras 2.1.3	Multiple	MNIST, ImageNet	4 cores (Intel i7-7700HQ@3.6GHz), 16GB of memory, a NVIDIA GTX 1070 GPU and Ubuntu 16.04.4 as the host OS	(GUO et al., 2018)
Black-Box	DeepFool (MOOSAVI-DEZFOOLI et al., 2015)	TensorFlow, Caffe, MatConvNet	Multiple	MNIST, CIFAR-10, ImageNet	Mid-2015 MacBook Pro, GTX 750 Ti GPU	(MOOSAVI-DEZFOOLI et al., 2015)
	Maxout (GOODFELLOW et al., 2014)	Theano	Single	MNIST		(GOODFELLOW et al., 2014)
	lime-experiments (RIBEIRO et al.,2016)	NumPy	Multiple	multi_polarity_books, multi_polarity_kitchen, multi_polarity_dvd, multi_polarity_kitchen		(RIBEIRO et al.,2016)
	influence-release (KOH et al., 2017)	Numpy, Tensorflow, Keras	Multiple	MNIST, ImageNet		(KOH et al., 2017)
	Aequitas (UDESHI et al., 2018)	Python, Numpy, Scikit-learn	Single	US censos	Intel i7 processor having 64GB of RAM and running Ubuntu 16.04	(UDESHI et al., 2018)
	DeepConcolic (SUN et al., 2018)	TensorFlow 1.5.0 and Keras	Single	MNIST, CIFAR-10	24 core Intel(R) Xeon(R) CPU E5-2620 v3 and 2.4 GHz and 125 GB memory	(SUN et al., 2018)
White-Box	DeepGauge (MA et al., 2018)	TensorFlow 1.5.0 and Keras 2.1.3	Single	MNIST, ImageNet	GNU/Linux system with Linux kernel 3.10.0 on a 18-core 2.3GHz Xeon 64-bit CPU with 196 GB of RAM and also an NVIDIA Tesla M40 GPU with 24G	(MA et al., 2018)
	DeepRoad (ZHANG et al., 2018)	TensorFlow and Keras	Multiple	Udacity Training, Udacity Test Ep1, Udacity Test Epi2, Youtube Epi1,		(ZHANG et al., 2018)

			Youtube Epi2		
DeepTest (TIAN et al., 2018)	TensorFlow, Keras and Theano	Single	Udacity HMB_3.bag		(TIAN et al., 2018)
DeepXplore (PEI et al., 2017)	TensorFlow 1.0.1 and Keras 2.0.3	Multiple	MNIST, ImageNet, Driving, Contagio/ VirusTotal, Drebin	Linux laptop running Ubuntu 16.04 (one Intel i7-6700HQ 2.60GHz processor with 4 cores, 16GB of memory, and a NVIDIA GTX 1070 GPU)	(PEI et al., 2017)
DeepCover (SUN et al., 2018)		Single	ImageNet, CIFAR-10	MacBook Pro (2.5 GHz Intel Core i5, 8 GB memory)	(SUN et al., 2018)
SADL (KIM et al., 2018)	Keras v.2.2.0	Multiple	MNIST, CIFAR-10, Udacity Self-driving Car Challenge	Intel i7-8700 CPU, 32GB RAM, running Ubuntu 16.04.4 LTS	SADL (KIM et al., 2018)
keras-prioritizer (BYUN et al., 2019)	Python on top of keras	Multiple	MNIST, EMNIST, Taxinet	Ubuntu 16.04 running on an Intel i5 CPU, 32GB DDR3 RAM, an SSD, and a single NVIDIA GTX 1080-Ti GPU	(BYUN et al., 2019)
BoostDNNTesting (LI et al., 2019)	Tensorflow 1.12.0 and Keras 2.2.4	Multiple	MNIST, ImageNet, Driving, Mutant1[a], Mutant2[a], Mutant3[a]	ImageNet: a Linux server with two 10-core 2.20GHz Xeon E5-2630 CPUs, 124 GB RAM, and 2 NVIDIA GTX 1080Ti GPUs, outrasdnns: a Linux laptop with an 2.20GHz i7-8750H CPU, 16 GB RAM, and a NVIDIA GTX 1050Ti GPU	(LI et al., 2019)

Fonte: Elaborada pelo autor (2020).

A Tabela 5 mostra as características de cada ferramenta, sendo definidas com base nos pontos em comum que elas possuem. As seguintes características foram consideradas: as tecnologias que a ferramenta utilizam para funcionar, quantidade de entradas que a rede neural suporta podendo ser apenas uma ou múltiplas entradas, depende da rede neural, os *datasets* utilizados pelo modelo, a configuração de processamento no qual a ferramenta foi testada e seu *link* para reprodução dos resultados.

Dois pontos importantes citados acima são o processamento e se a ferramenta é disponibilizada. Sendo informado o ambiente em que a ferramenta foi executada, podemos

verificar a possibilidade de reprodução em uma máquina diferente. No quesito *link*, se a ferramenta não era disponibilizada, visto que nem todas os artigos disponibilizaram o acesso, não foi possível validar os resultados e por isso foi descartada.

Muitas das ferramentas não conseguimos executar por algumas razões. Por exemplo, em alguns casos, as ferramentas não estavam disponíveis na mesma versão exposta no artigo. Em outros casos, algumas ferramentas apresentam erros de execução ou falta de documentação para entender todos os requisitos para que as ferramentas funcionem corretamente. De fato, apenas metade das ferramentas possui um guia de usuário disponível, enquanto o restante disponibilizava apenas o código.

3.3 Considerações Finais

Neste capítulo foram apresentadas as técnicas e ferramentas mais recorrentes para teste em DL atualmente após uma revisão bibliográfica, no qual foram divididas em quatro áreas: Dados, Implementação, Black-Box e White-Box. A partir disso, foi possível fazer uma relação de quais critérios e técnicas de teste foram mais recorrentes nos estudos, assim como as principais características que as ferramentas encontradas apresentam. Com esses resultados, no próximo capítulo veremos como foi feita a seleção e como forma usadas as ferramentas selecionadas para o estudo empírico.

4 ESTUDO EMPÍRICO

Nesta parte de nossa pesquisa, realizamos um estudo comparativo para avaliar duas ferramentas de teste selecionadas a partir dana nossa revisão bibliográfica. Nosso principal objetivo nesta parte é identificar vantagens e desvantagens das ferramentas de teste que utilizam técnicas de teste em ML e, se possível, verificar se é capaz uni-las para melhorar a confiabilidade das ferramentas. A seção 4.1 apresenta as questões de pesquisa. A seção 4.2 descreve as etapas do estudo. A seção 4.3 explica os critérios usados para selecionar as 2 ferramentas para o estudo comparativo. A Seção 4.4 apresenta os sistemas-alvo utilizados e a Seção 4.5 mostra a preparação do ambiente de estudo. A Seção 4.6 descreve a coleta de dados do estudo proposto. A Seção 4.7 nos traz os resultados obtidos, em seus subpontos são respondidas as QP3 e QP4. Finalmente a seção 4.8 apresenta alguns problemas encontrados durante a pesquisa.

4.1 Questões de Pesquisa

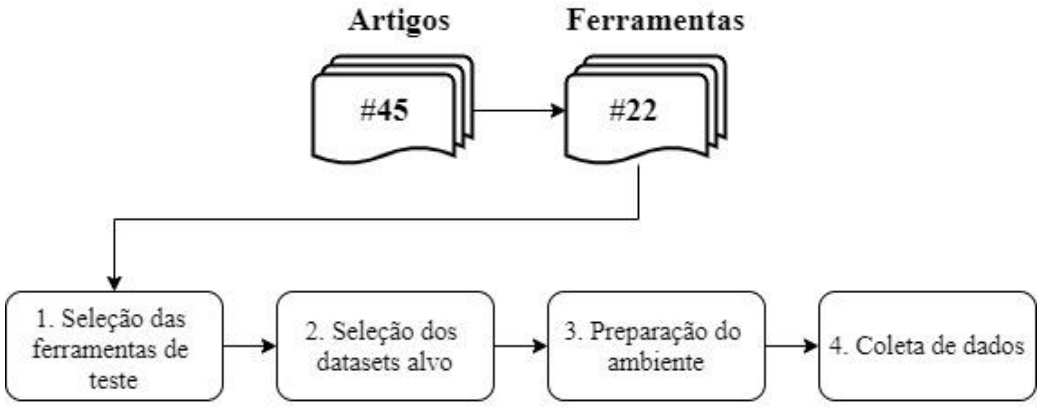
O objetivo desse estudo é além de comparar duas ferramentas, verificar se possíveis junções podem melhorar ou não seu desempenho. Para dar suporte a esse objetivo, definimos as seguintes questões de pesquisa:

QP3 - *Quão eficientes são as ferramentas de teste de sistemas baseados em ML?*
QP4 - *Quão eficazes são as ferramentas de teste de sistemas baseados em ML?*

4.2 Passo-a-passo do estudo

Para responder às perguntas de pesquisa apresentadas na Seção 4.1, apresentamos o estudo empírico para avaliar duas ferramentas de teste para ML. A Figura 13 mostra as quatro etapas do estudo que seguimos no estudo empírico proposto. Etapa 1 é a filtragem das ferramentas de teste selecionadas. Mostramos os critérios de seleção adotados para a redução do conjunto de ferramentas. Escolhemos duas ferramentas para o estudo empírico proposto. A etapa 2 apresenta os *datasets* que serão usados com as ferramentas de teste. Além disso, demonstramos o ambiente empírico de estudo na Etapa 3. Finalmente, a Etapa 4 consiste em executar as ferramentas selecionadas e coletar dados de interesse. Essas quatro etapas são detalhadas nas próximas seções.

Figura 13 – Etapas do estudo empírico



Fonte: Elaborado pelo autor (2020).

4.3 Selecionando as ferramentas de teste

Encontramos 22 ferramentas de teste para ML na pesquisa realizada. Após a aplicação de um conjunto de critérios de filtragem, realizamos um estudo com DeepXplore (PEI et al., 2017) e DeepFool (MOOSAVI-DEZFOOLI et al., 2015). O processo de seleção foi conduzido como segue.

Primeiro, fizemos uma verificação de todas as ferramentas que não demandavam de uma configuração de processamento muito grande, analisando conforme mostrado na tabela 2. Adotamos também como critério ferramentas que possuíssem características parecidas como: mesmo datasets utilizados, mesmos modelos e tecnologias para o estudo ser o mais justo possível em seu resultado. Obtivemos 8 ferramentas: DeepXplore (PEI et al., 2017), DeepFool (MOOSAVI-DEZFOOLI et al., 2015), DeepTest (TIAN et al., 2018), Maxout (GOODFELLOW et al., 2014), DLFuzz (GUO et al., 2018), TensorFuzz (ODENA et al., 2018), VerifyML (DWARAKANATH et al., 2018) e SADL (KIM et al., 2018).

Em seguida, restringimos o conjunto de ferramentas para incluir apenas ferramentas que possuíssem um guia de passo-a-passo para o usuário. Após a aplicação desses critérios, terminamos com quatro das oito ferramentas relatadas anteriormente. Portanto, selecionamos duas ferramentas de teste de áreas diferentes para ML neste estudo: DeepXplore (PEI et al., 2017) como White-box e DeepFool (MOOSAVI-DEZFOOLI et al., 2015) como Black-box, as outras duas ferramentas foram selecionadas para verificar a implementação das duas citadas anteriormente, sendo elas: TensorFuzz (ODENA et al., 2018) e DLFuzz (GUO et al., 2018). Uma breve descrição das ferramentas DeepXplore (PEI et al., 2017) e DeepFool (MOOSAVI-DEZFOOLI et al., 2015) está abaixo.

O DeepXplore é uma ferramenta de teste diferencial que gera entradas de teste para um sistema de DL. Inspirados pela abordagem *test coverage* dos sistemas de software tradicionais, os autores propuseram o *neuron coverage* (cobertura de neurônios) para impulsionar a geração de testes. Esse critério é definido como a proporção de neurônios únicos que são ativados para determinadas entradas e o número total de neurônios em um DNN. Um neurônio individual é considerado ativado se a saída do neurônio (dimensionada pelas saídas da camada geral) for maior que um limite de DNN (definido no algoritmo). Na Figura 14 é mostrada a fórmula de cálculo dessa métrica de cobertura.

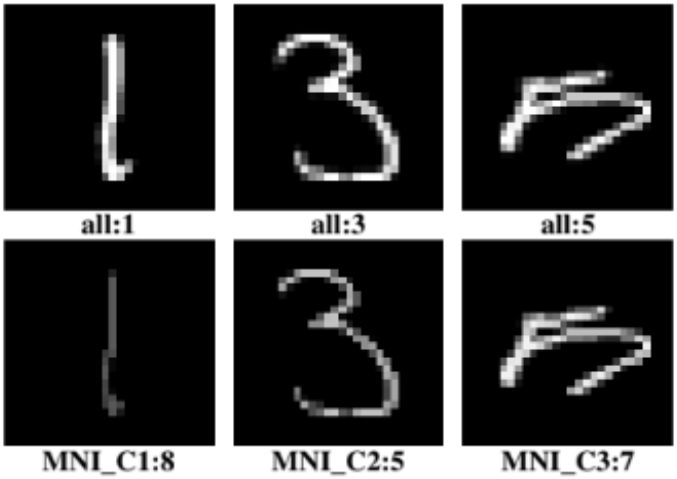
Figura 14 – Fórmula do *Neuron Coverage*

$$Neuron\ Coverage = \frac{|Activated\ Neurons|}{|Total\ Neurons|}$$

Fonte: PEI et al. (2017).

Além disso, as entradas precisam expor diferenças entre os diferentes modelos DNN usados, bem como ser o máximo possível dos dados do mundo real. O algoritmo de otimização conjunta usa iterativamente uma pesquisa de gradiente para encontrar uma entrada modificada que satisfaça todos esses objetivos. Na figura 15 temos um exemplo de saída gerada pelo DeepXplore. Nas linhas de cima são os dados de teste que serão usados de *seed* pelo modelo, todos possuem um *label* de classificação e sem alteração na imagem são classificados corretamente. Depois de feita uma transformação na iluminação da imagem e levada para classificar novamente, foi percebido que os modelos identificaram erroneamente a imagem visto o label original da imagem.

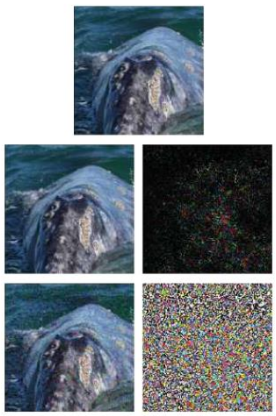
Figura 15 – Exemplo de imagens geradas pelo DeepXplore



Fonte: PEI et al. (2017).

O DeepFool é uma ferramenta desenvolvida para calcular exemplos contraditórios que enganem os classificadores de última geração. É baseado em uma linearização iterativa do classificador para gerar perturbações mínimas suficientes para alterar os rótulos de classificação (*labels*). Um exemplo dessas perturbações contraditórias é mostrado na Figura 16. Em sua primeira linha: a imagem original é classificada como “baleia”. Na segunda linha, a imagem original sobreposta com os ruídos é classificada como "tartaruga" e a perturbação correspondente calculada pelo DeepFool é imagem mais escura a direita. Na terceira linha, temos uma representação da imagem classificada como “tartaruga” e a perturbação correspondente calculada pelo método do sinal rápido de gradiente (GOODFELLOW et al., 2014). O DeepFool leva a uma perturbação menor que dificulta a visualização desses ruídos a olho nu.

Figura 16 – Comparação da perturbação do DeepFool e do método sinal rápido de gradiente

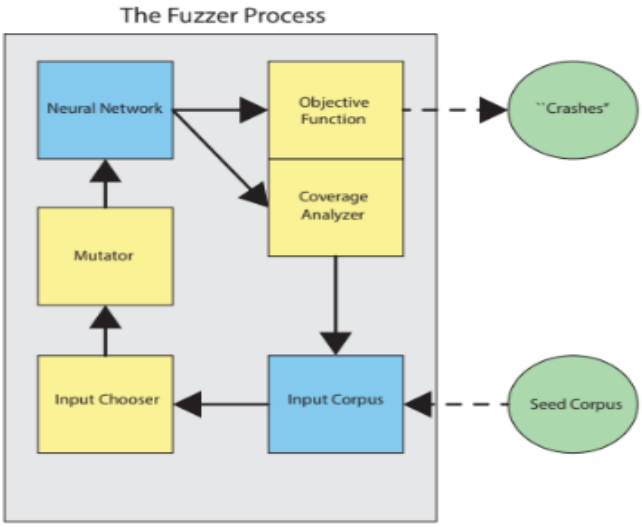


Fonte :MOOSAVI-DEZFOOLI et al. (2015).

Agora faremos uma breve descrição das ferramentas que utilizaremos como teste na implementação dessas duas ferramentas: o TensorFuzz (ODENA et al., 2018) e DLFuzz (GUO et al., 2018).

O TensorFuzz desenvolve métodos de difusão guiada por cobertura para redes neurais. Nessa técnica, mutações aleatórias de entradas em uma rede neural são guiadas por uma métrica de cobertura em direção ao objetivo de satisfazer as restrições especificadas pelo usuário. Com isso, é possível dizer com que rapidez os algoritmos de clusterização podem fornecer essa métrica de cobertura. A partir disso, a ferramenta é utilizada para encontrar problemas numéricos em redes neurais treinadas, divergências entre as redes neurais e suas versões quantizadas e comportamentos indesejáveis nos modelos de linguagem no nível dos caracteres. Na Figura 17 temos uma representação de como funciona o processo da ferramenta para encontrar esses erros pontuados.

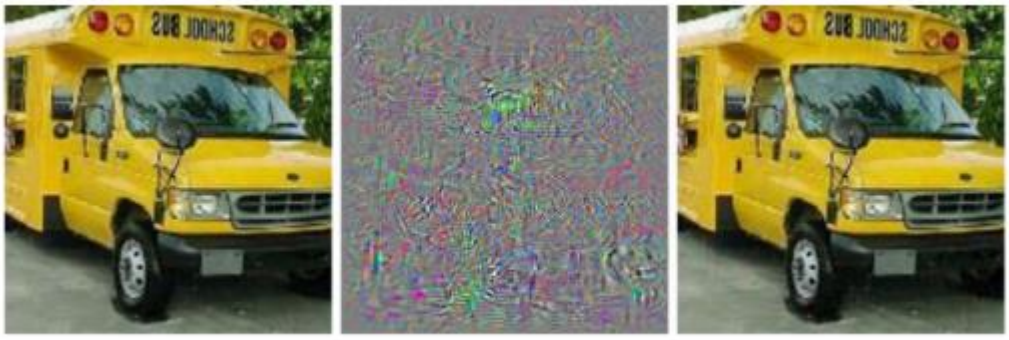
Figura 17 – Processo do TensorFuzz



Fonte: ODENA et al. (2018).

O DLFuzz foi a primeira estrutura de teste diferencial para orientar os sistemas de DL que expõem comportamentos incorretos. O DLFuzz mantém uma mutação minuciosa da entrada para maximizar a cobertura de neurônios e a diferença de previsão entre a entrada original e a entrada mutada, sem esforço manual de rotulagem ou oráculos de referência cruzada de outros sistemas de DL com a mesma funcionalidade. A ferramenta seleciona iterativamente os neurônios que devem ser ativados para cobrir mais lógica e modifica as entradas de teste aplicando pequenas perturbações para orientar os sistemas de DL que expõem comportamentos incorretos. Durante o processo de mutação, ele mantém as entradas mutadas que contribuem para um certo aumento da cobertura dos neurônios para a subsequente difusão, e a perturbação é restrita para ficar invisível e garante que os resultados da previsão da entrada original e das entradas mutadas sejam o mesmo. A Figura 18 nos mostra um exemplo de como é essa mutação que a ferramenta gera, juntamente com os ruídos que são inseridos sobrepostos a imagem original para gerar a mutação.

Figura 18 – Um exemplo de teste do DLFuzz



Fonte: GUO et al. (2018).

4.4 Seleção do *dataset* alvo

Nesta etapa, selecionamos três *datasets* de vários tamanhos para compor nosso estudo empírico. Esses *datasets* fornecem variabilidade por meio de variadas imagens disponibilizadas. Os três *datasets* de destino estão listados na Tabela 6 em ordem crescente no número de recursos.

Tabela 6 – *Datasets* alvo

Dataset	**Tipo de dado**	***Training set***	***Test set***
MNIST (LECUN et al., 1998)	Imagens de números de 0 a 9	60.000	10.000
ImageNet (DENG et al., 2009)	Imagens diversas rotuladas manualmente	100.000	1.000
Driving (UDACITY-CHALLENGE, 2016)	Imagens capturadas por uma câmera montada atrás do para-brisa de um carro	101.396	5.614

Fonte: Elaborada pelo autor (2020).

Foram levadas em consideração para a escolha do *dataset* as três características mostradas na tabela acima e a compatibilidade deles com as ferramentas selecionadas no ponto 4.3. O Driving foi descartado pois pertencia apenas ao DeepXplore, se tornando inviável para uso no estudo. Após isso, para decidir entre os dois restantes verificamos o tamanho tanto do seu *training* e *test set*. O *training se*t é a parte do *dataset* que será utilizada para gerar o modelo após execução com a rede neural. Após isso, o *test set* é utilizado como oráculo e verifica se o modelo está conseguindo ou não fazer uma classificação satisfatória

das imagens que serão nossos resultados. Sendo assim, o ImageNet foi descartado por apresentar um número pequeno e não tão variado de imagens em seu *test set*, nos resultando com a escolha do MNIST para uso nesta pesquisa.

4.5 Preparação do ambiente

Nesta etapa, apresentamos o ambiente do estudo. Primeiro instalamos as ferramentas selecionadas com todas as dependências em dois computadores: um de uso apenas para essa pesquisa e outro pessoal preparado para o estudo. As ferramentas foram instaladas em máquinas diferentes devido a diferença em muitas dependências e como uma forma de agilizar o processo. O de uso exclusivo para a pesquisa foi um Mac Pro 3,5 GHz 6-Core Intel Xeon E5 64GB DDR3, o pessoal possui 8 GB de RAM, processador i5 2,40 GHz e sistema operacional Windows 10. Para usar as ferramentas DeepXplore e DeepFool em nossa pesquisa, instrumentamos o código fonte e o conjunto de testes dos sistemas de destino em cada ponto de variabilidade.

As duas ferramentas usadas no estudo se baseiam no modelo LeNet (LECUN et al., 1998) para validar os *labels* de imagens encontradas no *dataset*. Precisamos saber o prazo máximo para executar um conjunto de testes para todas as configurações válidas em todos os sistemas de destino. Para atingir esse objetivo, em todos os casos que executávamos uma ferramenta, seu tempo de execução era guardado para comparação futura.

4.6 Coleta de dados

Esta etapa consiste em executar as ferramentas de teste selecionadas no conjunto de testes dos sistemas de destino e coletar os dados para avaliar essas ferramentas em termos de eficiência (QP3) e eficácia (QP4). Para a avaliação de eficiência, queremos verificar o tempo de execução. Para mitigar efeitos aleatórios, realizamos os cenários da pesquisa executando cada ferramenta em cada conjunto de testes do sistema de destino 3 vezes e calculamos o tempo médio de execução. Para a avaliação da eficácia, queremos dizer a quantidade de configurações que geraram erros que cada ferramenta de teste foi capaz de identificar.

Antes da execução, é necessário fazer uma breve explicação dos hiperparâmetros que foram utilizados nas duas ferramentas. Os hiperparâmetros do modelo são parâmetros para o processo de construção do modelo que não são aprendidos durante o treinamento. Eles podem fazer uma diferença substancial no desempenho de um modelo de aprendizado de máquina, definindo a arquitetura do modelo e afetando a capacidade do modelo, influenciando sua flexibilidade.

No DeepXplore é possível configurar 10 hiperparâmetros, sendo três deles opcionais. Seu primeiro hiperparâmetro é o *transformation*, ele nos indica que tipo de transformação será aplicado na imagem, sendo seus tipos: *light* que irá aumentar o brilho na imagem, *occl* que causa um pequeno ruído em algum ponto da imagem, e o *blackout* que causa um ruído maior como uma figura geométrica em algum ponto da imagem. O segundo é o *weight_diff*, no artigo descrito como lambda 1, é o parâmetro que vai lidar com o controle do comportamento diferencial. Quanto maior, mais diminui o valor/confiança de um DNN específico, quanto menor, bota mais *weight* (peso) para manutenção dos outros DNN's. O terceiro é o *weight_nc*, no artigo descrito como lambda 2, é o parâmetro para controle do *neuron coverage*. Quanto maior, cobre um número maior de neurônios, quanto menor, gera mais inputs para teste. O quarto chamamos de *step*, que significa o tamanho de passos do gradiente descendente no algoritmo. Mais *steps* pode gerar uma oscilação na classificação e o modelo se torna menos preciso para conseguir mais interações para chegar ao objetivo (um comportamento diferente). O quinto e um dos mais importante é o número de sementes (*inputs*) vindas do *test set* que serão utilizadas, as chamadas *seeds*. O sexto é chamado de *grad_iterations* que significa o número de interações que o gradiente vai realizar. E por sétimo e último temos o *threshold*, ele é responsável por determinar a ativação dos neurônios, achar inputs o ativem fica mais difícil de acordo com o crescimento do seu valor. O algoritmo ainda disponibiliza de outros parâmetros com valores default, ou seja, um valor fixo, muitas vezes não sendo preciso defini-los, apenas se você buscar um resultado muito específico. São eles: *target_model* que indica qual modelo queremos predizer uma mudança dos três que é disponibilizado, *start_point* nos possibilita dizer por onde começar a busca na rede neural e *occl_size* que controla o tamanho da transformação *occl* que será aplicada na imagem.

Para a versão disponibilizada do DeepFool, diferente do DeepXplore não foi necessária nenhuma passagem de atributos para ser possível a execução da ferramenta, sua implementação já vinha configurada e era apenas necessário executar o arquivo de teste.

4.7 Resultados e Discussão

Nesta seção, discutiremos alguns dos resultados alcançados neste estudo comparativo de ferramentas de teste para ML. A Seção 4.7.1 responde ao QP3 e a Seção 4.7.2 responde ao QP4.

4.7.1 *Eficiência das ferramentas*

QP3 - *Quão eficiente são as ferramentas de teste de sistemas baseados em ML?*

A Tabela 7 e 8 ilustram os dados coletados referentes ao tempo gasto pela ferramenta DeepXplore para executar dado os hiperparâmetros configurados. Observamos que o parâmetro *seed* era o que mais influenciava no tempo de execução, pois quanto mais entradas utilizávamos, mais o tempo que a ferramenta levava para gerar algum resultado crescia. Primeiramente executamos com mil seeds para verificar o comportamento e medir o desempenho da ferramenta.

Tabela 7 – Casos de teste no DeepXplore com mil seeds

Hiperparâmetros	Tipo	Caso 1	Caso 2	Caso 3
transformation	choices= ['light', 'occl', 'blackout']	light	light	light
weight_diff	float	1	1	1
weight_nc	float	0.1	0.1	0.1
step	float	10	10	10
seeds	int	1000	1000	1000
grad_iterations	int	1	1	1
threshold	float	0	0	0
target_model	int	choices= [0, 1, 2], default=0	choices= [0, 1, 2], default=0	choices= [0, 1, 2], default=0
start_point	tuple	occlusion upper left corner coordinate, default= (0, 0)	occlusion upper left corner coordinate, default= (0, 0)	occlusion upper left corner coordinate, default= (0, 0)
occl_size	tuple	occlusionsize, default= (10, 10)	occlusionsize, default= (10, 10)	occlusionsize, default= (10, 10)
Tempo de execução		start: 09/01/2020 at 8:43 AM end: 16/01/2020 at 6:13 AM	start: 17/01/2020 at 08:04 AM end: 21/01/2020 at 22:54 PM	start: 24/01/2020 at 08:22 AM end: 29/01/2020 at 12:04 PM
Tempo total		166 hrs e 30 min	110 hrs e 50 min	124 hrs e 18 min
***image_diffbehavior*[1]**		31	36	33
***averaged covered neurons*[2]**		0.979	0.968	0.981

[1]TOTAL DE IMAGENS QUE CAUSARAM UM COMPORTAMENTO CONTRÁRIO
[2]COBERTURA FINAL ENCONTRADA

Fonte: Elaborada pelo autor (2020).

Após verificar o tempo de execução, fizemos o teste com a metade do número de entradas que foram utilizadas nos casos descritos na Tabela 7 para verificar se o tempo de execução alteraria ou se manteria na mesma média. Esses resultados estão dispostos na Tabela 8 abaixo.

Tabela 8 – Casos para teste no DeepXplore com quinhentas seeds

Hiperparâmetros	Tipo	Caso 1	Caso 2	Caso 3
transformation	choices= [‘light’, ‘occl’, ‘blackout’]	light	light	light
weight_diff	float	1	1	1
weight_nc	float	0.1	0.1	0.1
step	float	10	10	10
seeds	int	500	500	500
grad_iterations	int	1	1	1
threshold	float	0	0	0
target_model	int	choices= [0, 1, 2], default=0	choices= [0, 1, 2], default=0	choices= [0, 1, 2], default=0
start_point	tuple	occlusion upper left corner coordinate, default= (0, 0)	occlusion upper left corner coordinate, default= (0, 0)	occlusion upper left corner coordinate, default= (0, 0)
occl_size	tuple	occlusionsize, default= (10, 10)	occlusionsize, default= (10, 10)	Occlusionsize, default= (10, 10)
Tempo de execução		start: 16/01/2020 at 10:02 AM end: 16/01/2020 at 18:38 PM	start: 24/01/2020 at 08:27 AM end: 25/01/2020 at 01:57 AM	start: 24/01/2020 at 08:27 AM end: 25/01/2020 at 01:16 AM
Tempo total		8 hrs e 36 min	17 hrs e 30 min	16 hrs e 49 min
***image_diffbehavior*[1]**		15	15	17
***averaged covered neurons*[2]**		0.974	0.970	0.970

[1]TOTAL DE IMAGENS QUE CAUSARAM UM COMPORTAMENTO CONTRÁRIO
[2]COBERTURA FINAL ENCONTRADA

Fonte: Elaborada pelo autor (2020).

Dentre todos os parâmetros observados na Tabela 8, observamos que o número de *seeds* tem um impacto maior no tempo de execução, visto que influencia no tempo gasto no processamento da rede neural. Os outros parâmetros são configurados de uma forma para desempenhar o mais minimamente possível enquanto as entradas que são responsáveis por ocasionar o processamento do modelo.

Tabela 9 – Sumário dos resultados

	Caso 1		Caso 2		Caso 3	
	500 seeds	1000 seeds	500 seeds	1000 seeds	500 seeds	1000 seeds
Tempo total	8 hrs e 36 min	166 hrs e 30 min	17 hrs e 30 min	110 hrs e 50 min	16 hrs e 49 min	124 hrs e 18 min
Imagens geradas	15	31	15	36	17	33
Cobertura	0.974	0.979	0.970	0.968	0.970	0.981

Fonte: Elaborada pelo autor (2020).

A Tabela 9 acima nos mostra de forma mais nítida uma comparação dos resultados encontrados nos seis casos executados no DeepXplore. Os casos com mil inputs demoram em torno de cinco dias para finalizar o processo, isso para um banco bem maior de imagens, o tempo sobe exponencialmente. Para um cenário mais simples, no qual você não possui uma configuração de processamento muito alta a disposição para execução de testes mais demorados, percebemos que os casos com quinhentas entradas se tornam mais interessantes, pois possuem quase a mesma cobertura gerada pelas outras abordagens em um tempo muito menor, seu único problema nítido foi que ele gera basicamente a metade das imagens que ocasionam comportamentos contrários que os de mil entradas. Isso se torna muito específico a depender do nível de problema que estamos lidando, a quantidade de imagens que possuímos para auxiliar a ferramenta nessa classificação. Em síntese, um maior número de *seeds* nos proporciona um melhor treinamento do modelo, apesar do tempo crescer exponencialmente, enquanto que um menor número é processado em um tempo muito menor com a mesma cobertura, mas as imagens geradas diminuem pela metade.

Para o DeepFool não foi necessário fazer esse mesmo cenário, pois a ferramenta disponibilizada não era tão complexa quanto o DeepXplore. Como foi dito, nela não era necessária sua execução passando os parâmetros, ela já era configurada para apenas o usuário compilar a ferramenta e checar os resultados. Sua execução levava em média apenas uma hora, pois como seu algoritmo era menos custoso (computacionalmente) e executava apenas um *input* por vez, gerava resultado em uma velocidade muito menor do que a demonstrada pelo DeepXplore.

4.7.2 *Eficácia das ferramentas*

QP4 - *Quão eficazes são as ferramentas de teste de sistemas baseados em ML?*

Para verificar a eficácia das ferramentas, fizemos um estudo utilizando outras duas já citadas anteriormente, o TensorFuzz e o DLFuzz. Seu objetivo é verificar se após a execução dessas ferramentas que visam procurar erros na implementação do modelo, os resultados são melhores ou piores que os relatados no ponto 4.7.1. Devido aos resultados mostrados no ponto anterior, foi decidido trabalhar para teste de eficácia apenas com 500 *seeds* com intuito de agilizar o processo, visto que a cobertura não sofreu impacto considerável. Essas ferramentas que tem o objetivo de checar a implementação o faz por meio de recomendação

de ajustes no modelo utilizado, justamente para otimizar o processo como veremos nos resultados a seguir.

A Tabela 10 abaixo nos mostra os resultados da execução do primeiro caso em que executamos o TensorFuzz na ferramenta DeepXplore. Após utilizar tal ferramenta no código do DeepXplore, foram notados alguns dados: primeiramente a ferramenta não encontrou erros significativos em sua implementação. Após checar a versão de estudo do artigo da ferramenta e a que o DeepXplore utiliza, não possui tanta diferença de versões e talvez seja isso o motivo de não encontrar erros. A diferença notada foi o tempo de execução da ferramenta, enquanto nos primeiros casos possuía uma média de 14 horas com 500 entradas, neste cenário o tempo foi de 9 horas e 15 minutos.

Tabela 10 – Resultado da execução do DeepXplore com o TensorFuzz

Hiperparâmetros	**Tipo**	**Xplore + TensorFuzz**
transformation	choices= ['light', 'occl', 'blackout']	light
weight_diff	float	1
weight_nc	float	0.1
step	float	10
seeds	int	500
grad_iterations	int	1
threshold	float	0
target_model	int	choices= [0, 1, 2], default=0
start_point	tuple	occlusion upper left corner coordinate, default= (0, 0)
occl_size	tuple	occlusionsize, default= (10, 10)
Tempo de execução		start: 31/01/2020 at 07:54 AM end: 31/01/2020 at 17:53 PM
Tempo total		9 hrs e 15min
***image_diffbehavior*[1]**		15
***averaged covered neurons*[2]**		0.979

[1]TOTAL DE IMAGENS QUE CAUSARAM UM COMPORTAMENTO CONTRÁRIO
[2]COBERTURA FINAL ENCONTRADA

Fonte: Elaborada pelo autor (2020).

Para o próximo caso, executamos o DLFuzz em uma versão diferente do DeepXplore que não foi alterada com o TensorFuzz, justamente para verificarmos se possui alguma diferença. A Tabela 11 abaixo nos mostra os resultados da execução do segundo caso em que executamos o DLFuzz na ferramenta DeepXplore. Como resultado, podemos perceber que o tempo de execução permaneceu estável, porém em comparação com a média dos casos envolvendo apenas as quinhentas seeds, o tempo diminuiu um pouco com o teste dessas duas

ferramentas. Diferente de apenas se testar com o TensorFuzz, esse caso envolvendo o DLFuzz obteve um resultado diferente. Apesar da cobertura se manter na média, as imagens que causaram um comportamento contraditório de classificação aumentaram um pouco, isso foi um ponto positivo pois cada imagem que causa um comportamento contraditório está aumentando em consequência o poder de classificação do nosso modelo.

Tabela 11 – Resultado da execução do DeepXplore com o DLFuzz

Hiperparâmetros	Tipo	Xplore + DLFuzz
transformation	choices= ['light', 'occl', 'blackout']	light
weight_diff	float	1
weight_nc	float	0.1
step	float	10
seeds	int	500
grad_iterations	int	1
threshold	float	0
target_model	int	choices=[0, 1, 2], default=0
start_point	tuple	occlusion upper left corner coordinate, default=(0, 0)
occl_size	tuple	occlusionsize, default=(10, 10)
Tempo de execução		start: 04/02/2020 at 07:54 AM end: 04/02/2020 at 17:20 PM
Tempo total		9 hrs e 26 min
***image_diffbehavior*[1]**		19
***averaged covered neurons*[2]**		0.970

[1]TOTAL DE IMAGENS QUE CAUSARAM UM COMPORTAMENTO CONTRÁRIO
[2]COBERTURA FINAL ENCONTRADA

Fonte: Elaborada pelo autor (2020).

Como última parte desse processo, executamos as duas ferramentas para ver que tipo de resultado geraria. Primeiramente utilizamos o TensorFuzz e após ele o DLFuzz, a Tabela 12 mostra os resultados encontrados. Como interpretação deduzimos as seguintes premissas: retreinar com imagens adversas não modifica o tempo de execução e nem varia muito a cobertura, mas ele ajuda muito a gerar novas imagens que são classificadas diferentemente de seus labels que é o objetivo final do xplore.

Tabela 12 – Resultado da execução do DeepXplore com o TensorFuzz e DLFuzz

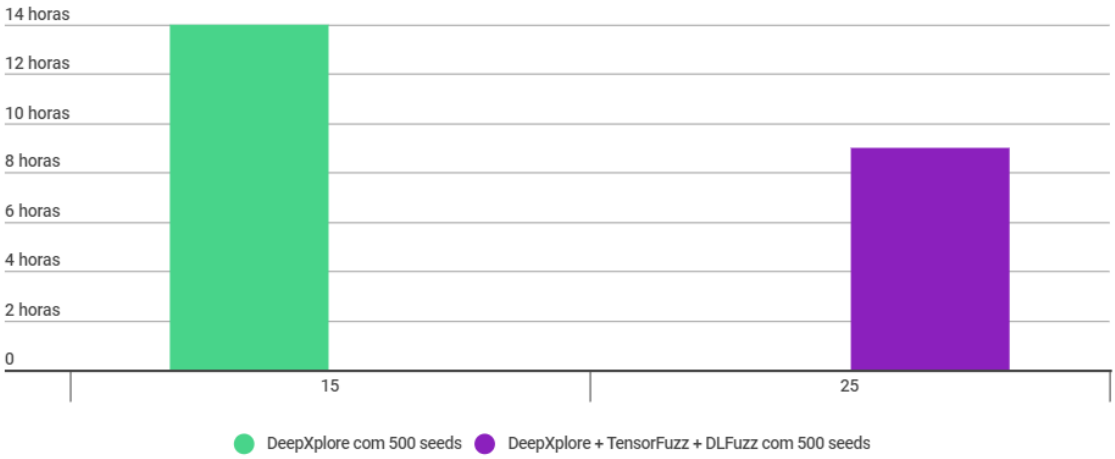
Hiperparâmetros	Tipo	Xplore + TensorFuzz + DLFuzz
transformation	choices= ['light', 'occl', 'blackout']	light
weight_diff	float	1
weight_nc	float	0.1
step	float	10
seeds	int	500
grad_iterations	int	1
threshold	float	0
target_model	int	choices= [0, 1, 2], default=0
start_point	tuple	occlusion upper left corner coordinate, default= (0, 0)
occl_size	tuple	occlusionsize, default= (10, 10)
Tempo de execução		start: 05/02/2020 at 07:52 AM end: 05/02/2020 at 16:47 PM
Tempo total		8 hrs e 55 min
***image_diffbehavior*[1]**		25
***averaged covered neurons*[2]**		0.976

[1]TOTAL DE IMAGENS QUE CAUSARAM UM COMPORTAMENTO CONTRÁRIO
[2]COBERTURA FINAL ENCONTRADA

Fonte: Elaborada pelo autor (2020).

Temos então como resultado um número bem maior que os gerados nos casos normais de execução, aumentaram em dez as imagens geradas chegando a vinte e cinco. Na Figura 19 abaixo podemos ver de forma nítida como essa aplicação de ferramentas aumentou a eficácia da ferramenta consideravelmente. Enquanto os casos do DeepXplore rodando com 500 entradas nos resultaram em 15 imagens em uma média de 14 horas, os casos envolvendo a ferramenta após aplicação do TensorFuzz e DLFuzz com 500 seeds nos resultaram em 25 imagens em uma média de tempo de 9 horas.

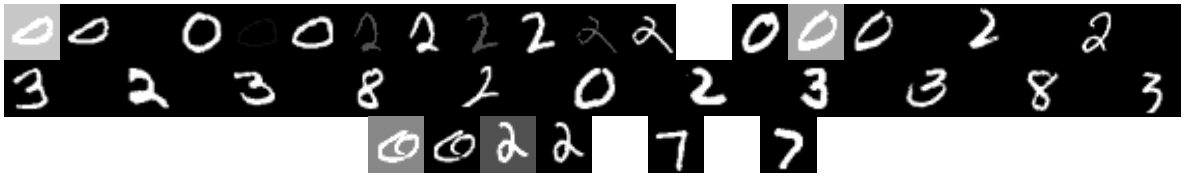
Figura 19 – Comparação de Tempo x Imagens geradas dos resultados gerados



Fonte: Elaborada pelo autor (2020).

Enquanto que os casos utilizando mil *seeds* possuem uma média entre 31 e 36 imagens, esse teste com apenas quinhentas se aproximou muito, deixando claro que não são necessárias tantas entradas para conseguir bons resultados, tudo depende da rotina que você está executando com sua ferramenta. Como exemplo de imagens, na Figura 20 estão algumas das que foram geradas no decorrer do estudo, todas as imagens se encontram em uma pasta do Google Drive[2].

Figura 20 – Exemplos de imagens geradas utilizando o DeepXplore

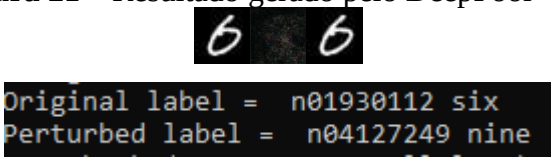


Fonte: Elaborada pelo autor (2020).

Para verificar a se a ferramenta DeepFool é mais eficaz ou não que o DeepXplore montamos o seguinte cenário: executamos primeiramente o DeepFool com uma imagem do MNIST que não foi passada por nenhuma das abordagens de teste citadas anteriormente. A Figura 21 nos mostra qual foi o resultado dessa aplicação.

[2]https://drive.google.com/open?id=14QWrHIIBHDbDuyW5qDE4h_HedPRIdByb

Figura 21 – Resultado gerado pelo DeepFool

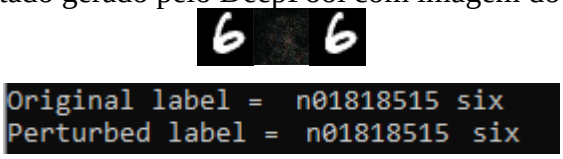


Fonte: Elaborada pelo autor (2019).

Na imagem acima a mais à esquerda é a imagem original utilizada para o estudo, sua classificação manual é tida como um 6. A segunda imagem são os ruídos que a ferramenta inseriu na imagem para gerar o exemplo contraditório, sempre lembrando que o objetivo do DeepFool é fazer com que esses exemplos sejam os mais realistas possíveis, até imperceptíveis ao olho humano. A última imagem foi a gerada pelo DeepFool como exemplo contraditório, tendo as classificações descritas abaixo. A imagem original foi classificada como um seis e a do exemplo gerado foi classificada como um nove, diferentemente da classificação feita inicialmente.

Com esse resultado, o próximo passo foi pegar uma imagem gerada pelo DeepXplore que mais se assemelha-se a utilizada no exemplo acima, visto que as imagens que o DeepXplore gera são pegas alheatórias do *test set*. Na Figura 22 vemos a execução dessa imagem no DeepFool. A imagem utilizada neste teste foi classificada pelo DeepXplore como possível 8, 6 ou 4, no DeepFool a original foi classificada como um 6 e após gerar as perturbações continuou na mesma classificação

Figura 22 – Resultado gerado pelo DeepFool com imagem do DeepXplore



Fonte: Elaborada pelo autor (2019).

Para concluir, as descobertas apresentadas reivindicam claramente um estudo exploratório aprofundado, a fim de identificar os mecanismos de implementação que fazem com que uma ferramenta encontre falhas que a outra não.

4.8 Ameaças à Validade

Uma questão-chave em estudos empíricos como o nosso é a validade dos resultados. Mesmo com um planejamento cuidadoso, diferentes fatores podem afetar esses resultados da

pesquisa. Esta seção discute as decisões que tomamos para reduzir o impacto desses fatores na validade do estudo. Essas ações são principalmente relacionadas ao método de pesquisa adotado para aumentar a confiança do estudo.

Análise de texto completo e extração de dados. O autor da monografia foi responsável por ler completamente os artigos selecionados e extrair informações sobre as ferramentas de teste. Semanalmente, o autor discutiu pelo menos um artigo com a orientadora. Embora tendo realizado essa condução cuidadosa, sem prazo para conclusão, não tomamos outras medidas para redução de riscos. Nesse contexto, algumas ferramentas podem ter sido descartadas de maneira errada. Por exemplo, descartamos ferramentas, como o DeepConcolic (SUN et al., 2018), porque consideramos que não seria viável devido sua configuração de processamento.

Identificando recursos e comparando ferramentas. Não conseguimos encontrar o nome e as características de algumas ferramentas. Nesses casos, nomeamos uma ferramenta como os autores que a propuseram e tentamos ao máximo deduzir algumas informações ausentes. Algumas ferramentas foram, de fato, uma evolução das ferramentas anteriores e, nesses casos, mantivemos o nome da ferramenta anterior ou com alguma pequena mudança. Além disso, algumas ferramentas fornecem personalização das estratégias de teste usadas. No entanto, decidimos usar todas as ferramentas avaliadas em suas configurações padrão e os mesmos procedimentos de execução descritos nos documentos originais. Outras configurações provavelmente dariam resultados diferentes.

Essa extração de dados pode, se mal realizada, nos levar a resultados e conclusões incorretos. Para mitigar essa ameaça, o autor analisou manualmente os pares de falhas e tentou inferir informações semelhantes do log fornecido por cada ferramenta de teste avaliada. Foi realizado com todos os pares gerados para DeepXplore, DeepFool e os casos envolvendo TensorFuzz e DLFuzz.

Generalização de resultados. Não podemos afirmar que nossos resultados sejam generalizados diretamente para outros ambientes e ferramentas, como práticas da indústria. Uma grande ameaça à validade, nesse caso, pode ser as ferramentas selecionadas e os sistemas de destino. Escolhemos duas ferramentas classificadas como White e Black box e não podemos garantir que nossas observações possam ser generalizadas para esta categoria. Tentamos minimizar essa ameaça escolhendo ferramentas que implementam diferentes técnicas de teste, como execução com uma métrica de teste com transformações específicas (DeepXplore) e uma que utiliza ruídos como abordagem metamórfica (DeepFool).

4.9 Considerações finais

Comparando as abordagens mais trabalhadas, foi percebido que cada uma é mais dedicada a um tipo especial de demanda como foi explicado. Avaliamos então a eficiência e a eficácia de duas ferramentas e observamos o DeepXplore e o DeepFool em geral. Como resultados preliminares, ambas as ferramentas apresentaram resultados e particularidades distintos para cada cenário. Uma rotina de testes se mostrou eficaz em melhorar como o modelo trabalha para alcançar o objetivo proposto de testes mais diversificados.

5 CONCLUSÃO

Diante do cenário atual, vemos como a computação, em específico IA, vem tomando sempre novos rumos e alcançando novas áreas a cada dia. Apesar de facilitador, como seu uso vem entrando também em sistemas que fazem trabalhos críticos, essas ferramentas que utilizam de DL precisam de um acompanhamento íntegro com testes que consigam capturar *bugs* que muitas vezes passam despercebidos por conta das peculiaridades existentes no paradigma envolvendo IA. Se a fase de teste da ferramenta não for bem trabalhada, os resultados que o software pode gerir são afetados também, portanto, a especificação dessa rotina deve ser bem definida e alinhada a necessidade do sistema a ser desenvolvido.

Contudo, como foi visto durante o estudo feito, existem ainda muitas ideias soltas quando o assunto é fazer testes concretos voltados para sistemas DL. A área ainda é muito nova e está tomando pequenos passos para consolidar uma ideia mais ampla que englobe essa verificação do sistema. Após os eventos dos carros autônomos que acabaram por sofrer acidentes por tomadas de decisão erradas, a área ganhou muito enfoque por ser algo extremamente necessário visto a evolução que a tecnologia está tomando.

A partir da pesquisa bibliografia e com a execução da proposta nesta pesquisa foi possível mostrar que juntando técnicas diferentes existentes para validar esses sistemas, é possível aumentar a confiabilidade desse software para uma melhor classificação. Assim, uma rotina de testes agrega para melhorar esses modelos que fazem uso de classificação juntamente com a IA.

O estudo aplicado ao DeepXplore obteve resultados bem assertivos, por mostrar que ao fazer uso de ferramentas para testar as diferentes partes do software resultam em uma melhora do resultado gerado. O mais interessante foi perceber que não só isso influencia no resultado, mas também a sequência que tais teste são executados é crucial para um melhor desempenho da ferramenta como classificador.

A abordagem utilizada possibilita, em oportunidades futuras, promover uma rotina contínua de testes para sistemas específicos que se assemelhem ao descrito nesta pesquisa, contribuindo positivamente para a evolução da IA sob a ótica de testes focados para sistemas DL. Além disso, viabiliza um material de visão geral para quem não possui conhecimento em tal área e quer se aprofundar para trabalhos futuros.

REFERÊNCIAS

RUSSEL, Stuart.; NORVIG, Peter. **Inteligência Artificial**. 3a. ed. Rio de Janeiro: Elsevier Editora, 2013.

THEGUARDIAN. **Uber's self-driving car saw the pedestrian but didn't serve - report**. 2018. Disponível em:https://www.theguardian.com/technology/2018/may/08/ubers-self-driving-car-saw-the-pedestrian-but-didnt-swerve-report.Acesso em: 06 de dez. de 2019.

CANALTECH. **Você sabe o que é machine learning? Entenda tudo sobre esta tecnologia**. 2017. Disponível em: https://canaltech.com.br/inovacao/voce-sabe-o-que-e-machine-learning-entenda-tudo-sobre-esta-tecnologia-104100/. Acesso em: 06 de dez. de 2019.

NAQA, Issanel; MURPHY, Martin J. **What is Machine Learning**. 2015

INFINITOMAIZUM. **Em 1956 Nasce a Inteligência Artificial**. 2009. Disponível em: https://infinitomaizum.wordpress.com/2009/11/05/em-1956-nasce-a-inteligencia-artificial/. Acesso em: 06 de dez. de 2019.

MEDIUM. **A diferença entre Inteligência Artificial, Machine Learning e Deep Learning**. 2016. Disponível em: https://medium.com/data-science-brigade/a-diferen%C3%A7a-entre-intelig%C3%AAncia-artificial-machine-learning-e-deep-learning-930b5cc2aa42. Acesso em: 06 de dez. de 2019.

ACCENTURE. **Inteligência Artificial: O que significa e por que é o futuro do crescimento?**.2016. Disponível em: https://www.accenture.com/br-pt/insight-artificial-intelligence-future-growth. Acesso em: 06 de dez. de 2019.

MEDIUM. **Aprendizagem de Máquina: Supervisionado ou Não Supervisionado?**.2016. Disponível em: https://medium.com/opensanca/aprendizagem-de-maquina-supervisionada-ou-n%C3%A3o-supervisionada-7d01f78cd80a. Acesso em: 06 de Dez. de 2019.

XIE, Xiaoyuan; ZHANGZhiyi; CHEN, TsongYueh; LIU, Yang; POON, Pak-lok; XU, Baowen. **METTLE: A Metamorphic Testing Approach to Validating Unsupervised Machine Learning Methods**. 2018.

STATSBOT. **Machine Learning Algorithms: Which One to Choose for Your**. 2017. Disponível em: https://blog.statsbot.co/machine-learning-algorithms-183cc73197c. Acesso em: 06 de dez. de 2019.

MA, Lei; XU,Felix Juefei; ZHANG, Fuyuan; SUN, Jiyuan; XUE, Minhui; LI, Bo; CHEN, Chunyang; SU, Ting; LI, Li; LIU, Yang; ZHAO, Jianjun; WANG, Yadong. **DeepGauge: Multi-granularity Testing Criteria for Deep Learning Systems**. Disponível em: Proceedings of the 33rdACM/IEEE International Conference on Automated Software Engineering, ASE 2018, https://github.com/oci-labs/DeepGauge-ML-Demo, pages 120–131, 2018.

LECUN, Yan; CORTES, Corinna; BURGES, CJ. **Deep learning. Nature**, *v. 521, p.436-444, 2015.*

FARABET, Clement; COUPRIE, Camille; NAJMAN, Laurent; LECUN, Yann. **Learning hierarchical features for scene labeling**. Disponível em: IEEE transactions on pattern analysis and machine intelligence, v. 35, p. 1915–1929, ago. 2013.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. **Imagenet classification with deep convolutional neural networks**. Disponível em: Advances in neural information processing systems, p. 1097–1105, 2012.

SZEGEDY, Christian; LIU, Wei; JIA, Yangqing; SERMANETE, Pierre; REED, Scot; ANGUELOV, Dragomir; ERHAM, Dumitru; VANHOUCKE, Vincent; RABINOVICH, Andrew. **Going deeper with convolutions**. Disponível em: Proceedings of the IEEE conference on computer vision and pattern recognition, p. 1–9, 2015.

HINTON, Geoffrey; DENG, Li; YU, Dong; DAHL, George E.; MOHAMED, Abdel-rahman; JAITLY, Navdeep; SENIOR, Andrew; VANHOUCKE, Vincent; NGUYEN, Patrick; SAINATH, Tara N. **Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups**. Disponível em: IEEE Signal processing magazine, v. 29, p. 82–97, jun. 2012.

JEAN, Sébastien; CHO, Kyunghyun; MEMISEVIC, Roland; BENGIO, Yoshua. **On using very large target vocabulary for neural machine translation**. Disponível em: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, v. 1,p. 1–10, 2015.

SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V. **Sequence to sequence learning with neural networks**. Disponível em: Advances in neural information processing systems, p. 3104–3112, 2014.

DENG, Jia; DONG, Wei; SOCHER, Richard; LI, Li-Jia; LI, Kai;FEI-FEI, Li. **Imagenet: A large-scale hierarchical image database**. Disponível em: Proceedings of the 22nd IEEE Conference on Computer Vision and Pattern Recognition. 2009.

BOJARSKI, Mariusz; TESTA, Davide Del; DWORAKOWSKI, Daniel; FIRNER, Bernhard; FLEPP, Beat; GOYAL, Prasoon; JACKEL, Lawrence D.; MONFORT, Mathew; MULLER, Urs; ZHANG, Jiakai. **End to end learning for self-driving cars**. Disponível em: arXiv preprint arXiv:1604.07316, 2016.

CHEN, Chenyi; SEFF, Ari; KORNHAUSER, Alain; XIAO, Jianxiong. **Deepdriving: Learning affordance for direct perception in autonomous driving**. Disponível em: Proceedings of the IEEE International Conference on Computer Vision,p. 2722–2730, 2015.

CUI, Zhihua; XUE, Fei; CAI, Xingjuan; CAO, Yang; WANG, Gai-ge; CHEN, Jinjun. **Detection of malicious code variants based on deep learning**. Disponível em: IEEE Transactions on Industrial Informatics, v. 14, p. 3187–3196, ago. 2018.

MA, Lei; ZHANG, Fuyuan; SUN, Jiyuan; XUE, Minhui; LI, Boi; JUEFEI-XU, Felix; XIE, Chao; LI, Li; LIU, Yang; ZHAO, Jianjun; WANG, Yadong. **DeepMutation: Mutation Testing of Deep Learning Systems**, 2018.

ABADI, Martín; BARHAM, Paul; CHEN, Jianmin; CHEN, Zhifeng; DAVIS, Andy; DEAN, Jeffrey; DEVIN, Matthieu; GHEMAWAT, Sanjay; IRVING; Geoffrey; ISARD, Michael; KUDLUR, Manjunath; LEVENBERG, Josh; MONGA, Rajat; MOORE, Sherry; MURRAY, Derek G.; STEINER, Benoit; TUCKER, Paul; VASUDEVAN, Vijay; WARDEN, Pete; WICKE, Martin; YU, Yuan; ZHENG, Xiaoqiang. **TensorFlow: A system for large-scale machine learning**. 12th USENIX Symposium on Operating Systems Design and Implementation, p. 265–283, mai.2016. Disponível em: https://ui.adsabs.harvard.edu/link_gateway/2016arXiv160508695A/EPRINT_PDF. Acesso em: 09 de dez. de 2019.

CHOLLET, F.; et al. **Keras**. 2015. Disponível em: https://github.com/fchollet/keras. Acesso em: 09 de Dez. de 2019.

GIBSON, Adam; NICHOLSON, Chris; PATTERSON, Josh; WARRICK, Melanie; BLACK, Alex D.; KOKORIN, Vyacheslav; AUDET, Samuel; ERALY, Susan. **Deeplearning4j: Distributed, open-source deep learning for Java and Scala on Hadoop and Spark**. 2016. Disponível em:https://figshare.com/articles/deeplearning4j-deeplearning4j-parent-0_4-rc3_8_zip/3362644. Acesso em: 09 de dez. de 2019.

GULSHAN, Varun; PENG, Lily; CORAM, Marc; STUMPE, Martin C.; WU, Derek; NARAYANASWAMY, Arunachalam; VENUGOPALAM, Subhashini; WIDNER, Kasumi; MADAMS, Tom; CUADROS, Jorge; et al. **Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs**. Jama, v. 316, p. 2402–2410, 2016.

ANGELOVA, Anelia; KRIZHEVSKY, Alex; VANHOUCKE, Vincent; OGALE, Abhijit S.; FERGUSON, Dave. **Real-time pedestrian detection with deep network cascades**. v. 32, p. 1-32, Jan. 2015.

HUVAL, Brody; WANG, Tao; TANDON, Sameweep; KISKE, Jeff; SONG, Will; PAZHAYAMPALLIL, Joel; ANDRILUKA, Mykhaylo; RAJPURKAR, Pranav; MIGIMATSU, Toki; CHENG-YUE, Royce; et al. **An empirical evaluationof deep learning on highway driving**. Disponível em: arXiv:1504.01716, 2015.Acesso em: 09 de dez. de 2019.

BERK, Richard; HEIDARI, Hoda; JABBARI, Shahin; KEARNS, Michael; ROTH, Aaron. **Fairness in criminal justice risk assessments: the state of the art**. Disponível em: arXiv:1703.09207, 2017. Acesso em: 09 de Dez. de 2019.

SCARBOROUGH, David; SOMERS, Mark Joan. **Neural networks in organizational research: Applying pattern recognition to the analysis of organizational behavior**. American Psychological Association, 2006.

KATZ, Guy; BARETT, Clark; DILL, David L.; JULIAN, Kyle; KOCHENDERFER, Mykel J. **Reluplex: An efficient smt solver for verifying deep neural networks**. Disponível em: International Conference on Computer Aided Verification. Springer, p. 97–117. 2017.

SIANO, Pierluigi; CECATI, Carlo; YU, Hao; KOLBUSZ, Janusz. **Real time operation of smart grids via FCN networks and optimal power flow**. Disponível em: IEEE Transactions on Industrial Informatics, v.8, p. 944–952, nov. 2012.

SCHMIDHUBER, Jürgen. **Deep learning in neural networks: An Overview**. Disponível em: Neural networks, v. 61, p. 85–117, 2015.

DIGITALVYDIA. **A Comprehensive Guide to Types of Neural Networks**. 2019. Disponível em: https://www.digitalvidya.com/blog/types-of-neural-networks/. Acesso em: 09 de dez. de 2019.

PATHMIND. **A Beginner's Guide to Backpropagation in Neural Networks**. 2019. Disponível em: https://pathmind.com/wiki/backpropagation. Acesso em: 10 de dez. de 2019.

DEVMEDIA. **Redes Neurais Artificiais: Algoritmo Backpropagation**. 2013. Disponível em: https://www.devmedia.com.br/redes-neurais-artificiais-algoritmo-backpropagation/28559. Acesso em: 10 de dez. de 2019.

IEEE. ***Standard Glossary of Software Engineering Terminology***, Standard 610.12. IEEE Press, 1990.

DIJKSTRA, Edsger W. **Structured programming**. Capítulo I: Notes on structured programming, p.1–82. Academic Press Ltd., London, UK, 1972. Disponível em: http://dl.acm.org/citation.cfm?id=1243380.1243381. Acesso em: 09 de dez. de 2019.

TOMELIN Marcio. **Testes de Software a partir da Ferramenta Visual Test**. Blumenau, junho. 2001.

PACULA, Maciej. **Unit-Testing Statistical Software**. 2011. Disponível em: http://blog.mpacula.com/2011/02/17/unit-testing-statistical-software/. Acesso em: 10 de dez. de 2019.

RAMANATHAN, Arvind; PULLUM, Laura L.; HUSSAIN, Faraz; CHAKRABARTY, Dwaipayan; JHA, SumitKumar. **Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision**. Design, Automation Test in Europe Conference Exhibition, p. 786–791, mar. 2016.

AMERSHI, Saleema; BEGEL, Andrew; BIRD, Christian; DELINE, Rob; GALL, Herald; KAMAR, Ece; NAGAPPAN, Nachi; NUSHI, Besmira; ZIMMERMANN, Tom. **Software engineering for machine learning: A case study**. Disponível em: Proc. ICSE, 2019. Acesso em: 09 de Dez. de 2019.

BARR, Earl T.; HARMAN, Mark; MCMINN, Phil; SHAHBAZ, Muzammil; YOO, Shin. **The oracle problem in software testing: A survey**. Disponível em: IEEE transactions on software engineering, v. 41, p. 507–525, 2015.

MURPHY, Chris; KAISER, Gail E.; ARIAS, Marta. **An approach to software testing of machine learning applications**. Disponível em: SEKE, v. 167, 2007.

DAVIS, Martin D.; WEYUKER, Elaine J. **Pseudo-oracles for non-testable programs**. Disponível em: Proceedings of the ACM 81 Conference, v. 81, p. 254–257, 1981.

XIE, Xiaofei; MA, Lei; JUEFEI-XU, Felix; CHEN, Hongxu; XUE, Minhui; LI, Boi; LIU, Yang; ZHAO, Jianjun; YIN, Jianxiong; SEE, Simon. **DeepHunter: Hunting Deep Neural Network Defects via Coverage-Guided Fuzzing**. 2018.

SEKHON, Jasmine; FLEMING, Cody. **Towards Improved Testing For Deep Learning**. Disponível em: Proc. ICSE, 2019. Acesso em: 09 de Dez. de 2019.

QI, Zhixin; WANG, Hongzhi; LI, Jianzhong; GAO, Hong. **Impacts of Dirty Data: An Experimental Evaluation**. Disponível em: https://github.com/qizhixinhit/Dirty-dataImpacts 2018.

ODENA, Augustus; GOODFELLOW, Ian. **TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing**. Disponível em: https://github.com/brain-research/tensorfuzz. Acesso em: 09 de dez. de 2019.

ZHANG, Yuhao; CHEN, Yifan; CHEUNG, Shing-Chi; XIONG, Yingfei; ZHANG, Lu. **An empirical study on tensorflow program bug**s. Disponível em: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, https://github.com/ForeverZyh/TensorFlow-Program-Bugs, p. 129–140, 2018.Acesso em: 12 de dez. de 2019.

GUO, Jianmin; JIANG, Yu; ZHAO, Yue; CHEN, Quan; SUN, Jiaguang. **Dlfuzz: differential fuzzing testing of deep learning systems**. Disponível em: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, https://github.com/turned2670/DLFuzz, p. 739–743, 2018.Acesso em: 12 de dez. de 2019.

MURPHY, Christian; KAISER, Gail; HU, Lifeng. **Properties of machine learning applications for use in metamorphic testing**. Disponível em: SEKE, 2008. Acesso em: 12 de dez. de 2019.

XIE, Xiaoyuan; HO, Joshua W. K.; MURPHY, Christian; KAISER, Gail; XU, Baowen; CHEN, TsongYueh. **Testing and validating machine learning classifiers by metamorphic testing**. Disponível em: Syst.Softw., v. 84, p. 544–558, abril. 2011.

DWARAKANATH, Anurag; AHUJA, Manish; SIKAND, Samarth; RAO, Raghotham M.; BOSE, R. P. Jagadeesh Chandra; DUBASH, Neville; PODDER, Sanjay. **Identifying implementation bugs in machine learning based image classifiers using metamorphic testing.** Disponível em: 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, https://github.com/verml/VerifyML, p. 118–128, 2018.

DELAMARO, Márcio Eduardo; MALDONADO, José Carlos, Mario Jino. **Introdução ao Teste de Software**. editora Elsevier, 2016. 2ª Edição.

NETO, Arilo Claudio Dias. **Engenharia de Software Magazine - Introdução a Teste de Software**. All content following this page was uploaded by Arilo Claudio Dias Neto on 14 May 2015. Disponível em: https://edisciplinas.usp.br/pluginfile.php/3503764/mod_resource/content/3/Introducao_a_Teste_de_Software.pdf. Acesso em: 02 de ago. de 2019.

MEDIUM. **Você sabe o que é Teste Caixa Preta e Teste Caixa Branca?**. 2019. Disponível em: https://medium.com/@ingrid.carvalho.mo/voc%C3%AA-sabe-o-que-%C3%A9-teste-caixa-branca-e-teste-caixa-preta-9a2d08fe9d0c. Acesso em: 12 de dez. de 2019.

PEI, Kexin; CAO, Yinzhi; YANG, Junfeng; JANA, Suman. **Deepxplore: Automated whitebox testing of deep learning systems**. Disponível em: Proceedings of the 26th Symposium on Operating Systems Principles, https://github.com/peikexin9/deepxplore, p. 1–18, 2017.

MA, Lei; JUEFEI-XU, Felix; XUE, Minhui; LI, Bo; LI, Li; LIU, Yang; ZHAO, Jianjun. **Deepct: Tomographic combinatorial testing for deep learning systems**. Disponível em: Proc. SANER, p. 614–618, fev. 2019.

ZHANG, Mengshi; ZHANG, Yuqun; ZHANG, Lingming; LIU, Cong; KHURSHID, Sarfraz. **DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems**. Disponível em: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, https://github.com/eweill/DeepROAD, p. 132–142, 2018.

TIAN, Yuchi; JANA, Suman; PEI, Kexin; RAY, Baishakhi. **DeepTest: Automated testing of deep-neural-network-driven autonomous cars**. Disponível em: Proceedings of the 40th International Conference on Software Engineering, https://deeplearningtest.github.io/deepTest/, p. 303–314, 2018.

ZHANG, Jie M.; HARMAN, Mark; MA, Lei; LIU, Yang. **Machine Learning Testing: Survey, Landscapes and Horizons**. 2019.

BSTQB. **Certified Tester Foundation Level Syllabus**. 2019. Disponível em: https://www.bstqb.org.br/uploads/syllabus/syllabus_ctfl_2018br.pdf. Acesso em: 19 de dez. de 2019.

GOODFELLOW, Ian J.; SHLENS, Jonathan; SZEGEDY, Christian. **Explaining and Harnessing Adversarial Examples**. 2014.

ENGSTROM, Logan; TRAN, Brandon; TSIPRAS, Dimitris; SCHMIDT, Ludwig; MADRY, Aleksander. **A Rotation and a Translation Suffice: Fooling cnns with Simple Transformations**. 2019.

GILMER, Justin; ADAMS, Ryan P.; GOODFELLOW, Ian J.; ANDERSEN, David; DAHL, GeordeE. **Motivating the Rules of the Game for Adversarial Example Research**. 2018.

BRAIEK, Houssem Ben; KHOMH, FOUTSE. **On Testing Machine Learning Programs**. 2018.

MOOSAVI-DEZFOOLI, Seyed-Mohsen; FAWZI, Alhussein; FROSSARD, Pascal. **DeepFool: a simple and accurate method to fool deep neural networks**. Disponível em: https://github.com/lts4/deepfool, 2015.

RIBEIRO, Marco Tulio; SINGH, Sameer; GUESTRIN, Carlos. **"Why Should I Trust You?": Explaining the Predictions of Any Classifier**. Disponível em: https://github.com/marcotcr/lime-experiments. 2016.

UDESHI, Sakshi; ARORA, Pryanshu; CHATTOPADHYAY, Sudipta. **Automated directed fairness testing**. Disponível em: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, https://github.com/sakshiudeshi/Aequitas, 2018.

COOK, R. Dennis; WEISBERG, Sanford. **Characterizations of an empirical influence function for detecting influential cases in regression**. Disponível em: Technometrics, v. 22, p. 495–508, 1980.

KOH, PangWei; LIANG, Percy. **Understanding Black-box Predictions via Influence Functions**. Disponível em: https://github.com/kohpangwei/influence-release. 2017.

SUN, Youcheng; WU, Min; RUAN, Wenjie; HUANG, Xiaowei; KWIATKOWSKA, Marta; KROENING, Daniel. **Concolic Testing for Deep Neural Networks**. Disponível em: https://github.com/TrustAI/DeepConcolic. 2018.

GOODFELLOW, Ian J.; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. **Generative Adversarial Networks**. Disponível em: https://github.com/lisa-lab/pylearn2/tree/master/pylearn2/scripts/papers/maxout. *2014.*

SUN, Youcheng; HUANG, Xiaowei; KROENING, Daniel; SHARP, James; HILL, Matthew; ASHMORE, Rob. **Testing Deep Neural Networks**. Disponível em: https://github.com/theyoucheng/deepcover *2018.*

HAYHURST, Kelly; VEERHUSEN, Dan; CHILENSKI, John; RIERSON, Leanna. **A practical tutorial on modified condition/decision coverage**. Disponível em: NASA Langley Technical Report Server, 2001.

LECUN, Yann; BOTTOU, Leon; BENGIO, Yoshua; HAFFNER, Patrick. **Gradient-Based Learning Applied to Document Recognition**. Disponível em: Proceedings of the, v. 86, p.2278–2324, 1998.

KRIZHEVSKY, Alex; HINTON, Geoffrey. **Learning Multiple Layers of Features from Tiny Images**. Disponível em: Technical Report Citeseer. 2009.

LECUN, Yann; CORTES, Corrina; BURGES, Christopher JC. **The MNIST database of handwritten digits**. 1998.

UDACITY-CHALLENGE. **Using Deep Learning to Predict Steering Angles**. Disponível em: https://github.com/udacity/self-driving-car. 2016.

KIM, Jinhan; FELDT, Robert; YOO, Shin. **Guiding Deep Learning System Testing using Surprise Adequacy**. Disponível em: https://github.com/coinse/sadl, *2018.*

LI, Zenan;MA, Xiaoxing;XU, Chang; CAO, Chun; XU, Jingwei; LU, Jian. **Boosting operational DNN testing efficiency through conditioning**. Disponível em: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, https://figshare.com/s/6bfb5fb0529a02b86cb5, 2019.

LECUN, Yann;CORTES, Corinna;BURGES, Christopher JC. **MNIST handwritten digit database**. Disponível em: http://yann.lecun.com/exdb/mnist 2. 2010.

KRISHNAM, Sanjay; WANG, Jiannan; WU, Eugene; FRANKLIN, Michael; GOLDBERG, Kenneth. **ActiveClean: interactive data cleaning for statistical modeling**. Disponível em: Proceedings of the VLDB Endowment, v. 9, p. 948-959. 2016.

KRISHNAM, Sanjay; WU, Eugene; FRANKLIN, Michael; GOLDBERG, Kenneth. **BoostClean: Automated Error Detection and Repair for Machine Learning**. 2017.

KRISHNAM, Sanjay; WANG, Jiannan; WU, Eugene; FRANKLIN, Michael; GOLDBERG, Kenneth; KRASKA, Tim; MILO, Tova. **SampleClean: Fast and Reliable Analytics on Dirty Data**. Disponível em: IEEE Data Eng. Bull, v. 38, p. 59-75. http://www.sampleclean.org/. 2015.

HYNES, Nick; SCULLEY, D.; TERRY, Michael. **The Data Linter: Lightweight Automated Sanity Checking for ML Data Sets**. Disponível em: http://github.com/brain-research/data-linter. *2017.*

SELSAM, Daniel; LIANG, Percy; DILL, David. **Developing Bug-Free Machine Learning Systems with Formal Mathematics**. Disponível em: https://github.com/dselsam/certigrad. *2017.*

PHAM, Hung Viet; LUTELLIER, Thibaud; QI, Weizhen; TAN, Lin; **CRADLE: Cross-Backend Validation to Detect and Localize Bugs in Deep Learning Libraries**. Disponível em: Proceedings ICSE 2019, v. 1, p. 1027-1038. 2019.

GROOSE, Roger; DUVENAUD, David. **Testing MCMC code**. 2014

SRISAKAOKUL, Siwakorn; WU, Zhengkai; ASTORGA, Angello; ALEBIOSU, Oreluwa; XIE, Tao. **Multiple-Implementation Testing of Supervised Learning Software**. Disponível em: AAAI Workshops. 2016

MA, Wei; PAPADAKIS, Mike; TSAKMALIS, Anestis; CORDY, Maxine; TRAON, Yves Le. **Test Selection for Deep Learning Systems**. 2019.

GU, Shixiang; RIGAZIO, Luca. **Towards Deep Neural Network Architectures Robust to Adversarial Examples**. 2014

MA, Lei; ZHANG, Fuyuan; XUE, Minhui; LI, Bo; LIU, Yang; ZHAO, Jianjun; WANG, Yadong. **Combinatorial Testing for Deep Learning Systems**. 2018

LI, Zenan; MA, Xiaoxing; XU, Chang; CAO, Chun. **Structural Coverage Criteria for Neural Networks Could Be Misleading**. Disponível em: Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, p. 89-92. 2019.

MANGAL, Ravi; NORI, Aditva; ORSO, Alessandro. **Robustness of Neural Networks: A Probabilistic and Practical Approach**. 2019.

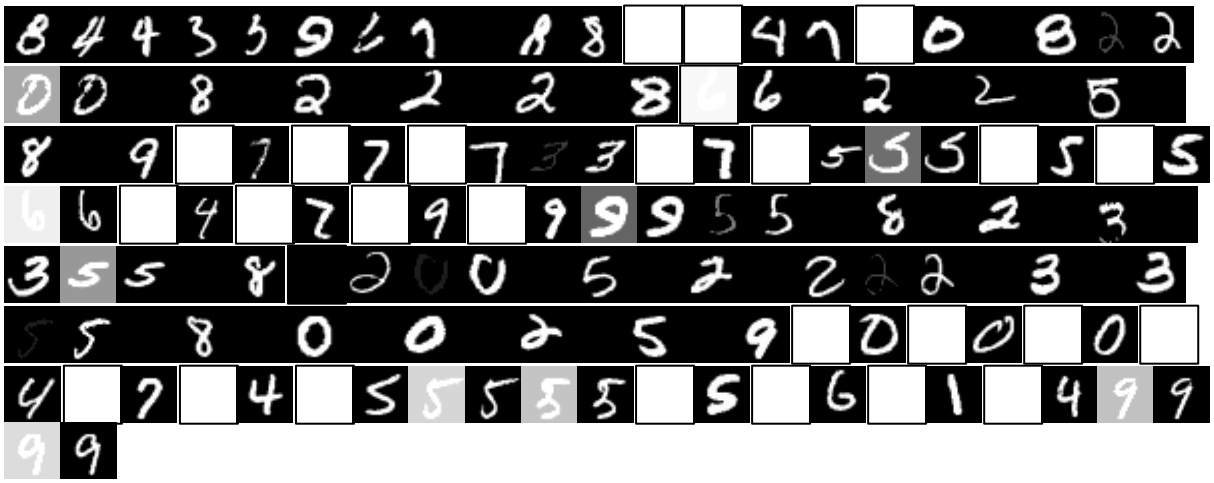
SHARMA, Arnab; WEHRHEIM, Heike. **Testing Machine Learning Algorithms for Balanced Data Usage**. Disponível em: *12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, p. 125-135. 2019.

BYUN, Taejoon; SHARMA, Vaibhav; VIJAYAKUMAR, Abhishek; RAYADURGAM, Sanjai; COFER, Darren**. Input Prioritization for Testing Neural Networks**. Disponível em: https://github.com/bntejn/keras-prioritizer, 2019.

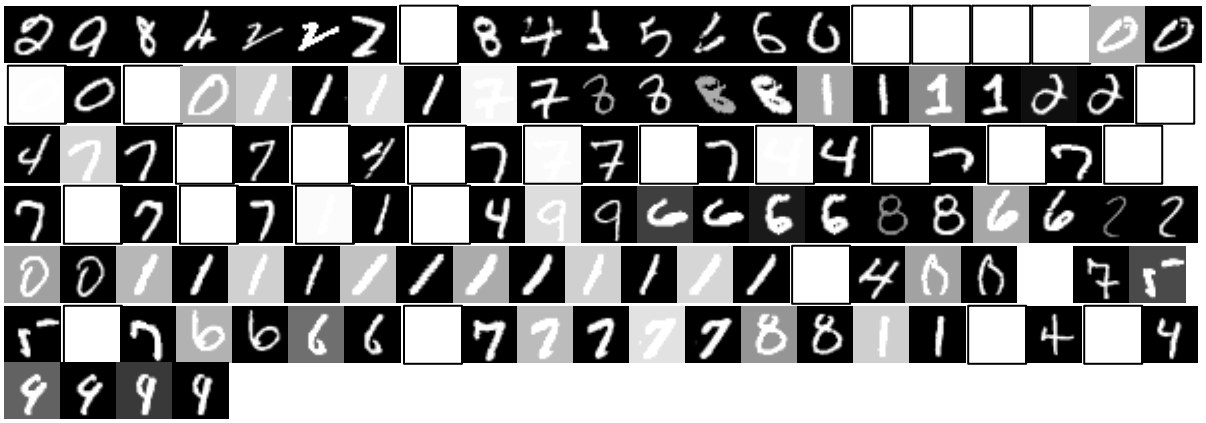
APÊNDICE A – IMAGENS GERADAS APÓS APLICAÇÃO DA ABORDAGEM NO DEEPXPLORE

Este apêndice é parte do conjunto de resultados produzidos para a aplicação da abordagem apresentada neste trabalho. Contém todas as imagens que foram geradas na execução da ferramenta.

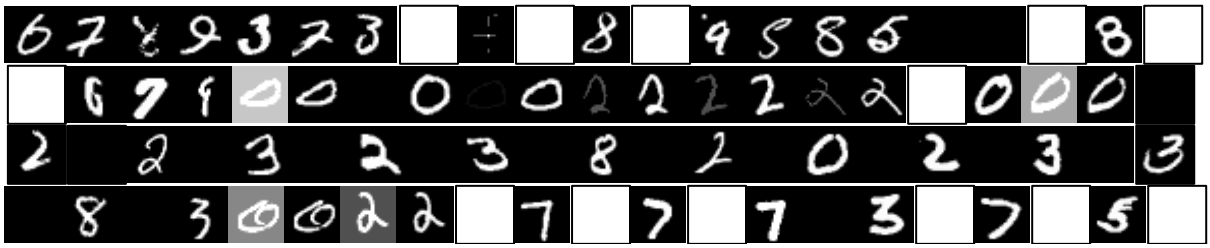
- **DeepXplore + TensorFuzz**

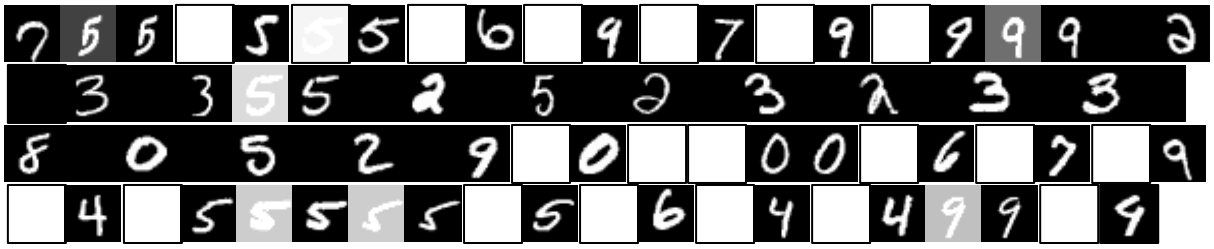


- **DeepXplore + DLFuzz**



- **DeepXplore + TensorFuzz + DLFuzz**





APÊNDICE B – CRITÉRIOS DE TESTE

Apêndice destinado a exposição dos critérios de teste encontrados nos artigos estudados.

Neuron-level Coverage Criteria

- **k-multisection neuron coverage:** Dado um neurônio n, a cobertura de neurônios de k-multisection mede a profundidade com que o conjunto dado de entradas de teste T cobre a faixa [lown, highn]. Para quantificar isso, dividimos o intervalo [lown, highn] em k seções iguais (isto é, k-multisections), para k> 0. Escrevemos (Sni) para denotar o conjunto de valores na i-ésima seção para 1 ≤ i ≤ k.

$$\mathrm{KMNCov}(T,k) = \frac{\sum_{n\in N} |\{S_i^n \mid \exists \mathbf{x} \in T : \phi(\mathbf{x},n) \in S_i^n\}|}{k \times |N|}.$$

  Se as entradas do teste seguem uma distribuição estatística similar com os dados de treinamento, uma saída de neurônio raramente localiza na região do caso de canto.

  ENCONTRADO EM: DeepGauge.

- **Neuron boundary coverage:** A cobertura de limite de neurônio mede quantas regiões de caso de canto (ambos os limites superior e inferior) foram cobertas pelo conjunto de entrada de teste fornecido T. É definido como a relação entre o número de casos de canto cobertos e o número total de casos de canto (2 × | N |)

$$\mathrm{NBCov}(T) = \frac{|\mathrm{UpperCornerNeuron}| + |\mathrm{LowerCornerNeuron}|}{2 \times |N|}.$$

  ENCONTRADO EM: ConcolicTesting, DeepGauge.

- **Strong neuron activation coverage:** A cobertura de ativação de neurônios fortes mede quantos casos de canto (o valor do limite superior highn) foram cobertos pelas entradas de teste T. É definido como a proporção do número de casos de canto superior cobertos e o número total de casos de canto (| N |):

$$\mathrm{SNACov}(T) = \frac{|\mathrm{UpperCornerNeuron}|}{|N|}.$$

  ENCONTRADO EM: DeepGauge.

- **Neuron coverage:** É definido como a proporção de neurônios únicos que são ativados para determinadas entradas e o número total de neurônios em um DNN

$$Neuron\ \ Coverage = \frac{|Activated\ \ Neurons|}{|Total\ \ Neurons|}$$

  Um neurônio individual é considerado ativado se a saída do neurônio (dimensionada pelas saídas da camada geral) for maior que um limite de DNN (definido no algoritmo).

ENCONTRADO EM: Concolic Testing, DeepTest, DeepXplore, Towards Improved Testing For Deep Learning, Structural Coverage Criteria for Neural Networks Could Be Misleading.

- **neuron-activation configuration:** Para um conjunto de neurônios Li = {n1, n2, ..., Nk}, uma configuração de ativação do neurônio é uma tupla c = (b1, b2, ..., bk), onde bi = {0, 1}. Uma configuração c = (b1, b2, ..., bk) de Li é coberta por T se houver uma entrada de teste x x T, tal que bi = A (ni, x) para 1 ≤ i ≤ k. FC NA (T, Li), para representar que todas as configurações de ativação neuronal de T. Li e NCNA (T, Li) representam o número de configurações de ativação neuronal de T. Li.

  ENCONTRADO EM: CombinatorialTesting.

- **Sign-Sign Coverage:** Para cobertura SCC, primeiro definimos um requisito RSSC (α) para um par de neurônios α = (nk, i, nk + 1, j):

$$\{\exists x_1, x_2.\ ap[x_1]_{k,i} \neq ap[x_2]_{k,i} \wedge ap[x_1]_{k+1,j} \neq ap[x_2]_{k+1,j} \wedge \bigwedge_{1 \leq l \leq s_k, l \neq i} ap[x_1]_{k,l} - ap[x_2]_{k,l} = 0\}$$

  e após é encontrado

$$\Re_{SSC} = \bigcup_{2 \leq k \leq K-2, 1 \leq i \leq s_k, 1 \leq j \leq s_{k+1}} \Re_{SSC}((n_{k,i}, n_{k+1,j}))$$

  ENCONTRADO EM: TestingDeep Neural Networks.

- **Metamorphic testing:** A força da MT reside na sua capacidade de resolver automaticamente o problema oracle de teste através de relações metamórficas (MRs). Em particular, seja p uma representação matemática do programa que mapeie entradas de programa para saídas de programa (por exemplo, p⟦i⟧ = o). Assumindo fI e fO são duas funções específicas para transformar o domínio de entrada e saída respectivamente, e elas satisfazem o seguinte MR formulação:

$$\forall i,\ p[\![f_I(i)]\!] = f_O(p[\![i]\!])$$

  ENCONTRADO EM: DeepRoad.

- **Input validation:** O objetivo da validação de entrada (IV) é garantir que apenas dados corretamente formados possam ser aceitos pelos sistemas, e os dados malformados devem ser rejeitados antes da execução. A razão é que uma entrada inválida pode desencadear um mau funcionamento dos componentes, o que torna o sistema inseguro.

  ENCONTRADO EM: Deep Road.

- **MC/DC:** E um método para medir até que ponto o software crítico para a segurança foi adequadamente testado. Em seu núcleo está a ideia de que, se uma escolha pode ser feita, todos os fatores possíveis (condições) que contribuem para essa escolha (decisão) devem ser testados.

  ENCONTRADO EM: TestingDeep Neural Networks.

- **SS coverage:** Um par de características α = (ψk, i, ψk + 1, j) é coberto por SS em dois casos de teste x1, x2, denotado por SS (α, x1, x2), se as seguintes condições forem satisfeitas pelas instâncias DNN [x1] e N [x2]:

– $sc(\psi_{k,i}, x_1, x_2)$ *and* $nsc(P_k \setminus \psi_{k,i}, x_1, x_2)$;
– $sc(\psi_{k+1,j}, x_1, x_2)$.

where P_k *is the set of nodes in layer* k.

ENCONTRADO EM: Testing Deep Neural Networks, Structural Coverage Criteria for Neural Networks Could Be Misleading.

- **VS coverage:** Dada uma função de valor g, um par de características α = (ψk, i, ψk + 1, j) é coberto por VS por dois casos de teste x1, x2, denotado por V Sg (α, x1, x2), se as seguintes condições estão satisfeitos com as instâncias DNN N [x1] e N [x2]:
  – vc(g, ψk,i, x1, x2) and nsc(Pk, x1, x2);
  –sc(ψk+1,j , x1, x2).

  ENCONTRADO EM: TestingDeep Neural Networks.

- **SV coverage:** Dada uma função de valor g, um par de características α = (ψk, i, ψk + 1, j) é coberto por SV por dois casos de teste x1, x2, indicado por SV g (α, x1, x2), se as seguintes condições estão satisfeitos com as instâncias DNN N [x1] e N [x2]:
  – sc(ψk,i, x1, x2) and nsc(Pk \ ψk,i, x1, x2);
  – vc(g, ψk+1,j , x1, x2) and nsc(ψk+1,j , x1, x2).

  ENCONTRADO EM: TestingDeep Neural Networks.

- **VV coverage:** Dadas duas funções de valor g1 e g2, um par de características α = (ψk, i, ψk + 1, j) é coberto por VV por dois casos de teste x1, x2, denotado por VV g1, g2 (α, x1, x2), se as seguintes condições forem satisfeitas pelas instâncias DNN N [x1] e N [x2]:
  – vc(g1, ψk,i, x1, x2) and nsc(Pk, x1, x2);
  – vc(g2, ψk+1,j , x1, x2) and nsc(ψk+1,j , x1, x2).

  ENCONTRADO EM: TestingDeep Neural Networks.

- **2-way coverage:** A cobertura 2-way e aplicada em três neurônios distintos, sendo capaz de cobrir, (1) o efeito independente de uma condição (ativação do neurônio em Lk-1) tem um resultado (valor de neurônio na próxima camada, Lk), (2) as falhas que podem surgir por causa da 'interação' ou valores de ativação de neurônios na mesma camada Lk − 1.

$$F_{n,t} = \phi(t, n_{i,k-1}) - \phi(t, n_{j,k-1}) + \phi(t, n_{q,k}).$$

  ENCONTRADO EM: Towards Improved Testing For Deep Learning.

- **Activation Trace and SurpriseAdequacy:** Dado um conjunto de treinamento T, primeiro calculamos AN (T) registrando os valores de ativação de todos os neurônios usando cada entrada no conjunto de dados de treinamento. Posteriormente, dada uma nova entrada x, nós medimos quão surpreendente x é quando comparado a T comparando o traço de ativação de x com AN (T). Essa medida de similaridade quantitativa é chamada Surprise Adequacy (SA).

  ENCONTRADO EM: Guiding Deep Learning System Testing using Surprise Adequacy.

- **Likelihood-based Surprise Adequacy:** A SA baseada em verossimilhança (LSA) usa o KDE para estimar a densidade de probabilidade de cada valor de ativação em AN (T) e obtém a surpresa de uma nova entrada em relação à densidade estimada. O KDE (Kernel Density Estimation) é uma maneira de estimar a função de densidade de probabilidade de uma dada variável aleatória. A função de densidade resultante permite a estimativa da probabilidade relativa de um valor específico da variável aleatória. Dada uma matriz de largura de banda H e a função de kernel Gaussian K, o rastreio de ativação da nova entrada x, e xi ∈ T, KDE produz a função de densidade ˆf da seguinte forma:

$$\hat{f}(x) = \frac{1}{|A_{N_L}(\mathbf{T})|} \sum_{x_i \in \mathbf{T}} K_H(\alpha_{N_L}(x) - \alpha_{N_L}(x_i))$$

  ENCONTRADO EM: Guiding Deep Learning System Testing using Surprise Adequacy.

- **Distance-based Surprise Adequacy:** O DSA tem como objetivo comparar a distância do AT de uma nova entrada x a ATs conhecidos pertencentes à sua própria classe, cx, à distância conhecida entre ATs na classe cx e ATs em outras classes em C \ {cx} .Se o primeiro é relativamente maior que este último, x seria uma entrada surpreendente para a classe cx para o sistema DL de classificação D. Embora existam várias maneiras de formalizar isso, selecionamos uma simples e calculamos DSA como a razão entre dista e distb:

$$DSA(x) = \frac{dist_a}{dist_b}$$

  ENCONTRADO EM: Guiding Deep Learning System Testing using Surprise Adequacy.

- **Surprise Coverage:** Dado um conjunto de entradas, também podemos medir o intervalo de valores de SA que o conjunto cobre, chamado Surprise Coverage (SC). Como tanto o LSA quanto o DSA são definidos em espaços contínuos, usamos o bucketing para diferenciar o espaço de surpresa e definir a cobertura surpresa baseada na probabilidade (LSC) e a cobertura surpresa baseada na distância (DSC - Distance-based Surprise Coverage). Dado um limite superior de U e buckets B = {b1, b2, ..., bn} que dividem (0, U) em n segmentos SA, SC para um conjunto de entradas X é definido da seguinte forma:

$$SC(X) = \frac{|\{b_i \mid \exists x \in X : SA(x) \in (U \cdot \frac{i-1}{n}, U \cdot \frac{i}{n}]\}|}{n}$$

  ENCONTRADO EM: Guiding Deep Learning System Testing using Surprise Adequacy.

- **Fast Gradient Sign Method:** Seja θ os parâmetros de um modelo, x a entrada para o modelo, y os alvos associados a x (para tarefas de aprendizado de máquina que tenham alvos) e J (θ, x, y) o custo usado para treinar a rede neural. Podemos linearizar a função de custo em torno do valor atual de θ, obtendo uma perturbação restrita max-norm ótima de η = sinal (∇xJ (θ, x, y)).

  ENCONTRADO EM: Explaining and Harnessing Adversarial Examples.

Layer-level Coverage Criteria

- **top-k neuron coverage:** A cobertura do neurônio k mede quantos neurônios já foram os k neurônios mais ativos em cada camada. É definido como a razão entre o número total de neurônios do topo-k em cada camada e o número total de neurônios em um DNN

$$\text{TKNCov}(T,k) = \frac{|\bigcup_{\mathbf{x}\in T}(\bigcup_{1\le i\le l}\text{top}_k(\mathbf{x},i))|}{|N|}.$$

  Os neurônios da mesma camada de um DNN geralmente desempenham papéis semelhantes e os principais neurônios ativos de diferentes camadas são indicadores importantes para caracterizar a principal funcionalidade de um DNN.

  ENCONTRADO EM: DeepGauge.

- **top-k neuron patterns:** Dada uma entrada de teste x, a sequência dos neurônios top-k em cada camada também forma um padrão. Intuitivamente, os padrões de neurônios top-k denotam diferentes tipos de cenários ativados dos principais neurônios hiperativos de cada camada.

  formally, a pattern is an element of $2^{L_1} \times 2^{L_2} \times \cdots \times 2^{L_l}$, where 2^{L_i} is the set of subsets of the neurons on i-th layer, for $1 \le i \le l$. Given the test input set T, the number of top-k neuron patterns for T is defined as:

$$\text{TKNPat}(T,k) = |\{(\text{top}_k(\mathbf{x},1),\ldots,\text{top}_k(\mathbf{x},l)) \mid \mathbf{x}\in T\}|.$$

  ENCONTRADO EM: DeepGauge.

- **t-way combination sparsecoverage:** Dado um conjunto de testes T e um conjunto de neurônios Li, usamos Θ para modificar o conjunto de todas as combinações t de neurônios em Li. Então a cobertura esparsa da combinação t-way representava das quais todas as configurações de ativação de neurônios são cobertas por T.

$$\text{TWsCov}(T,L_i) = \frac{|\{\theta\in\Theta|\ FC\_NA(T,\theta)\}|}{|\Theta|}$$

  ENCONTRADO EM: Combinatorial Testing, Structural Coverage Criteria for Neural Networks Could Be Misleading.

- **t-way combination dense coverage:** Para um conjunto de entradas de teste T e um conjunto de neurônios Li, a cobertura densa de combinação t-way pode ser calculada como abaixo.

$$\text{TWdCov}(T,L_i) = \frac{\sum_{\theta\in\Theta} NC\_NA(T,\theta)}{2^t|\Theta|}$$

  ENCONTRADO EM: Combinatorial Testing, Structural Coverage Criteria for Neural Networks Could Be Misleading.

- **Lipschitz Continuity:** Uma rede N é dita ser Lipschitz contínua se houver uma constante real c ≥ 0 tal que, para todo x1, x2 ∈ DL1

$$||v[x_1]_1 - v[x_2]_1|| \leq c \cdot ||x_1 - x_2||$$

ENCONTRADO EM: Concolic Testing, Robustness of Neural Networks: A Probabilistic and Practical Approach.

- **Gradient:** G pode ser calculado utilizando a regra de cadeia no cálculo, isto é, calculando as derivadas de camada iniciando a partir da camada do neurônio que gera y até alcançar a camada de entrada que toma x como entrada. Observe que a dimensão do gradiente G é idêntica àquela da entrada x.

  ENCONTRADO EM: DeepXplore.

- **Balanced Data Usage:** baseia-se na ideia de que o aluno deve tratar todos os dados do conjunto de treinamento igualmente, desconsiderando questões como nomes ou pedidos de recursos ou pedidos de instâncias de dados. Em vez de comparar com alguma verdade básica, definimos "aprender o que está nos dados" usando todos os dados de treinamento da mesma maneira**.**

ENCONTRADO EM: Testing Machine Learning Algorithms for Balanced Data Usage.