MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

EDUARDO MARTÍN MALVACIO

IDENTIFICAÇÃO DE MALWARE UTILIZANDO APRENDIZADO POR TRANSFERÊNCIA

RIO DE JANEIRO
2022

EDUARDO MARTÍN MALVACIO

IDENTIFICAÇÃO DE MALWARE UTILIZANDO APRENDIZADO POR TRANSFERÊNCIA

Dissertação apresentada ao Programa de Pós-graduação em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientador(es): Julio Cesar Duarte, D.Sc.

Rio de Janeiro

2022

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro – RJ CEP: 22290-270

Malvacio, Eduardo Martín.
Identificação de malware utilizando Aprendizado por Transferência / Eduardo Martín Malvacio. – Rio de Janeiro: Instituto Militar de Engenharia, 2022.
105 f.

Orientador(es): Julio Cesar Duarte.

Dissertação (mestrado) – Instituto Militar de Engenharia, Sistemas e Computação, 2022.

1. Malware. 2. Detecção de Malware. 3. Redes Neurais Convolucionais. 4. Visão computacional. 5. Aprendizado por Transferência. i. Duarte, Julio Cesar (orient.) ii. Título

EDUARDO MARTÍN MALVACIO

Identificação de malware utilizando Aprendizado por Transferência

Dissertação apresentada ao Programa de Pós-graduação em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientador(es): Julio Cesar Duarte.

Aprovada em 14 de setembro de 2022, pela seguinte banca examinadora:

Prof. **Julio Cesar Duarte** - D.Sc. do IME - Presidente

Prof. **Ronaldo Ribeiro Goldschmidt** - D.Sc. do IME

Prof. **Eugênio da Silva** - D.Sc. da UERJ-ZO

Rio de Janeiro
2022

DEDICATÓRIA

Este trabalho é dedicado à minha esposa Florencia, à minha filha Helena, pelo seu amor e por serem motivo de constante inspiração e ao meu pai Emilio (In Memoriam).

AGRADECIMENTOS

Os agradecimentos principais são direcionados a Deus por iluminar minha trajetória, me dando a força espiritual e determinação para superar as adversidades, a minha família pelo apoio durante esta difícil jornada, sempre incentivando-me e mantendo-me focado a cumprir com minha missão e realizar minha vocação.

Ao Exército Argentino por confiar em mim, atribuindo-me a grande responsabilidade de representá-lo em uma nação amiga, incentivando a troca de conhecimentos e experiências para cooperação mútua.

Ao Instituto Militar de Engenharia do Exército Brasileiro por me oferecer a oportunidade de cursar o mestrado, aumentando o conhecimento profissional e permitindo conhecer seu excelente grupo humano gerando novas amizades. À Seção de Engenharia de Computação, pelo acolhimento.

A todos os professores do programa de Mestrado em Engenharia em Sistemas e Computação do IME, em especial ao Cel Anderson F. Pereira dos Santos, Prof. Marcos Veloso Peixoto, Prof. Alex de Vasconcellos Garcia, Prof. Ronaldo Ribeiro Goldschmidt e à Profa. Claudia Marcela Justel.

Ao meu orientador e professor Cel Julio Cesar Duarte que desde o início do curso tem me apoiado, sempre com extrema boa vontade, paciência e atenção, e pela valiosa orientação e sugestões dadas durante a realização deste trabalho.

Aos meus pais por seu esforço e carinho em me educar e me formar em valores éticos e morais com seu exemplo diário de vida. Aos meus irmãos por me acompanharem dos meus primeiros passos incondicionalmente.

Agradecimentos especiais para todos aqueles que de alguma forma contribuíram e me ajudaram para que eu conseguisse superar esta etapa, minha eterna gratidão.

"O soldado acredita na pátria, na liberdade,
na justiça; não na tirania, no despotismo
ou no fanatismo..."
Nicolas Levalle (1840-1902)

RESUMO

Com o avanço da tecnologia, criminosos cibernéticos estão desenvolvendo novas formas de *malware* que são mais sofisticadas e mais difíceis de se detectar. Pesquisadores de segurança devem estar constantemente alertas para estas novas ameaças e trabalhar incansavelmente para analisar e compreender cada nova amostra de *malware*. Este processo é extremamente complicado, pois cada amostra é única e pode exigir diferentes abordagens de análise. Além disso, os cibercriminosos estão continuamente modificando seu *malware* para impedir que os pesquisadores o detectem e o analisem. Isto significa que o processo de descoberta de novas amostras é uma corrida de longa distância contra um inimigo que está sempre um passo à frente. Ainda assim, os pesquisadores de segurança são cruciais para a luta contra o *malware*. À medida que novas amostras são descobertas, melhores ferramentas e técnicas de detecção e análise em que podem ser desenvolvidas, ajudando a proteger os usuários da Internet contra essas ameaças. A análise de *malware* é um processo complexo que requer o uso de muitas técnicas diferentes. Um dos métodos mais importantes e eficazes para analisar o *malware* é o aprendizado por transferência, que é um método de aprendizado de máquina que se baseia no uso de um modelo pré-treinado para analisar novas amostras de *malware*. Este método é especialmente útil para a análise de *malware* porque este tende a ser muito semelhante entre famílias diferentes. Por exemplo, um trojan bancário pode ser muito semelhante a outro trojan bancário, mas muito diferente de um vírus de computador. O aprendizado por transferência permite que um modelo pré-treinado para detectar trojans bancários, por exemplo, seja reutilizado para detectar novas amostras desse tipo de *malware*, sem a necessidade de retreinar o modelo a partir do zero. Este documento apresenta e testa uma abordagem do problema de identificação de *malware* usando o aprendizado de transferência, usando amostras de arquivos benignos que têm alguma função ou comportamento similar ao *malware*, ao contrário de trabalhos relacionados que usam amostras de arquivos benignos muito diferentes das amostras de malware, implementando redes neurais pré-treinadas para problemas gerais de classificação de imagens, permitindo criar um método de detecção eficiente e aplicável a um grande conjunto de amostras. Os resultados analisados confirmam que esta abordagem pode ser utilizada para esta tarefa.

Palavras-chave: Malware. Detecção de Malware. Redes Neurais Convolucionais. Visão computacional. Aprendizado por Transferência.

ABSTRACT

Malware, or malicious software, is an increasingly common threat on the Internet. As technology advances, cybercriminals are developing new forms of malware that are more sophisticated and harder to detect. Security researchers must constantly be alert to these new threats and work tirelessly to analyze and understand each new malware sample. This process is extremely complicated, as each malware sample is unique and may require different analysis approaches. In addition, cybercriminals are continually modifying their malware to prevent researchers from being able to detect and analyze it. This means that the process of discovering new malware samples is a long-distance race against an enemy that is always one step ahead. Still, security researchers are crucial to the fight against malware. As new samples are discovered, better detection and analysis tools and techniques can be developed, helping to protect Internet users from these threats. Malware analysis is a complex process that requires the use of many different techniques. One of the most important and effective methods for analyzing malware is transfer learning which is a machine learning method based on the use of a pre-trained model to analyze new malware samples. This method is especially useful for malware analysis because malware tends to be very similar between different families. For example, a banking Trojan may be very similar to another banking Trojan, but very different from a computer virus. Transfer learning allows a pre-trained model for detecting banking Trojans, for example, to be reused to detect new malware samples, without having to retrain the model from scratch. The present work exposes and experiments an approach to the problem of malware identification using transfer learning, using benign file samples that have some function or behavior similar to that of malware, implementing pre-trained neural networks for image classification problems in general, allowing to create an efficient detection method applicable to a large set of samples. The analyzed results confirm that this approach can be used for this task.

Keywords: Malware. Malware Detection. Convolutional Neural Networks. Computer Vision. Transfer Learning.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DDoS	Negação de Serviço Distribuída
IP	Protocolo de Internet
VM	Máquina Virtual
ASCII	Código Padrão Americano para Intercâmbio de Informações
RNA	Rede Neural Artificial
CNN	Rede Neural Convolucional
PE	Elemento Processador
NLP	Processamento de Linguagem Natural
SFT	Ajuste Fino Superficial
CV	Visão computacional
API	Interface de Programação de Aplicativos
Win PE	Formato Executável Portátil do Windows
GPU	Unidade de Processamento Gráfico
TPU	Unidade de Processamento de Tensores
AM	Aprendizado de Máquina
DCNN	Deep Convolutional Neural Network
ANN	Artificial Neural Network
SFTA	Scale Feature Texture Analyzer
KNN	K Nearest Neighbor
LGMP	Local-Global Malicious Pattern

SUMÁRIO

1 INTRODUÇÃO

Malware é um tipo de software projetado para danificar ou desativar computadores e sistemas computacionais (1). Esse software geralmente é instalado sem o conhecimento ou consentimento do usuário e pode ser usado para roubar informações pessoais, desabilitar arquivos críticos do sistema ou até mesmo destruir o sistema completamente. *Malware* é uma séria ameaça à segurança e pode ser muito difícil de remover, uma vez instalado (2).

As motivações dos cibercriminosos para desenvolver *malware* podem ser muito variadas. Às vezes, eles fazem isso para ganhar dinheiro diretamente, roubando informações pessoais ou bancárias das vítimas. Outras vezes, eles fazem isso para obter acesso não autorizado a um sistema ou para sabotar uma empresa ou organização. Eles também podem fazer isso apenas por diversão ou para mostrar suas habilidades (1). Não importa a motivação, o resultado é o mesmo: o *malware* pode causar sérios danos às vítimas.

A quantidade de *software* maliciosos aumentou significativamente nos últimos anos, chegando a mais de **1.363 bilhões de amostras** em 2022, conforme mostra a Figura 1, com dados atualizados até o primeiro semestre de 2022.

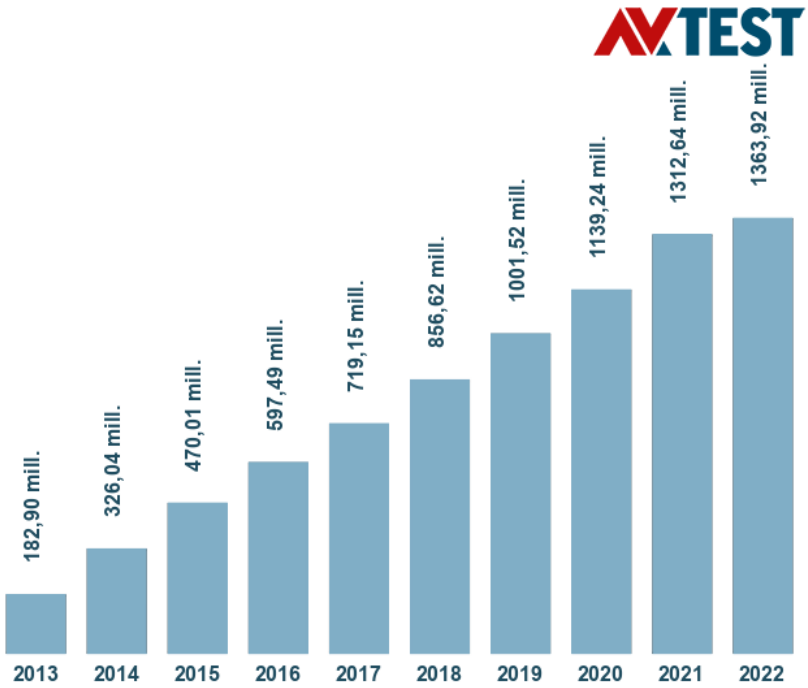

Figura 1 – Quantitativo acumulado de *malware* na última década, de acordo com o Instituto AV-Test. Fonte: AV-Test(3)

Embora o número de novos *malware* descobertos a cada ano tenha se mostrado variável, é certo que também apresentará uma tendência de aumento ate 2021, dados os

avanços tecnológicos e o número crescente de dispositivos.

Consequentemente, a detecção e classificação de *malware* tornou-se um dos campos de pesquisa mais importantes. A detecção de *malware* é feita principalmente pela aplicação de análise de superficial, análise estática ou análise dinâmica (4). Na análise estática e na análise de superficial, o código-fonte original do *malware* é verificado sem executá-lo e, embora essa abordagem seja barata, é insuficiente se o *malware* for criptografado ou ofuscado. Por outro lado, uma abordagem baseada em análise dinâmica analisa as características comportamentais do *malware* executando o código em ambientes isolados ou virtuais (5). Portanto, essa abordagem consome muito tempo e recursos para o analista.

Outra alternativa para identificar *malware* é a abordagem de visualização. Em muitos trabalhos recentes, a abordagem de análise visual de *malware* (6, 7) tem sido usada como uma solução eficaz para identificar *malware*, pois analisa o seu arquivo executável. A visualização de *malware* é, portanto, um método de usar *software malicioso* convertendo-o em uma imagem e extraindo seus binários (8).

1.1 Motivação

Usar uma estratégia de aprendizado automática em *malware* não é fácil e já é feito há muitos anos e, devido à complexidade do *malware*, há uma corrida constante entre pesquisadores de segurança e criadores de *malware*. Assim, a pesquisa neste campo será sempre relevante. Mesmo que os pesquisadores encontrem um novo método que detecte todos os *malware*, os criadores de *malware* acabarão encontrando uma maneira de contornar o novo método. A IBM Security publicou um relatório (9) afirmando que o custo médio de uma violação de segurança é de US$ 3,86 milhões e que leva em média 280 dias para identificar e conter a violação. Não só as grandes empresas podem arcar com esse custo, mas o setor mais afetado é o da saúde.

Quando uma nova variante de *malware* é adquirida, os especialistas geralmente analisam a amostra manualmente, pois sua remoção requer conhecimento das funções do *malware* (10). Essa análise manual requer várias horas a várias semanas, dependendo da complexidade do malware (11). Uma razão pela qual a análise de malware demora tanto é porque a região de dados binários que caracteriza a funcionalidade do *malware* não é facilmente identificável.

Desde que o estudo de vírus de computador começou no final da década de 1980, a análise de *malware* tornou-se uma tarefa vital para a segurança da informação. À medida que os vírus e outros *malwares* se tornaram mais sofisticados, a análise de *malware* teve que se adaptar para acompanhar.

O trabalho proposto nesta dissertação é necessário devido à grande quantidade

de informações sobre classificação de *malware*. Muitos métodos são propostos e testados com seu próprio conjunto de dados privado, dificultando a validação de seus resultados por outros pesquisadores. Além disso, esses métodos são altamente otimizados para cada conjunto de dados específico, a fim de obter o maior desempenho possível. Isso significa que há uma chance de que um método com um bom desempenho não seja tão eficaz no mundo real.

1.2 Caracterização do Problema

As ameaças de *malware* estão evoluindo vertiginosamente em todo o mundo em várias frentes, como: usuários, dispositivos, aplicativos e principalmente, considera-se que, ao longo dos anos, as redes sociais, a caixa de correio, as redes eletrônicas, empresariais e financeiras serão mais vulneráveis (12).

No mundo da cibersegurança, há uma batalha constante entre quem cria *malware* e quem o analisa. Ambas as partes estão constantemente tentando ser mais inteligentes na tentativa de infectar mais computadores ou proteger melhor os usuários contra ataques. Esta competição está se tornando cada vez mais intensa, pois as apostas estão se tornando cada vez mais importantes. À medida em que mais e mais empresas e indivíduos dependem de computadores para armazenar informações confidenciais, o dano potencial que um ataque de *malware* bem-sucedido pode causar está sempre aumentando. Ao mesmo tempo, as recompensas pela criação bem-sucedida de *malware* também aumentam, pois os hackers podem exigir grandes somas de dinheiro de suas vítimas. Consequentemente, ambos lados investem continuamente em mais recursos para obter vantagem sobre seus oponentes. Os analistas de *malware* estão constantemente fazendo engenharia reversa de novas amostras para entender melhor como elas funcionam e como se proteger contra elas. De sua parte, os criadores de *malware* estão constantemente criando novas maneiras de evitar a detecção e tornar suas criações mais eficazes.

Um papel importante e em rápido crescimento na segurança cibernética é o de analista de *malware*. Parte engenheiro de segurança, parte especialista em forense digital e parte programador, essa função crucial fornece inteligência detalhada após um evento de segurança cibernética. Uma vez que o ataque cibernético inicial tenha sido identificado e contido, é fundamental que ocorra análise e revisão completas do incidente. Isso necessariamente incluirá um olhar atento às ferramentas e métodos usados pelo adversário. Ao analisar o *malware* usado em um ataque, novas defesas podem ser implementadas ou refinadas conforme necessário. A capacidade de fazer engenharia reversa de código malicioso é primordial em uma estratégia defensiva e é aí que o analista de malware agrega valor à equipe de segurança cibernética. A principal função de um *analista de malware* é identificar, examinar e entender várias formas de *malware* e seus métodos de entrega. Este

software malicioso inclui todas as várias formas de *adware*, *bots*, *bugs*, *rootkits*, *spyware*, *ransomware*, *trojans*, vírus e *worms*.

A classificação de *malware* é um problema de alta prioridade que pode ser resolvido por uma abordagem de aprendizado automático. Ao mesmo tempo, a presença de uma grande quantidade de dados é vital para alcançar alta precisão. No entanto, uma enorme quantidade de dados aumenta os custos de pré-processamento e treinamento. A aplicação de modelos pré-treinados de outra área, como *visão computacional*[1] (CV) e *processamento de linguagem natural*[2] (PNL), para a classificação de malware, ou seja, um método baseado em aprendizado por transferência[3], poderia reduzir esses custos e acelerar a introdução do modelo num sistema de segurança.

O presente trabalho visa abordar, usando redes neurais pré-treinadas e aprendizado de transferência, o problema de identificação de *malware* quando amostras de *malware* e amostras de arquivos benignos compartilham funções semelhantes. Olhando de outra forma, pode-se dizer que as imagens das amostras de *malware* são muito semelhantes às imagens dos arquivos benignos, o que aumenta a dificuldade de classificação.

1.3 Objetivos

O objetivo deste trabalho é investigar e propor o uso de redes neurais pré-treinadas na identificação de arquivos maliciosos (*malware*) através de um estudo comparativo entre várias redes pré-treinadas, utilizando um conjunto de dados de arquivos benignos com algum comportamento ou função semelhante ao de *malware*, verificando se o desempenho desses classificadores é comparável ao dos classificadores criados especificamente para o problema.

Como objetivos específicos pode-se destacar:

- Criar um *benchmark* usando um banco de dados com arquivos maliciosos e benignos representativos da tarefa que está sendo modelada; e
- Realizar uma análise de sensibilidade das redes neurais testadas, verificando seu desempenho e sua reação a mudanças em seus hiper-parâmetros.

Quando fala-se sobre a tarefa que está sendo modelada, estamos nos referindo ao fato de que as amostras benignas deste trabalho possuem algum comportamento ou função semelhante ao do *malware*, é neste ponto que reside a importância do *especialista em malware*. A tarefa desse especialista é estudar a funcionalidade desses programas

[1] Do inglês, *Computer Vision*
[2] Do inglês, *Natural Language Processing*
[3] Do inglês, *Transfer Learning*

maliciosos para tentar entendê-los e classificá-los. No entanto, um analista não pode analisar todas as novas amostras de *malware*, mas é essencial que analise as amostras mais representativas da evolução dos ataques. Portanto, selecionar as amostras de *malware* mais representativas para análise é uma tarefa crucial para o analista de malware. Uma maneira de selecionar essas amostras de *malware* é comparar os códigos-fonte de todas as amostras para encontrar aqueles com o código-fonte mais representativo. No entanto, este processo é muito demorado e requer grandes quantidades de recursos. Portanto, há uma alta demanda por métodos para selecionar automaticamente as amostras de *malware* mais representativas para análise. Isso é o que é conhecido como amostragem de *malware* (13).

1.4 Justificativa

O *malware* é uma ameaça crescente para a economia global. De acordo com um relatório da McAfee (14), o custo global dos danos causados por *malware* atingiu US$ 600 bilhões em 2018, ou 0,8% do PIB global.

O *malware* não é apenas uma ameaça à economia, é também uma ameaça à segurança nacional. Em 2016, o *malware* ***Stuxnet*** atacou equipamentos de controle de centrífugas nucleares no Irã, mostrando que o *malware* pode ter um impacto real na capacidade de um país se defender (15).

A República Argentina e a União de Nações Sul-Americanas (UNASUL) não ficaram alheias a essa nova dinâmica de problemas. O Conselho de Defesa da UNASUL considera este problema em seus Planos de Ação desde 2012. Por sua vez, foram incorporadas atividades específicas sobre políticas, mecanismos e capacidades regionais para lidar com ameaças cibernéticas no campo da defesa. Por exemplo, no Plano de Ação de 2015, o tema foi incorporado como Grupo de Trabalho do Plano de Ação Extra, cuja finalidade era dar continuidade ao Grupo de Trabalho de Defesa Cibernética e coordenar com o COSIPLAN[4] a realização de um seminário. Ainda em 13 de setembro de 2013, os ministros da Defesa da Argentina e do Brasil expressaram, no âmbito de uma reunião bilateral, a importância de trabalhar juntos sobre o assunto e incluíram a defesa cibernética na Declaração de Buenos Aires, além de acordarem a cooperação cibernética em defesa e a criação de um subgrupo de trabalho bilateral específico (16).

Um software malicioso (ou *malware*) pode ser reconhecido como uma das ameaças mais perigosas aos sistemas de computador. O *malware* há muito deixou de ser mero entretenimento para hackers e agora se tornou uma arma reconhecida de serviços de inteligência patrocinados pelo Estado e gangues de criminosos cibernéticos. De acordo com estatísticas de empresas de pesquisa, o número de amostras únicas de *malware* está aumentando a cada ano e chegará a 1,323 milhões até 2022 (3). O crescimento de ataques de

[4] Conselho de Infraestrutura e Planejamento Sul-Americano da UNASUL

ransomware[5] contra os setores médico, de energia e financeiro forçou os principais estados a considerar os pagamentos de *ransomware* como financiamento do terrorismo devido a ataques a infraestrutura crítica. Além disso, os ataques cibernéticos de grupos de hackers suspeitos de serem patrocinados pelos governos de vários países do *G20* tornaram-se um fator contribuinte para a tensão geopolítica (17). Portanto, o desafio de detectar e prevenir programas maliciosos é uma das prioridades para o desenvolvimento sustentável do mundo.

A abordagem tradicional de detecção de *malware* inclui análise de assinatura e comportamento que compara as especificações de uma amostra desconhecida com modelos existentes baseados em regras criadas por humanos. Essas regras são criadas a partir de atividades suspeitas de *malware* e exigem um grande número de amostras. O processo de geração de modelos pode ser automatizado usando métodos empíricos, mas esta abordagem tem suas limitações (18).

Assim, a gravidade do problema e a rapidez com que as ameaças mudam exigem uma solução para melhorar rapidamente os algoritmos de detecção de *malware*. Descobertas recentes em Inteligência Artificial levaram à aplicação do aprendizado de máquina em vários campos econômicos, como medicina, varejo, transporte e muitos outros. As empresas aplicam técnicas de inteligência artificial para aumentar o desempenho de operações de rotina que exigem muito tempo de análise humana. Uma abordagem semelhante encontrou aplicação em várias direções de segurança cibernética, incluindo detecção e prevenção de intrusão, reconhecimento de *spam*, detecção de fraude e detecção de *malware* (19), para a qual o aprendizado de máquina é uma solução adequada para detectar e prevenir ataques cibernéticos devido à natureza do aprendizado de máquina para lidar com a rápida variabilidade.

1.5 Metodologia

Define-se o trabalho realizado como um estudo experimental, onde é levantada a possibilidade de utilização de Redes Neurais Convolucionais e Aprendizado por Transferência para identificação de *malware*.

Para obter o conhecimento necessário e validar o problema selecionado, foi realizado um estudo bibliográfico aprofundado focado nos principais tópicos de pesquisa: aprendizado de máquina, mais especificamente aprendizado por transferência e análise de *malware*. A pesquisa é realizada experimentalmente, buscando assim mensurar e validar a abordagem proposta em relação ao problema a ser resolvido.

Em relação aos conjuntos de dados utilizados, deve-se notar que não foram encontrados conjuntos confiáveis de amostras de arquivos benignos, disponíveis na Internet, e

[5] Ransomware é um software de extorsão que pode bloquear um computador e depois exigir um resgate para desbloqueá-lo.

esse problema não existe para amostras de *malware*, já que milhares de amostras podem ser encontradas.

A divisão do problema geral em fases ou estágios foi essencial para poder enfrentar problemas menores e mais complexos com maior eficiência. A separação das fases do problema geral possibilitou analisar cada um dos subproblemas de forma independente, podendo assim identificar e resolver mais facilmente os erros em cada um deles.

Quanto aos experimentos, eles foram divididos em seis fases: coleta da amostra, pré-processamento da amostra, geração da imagem, geração do modelo, treinamento do modelo e classificação da amostra.

Como parte do trabalho, o artigo "An Assessment of the Effectiveness of Pretrained Neural Networks for Malware Detection" foi enviado para a revista *IEEE Latin America Transactions*, onde após algumas correções foi aprovado e publicado.

1.6 Organização da Dissertação

Esta dissertação está organizada da seguinte forma: O Capítulo 2 - Análise de malware - apresenta o conceito de *malware*, apresentando suas características, técnicas de evasão e conceitos relacionados à análise de *malware*. O aprendizado de máquina, enfatizando aspectos relacionados a redes neurais convencionais, aprendizado por transferência e suas arquiteturas mais conhecidas são abordados no Capítulo 3 - Aprendizado de Máquina. O Capítulo 4 - Trabalhos relacionados - descreve os trabalhos científicos que têm alguma relação com este estudo. O Capítulo 5 - Aplicando Aprendizado por Transferência na Análise de Malware - descreve a abordagem criada durante este estudo. O Capítulo 6 - Análise dos Resultados - especifica a configuração dos experimentos executados e os resultados de suas execuções são discutidos. Por fim, o Capítulo 7 - Conclusão - apresenta a conclusão deste trabalho, as considerações finais e sugestões para trabalhos futuros.

2 ANÁLISE DE MALWARE

A análise de *malware* é definida como um processo complexo, pelo qual o investigador procura entender quais ações um código malicioso realiza e qual o seu objetivo. Também pode ser vista como uma ferramenta que fornece a um investigador a informação necessária para responder a uma intrusão na rede (10). Tem o fundamento necessário para determinar o que aconteceu, ou garantir que um computador e seus arquivos tenham sido infectados.

Por outro lado, os desenvolvedores de *malware* podem usar várias técnicas para esconder seu código e evitar que sejam detectados. Alguns métodos comuns de ocultação de *malware* incluem empacotamento, ofuscação e cifragem. Também podem usar técnicas de evasão, como o uso de exploração de vulnerabilidades conhecidas para pular as proteções de segurança, e técnicas de anti-análise para dificultar que os investigadores de segurança examinem seu código. Isto pode incluir o uso de instruções de códigos confusas ou a constante mudança de código para evitar que os padrões sejam detectados. Além disso, podem incorporar análises para detectar se o código está sendo analisado e tomar medidas para evitar que a análise continue.

Nas seções a seguir, define-se o que é *malware*, destacando algumas de suas características, potenciais de dano e métodos de evasão comumente empregados, além de abordar as técnicas para sua análise.

2.1 Terminologia e Conceitos Básicos

Shyamasundar, Shah e Kumar(20) afirmam que o ponto de partida dos vírus e do *malware* foi trabalho de John von Neumann em seus estudos sobre um autômato capaz de se reproduzir. Em 1951, von Neumann já havia proposto métodos para demostrar como criar tal autômato. Mas foi a partir de 1984 que Cohen(21) e Adleman(22) estabeleceram as bases do que hoje se conhecem como vírus informático. Desde então, tem existido várias variações destes programas, algumas bastante perigosas.

Malware é uma contração da expressão inglesa "**mal**icious soft**ware**", que significa literalmente "software malicioso". Trata-se de um termo geral que se utiliza para denominar todo tipo de software malicioso, seja um vírus, um *worm*, um *trojan* ou qualquer outro tipo de programa malicioso (23).

Existem muitas definições de *malware*, software malicioso, código malicioso e conteúdo malicioso. A seguir são indicadas duas definições similares de código malicioso e software malicioso que são viáveis para este trabalho (24).

1. Código malicioso se define como:

   > *É um código de programação capaz de causar danos à disponibilidade, à integridade do código ou dos dados, ou à confidencialidade de um sistema informático; engloba os trojans, vírus, worms, entre outros* (25).

2. Software Malicioso (*malware*) é definido como:

   > *O malware é um conjunto de instruções que se executam no seu computador e fazem que seu sistema faça algo que um atacante quer que faça* (24).

No mundo digital, o desenvolvimento e a distribuição de programas maliciosos são de interesse para indivíduos e organizações com intenções antiéticas ou ilegais. Alguns exemplos do comportamento do *malware* são:

- Eliminação de arquivos cruciais em um computador para torná-lo inutilizável sem um processo de recuperação;
- Registar todas as teclas para ver o que os usuários estão digitando;
- Roubar informação ou arquivos pessoais ou sensíveis de um computador; e
- Usar os recursos de um computador para o propósito do *malware*, por exemplo, enviar e-mails, enviar e-mails de spam, realizar ataques DDoS[1] para outro sistema ou forçar chaves de criptografia.

Existem diferentes famílias de *malware* (10, 26, 27), cada uma possuindo características diferenciadas, como exemplo, podem-se citar algumas:

- A mais comum é o ***vírus***, que é um programa capaz de se replicar e disseminar pelo computador afetando os arquivos e pastas, mas também pode afetar outros programas e o sistema operacional.
- Os ***worms*** são *malware*s que se espalham pela internet e se reproduzem, geralmente utilizando a função de e-mail para se enviar a múltiplos destinatários.
- Os ***trojans*** são programas que parecem inofensivos e são considerados como garantidos pelo usuário, mas na realidade são uma porta de entrada para que um criminoso cibernético tenha acesso remoto ao computador.
- Os ***rootkits*** são uma classe de programas responsáveis por ocultar a existência de outros programas maliciosos no computador, aumentando a dificuldade da sua detecção e eliminação.

[1] *Negação de Serviço Distribuída* ou Ataque de Negação de Serviço Distribuído

- Os ***ransomware*** são programas capazes de encriptar arquivos para torná-los inacessíveis, e depois exigem um resgate na forma de dinheiro dos usuários para liberar os arquivos.
- Os ***spyware*** são programas que coletam informações pessoais e as enviam sem permissão ao autor do programa.
- Os ***adware*** são programas que são baixados com outros programas e que mostram anuncio sem consentimento do usuário.

2.2 Técnicas de Análise

A análise de *malware* se entende como o processo de investigação do comportamento de uma amostra específica de *malware*. Este processo consiste em três etapas diferentes com objetivos, abordagens e métodos diferentes. É importante levar em conta a complexidade e o potencial de informação estimado e o consumo de tempo para cada uma das três etapas: *análise superficial*, *análise dinâmica* e *análise estática* (Figura 2). Tradicionalmente, um estudo ou análise de *malware* seguirá este procedimento: análise superficial(28), análise dinâmica(29) e análise estática(30). No entanto, o processo da análise (Figura 3) deve refletir o propósito e o objetivo da análise e justificar a extensão e se uma, duas ou as três etapas são necessárias. A seguir se descrevem as três partes diferentes.

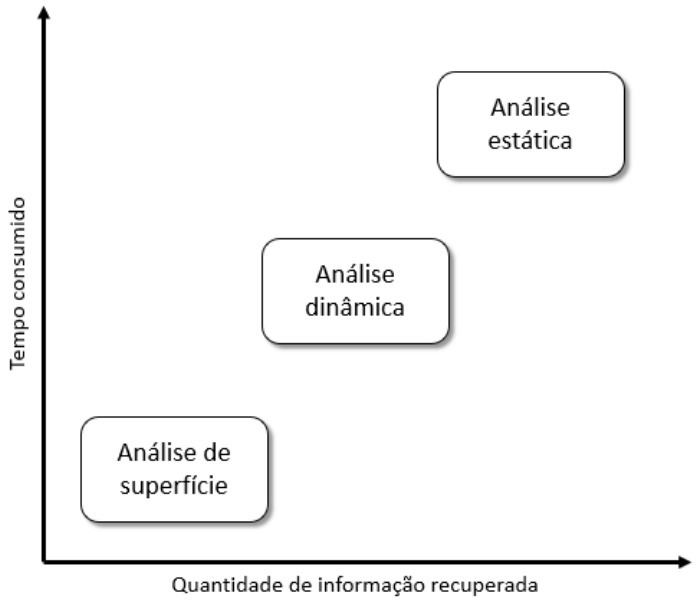

Figura 2 – Consumo de tempo e estimativa do ganho de informação na análise de *malware*. Adaptação de Vinod et al.(28)

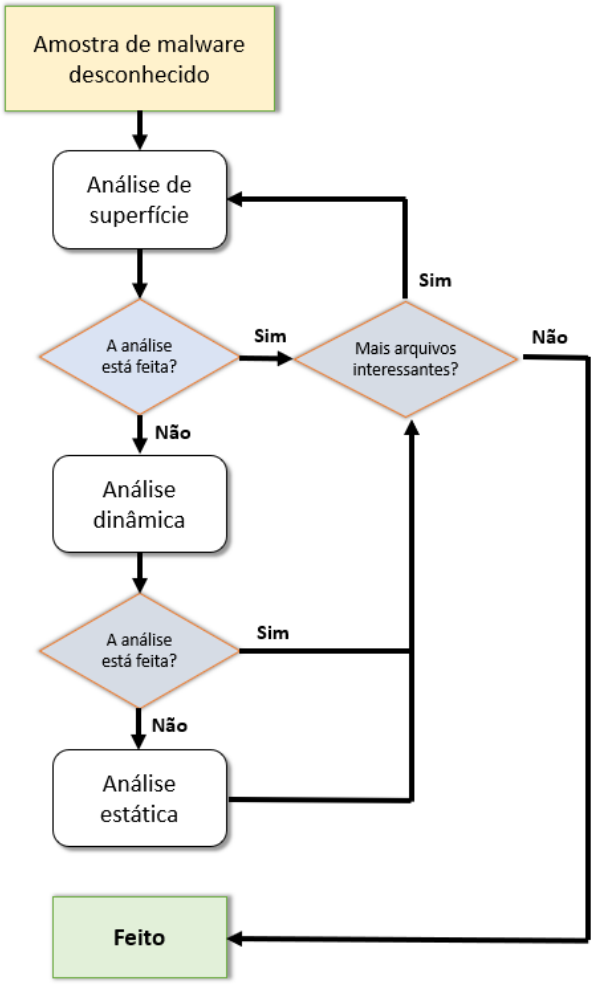

Figura 3 – Fluxograma de análise de *malware*s mostrando o processo de análise de amostras. O analista deve julgar quais arquivos da amostra são interessantes e valiosos para analisar, e até que ponto a análise deve ser realizada. Adaptação de Higuera et al.(31)

2.2.1 Análise superficial

A análise superficial é geralmente a primeira etapa de um processo de análise de *malware* e é quase sempre realizada. Isto também é verdade para os programas antivírus que verificam apenas uma amostra em busca de uma assinatura. Uma análise superficial consiste em abrir a amostra e procurar rapidamente por informações no arquivo de amostra sem executá-lo.

A análise superficial também inclui a execução de programas de varredura na amostra para detectar se eles podem fornecer alguma informação. Os programas antivírus e escâneres on-line podem reconhecer uma assinatura e serem capazes de revelar alguma informação sobre a amostra. Esta fase da análise é considerada simplesmente uma verificação da amostra de *malware*, que normalmente provê uma ideia geral do tipo de *malware* (28).

2.2.2 Análise dinâmica

A análise dinâmica consiste principalmente em executar a amostra-alvo e coletar os resultados de diagnóstico e comportamento com base nos registros e ferramentas de monitoramento. Trata-se de um processo muito mais complexo e que requer mais tempo do que a análise superficial. A razão é que a amostra é executada e frequentemente pode mostrar um comportamento complexo (32). As principais áreas que são monitoradas durante a execução inicial da amostra são:

- **Memória**: Principalmente processos executados, com *threads*. Mostra quais processos têm sido criados e com quais comandos;
- **Disco**: Acesso e alterações de arquivos e do registro. Mostra as leituras, escrituras, criações e eliminações de arquivos e do registro; e
- **Rede**: Todo o trafego da rede.

Ao monitorar estas áreas e procurar comportamentos anormais ou suspeitos, pode se revelar uma imagem completa da funcionalidade da amostra. O monitoramento da memória e do disco fornecerá informação sobre o que está acontecendo no computador local, enquanto o monitoramento da rede indicará o contato da amostra com outros computadores através de redes locais ou externas.

Além disso, podem-se monitorar os possíveis arquivos que estão escrevendo e voltá-los para estudar seu conteúdo. Este procedimento dará informação sobre como se instala a amostra no computador, como tenta se esconder ou se tornar persistente na inicialização. Ao analisar os pacotes de rede enviados, é possível descobrir os endereços de IP[2], o conteúdo enviado pela rede, os protocolos utilizados, etc.

Uma opção é executar a amostra dentro de um depurador. Dependendo do depurador, podem-se ver a amostra, ou parte dela, em seu código de montagem. Aqui podem-se seguir cada instrução conforme se executa e continuar descobrindo a funcionalidade da amostra, além de pausar o objetivo em qualquer momento, dando o tempo necessário para a investigação do comportamento. A análise do código real será melhor realizada na análise estática. No entanto, pode-se fazer uso de um depurador para monitorar mais de perto a amostra em funcionamento.

Para poder executar a amostra várias vezes e realizar experimentos adequados, precisa-se dos mesmos parâmetros do ambiente para cada execução. Também precisam-se ser capazes de realizar várias experiências consecutivas sem gastar tempo demais na inicialização. Pode-se conseguir isto executando a amostra em um ambiente controlado

[2] Internet Protocol, do Inglês, ou Protocolo de Internet

utilizando a virtualização. A virtualização neste contexto significa uma máquina virtual (VM) na qual a amostra é executada.

Definitivamente, a análise dinâmica é mais abrangente do que a análise superficial. Durante a análise dinâmica, muitos dados e informação podem ser descobertos, proporcionando uma compreensão quase completa da amostra. Normalmente, isto será suficiente para mapear a maior parte do comportamento da amostra, como ela entra em contato com o seu C&C (Comando e Controle), a instalação no sistema, etc. A informação que não é encontrada durante as análises superficial e dinâmica, pode ser encontrada utilizando a análise estática. Alguns autores (10, 29) dividem a análise dinâmica em: Análise Básica e Avançada.

2.2.3 Análise Dinâmica Básica

A análise dinâmica básica envolve executar o *malware* e observar seu comportamento no sistema para remover a infecção, produzir assinaturas eficazes ou ambos. Entretanto, antes de poder executar o *malware* de forma segura, é necessário configurar um ambiente de testes que não represente um risco para o sistema ou para a rede. Assim como a análise estática, este tipo de análise pode ser usado pela maioria das pessoas mesmo sem fortes habilidades de programação, mas não será eficaz com todos os *malware*.

2.2.4 Análise Dinâmica Avançada

A análise dinâmica avançada consiste em utilizar um depurador para examinar o estado interno de um arquivo executável em execução. Este tipo de análise fornece outra forma de obter informações detalhadas de um arquivo executável, e é mais eficaz em *malware* sofisticados.

O depurador é uma peça de software ou hardware usado para testar ou examinar a execução de outro programa, fornecendo uma visão dinâmica de um programa em execução. Por exemplo, podem exibir os valores de endereços de memória à medida que mudam durante a execução de um programa (10).

2.2.5 Análise estática

A análise estática consiste em examinar o código da máquina da amostra binária para descobrir a funcionalidade e as técnicas utilizadas pela amostra. A amostra não é executada, mas sim examina o código de máquina do arquivo binário. Outra maneira de diferenciar a análise dinâmica da estática é que a análise dinâmica é uma análise de comportamento, enquanto que a análise estática é uma análise do código. Portanto, mesmo que a amostra seja executada em um depurador para examinar o código, ela continua sendo uma análise estática. A maior desvantagem da análise estática é que ela pode ser

muito demorada e complexa de realizar, às vezes pode até ser quase impossível obter as informações ou respostas necessárias. A grande vantagem é que a análise estática é segura, ou seja, a amostra nunca é executada realmente de forma descontrolada e, portanto, não pode criar nenhum dano ou fazer algo não previsto pelo analista. A análise pode fornecer resposta para cada pergunta sobre a amostra, já que pode examinar cada instrução do arquivo binário (10, 33).

A análise estática é frequentemente referida como engenharia reversa. A engenharia reversa é o oposto da engenharia, ou seja, desmontar um produto em vez de criá-lo. Neste contexto, o termo engenharia reversa é usado porque um programa compilado, um arquivo binário, é estudado para revelar o programa original.

A análise superficial pode ser percebida como uma forma de análise estática, já que a análise é de fato estática porque a amostra nunca é executada. No entanto, é importante distinguir a análise superficial da análise estática por causa de seus diferentes objetivos: a análise superficial supõe que deve oferecer apenas uma breve visão da amostra em um período de tempo muito curto, enquanto que a análise estática, por outro lado, pode fornecer informações detalhadas da amostra, mas o tempo necessário para a análise pode ser de várias horas, ou mesmo dias.

Na análise estática, o código de máquina do binário é examinado. Para isso existem várias ferramentas, sendo a mais básica o **editor hexadecimal**. Este editor lê a amostra binária como valores hexadecimais e os apresenta, muitas vezes com apresentação ASCII[3] ao mesmo tempo. Isto pode ser valioso para a visualização de cadeias ASCII, mas além disso é muito básico e difícil de usar na análise de um binário grande.

Outro tipo de ferramenta são os depuradores. A desvantagem de alguns depuradores é que eles exigem que a amostra seja executada e durante a execução é possível controlar cada instrução enquanto ela é enviada à CPU. Todo depurador tem que interpretar o código da máquina, mas os depuradores têm até certo ponto a capacidade de mostrar o código da máquina desmontada. A linguagem de montagem pode ser vista como a tradução legível por humanos do código da máquina. O desmontador interpretará cada valor hexadecimal no binário e o traduzira em instruções e dados da CPU.

É importante lembrar que muitas amostras de *malware* utilizam algum tipo de técnica para ofuscar o arquivo binário tornando-o quase ilegível, pelo que a tradução da linguagem do montador não funcionará, ou seja, haverá uma tradução ao código montador, mas o código não será legível ou poderá estar incorreto.

A última categoria de ferramentas na análise estática são os descompiladores. Como o nome sugere, estas ferramentas tentam descompilar um executável binário em código de nível superior, como, por exemplo, uma linguagem de programação como o C. Devido à

[3] Do inglês, American Standard Code for Information Interchange

complexidade dos compiladores atuais, isto é muitas vezes inviável. Muitos compiladores realizam otimização no que diz respeito tanto ao tempo de execução quanto ao tamanho do executável. Embora o executável compilado seja executado conforme projetado em código de alto nível, pode não ser trivial ir em sentido contrário e descompilar o executável em código de alto nível que funcione (34). No entanto, descompilar uma pequena parte de binário pode ser útil para entender a estrutura e a funcionalidade do código. Algumas linguagens e compiladores incluem informações sobre o código de alto nível original, o que permite que os descompiladores forneçam código de alto nível funcional. Infelizmente, as amostras de *malware* raramente se apresentam dessa maneira.

A análise estática completa de um arquivo binário geralmente não é necessária. Quando uma análise estática é realizada, a informação sobre a amostra de *malware* já foi recompilada normalmente por meio da análise superficial e dinâmica, reduzindo a necessidade de análise estática. De forma semelhante à análise dinâmica, alguns autores (10, 30) a dividem em básica e avançada.

2.2.5.1 Análise Estática Básica

A análise estática básica consiste em examinar um determinado arquivo executável sem observar as instruções reais. Por intermédio deste tipo de análise é possível confirmar se um arquivo é malicioso, fornecer informação sobre sua funcionalidade, e às vezes até informações comuns podem ser obtidas para criar assinaturas de rede simples. Embora este tipo de análise seja geralmente rápida e fácil de realizar, muitas vezes é de grande ajuda quando o *malware* é sofisticado, obtendo comportamentos importantes (10).

***Hashing*:** Consiste em obter um conjunto de caracteres únicos, aplicando uma função matemática a uma entrada de dados, neste caso, um arquivo executável. O resultado desta operação é único para cada arquivo, portanto serve para comparar se um arquivo sofreu algum tipo de alteração.

***Busca de Strings*:** Consiste em procurar cadeias de caracteres dentro de um executável. Por exemplo, ao imprimir uma mensagem, ao copiar um arquivo para um local específico, ou conectar-se a um URL, ficará armazenada um *string*, geralmente no formato ASCII ou Unicode[4]. Para este fim, programas especiais são usados para analisar tais strings.

2.2.5.2 Análise Estática Avançada

A análise estática avançada consiste em fazer engenharia reversa das partes internas do *malware*, ou seja, executar a parte obtida em um desmontador e observar as instruções do programa, descobrindo desta maneira o que o programa faz. Para realizar este tipo de

[4] **Unicode** é um padrão adotado mundialmente que possibilita com que todos os caracteres de todas as linguagens escritas utilizadas no planeta possam ser representados em computadores.

análise é necessário ter conhecimento avançado do desmontador, sistemas operacionais e construções de código (10).

***Engenharia Reversa*:** é um método que permite obter informação a partir de um software finalizado. Na análise de *malware* o objetivo é obter como saída o código montador a partir de um binário. Geralmente, quando o *malware* se posiciona no disco rígido, ele o faz em um formato binário permitindo que a máquina entenda mais rapidamente e execute as instruções de forma eficiente. Um programa desmontador, portanto permitirá, que a linguagem da máquina (binária) seja transformada em código de montagem.

2.3 Técnicas de Evasão

Os desenvolvedores de programas maliciosos parecem estar bem cientes do risco que envolve a engenharia reversa dos mesmos. A engenharia reversa revela as técnicas utilizadas, quem controla o *malware*, os endereços IP e pode, em última instância, levar à identificação e à perseguição dos desenvolvedores. Para dificultar a análise e a engenharia reversa do *malware*, são desenvolvidas várias técnicas para ofuscar as amostras de *malware*. Estas técnicas não impedem a análise do software, tampouco são seguras. As técnicas seguem o princípio de Kerckhoffs de segurança por obscuridade (35), no entanto, tornam a análise mais difícil e entediante. Os principais métodos de evasão são aqueles detalhados nas seguintes subsecções (10, 36, 37, 38).

2.3.1 Ofuscação

A ofuscação é uma técnica pacífica que tem como objetivo modificar o programa para torná-lo menos legível, sem alterar sua funcionalidade. Isto é feito alterando o projeto, a estrutura, a organização e os dados de tal maneira que o programa seja funcionalmente idêntico. Os empacotadores executáveis são talvez a técnica de ofuscação mais utilizada uma vez que comprimem e podem também codificar o programa, tornando-o ilegível para uma análise estática. Dessa maneira, o programa se descomprime e é descriptografado de forma transparente no momento em que é carregado na memória (39, 36).

Já que os criadores de *malware* frequentemente usam a ofuscação para escapar dos software antivírus, é importante compreender como é utilizada esta técnica em *malware*. Aqui temos algumas técnicas de ofuscação que são utilizadas para empacotar cadeias maliciosas:

2.3.1.1 Inserção de código morto

Uma inserção de código morto[5] é uma abordagem simples para mudar a aparência de um programa enquanto mantém sua funcionalidade. *NOP* é um exemplo de tal comando. O código original se ofusca facilmente inserindo instruções *NOP*. Os software antivírus baseados em assinatura, por outro lado, podem resistir a esta estratégia, simplesmente eliminando as instruções falhadas antes de analisá-las (36).

A instrução *NOP* não faz nada, e a execução continua com a instrução seguinte. Esta instrução não afeta os registradores nem as *flags*. *NOP* é normalmente usado para gerar um atraso na execução ou para reservar espaço na memória do código (36).

2.3.2 Método XOR

Este popular método de ofuscação esconde os dados para que não possam ser analisados. Isto é feito através do intercâmbio do conteúdo de duas variáveis dentro do código, como por exemplo:

- XOR EBX, EAX
- XOR EAX, EBX
- XOR EBX, EAX

OR exclusivo, ou *XOR*, é uma operação que retorna um valor que depende de apenas um bit sendo definido. Isso geralmente é mais fácil de explicar com um exemplo. Aqui está um exemplo simples usando Python:

```
>>> bin(5)
'0b101'
>>> bin(6)
'0b110'
```

Para este exemplo *XOR*, aplica-se esta operação aos valores 101 e 110, após realizar 101 *XOR* 110, obtemos 011 (em decimal 3), em Python o *XOR* é representado com o símbolo ^.

```
>>> 5^6
3
```

Alguns criadores de *malware* usam *XOR* porque é simétrico (a mesma chave usada para criptografar os dados também é usada para descriptografar os dados) e muito fácil de

[5] Do inglês, Dead-Code Insertion

executar. Eles podem criar seu próprio tipo de "chave", com uma sobrecarga matemática mínima, o que pode ser difícil para ferramentas e/ou analistas adivinharem.

2.3.2.1 Retribuição de registradores

A retribuição de registradores[6] é outra técnica simples que muda os registradores de tempos em tempos enquanto mantém o código do programa e seu comportamento igual (36).

2.3.2.2 Reordenação de Sub-rotina

A Reordenação de Sub-rotina[7] ofusca um código original reorganizando aleatoriamente suas sub-rotinas. Este método pode gerar $n!$ diferentes variações, onde n denota o número de sub-rotinas (36).

2.3.2.3 Substituição de Instruções

A substituição de instruções[8] evolui um código original substituindo algumas instruções por outras equivalentes. Esta técnica pode efetivamente mudar o código com uma biblioteca de instruções equivalentes (36).

2.3.2.4 Transposição do Código

A transposição do código[9] reordena a sequência de instruções de um código original sem afetar o comportamento do código. Este procedimento pode ser realizado de duas maneiras (36):

- A primeira técnica embaralha as instruções aleatoriamente, depois insere ramificações ou saltos incondicionais para restaurar a ordem de execução original. Não é difícil vencer este método porque o programa original pode ser facilmente restaurado eliminando os galhos ou saltos incondicionais.
- O segundo método cria novas gerações escolhendo e reordenando as instruções independentes que não têm impacto umas nas outras. Como é um problema complexo encontrar as instruções independentes, este método é difícil de implementar, mas pode fazer com que o custo da detecção seja alto.

[6] Do inglês, Register Reassignment
[7] Do inglês, Subroutine Reordering
[8] Do inglês, Instruction Substitution
[9] Do inglês, Code Transposition

2.3.2.5 Incorporação de Código

A incorporação de código[10] foi detectada pela primeira vez no vírus *Zmist/Win95* (também conhecido como *Zmist*) e instrui o código malicioso a se unir ao código do programa de destino. O *malware* descompila o programa em bits gerenciáveis, se insere entre eles e então volta a montar o código injetado em uma nova variante para usar a técnica (36).

2.3.2.6 Base64

Base64 é outra técnica de ofuscação muito conhecida e utilizada por adversários. É essencialmente um esquema de codificação de 64 caracteres, sendo o caractere de preenchimento o sinal de igual (=). O alfabeto também inclui as letras az, AZ, + y /, y 0-9 caracteres e a codificação funciona encadeando três caracteres para gerar uma cadeia de 24 bits, que é então dividida em quatro fragmentos de 6 bits, cada um dos quais se traduz em um dos caracteres *Base64* (36).

2.3.2.7 Empacotadores

Em algumas situações, todo o programa se ofusca para evitar a detecção do *malware* até que ele seja colocado na memória. Isto é conseguido com a ajuda de um software que comprime um executável para torná-lo menor. Depois, o executável comprimido é empacotado dentro do código necessário para se descomprimir durante o tempo de execução. O procedimento de descompressão garante que o arquivo executável não se assemelhe ao seu estado natural (36).

2.3.3 Técnicas Anti-Engenharia Reversa

As técnicas anti-engenharia reversa de *malware* são desenvolvidas para dificultar ou impedir que um analista faça a engenharia reversa do programa malicioso. Pode-se subdividir algumas dessas técnicas em três grupos: ***Anti-depuração***, ***Anti-desmontagem*** e ***Anti-emulação***.

2.3.3.1 Anti-depuração

É uma técnica ativa na qual o desenvolvedor de *malware* insere o código para dificultar a análise. Este código não muda a funcionalidade do programa quando é executado normalmente, mas pode mudar a funcionalidade quando é interpretado por um depurador. Algumas abordagens são as funções que comprovam a existência de depuradores e agem de acordo, sem revelar a funcionalidade real do programa se forem detectados depuradores (10).

[10] Do inglês, Code integration

A seguir, apresentam-se as técnicas mais comuns.

- ***API de Windows***: A técnica mais comum utilizada pelo *malware* é a API do Windows, já que fornece várias funções que o *malware* pode usar para detectar os depuradores. A mais utilizada é ***IsDebuggerPresent*** que verifica se há um indicador específico no bloco de ambiente do processo (PEB) para o campo e o processo de depuração, que retornará zero se o processo não estiver sendo executado em um depurador ou diferente de zero se o depurador for anexado. Outra função similar é a ***CheckRemoteDebuggerPresent***, que verifica se um processo remoto está depurando o atual (10, 38).
- ***CloseHandle/NtClose***: CloseHandle/NtClose é um outro uso de malware contra a depuração. Chamar o expedidor com um identificador inválido gera uma exceção de identificadores inválido, ***STATUS_INVALID_HANDLE*** (10, 38).
- ***FindWindow***: FindWindow também é usado para encontrar o depurador fornecendo um tipo de janela (por exemplo, OLLYDBG).
- ***NtGlobalFlag***: O malware também pode tirar proveito do indicador ***NtGlobalFlag*** de *PEB*[11] no deslocamento *0x68* (que é a chamada para verificar se está depurando) ao verificar se seu valor é igual a *0x70*.

2.3.3.2 Anti-desmontagem

Os autores de *malware* utilizam técnicas anti-desmontagem para atrasar, prevenir e/ou evitar a engenharia reversa do seu código. Tal técnica utiliza código criado manualmente para fazer com que as ferramentas de análise de desmontagem produzam uma lista incorreta de programas. Aqui expõem-se algumas técnicas comuns contra a desmontagem (10, 38).

Ofuscação da API: A ofuscação da API muda o nome das identificações (nomes de classes, nomes de métodos, nome de campos) para nomes aleatórios, de maneira que o leitor do código não saiba o que o código realmente está fazendo.

Ofuscação do código de operação/código montador: A ofuscação do código de montar/opcode torna mais difícil a desmontagem do *malware* usando táticas como executáveis com seções decifradas e instruções de código que são difíceis de ler ou sem sentido.

Código lixo/espaguete: O código lixo/espaguete é usado para confundir a engenharia reversa e ocultar o que o código atual está tentando realizar.

[11] Process Environment Block ou Processo Entorno Bloque

Achatamento do fluxo de controle: Seu objetivo é ofuscar o fluxo do programa achatando-o. Para conseguir isto, a transformação divide todos os blocos básicos do código fonte, como o corpo da função, os laços e as ramas condicionais, e os coloca dentro de um único laço infinito com uma instrução *switch* que controla o fluxo do programa. Isto faz com que o fluxo do programa seja significativamente mais difícil de seguir porque as construções condicionais naturais que facilitam a leitura do código desapareceram.

Instrução de salto com o mesmo objetivo: A instrução de salto com o mesmo objetivo é produzida por uma combinação ***jz*** com o ***jnz***. Este é um salto incondicional que o desmontador não reconhece porque desmonta apenas uma instrução de cada vez.

2.3.3.3 Anti-Emulação

Os desenvolvedores de *malware* sabem que a maioria dos analistas preferem executar seu *malware* dentro de um ambiente de máquina virtual para evitar que seus equipamentos sejam afetados pelo *malware* que estão tentando analisar. É por isso que a maioria dos criadores de *malware* utilizam técnicas *anti-VM* para dificultar a visualização ou análise de seu código dentro de uma máquina virtual (10). A seguir, listamos algumas dessas técnicas:

- ***ID de CPU***: Esta é a mais popular entre estas técnicas. Esta instrução é executada com *EAX=0x1* como entrada e o valor de retorno descreve as características do processador. O *bit 31* de *ECX* em uma máquina física será igual a 0 e, em uma máquina virtual convidada, será igual a 1. Outro método que utiliza *CPUID* o executa com *EAX=0x 40000000*, que é chamado de "marca de hipervisor". O resultado retornado será o fornecedor de virtualização ("*VMwareVMware*" para *VMWare* ou "*Microsoft HV*" para *Microsoft*). O resultado é armazenado em *ECX* e *EDX*.

- ***Pílula Vermelha e Nenhuma Pílula***[12]: Pílula Vermelha (Red Pill) é uma técnica *anti-VM* que executa a instrução *SIDT*[13] para obter o valor do registro *IDTR*[14]. O monitor de *VM* deve realocar o *IDTR* do hóspede para evitar conflito com o *IDTR* do anfitrião. Dado que o monitor de *VM* não recebe uma notificação quando a *VM* executa a instrução *SIDT*, é retornado o *IDTR* para a *VM*. Nenhuma Pílula (No Pill) se baseia no fato de que a estrutura *LDT*[15] é atribuída a um processador, não a um sistema operativo. A localização de uma máquina anfitriã será zero e em uma máquina virtual será diferente de zero.

[12] Do inglês, Red Pill and No Pill
[13] Store Interrupt Descriptor Table ou Tabela de Descritores de Interrupção Armazenados
[14] Interrupt Descriptor Table ou Tabela de Descritores de Interrupções
[15] Local Descriptor Table ou Tabela de Descritor Local

3 APRENDIZADO DE MÁQUINA

Mitchell(40) afirma que o aprendizado de máquina é uma área que estuda como construir programas de computador que melhoram seu desempenho em alguma tarefa graças à experiência. Baseia-se nas ideias de várias disciplinas, como inteligência artificial, estatística e probabilidade, teoria da informação, psicologia e neurobiologia, teoria do controle e complexidade computacional (41). Assim, afirma que, para utilizar a abordagem de aprendizado, é preciso considerar uma série de decisões que incluem a seleção do tipo de treinamento, a função objetiva a ser aprendida, sua representação e o algoritmo para aprender essa função a partir de exemplos de treinamento (41).

De acordo com Mitchell(40), algoritmos de aprendizado têm se mostrado úteis em vários domínios de aplicativos, como na mineração de dados em grandes bancos de dados contendo regularidades implícitas, que podem ser descobertas de forma automatizada, em domínios mal compreendidos e onde os humanos não possuem o conhecimento necessário para desenvolver algoritmos eficazes, e em domínios onde os programas devem se adaptar dinamicamente para responder às mudanças nas condições do ambiente.

Por sua vez, Lu e Bai(42) mencionam que as técnicas de aprendizado são a abordagem predominante para a categorização de textos. Esses autores definem as técnicas como um processo geral indutivo que constrói automaticamente um classificador, aprendendo a partir de um conjunto de documentos pré-classificados.

3.1 Paradigmas de Aprendizado

As técnicas de aprendizado são classificadas em supervisionadas, não supervisionadas e semi-supervisionadas. Nas supervisionadas, o objetivo é aprender o mapeamento das respostas corretas para os dados de entrada fornecidos; para isso, é utilizado um conjunto de dados de treinamento composto por pares que consistem em padrões de entrada e saída corretos. Desta forma, o sistema aprende o mapeamento da saída correta para cada padrão de entrada apresentado a ele (43).

Wanjun e Xiaoguang(44) afirmam que as etapas do aprendizado supervisionado aplicadas na classificação são inicialmente um conjunto de exemplos já classificados que são apresentados ao algoritmo com o qual um classificador é construído. Posteriormente, exemplos não classificados são apresentados ao classificador. Finalmente, medidas devem ser tomadas para avaliar o desempenho do classificador.

Alguns exemplos de técnicas de aprendizado de máquina supervisionado são: Árvores de Decisão, Entropia Máxima, Naive Bayes, Máquinas de Vetores de Suporte, dentre outras

(42).

Em técnicas de aprendizado não-supervisionado, a intervenção humana não é necessária para criar um conjunto de dados previamente classificado para ser apresentado ao algoritmo de aprendizado. O objetivo do aprendizado não supervisionado é encontrar padrões interessantes considerando a distribuição e composição dos dados apresentados (43). Exemplos de técnicas de aprendizado não supervisionado são as técnicas de agrupamento, tais como: Técnicas baseadas em Partição e Técnicas baseadas em Hierarquia.

As técnicas de aprendizado semi-supervisionado são um compromisso entre o aprendizado supervisionado e o não-supervisionado. Nesse tipo de aprendizado, os dados são divididos em duas partes: um grupo de dados classificados e outro de dados não classificados.

Para Chapelle, Scholkopf e Zien(43), o aprendizado semi-supervisionado é mais útil quando há uma quantidade maior de dados não classificados do que dados classificados. Especialmente quando para conseguir os classificados exige muito esforço, leva muito tempo ou é muito caro; uma vez que, além disso, obter dados não classificados é geralmente mais barato. Exemplos de técnicas de aprendizado semi-supervisionado são as técnicas de máquinas de vetores de suporte transdutivas, maximização de expectativas[1], etc. Algumas aplicações de aprendizado semi-supervisionado mencionadas pelos autores são o reconhecimento de fala, a classificação de páginas da web e o sequenciamento de proteínas.

Chen e Chau(45) mencionam os principais paradigmas do aprendizado de máquina, que são arquiteturas probabilísticas, aprendizado simbólico e indução de regras, redes neurais, algoritmos baseados em evolução, aprendizado analítico e métodos híbridos.

O *modelo probabilístico* é um dos métodos de aprendizado mais antigos e frequentemente utilizado para classificar diferentes objetos em classes previamente definidas com base em um conjunto de características. Exemplos disso são o modelo Bayesiano e o modelo Naive Bayes (45).

O *Aprendizado simbólico ou indução de regras* pode ser classificado de acordo com a estratégia de aprendizado subjacente como aprendizado por rotina, por instruções, por analogia, a partir de exemplos e por descoberta(45). Exemplos de tais técnicas são o algoritmo de árvore de decisão ID3 e sua variação, o algoritmo C4.5 que apresentam o resultado da classificação na forma de árvores de decisão ou um conjunto de regras de produção.

As redes neurais artificiais imitam os neurônios humanos. Aqui os conhecimentos são representados por descrições numéricas e o conhecimento é aprendido e lembrado por redes de neurônios artificiais interconectados por sinapses com pesos e unidades lógicas de limiar (45). Existem diferentes modelos de redes neurais, sendo alguns exemplos o

[1] Do inglês, Expectation Maximization

feedforward/backpropagation, os *mapas auto-organizados de Kohonen* e o *modelo de rede neural de Hopfield* (45).

Os *Algoritmos evolutivos* imitam o processo de evolução na natureza. Fogel, Owens e Walsh(46) identificam três categorias destes: genética, estratégias evolutivas e programação evolutiva. Os algoritmos genéticos imitam os princípios da seleção natural e utilizam operadores de mutação e cruzamento na população para selecionar os indivíduos[2] mais adaptados e repetem essa operação em várias gerações até obter o melhor indivíduo. Nas estratégias evolutivas, evolui-se uma população de números reais que codifica as possíveis soluções de um problema numérico e os tamanhos dos saltos; nas estratégias evolutivas a seleção está implícita. Na programação evolutiva, uma população de máquinas de estados finitos é desenvolvida submetendo-as a transformações unitárias (47).

O *aprendizado analítico* representa o conhecimento como regras lógicas e as razões para buscar provas, que são compiladas em regras mais complexas para resolver problemas com um pequeno número de buscas (45).

Deve-se notar que, na prática, esses paradigmas de aprendizado de máquina são muitas vezes usados em combinação com várias técnicas para melhor aproveitar as vantagens que cada uma apresenta e também corrigir as fraquezas que possuem se fossem usados individualmente. Isso é conhecido como métodos híbridos.

3.2 Redes Neurais Artificiais

As *Redes Neurais Artificiais*[3], são inspiradas nas redes neurais biológicas do cérebro humano e são constituídas por elementos que se comportam de forma semelhante ao neurônio biológico em suas funções mais comuns. Esses elementos são organizados de maneira semelhante à do cérebro humano (48).

Além de "semelhantes" ao cérebro, as redes neurais artificiais possuem uma série de características típicas do cérebro. Por exemplo, as Redes Neurais Artificiais (RNAs) aprendem com a experiência, generalizam de exemplos anteriores para novos exemplos e absorvem os principais recursos de um conjunto de dados (48). Algumas características são:

Aprender: é adquirir conhecimento de algo por intermédio do estudo, exercício ou experiência. As RNAs podem mudar seu comportamento com base no ambiente. São mostrados um conjunto de entradas e elas se ajustam para produzir saídas consistentes (48).

Generalizar: é estender ou ampliar algo. As RNAs generalizam automaticamente

[2] Um indivíduo é a representação de uma potencial solução para o problema.
[3] Do inglês, *Artificial Neural Networks*

devido à sua própria estrutura e natureza. Essas redes podem fornecer, dentro de uma faixa, respostas corretas para entradas que apresentam pequenas variações devido aos efeitos de ruído ou distorção (48).

Abstrair: é isolar mentalmente ou considerar separadamente as qualidades de um objeto. Algumas RNAs são capazes de abstrair a essência de um conjunto de entradas que aparentemente não apresentam aspectos comuns ou relativos (48).

Embora estudos anteriores tenham sido realizados, considera-se o primeiro modelo, o apresentado por McCulloch e Pitts(49), que em 1943 apresentaram um elemento conhecido como neurônio, que recebia valores de entrada $\boldsymbol{n}$ e gerava uma saída através de uma função, que por meio de alguns valores que ajustam os valores de entrada e um viés poderia realizar determinadas classificações. A forma geral desta função (Equação 3.1) é seguindo o que foi sugerido por Zhang e Zhang(50):

$$y = sen(\omega x - \Theta) \tag{3.1}$$

onde:

$$sgn(v) = \begin{cases} -1 & \text{se} \quad v \leq 0 \\ \\ 1 & \text{se} \quad v > 0 \end{cases}$$

$\omega = \omega_1, \omega_2, ..., \omega_n$	valores que ajustam a entrada ou pesos sinápticos
$x = (x_1, x_2, ..., x_n$	valores de entrada
Θ	é um viés introduzido para modificar a saída

Em 1949, Hebb(51) desenvolveu uma das **regras de aprendizado**, um método de redução de erros, mais usado, *The Hebbian rule*, que foi publicado em seu livro (51). Em 1958, Rosenblatt apresenta o **perceptron** inspirado no trabalho de McCulloch e Pitts(49), uma rede neural que recebe várias entradas binárias e produz uma única saída binária. Já em 1960, Widrow e Hoff(52) apresentam o **ADALINE**, um modelo que oferecia bons resultados quando os demais modelos estavam estagnados. O perceptron multicamada criado em 1986 também soube resolver os problemas existentes (53). Anos depois, em 1989, Cun et al.(54) criam a primeira rede neural convolucional, a LeNet, que abriu um amplo leque de possibilidades no reconhecimento de dados com mais de uma dimensão (reconhecimento de imagem ou voz) e em 2012 com a criação da rede AlexNet por Krizhevsky, Sutskever e Hinton(55), observou-se todo o potencial que esse tipo de rede neural pode oferecer.

3.2.1 Vantagens das redes neurais

Para entender o grande avanço que as redes neurais representam, é necessário entender as vantagens que elas oferecem:

- São sistemas distribuídos não lineares. Um neurônio operando de forma não linear implica que a rede neural não opere linearmente, permitindo que sistemas não lineares ou caóticos operem (53).
- São tolerantes a falhas, ou seja, a falha de um neurônio não obtendo a saída ótima não deve afetar o bom funcionamento da rede (53).
- Adaptabilidade, o sistema oferece uma certa capacidade para mudanças no ambiente de trabalho, presença de ruídos, modificações na entrada, embora também não ofereça grande capacidade de adaptabilidade, pois isso afetaria a convergência do modelo (53).
- Tem a capacidade de estabelecer relacionamentos complexos entre os dados de entrada (53).

Todas essas propriedades geram a grande capacidade que as redes neurais possuem, seu bom funcionamento e sua alta aplicabilidade no mundo moderno.

3.2.2 Estrutura básica de uma rede neural

3.2.2.1 Analogia com o cérebro

O neurônio é a unidade fundamental do sistema nervoso e, em particular, do cérebro. Cada neurônio é uma unidade de processamento simples que recebe e combina sinais de e para outros neurônios. Se a combinação de entradas for forte o suficiente, a saída do neurônio é ativada. A Figura 4 mostra as partes que compõem um neurônio (48). Os Neurônios (ou células nervosas) são células especializadas que transmitem e recebem sinais elétricos que não estão no corpo. Os neurônios são compostos de três partes principais: dendritos, um corpo celular e um axônio. Os sinais são recebidos através de dois dendritos, viajam até o corpo celular e continuam até o eixo para alcançar a sinapse (ou ponto de comunicação entre dois neurônios).

Nas Redes Neurais Artificiais, RNAs, a unidade análoga ao neurônio biológico é o elemento processador[4]. Um elemento processador recebe várias entradas e as combina, geralmente com uma soma básica. A soma das entradas é modificada por uma função de transferência e o valor da saída desta função de transferência é passado diretamente para a saída do elemento processador (48). A saída do elemento processador pode ser conectada

[4] Do inglês, process element (PE)

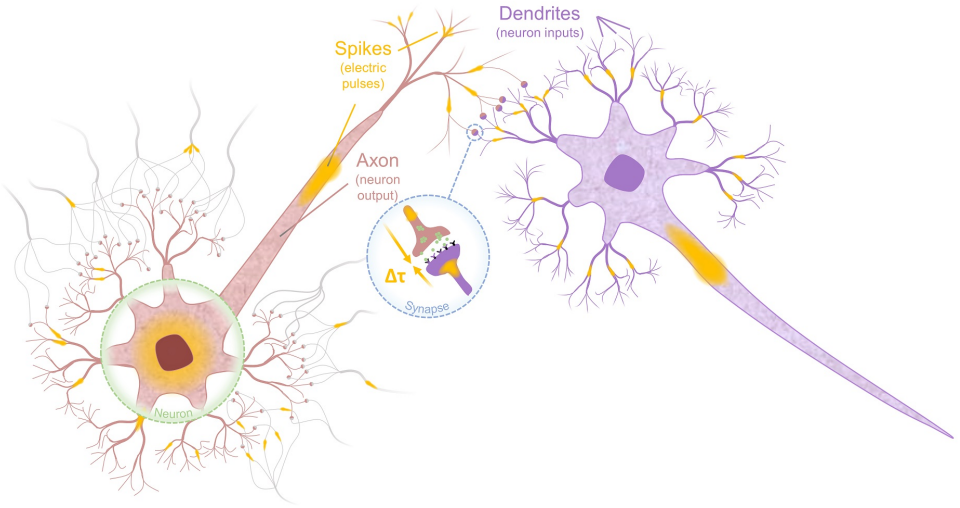

Figura 4 – Componentes de um neurônio. Fonte: Kurenkov et al.(56)

às entradas de outros neurônios artificiais por conexões ponderadas correspondentes à eficiência das sinapses das conexões neurais (48).

A Figura 5 representa um elemento processador de uma rede neural artificial implementada em um computador. Na figura pode-se ver: as *entradas* podem ser as saídas de outros neurônios, entradas extrenas, um bias (tendência) ou qualquer combinação destes elementos; o somatório de todas estas entradas, multiplicadas por suas respectivas forças de conexão sináptica (os *pesos*), dá origem ao chamado "net" de um neurônio; a *função de ativação* estados futuros de um neurônio são afetados pelo estado atual do neurônio e pelo valor do net de entrada, e a *função de saída*, essencialmente, qualquer função contínua e monotômica crescente ser utilizada como função de saída na modelagem neural.

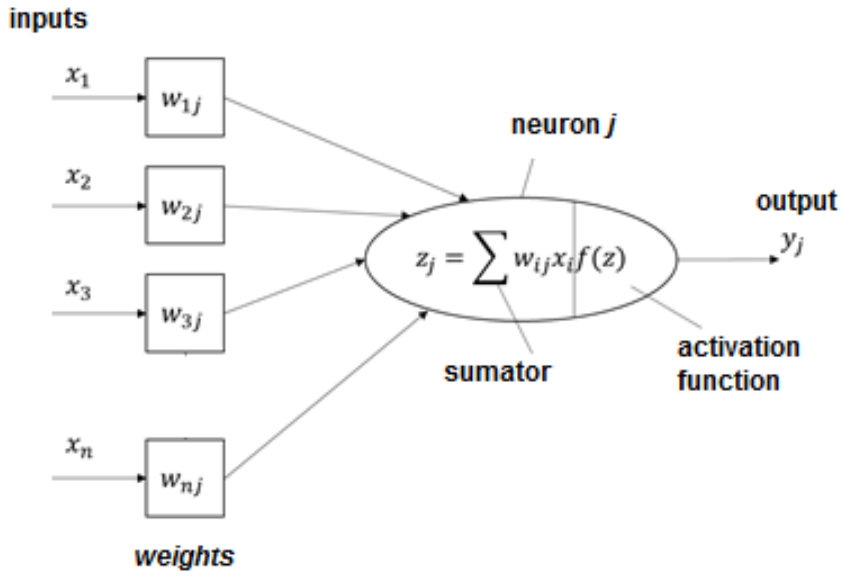

Figura 5 – Esquema de um Neurônio Artificial. Fonte: Karić et al.(57)

Uma rede neural consiste em um conjunto de unidades elementares PE conectadas de uma maneira específica. O interesse das RNAs reside não apenas no modelo do elemento PE, mas nas formas como esses elementos processadores estão conectados. Geralmente os elementos PE são organizados em grupos chamados níveis ou camadas. Uma rede típica consiste em uma sequência de camadas com conexões entre camadas adjacentes consecutivas (58).

Existem duas camadas com conexões com o mundo exterior. Uma camada de entrada, ou *buffer* de entrada, onde os dados são apresentados à rede, e uma camada ou *buffer* de saída que contém a resposta da rede a uma entrada. As demais camadas são chamadas de camadas ocultas. A Figura 6 mostra uma possível arquitetura de uma Rede Neural Artificial (58).

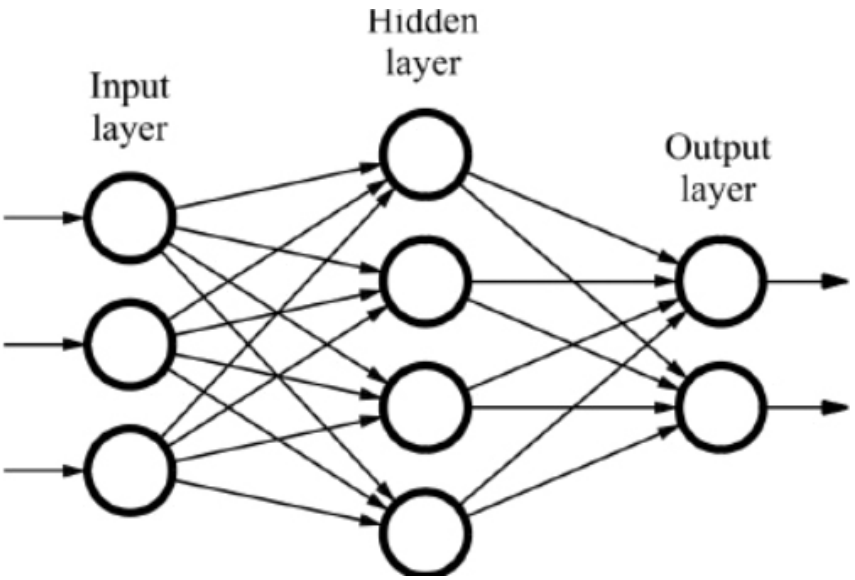

Figura 6 – Arquitetura de uma rede neural simples. Adaptação de Abdelrazek et al.(58)

O neurônio artificial foi projetado para "emular" as características básicas de funcionamento do neurônio biológico. Em essência, um conjunto de entradas é aplicado ao neurônio, cada um representando uma saída de outro neurônio. Cada entrada é multiplicada por seu ***peso*** correspondente ou ponderação análoga ao grau de conexão da sinapse (48). Todas as entradas ponderadas são somadas e o nível de excitação ou ativação do neurônio é determinado. Uma representação vetorial do funcionamento básico de um neurônio artificial é indicada pela seguinte expressão da Equação 3.2.

$$NET = X * W, \tag{3.2}$$

sendo NET a saída, X o vetor de entrada e W o vetor de pesos.

Normalmente o sinal de saída ***NET*** é processado por uma função de ativação ***F*** para produzir o sinal de saída do neurônio ***OUT***. A função ***F*** pode ser uma função

linear, uma função de limiar, ou uma função não-linear que simula com mais precisão as características de transferência não-linear de neurônios biológicos (48). A figura 7 representa um neurônio artificial com função de ativação F.

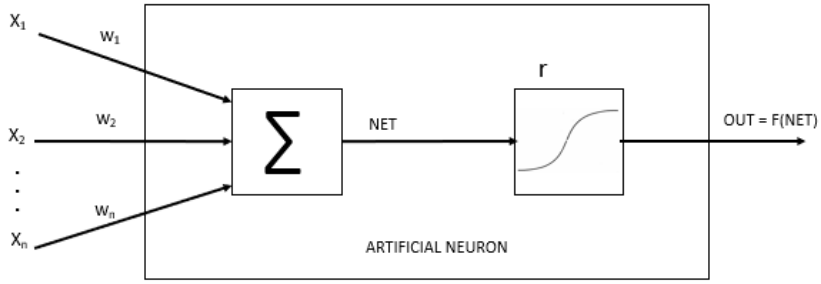

Figura 7 – Modelo de Neurônio Artificial. Adaptação de Carlucci et al.(59)

As funções ***F*** mais utilizadas são as funções *Sigmoid* e *Tangente hiperbólica* expressas na Tabela 1.

Tabela 1 – Funções de ativação

Sigmoid	$OUT = \frac{1}{(1+e^{-NET})}$
Tangente hiperbólica	$OUT = tanh(NET)$

Esse tipo de modelo de neurônio artificial ignora muitas das características dos neurônios biológicos. Dentre elas, destaca-se a omissão de atrasos e sincronismo na geração da saída. No entanto, apesar dessas limitações, as redes construídas com esse tipo de neurônio artificial possuem qualidades e atributos um pouco semelhantes aos dos sistemas biológicos (48).

3.2.2.2 Redes Neurais Artificiais de camada única e multicamada

A capacidade de cálculo e potência das redes neurais vem das múltiplas conexões dos neurônios artificiais que constituem as redes RNA. A rede mais simples é um grupo de neurônios organizados em uma camada conforme apresentado na Figura 8. Os nós circulares são apenas distribuidores das entradas e não são considerados constituintes de uma camada (60).

Cada uma das entradas é conectada através de seu peso correspondente a cada neurônio artificial. Na prática, há conexões deletadas e até conexões entre as saídas e entradas dos neurônios de uma camada. No entanto, a figura mostra conectividade total por razões de generalização (60).

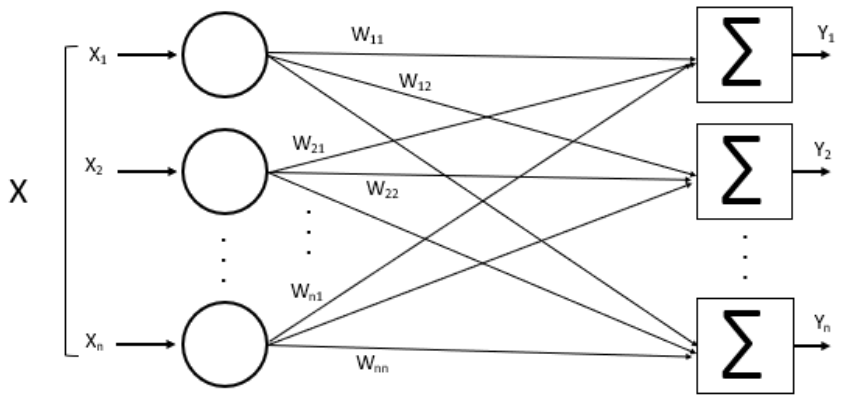

Figura 8 – Exemplo de uma Rede neural de uma camada. Adaptação de Carelli, Silani e Stella(61)

Normalmente, redes maiores e mais complexas oferecem melhor desempenho computacional do que redes menores e simples. As configurações das redes construídas têm aspectos muito diferentes, porém têm um aspecto comum, a ordenação dos neurônios em camadas ou níveis, imitando a estrutura em camadas que o cérebro apresenta em algumas de suas partes (60).

As redes multicamadas são formadas por um grupo de camadas simples em cascata. A saída de uma camada é a entrada da próxima camada. As redes multicamadas demonstraram ter qualidades e aspectos acima das redes de camada única. A Figura 9 mostra uma rede de duas camadas (62).

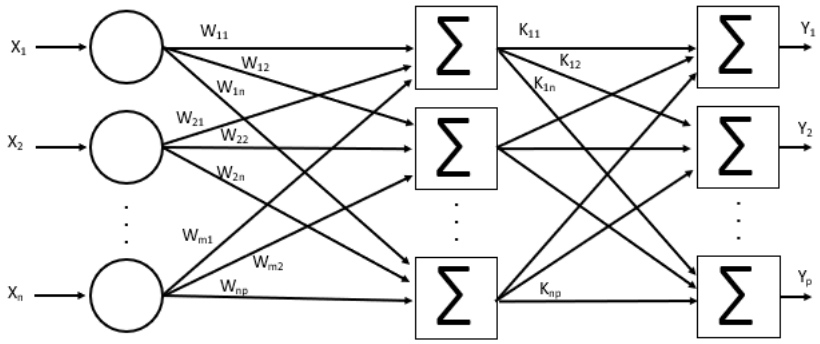

Figura 9 – Exemplo de uma Rede Neural de duas camadas. Adaptação de Wang e Raj(62)

Ressalta-se que o aprimoramento das redes multicamadas está na função de ativação não linear entre camadas, podendo projetar uma rede de camadas simples equivalente a uma rede multicamadas se a função de ativação não linear entre camadas não for utilizada.

3.3 Redes Neurais Convolucionais

Na Seção 3.2 foi desenvolvida uma introdução básica às redes neurais, apresentando e explicando as redes neurais mais simples que existem, com o objetivo de facilitar o entendimento de uma Rede Neural Convolucional (63).

Uma Rede Neural Convolucional (CNN) é um tipo de Rede Neural *feed-forward* em que o padrão de conectividade entre seus neurônios é inspirado na organização do córtex visual dos animais, cujos neurônios individuais estão dispostos de tal forma que respondem a regiões sobrepostas que esculpem o campo visual (63).

As CNNs são compostas por três tipos de camadas: (1) totalmente conectadas, (2) convolucionais e (3) pooling (63). Todas as implementações de CNNs podem ser descritas aproximadamente como um processo que envolve o seguinte:

1. Converter vários filtros pequenos para a imagem de entrada;
2. Sub-amostrar o espaço de ativações de filtro;
3. Repetir os passos 1 e 2 até ter atributos de alto nível suficientes; e
4. Aplicar uma Rede Neural de *feed-forward* padrão aos atributos resultantes.

A Figura 10 corresponde à arquitetura usada em (55) que foi aplicada ao concurso de classificação ImageNet. A arquitetura consiste em 8 camadas que podem ser aprendidas, as cinco primeiras são convolucionais, sendo as demais totalmente conectadas (55).

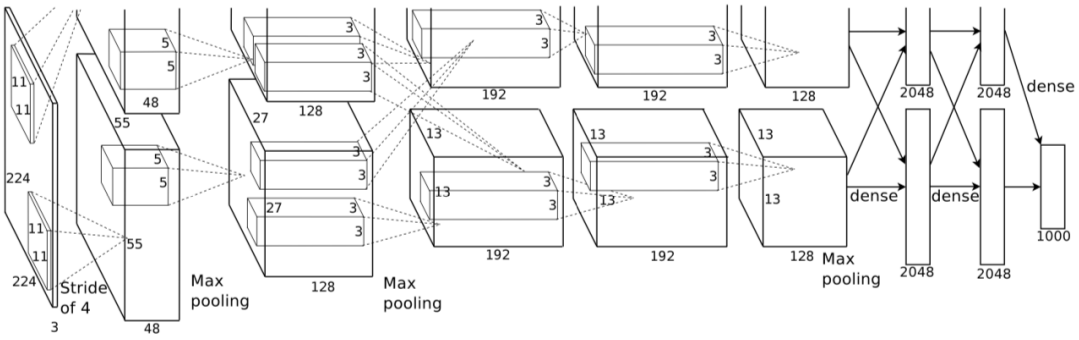

Figura 10 – Arquitetura da Rede Neural Convolucional Alexnet. Fonte Krizhevsky, Sutskever e Hinton(55)

3.3.1 Conexão local

É impraticável conectar neurônios a todos os neurônios da camada anterior ao lidar com entradas de alta dimensão, como imagens, porque a estrutura espacial dos dados não é levada em consideração nessas arquiteturas de rede. Nas redes neurais convolucionais, cada

neurônio está conectado a uma pequena região dos neurônios de entrada (cada neurônio se conecta apenas a uma pequena região contígua de pixels na entrada), e assim as CNNs são capazes de explorar a correlação espacial local impondo um padrão de conectividade local entre neurônios em camadas adjacentes (63).

3.3.2 Camada convolucional

As camadas convolucionais são o núcleo de uma CNN. Uma camada convolucional consiste em um conjunto de núcleos aprendíveis que coexistem ao longo da largura e altura dos recursos de entrada durante a passagem para frente, produzindo um mapa de ativação bidimensional do núcleo (64).

Resumindo, um núcleo consiste em uma camada de pesos de conexão onde a entrada é do tamanho de um pequeno remendo 2D e a saída é uma única unidade.

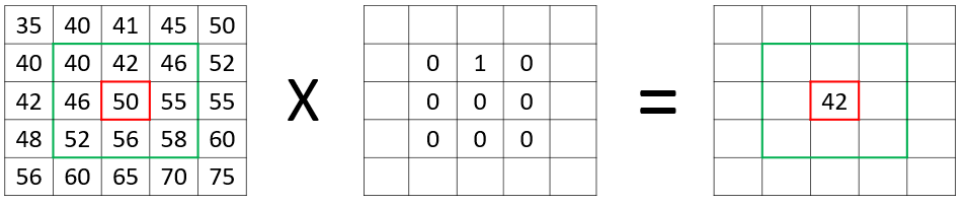

Figura 11 – Exemplo de uma camada de convolução. Fonte: Kimura et al.(64)

Na Figura 11, mostra-se como funciona a convolução, considerando uma imagem de 5x5 pixels como a da figura, onde 0 significa valores completamente pretos e 255 completamente brancos. No centro da figura, um *kernel* de 3x3 pixels foi definido com todos os oito 0s, exceto o *peso* definido como 1. A saída é o resultado da computação do *kernel* em cada posição possível na imagem (63).

A convolução do núcleo através de todas as posições é determinada pela passada. Por exemplo, para o passo 1, a convolução típica é feita, mas para o passo 2, metade das convoluções é evitada porque deve haver 2 pixels de distância entre os centros (63).

O tamanho da saída após a convolução de um *kernel* de tamanho Z sobre uma imagem N com borda S é definido como (Equação 3.3):

$$saida = \frac{N - Z}{S} + 1 \tag{3.3}$$

3.3.3 Camada de *pooling*

A camada de *pooling* é uma forma de redução não linear. Existem várias funções não lineares para implementar o agrupamento, como a mínima, a máxima e a média, mas a mais comum é a máxima. O agrupamento de máximas funciona dividindo a imagem em

um conjunto de retângulos não sobrepostos e, para cada sub-região, o valor máximo é retornado (63). Na Figura 12 pode-se ver um exemplo de Max-pooling..

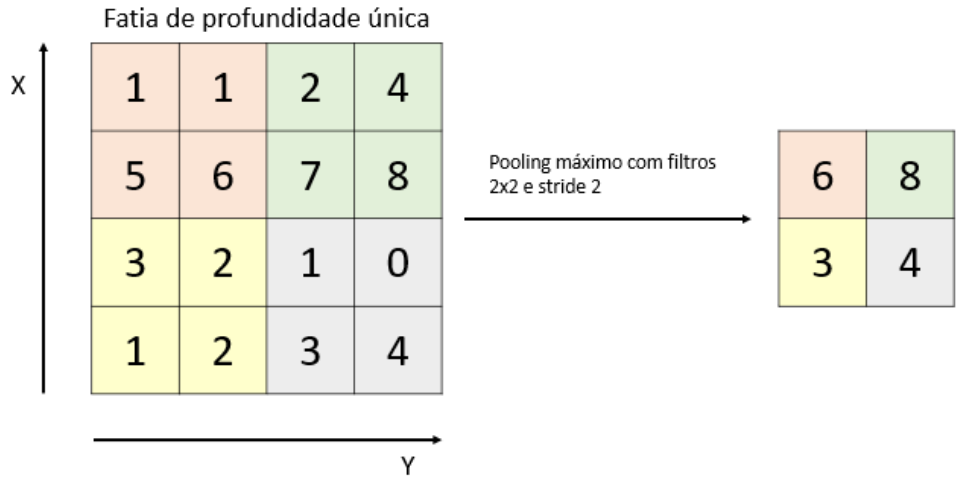

Figura 12 – Exemplo de Max-pooling. Fonte: Yani, Irawan e Setianingsih(65)

Segundo Grus(63), as principais vantagens do max-pooling são:

1. Reduz o cálculo das camadas superiores removendo valores não máximos; e
2. Fornece uma forma de invariância de translação e, consequentemente, fornece robustez adicional à posição por ser uma forma de reduzir a dimensionalidade das representações intermediárias.

3.3.4 Vantagens das redes neurais convolucionais

As redes neurais convolucionais são semelhantes às redes neurais, e sua principal vantagem é que cada parte da rede é treinada para realizar uma tarefa, o que reduz significativamente o número de camadas ocultas, tornando o treinamento mais rápido. Além disso, tem invariância de tradução dos padrões a serem identificados (66).

As redes neurais convolucionais (CNN) são um dos modelos mais populares em uso atualmente. Este modelo de rede neural computacional utiliza uma variação de perceptrons multicamadas e contém uma ou mais camadas convolutivas que podem ser totalmente conectadas ou agrupadas. Estas camadas convolutivas criam mapas de características que registram uma região da imagem que é eventualmente dividida em retângulos e enviada para o processamento não-linear (66).

As redes neurais convolucionais são muito poderosas para tarefas relacionadas à análise de imagens, uma vez que são capazes de detectar características simples como detecção de bordas, linhas, etc., e mapeá-las em características mais complexas até detectarem o que estão procurando (66).

As vantagens incluem:

- precisão muito alta nos problemas de reconhecimento de imagem;
- detecção automática de funções importantes sem supervisão humana; e
- compartilhamento de pesos, onde um grupo de conexões pode compartilhar o mesmo peso, ao invés de cada conexão ter um peso diferente, reduzindo assim a quantidade de parâmetros.

3.4 Aprendizado por Transferência

Os humanos têm uma habilidade inerente de transferir conhecimento por meio de tarefas, ou seja, reconhecer e aplicar conhecimentos relevantes de experiências de aprendizado anteriores quando confrontados com novas tarefas (60). Quanto mais relacionada uma nova tarefa estiver com uma experiência anterior, mais facilmente pode ser dominada. Os algoritmos convencionais de aprendizado de máquina e aprendizado profundo tradicionalmente são projetados para trabalhar de forma isolada, sendo treinados para resolver tarefas específicas (60).

O aprendizado por transferência tenta mudar esta situação desenvolvendo métodos para transferir o conhecimento aprendido em uma tarefa ou tarefas anteriores e usá-lo para melhorar o aprendizado de uma nova tarefa, como exemplificado na Figura 13.

Em resumo, transferir um aprendizado é o processo de treinamento de uma arquitetura em um conjunto de dados em larga escala e depois usar esse modelo previamente treinado para realizar o aprendizado para uma tarefa subsequente (ou seja, a tarefa alvo).

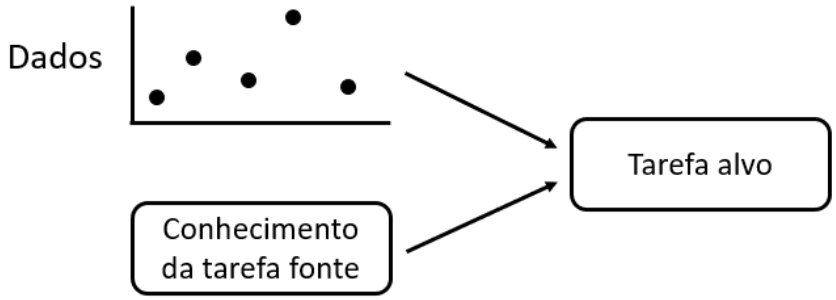

Figura 13 – Aprendizado por transferência. Adaptação de A e J(67)

Os métodos de transferência tendem a depender muito dos algoritmos de aprendizado por transferência usados para aprender tarefas e muitas vezes podem ser considerados simplesmente extensões desses algoritmos. Parte do trabalho do aprendizado por transferência está no contexto de aprendizado indutivo e envolve a extensão de algoritmos de

classificação e inferência bem conhecidos, como redes neurais (68), redes Bayesianas (69) e cadeias de Markov (70).

O objetivo do aprendizado por transferência é melhorar o aprendizado na tarefa objetiva, aproveitando o conhecimento da tarefa fonte. Existem três medidas comuns pelas quais a transferência pode melhorar o aprendizado. Primeiro, o desempenho inicial, que pode ser alcançado na tarefa alvo usando apenas o conhecimento transferido, antes de qualquer aprendizado adicional ser feito, em comparação com o desempenho inicial de um agente ignorante. O segundo é a quantidade de tempo que leva para aprender completamente a tarefa alvo, dado o conhecimento transferido em comparação com a quantidade de tempo para aprendê-lo do zero. Já a terceira é o nível de desempenho final alcançável na tarefa alvo em comparação com o nível final sem transferência. Essas três medidas são ilustradas na Figura 14.

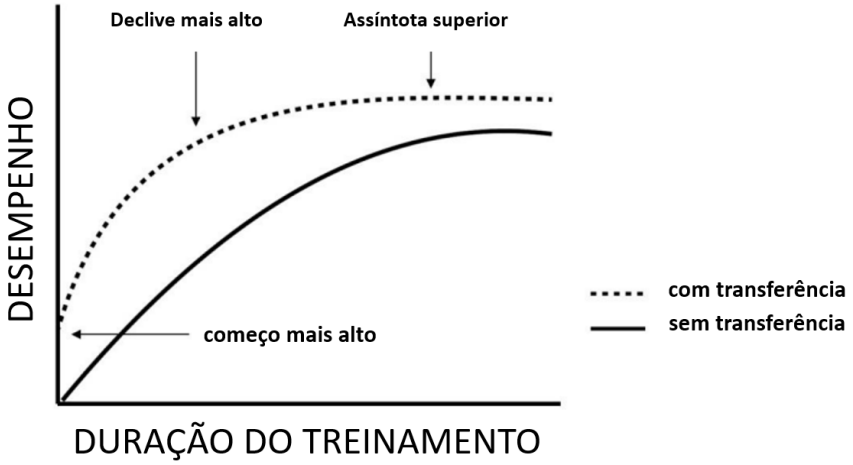

Figura 14 – Três maneiras de melhorar o aprendizado. Fonte: A e J(67)

Para Howard e Gugger(60), o aprendizado por transferência pode ser definido como "Modelos sofisticados de aprendizado profundo possuem milhões de parâmetros (pesos) e treiná-los do zero geralmente requer grandes quantidades de dados e recursos computacionais. O aprendizado por transferência é uma técnica que abrevia muito isso, pegando uma parte de um modelo que já foi treinado em uma tarefa relacionada e reutilizando-o em um novo modelo" (71).

3.4.1 Comparação com aprendizado de máquina

O aprendizado por transferência não é um conceito novo e específico de aprendizado de máquina (72). Há uma grande diferença entre a abordagem tradicional de construção e treinamento de modelos de aprendizado de máquina e o uso de uma metodologia que segue os princípios do aprendizado por transferência. Tal fato é melhor ilustrado na Figura 15.

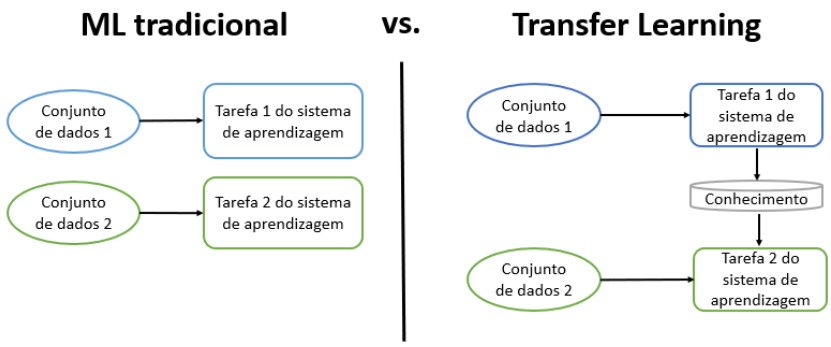

Figura 15 – Aprendizado de Máquina tradicional vs. Aprendizado por Transferência. Adaptação de Sarkar(72)

O aprendizado tradicional é isolado e ocorre apenas com base em tarefas específicas, conjuntos de dados e treinamento de modelos isolados separados uns dos outros. Nenhum conhecimento é retido que não possa ser transferido de um modelo para outro. Com o aprendizado por transferência, o conhecimento (características, pesos, etc.) de modelos previamente treinados pode ser usado para treinar modelos mais novos e até mesmo resolver problemas como ter menos dados para a nova tarefa (72).

3.4.2 Estratégias de Aprendizado por Transferência

Existem diferentes estratégias e técnicas de aprendizado por transferência que podem ser aplicadas dependendo do domínio, da tarefa em questão e da disponibilidade de dados.

Como pode ser visto na Figura 16, os métodos de aprendizado por transferência (73) podem ser classificados de acordo com os tipos de algoritmos de aprendizado de máquina tradicionais envolvidos, tais como:

- *Aprendizado por transferência indutiva*: neste cenário, os domínios de origem e destino são os mesmos, mas as tarefas de origem e destino são diferentes umas das outras. Os algoritmos tentam usar as tendências indutivas do domínio de origem para ajudar a melhorar a tarefa de destino. Dependendo se o domínio de origem contém dados marcados ou não, isso pode ser dividido em duas subcategorias, semelhantes ao aprendizado multitarefa e ao autoaprendizado, respectivamente;
- *Aprendizado sem supervisão de transferência*: esta configuração é semelhante à transferência indutiva em si, com foco em tarefas não supervisionadas no domínio de destino. Os domínios de origem e destino são semelhantes, mas as tarefas são diferentes. Nesse cenário, os dados marcados não estão disponíveis em nenhum dos domínios; e

- *Aprendizado por transferência transdutiva*: neste cenário, existem semelhanças entre as tarefas de origem e destino, mas os domínios correspondentes são diferentes. Nesta configuração, o domínio de origem tem muitos dados marcados, enquanto que o domínio de destino não tem nenhum. Isso pode ser classificado em subcategorias, referindo-se a configurações onde os espaços de características são diferentes ou as probabilidades são marginais.

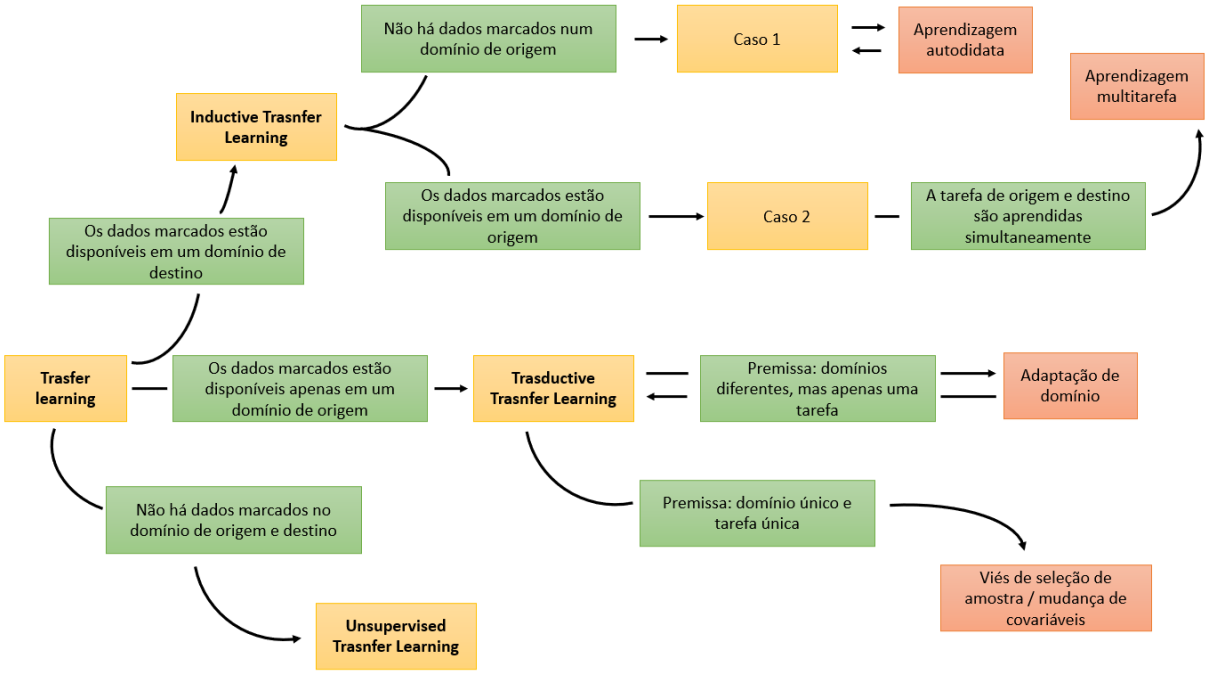

Figura 16 – Estratégias de aprendizado por transferência. Fonte: Pan e Yang(73)

3.4.3 Modelos Pré-Treinados

Simplificando, um modelo pré-treinado é um modelo criado para uma tarefa e retreinado com dados extensos para resolver um problema semelhante. No lugar de construir um modelo ajustando cada um de seus parâmetros para resolver um problema semelhante, o modelo treinado em outro problema é utilizado como ponto de partida.

Por exemplo, se quiserem construir um sistema de reconhecimento de plantas, podem passar meses construindo um algoritmo de reconhecimento de imagem aceitável do zero ou podem pegar um modelo inicial (um modelo pré-treinado) do *Google* que foi construído com dados do ImageNet (74) para identificar características nessas fotos. Um modelo pré-treinado pode não ser 100% preciso em sua aplicação, mas economiza um enorme esforço necessário para reinventar os mesmos conceitos continuamente.

O objetivo de treinar uma rede neural é identificar os pesos corretos para a rede por meio de várias iterações de ida e volta. Usando modelos pré-treinados que foram

previamente treinados em grandes conjuntos de dados, podem usar diretamente os pesos e a arquitetura obtidos e aplicar o aprendizado em a declaração de problema desse trabalho. Isso é conhecido em inglês como *Transfer Learning*, onde transferi-se o aprendizado do modelo pré-treinado para a definição do problema específico.

Deve-se tomar muito cuidado ao escolher qual modelo pré-treinado usar em cada caso. Se o problema for muito diferente daquele utilizado no modelo pré-treinado, a previsão obtida terá um baixo desempenho. Por exemplo, um modelo previamente treinado para reconhecimento de fala terá um desempenho muito ruim se for usado para identificar objetos em imagens.

Felizmente, existem muitas arquiteturas pré-treinadas que estão diretamente disponíveis para uso. O conjunto de dados ImageNet (74) tem sido amplamente utilizado para construir várias arquiteturas, pois é extenso (1.2 milhões de imagens), para criar um modelo generalizado. A especificação do problema é treinar um modelo que possa classificar corretamente as imagens em 1.000 categorias de objetos separadas. Essas 1.000 categorias de imagens representam classes de objetos encontrados na vida cotidiana, como espécies de cães, gatos, vários objetos domésticos, tipos de veículos, etc.

Tais redes pré-treinadas demonstram uma grande capacidade de generalizar para imagens fora do conjunto de dados ImageNet por meio de aprendizado por transferência e modificações no modelo existente que podem ser feitas ajustando o modelo. Uma vez que a rede pré-treinada deve ter sido muito bem treinada, não é desejável modificar os seus pesos tão cedo e demais. Para modificá-los, geralmente é utilizada uma taxa de aprendizado menor do que aquela usada para treinar inicialmente o modelo.

3.4.3.1 Exemplos de redes pré-treinadas

Existem vários modelos de aprendizado por transferência (TL) baseados em redes neurais convolucionais (CNNs). A fim de dar uma breve visão geral de sua utilidade, serão apresentados cinco exemplos de casos práticos e úteis em que os autores utilizaram um modelo de aprendizado por transferência baseado em redes neurais convolucionais. Estes exemplos utilizam os seguintes modelos: ResNet18, ResNet34, VGG16, DenseNet201 e AlexNet.

Al-Falluji, Katheeth e Alathari(75) utilizaram um modelo de aprendizado por transferência, em particular o modelo ResNet18, para detectar automaticamente a doença COVID-19 a partir de um conjunto de dados de imagens de raios X de pacientes com pneumonia bacteriana grave, doença COVID-19 e outro conjunto de dados de imagens de casos normais. O modelo proposto alcançou uma precisão de 96,73%. De acordo com suas descobertas, o sistema proposto pode ajudar os especialistas médicos a emitir diagnósticos a respeito da detecção da COVID-19.

Kumar et al.(76) utilizou o modelo ResNet34 e um conjunto de dados aberto incluindo 15.200 imagens de folhas de cultura para detectar e classificar as doenças das folhas das plantas. Os autores alcançaram uma precisão de 99,40% em um conjunto de testes. Este é outro exemplo da utilidade da aprendizado de transferência.

Das, Acharjee e Marium-E-Jannat(77) utilizou o modelo de aprendizado por transferência VGG16 para a classificação de gêneros musicais com a representação espectrográfica de peças musicais. O autor utilizou clipes de áudio de cada gênero, neste caso dez gêneros, e obteve as imagens de espectrograma desses gêneros com 84% de precisão, o que proporcionou uma abordagem promissora para o problema de classificação do gênero.

Rybiałek e Jeleń(78) utilizou um conjunto de dados limitado (2.247 imagens por classe) para classificar as mamografias de câncer de mama. Os autores descrevem a DenseNet-201 como o melhor modelo, alcançando 91% de precisão. Este é outro exemplo do uso e da utilidade dessas redes pré-treinadas.

Alawi et al.(79) propôs um sistema de diagnóstico inteligente que utiliza técnicas baseadas em um algoritmo de aprendizado profundo, precisamente o modelo pré-treinado AlexNet. O método proposto pelos autores mostra resultados encorajadores. Em termos de precisão, o modelo proposto alcançou 97,33% na fase de validação. Portanto, em alguns lugares onde não há serviços médicos, esta abordagem pode ser amplamente utilizada para o diagnóstico de células parasitas.

3.4.3.2 Modelos usados nesta dissertação

Conforme visto nas seções anteriores, as redes neurais convolucionais (CNNs) são uma classe de redes neurais de aprendizado profundo que foram usadas com sucesso no reconhecimento de objetos a partir de imagens. Oito arquiteturas CNN são testadas neste trabalho: **ResNet-18; ResNet-34; ResNet-50; ResNet-152; VGG16; DenseNet-121; DenseNet-201 e AlexNet**. Todas as CNNs foram selecionadas, após extensa revisão da literatura para o problema, com base em seu bom desempenho em tarefas de classificação de imagens.

Desde a sua introdução em 2015, a arquitetura rede residual (***ResNet***) revolucionou o campo da visão computacional, estabelecendo novos padrões em termos de precisão e desempenho. Mesmo com a sua simplicidade, a ResNet provou ser extremamente eficaz em uma ampla variedade de tarefas de visão computacional, incluindo classificação, detecção e localização de objetos (80). A chave para seu sucesso está em sua arquitetura de "rede residual", que permite que a rede seja melhor treinada, permitindo que as informações fluam pela rede com mais eficiência. A ResNet também possui outros recursos úteis, como uma estrutura de blocos "*bottleneck*" que reduz o número de parâmetros e melhora a eficiência computacional (81).

A Figura 17 mostra a arquitetura da Rede Neural Convolucional *ResNet-50*.

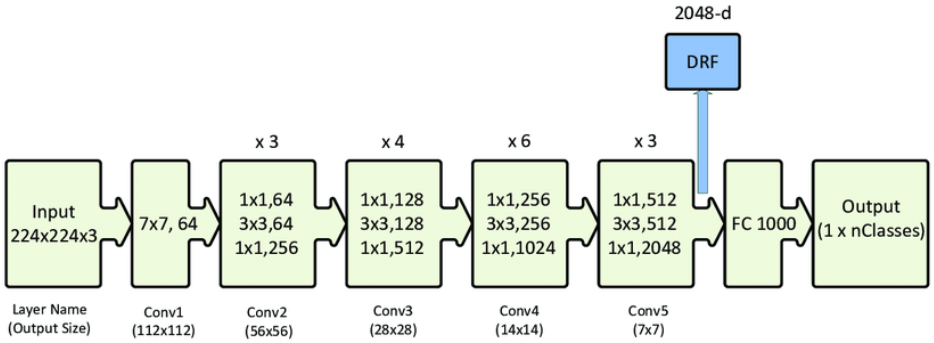

Figura 17 – Arquitetura da ResNet-50. Fonte: Mahmood et al.(82)

A ***VGG16*** e a ***VGG19*** (83) são dois modelos de redes neurais convolucionais muito populares entre os pesquisadores de visão computacional. Ambos foram desenvolvidos pelo grupo de pesquisa Visual Geometry Group (VGG) da Universidade de Oxford. O VGG16 é o modelo usado no ImageNet Large Scale Visual Recognition Challenge 2014 (ILSVRC2014), onde conquistou o primeiro lugar. Neste trabalho é utilizada a arquitetura VGG-16 com normalização por lotes, devido à sua simplicidade e robustez. É importante notar que a normalização por lotes (*Batch Normalization*) é muito eficaz para superar os desafios do treinamento profundo (84).

A Figura 18 mostra a arquitetura da Rede Neural Convolucional *VGG16*.

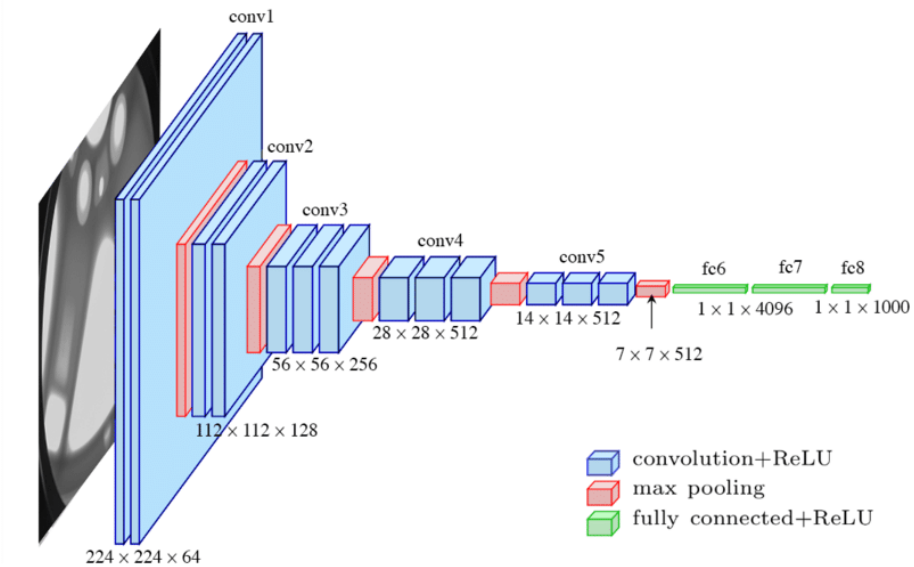

Figura 18 – Arquitetura da VGG16. Fonte Ferguson et al.(85)

DenseNet é uma rede neural do tipo *feed-forward* que conecta todas as camadas entre si. A ideia por trás do DenseNet é simplificar a propagação do sinal na rede conectando

todas as camadas densamente. Isso é conseguido através do uso de uma conexão *skip*, que permite que o sinal se propague diretamente de uma camada para outra, sem passar pelas camadas intermediárias. Isso reduz o número de parâmetros e melhora o fluxo de sinal, o que, por sua vez, melhora o desempenho da rede (86).

DenseNet foi originalmente proposto como uma rede para reconhecimento de objetos, mas tem se mostrado eficaz para uma variedade de tarefas, como segmentação de imagens e classificação de texto. O DenseNet também foi usado com sucesso em aplicações de aprendizado profundo não supervisionadas, como aprendizado de representação de imagem (87). A Figura 19 mostra a arquitetura da Rede Neural Convolucional *DenseNet-121*.

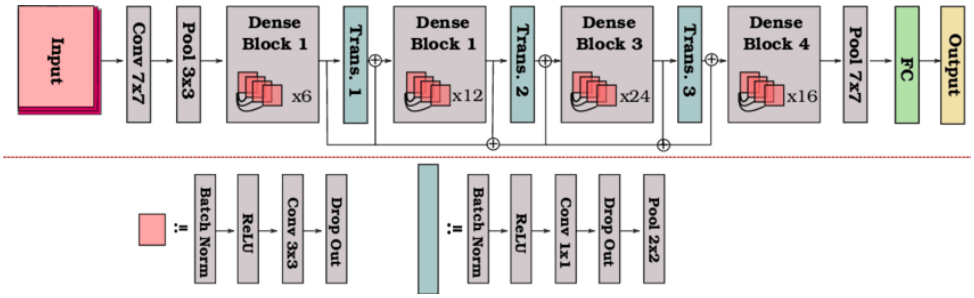

Figura 19 – Arquitetura da DenseNet-121. Fonte Radwan(88)

AlexNet é uma arquitetura de rede neural convolucional desenvolvida por Alex Krizhevsky, Geoffrey Hinton e Ilya Sutskever. A AlexNet competiu no ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) (89) e ganhou o primeiro lugar na classificação de objetos e o segundo na localização de objetos. AlexNet foi a primeira rede neural a usar retificadores em todas as camadas ocultas, permitindo um treinamento mais rápido. A Figura 20 mostra a arquitetura da Rede Neural Convolucional *AlexNet*.

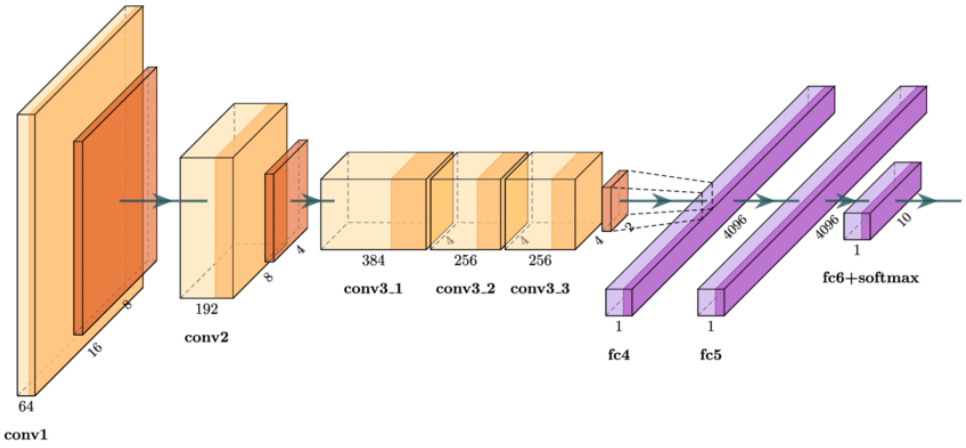

Figura 20 – Arquitetura da AlexNet. Fonte Strisciuglio, Antequera e Petkov(90)

3.5 Ajuste fino pouco profundo

O ajuste fino superficial ou pouco profundo, do Inglês *Shallow Fine-Tuning* (SFT), é um processo usado para melhorar o desempenho dos modelos de aprendizado. É uma forma de seleção de modelos que é usada para escolher o melhor modelo para um determinado problema. O processo de ajuste fino superficial é semelhante ao aprendizado profundo, mas com algumas diferenças importantes. Primeiro, o ajuste fino superficial requer menos dados para treinar o modelo. Em segundo lugar, o ajuste fino superficial é menos intensivo computacionalmente do que o aprendizado profundo. Por fim, o ajuste fino superficial tem mais probabilidade de superajustar os dados de treinamento (91).

Os pesos em todas as camadas convolucionais são inicializados com os valores correspondentes do modelo pré-treinado. Essas camadas são consideradas mais gerais e retêm informações sobre textura, cor e forma. O SFT executa o ajuste fino apenas na última camada totalmente conectada, que é mais especializada. Em geral, esta estratégia é a mais comum, pois permite que os pesos das últimas camadas sejam adaptados ao problema de classificação (92).

4 TRABALHOS RELACIONADOS

Pesquisadores têm estudado *malware* para entender melhor seus comportamentos e estruturas. No entanto, para classificá-los com mais precisão e eficiência, é necessário contar com a ajuda da inteligência artificial. Esta seção apresenta os trabalhos relacionados à classificação de *malware* com base em análise estática e dinâmica, aprendizado de máquina e uso de aprendizado profundo.

4.1 Trabalhos relacionados baseados na análise de características

Yang et al.(93) propuseram um método de análise estática usando um gráfico de chamada de função, semelhante ao feito por Nar et al.(94) extraindo atributos do arquivo executável portátil (PE) do *malware* e sequências de *opcode*. Alsoghyer e Almomani(95) e Faris et al.(96) implementaram análise estática para extrair vários atributos estáticos de binários de *malware* do *Android*, como permissões e chamadas de API, e então executaram técnicas de aprendizado de máquina para detectar aplicativos maliciosos. Jeon et al.(97) realizaram uma análise estática para detecção de *malware* usando o *Tensorflow*.

Já na análise dinâmica, são obtidas atributos comportamentais do *malware*, como chamadas para APIs, atividades na rede e arquivos de log, entre outros. Mohaisen, Alrawi e Mohaisen(98) desenvolveu um esquema automatizado de marcação de *malware*. No sistema proposto, vários recursos baseados em comportamento foram extraídos durante a análise dinâmica, como atividades da rede, sistemas de arquivos e logs. Sihwail et al.(99) usou técnicas forenses de memória com uma abordagem de análise dinâmica. Os artefatos maliciosos foram extraídos primeiro da memória e, em seguida, o *Cuckoo Sandbox* foi usado para monitorar o comportamento do *malware* durante a execução. Por fim, os artefatos maliciosos e o relatório de comportamento foram combinados para criar o conjunto de dados do recurso para classificação posterior. No entanto, softwares maliciosos podem alterar seu comportamento durante a execução em um ambiente virtual, portanto, a análise dinâmica baseada em atributos pode não capturar o comportamento real do *malware*.

4.2 Trabalhos relacionados baseados em visualização

Recentemente, uma extensa pesquisa foi realizada sobre a classificação de *malware* aplicando a abordagem baseada em visualização (7, 8, 100, 101, 102, 103).

Alguns autores desenvolveram soluções baseadas em CNN, nas quais não utilizaram nenhum modelo previamente treinado. Moussas e Andreatos(7) desenvolveram um sistema de classificação de *malware* visualizado baseado em uma rede neural artificial (RNA).

O sistema de classificação proposto utiliza as características extraídas da base de dados *Malimg* (104) para treinar a RNA. Posteriormente, o modelo treinado é utilizado na classificação de diferentes amostras dessa base de dados. A acurácia dos resultados obtidos com uma camada oculta foi de 96%, por outro lado, a implementação com duas camadas ocultas atingiu uma acurácia de 99,135%.

Roseline et al.(100) usaram o aprendizado por transferência e o ajuste fino para introduzir a detecção eficiente de *malware* no contexto de classes desequilibradas sem a necessidade de processos complexos de extração de atributos ou aumentação de dados, alcançando 99,97% de precisão.

Gibert et al.(101) converteram o *malware* em imagens em tons de cinza para desenvolver um esquema de aprendizado profundo baseado em uma CNN. O esquema proposto extrai padrões para classificar software malicioso. Além dos padrões, diferentes funções podem ser implementadas no processo de visualização de *malware*.

Xiao et al.(105) mostraram que a combinação de uma CNN profunda com um gráfico de entropia ajuda a melhorar o processo de classificação de padrões de *malware*.

Vasan et al.(103) também usaram imagens coloridas nas quais eles implementaram o IMCFN, um sistema de classificação de *malware* baseado em imagem usando uma *Fine-tuned* CNN. Inicialmente, eles converteram o *malware* em imagens coloridas usando um algoritmo de mapa de cores. Para superar o problema de desequilíbrio do conjunto de dados, eles aplicaram técnicas de aumentação de dados durante o processo de ajuste. Além disso, também compararam a aplicação da classificação entre imagens em tons de cinza e coloridas, e obtiveram o mesmo resultado, tendo um melhor desempenho de classificação no caso de utilizar imagens coloridas. Embora o IMCFN tenha alcançado uma precisão de 98,82% no conjunto de dados Malimg, o sistema proposto possui complexidade adicional devido às técnicas de aumento utilizadas e ao algoritmo de mapa de cores.

Alguns pesquisadores optaram por combinar vários modelos de treinamento para melhorar o processo de classificação visual (8, 106, 107). Para Nisa et al.(8), o *Scale Feature Texture Analyzer* (SFTA) é combinado com dois modelos de Deep CNN Techniques (DCNN), AlexNet e Inception-v3, para melhorar a precisão da detecção de *malware*.

Vasan et al.(107) utilizam VGG16 e ResNet50 para extração de características, porém, no conjunto de arquiteturas CNN, dois classificadores SVMs, SoftMax e Multiclass, foram implementados. Então, um processo de Análise de Componentes Principais (PCA) foi aplicado para diminuir a dimensionalidade das características, enquanto um processo de fusão foi usado antes do processo de classificação. Além disso, as CNNs podem ser combinadas com outras técnicas, por exemplo, a solução proposta por Narayanan e Davuluru(108) implementou um esquema de classificação baseado em redes neurais recorrentes e convolucionais usando imagens de *malware*.

O modelo proposto por Hemalatha et al.(6) foi construído com uma camada convolucional inicial, uma camada de pooling máximo, quatro blocos de convolução densos (Dense Conv) e quatro camadas de transição (convolução 1 × 1 e 2 × 2 de pooling médio). Os blocos Dense Conv consistem em uma coleção de blocos de convolução 1×1 e 3×3. Após cada camada de transição alternativa, essas convoluções (1×1 e 3×3) são repetidas 6, 12, 48 e 32 vezes dentro de cada bloco Dense Conv. Os mapas de características de saída obtidos após passar por essas camadas são fornecidos como entrada para o bloco Global Average Pooling (GAP). Por fim, uma camada totalmente conectada (FC) classifica as amostras de malware em suas classes correspondentes.

Em outros cenários, a CNN pode ser aplicada para extrair atributos ao implementar técnicas de aprendizado de máquina no processo de classificação de *malware* (109, 110, 111). Em Roseline et al.(109), o *malware* foi classificado pela implementação da técnica de comitê *Random Forest* sequencialmente em várias camadas. A solução proposta é realizada em duas etapas. Inicialmente, os atributos brutos são analisados usando diferentes tamanhos de janela. Diferentes técnicas de classificação são então aplicadas, incluindo *Random Forest* (RF), *Classificador de árvore extra* (ETC) e *Regressão logística* (LR).

Ouahab et al.(110) desenvolveram um descritor de atributos de imagem para extrair semelhanças entre as imagens de *malware*. Em seguida, eles implementaram o algoritmo *K-Nearest Neighbor* (KNN) para realizar o processo de classificação.

Finalmente, Naeem et al.(112) desenvolveu um método para converter arquivos binários de *malware* em imagens em tons de cinza e usou dois padrões, local e global (LGMP), para classificar o *malware*. A partir da parte experimental de seu estudo, a precisão da classificação atingiu 98,4%.

Embora muitos algoritmos que usam a exibição de imagens de *malware* tenham sido propostos, nenhum foi encontrado que usasse amostras de arquivos benignos com funções semelhantes a *malware*. Os métodos usados até o momento são úteis para detectar arquivos maliciosos cujo comportamento está longe do comportamento de arquivos benignos. No entanto, dado que o número de analistas é significativamente menor que o número de novas aparições de *malware*, a necessidade de melhorar a pré-classificação de *malwares* que devem ser analisados manualmente pelos analistas.

O presente trabalho visa abordar, usando redes neurais pré-treinadas e aprendizado de transferência, o problema de identificação de *malware* quando amostras de *malware* e amostras de arquivos benignos compartilham funções semelhantes. Olhando de outra forma, pode-se dizer que as imagens das amostras de *malware* são muito semelhantes às imagens dos arquivos benignos, o que aumenta a dificuldade de classificação. Dessa forma, espera-se uma precisão menor do que a dos trabalhos expostos recentemente.

4.3 Comparativo sobre os trabalhos apresentados

A Tabela 2 apresenta uma comparação entre os estudos relacionados mais recentes que utilizaram imagens para identificar *malware*. Pode-se observar que a maioria das abordagens analisadas não utilizam arquivos benignos, eles se concentram apenas na identificação de categorias de *malware* Malimg (104). Os trabalhos que utilizam arquivos benignos são executáveis que não têm nenhum comportamento anômalo, eles montaram conjuntos de dados com arquivos executáveis normais instalados no computador.

Os trabalhos recentes se concentram na identificação de famílias de *malware* e não em identificar se um arquivo é malicioso ou não, ou seja, eles já assumem que o arquivo é malicioso. Outra característica dos trabalhos relacionados é que a maioria utiliza o conjunto de dados Malimg, no qual todas as amostras são *malware*.

Esta proposta propõe, ao contrário dos trabalhos relacionados apresentados, utilizar dois conjuntos de dados, o primeiro conjunto de dados são amostras de arquivos maliciosos e o segundo conjunto de dados são amostras de arquivos benignos que tenham um comportamento que pareça anômalo, e por isso foram analisados manualmente por analistas de *malware*; isto dará a possibilidade de identificar a probabilidade de que um arquivo desconhecido seja ou não *malware*, tendo entrado na rede não apenas com arquivos benignos que não têm comportamento anômalo, mas sim treinado com arquivos benignos com comportamento anômalo, o que se supõe que produza uma probabilidade menor do que os trabalhos aqui analisados.

Além disso, serão utilizados modelos de rede pré-treinados e bibliotecas de inteligência artificial, o que deverá reduzir os tempos de modelagem e treinamento, permitindo que se experimente mais modelos e realize-se uma comparação mais diversificada.

Tabela 2 – Tabela comparativa dos trabalhos relacionados

Autor	**Objetivo**	**Método**	**Dataset**	**Classifica famí-lias**	**Entrada**	**Acurácia**
Hemalatha et al.(6)	Detecção de *malware* que lide com dados desequilibrados	CNN - DenseNet	Malimg	Sim	Imagens em escala de cinza	97,55%
Roseline et al.(100)	Desenvolver um modelo florestal profundo através da implementação de uma abordagem de conjunto em camadas	Deep Random Forest - Sliding Window	Próprio	Não	Imagens em escala de cinza	98,65%
Moussas e Andreatos(7)	Desenvolver um sistema de classificação de *malware* visualizado com base na ANN	ANN - Scratch model	Malimg	Sim	Propriedades do arquivo e imagem em escala de cinza	99,14%
Nisa et al.(8)	Melhorar a precisão da detecção de *malware* combinando as técnicas AlexNet e Inception-v3	DCNN, SFTA - AlexNet	Malimg	Sim	Imagens em escala de cinza	99,30%

Continua na próxima página

Autor	Objetivo	Método	Dataset	Classifica famí-lias	Entrada	Acurácia
Roseline et al.(109)	Implementar uma CNN leve para classificar imagens de *malware*	Lightweight CNN - Scratch model	Malimg	Sim	Imagens em escala de cinza	97,68%
Gibert et al.(101)	Agrupar eficazmente *malwares* em famílias, com base em seus padrões	CNN - Scratch model	Malimg	Sim	Imagens em escala de cinza	98,40%
Vasan et al.(103)	Implementar um sistema de classificação CNN baseado em imagens coloridas de *malware*	CNN - VGG16, ResNet50	Malimg	Sim	Imagens coloridas	98,82%
Saadat e Raymond(106)	Desenvolver um sistema de classificação preciso e rápido, integrando a CNN com o algoritmo Gradient Boosting	CNN - XG-Boost	Malimg	Sim	Imagens em escala de cinza	97,70%
Xiao et al.(105)	Combinar uma CNN profunda com um gráfico de entropia	CNN - Scratch model	Malimg	Sim	Imagens em escala de cinza	99,64%

Continua na próxima página

Autor	Objetivo	Método	Dataset	Classifica famílias	Entrada	Acurácia
Naeem et al.(111)	Implementar um sistema industrial de classificação de *malwares* na Internet baseado na visualização de imagens coloridas	DCNN - Scratch model	Malimg	Sim	Imagens coloridas	98,79%
Vasan et al.(107)	Implementar um sistema de detecção de *malware* baseado em visualização usando os modelos VGG16 e ResNet50 CNN	CNN - VGG16, ResNet50	Próprio	Não	Imagens em escala de cinza	98,11%
Roseline et al.(113)	Realizar a classificação de *malware* implantando uma técnica sequencial de conjuntos florestais aleatórios em várias camadas	CNN - Xgboost	Malimg	Sim	Imagens em escala de cinza	98,91%
Ouahab et al.(110)	Classificar as imagens de *malware* usando um descritor de características de imagem para extrair as semelhanças entre elas	KNN	Malimg	Sim	Imagens em escala de cinza	97,00%

Continua na próxima página

Autor	**Objetivo**	**Método**	**Dataset**	**Classifica famí-lias**	**Entrada**	**Acurácia**
Naeem et al.(112)	Implementar os padrões maliciosos locais e globais no processo de classificação de *malware*	CNN - LGMP	Próprio	Não	Imagens em escala de cinza	98,00%

5 APLICANDO APRENDIZADO POR TRANSFERÊNCIA NA ANÁLISE DE MALWARE

Neste capítulo, detalhamos uma solução proposta para o problema de classificação de imagens de *malware*, a partir de um conjunto de dados com amostras de imagens de arquivos benignos com algum comportamento ou função semelhante ao *malware*, que é um problema complexo mas fundamental na análise de *malware*. Em vários trabalhos anteriores (100, 111, 103) foi investigado o uso de Redes Neurais Pré-Treinadas para classificar amostras, mas essas redes foram treinadas com amostras de arquivos benignos que não apresentam nenhum comportamento suspeito. Por isso, neste capítulo, propôs-se o uso de Redes Neurais Pré-Treinadas e seu ajuste fino com um conjunto de dados que contém amostras de arquivos benignos com comportamento ou funcionalidade semelhante ao do *malware*. Por outro lado, este trabalho foca no uso de *Transfer Learning* com ajuste fino de redes neurais, uma técnica de aprendizado automático que permite reutilizar os pesos treinados de uma rede neural obtidos a partir de um grande e geral conjunto de dados para obter um bom desempenho com um conjunto de dados pequeno e específico.

Pode-se dividir a solução proposta em duas partes:

1. Construção do conjunto de dados, onde as amostras já classificadas são obtidas, reclassificadas, normalizadas e convertidas em imagens.

2. Treinamento e validação do modelo, onde são treinados os modelos de redes neurais, a fim de descobrir automaticamente padrões nas imagens das amostras, para posteriormente classificá-las.

5.1 Metodologia

O aprendizado por transferência é uma solução adequada para a classificação de imagens, pois fornece uma metodologia pela qual o conhecimento pode ser transferido do domínio de origem para um domínio de destino de um problema semelhante com pouca personalização para modelos CNN profundos.

Os experimentos foram divididos em seis fases: coleta da amostra, pré-processamento da amostra, geração da imagem, geração do modelo, treinamento do modelo e classificação da amostra. Um modelo com as seis fases diagramadas pode ser visto na Figura 21. A divisão do problema geral em fases ou estágios foi essencial para poder enfrentar problemas menores e mais complexos com maior eficiência. A separação das fases do problema geral possibilitou analisar cada um dos subproblemas de forma independente, e assim poder identificar e resolver mais facilmente os erros em cada um deles.

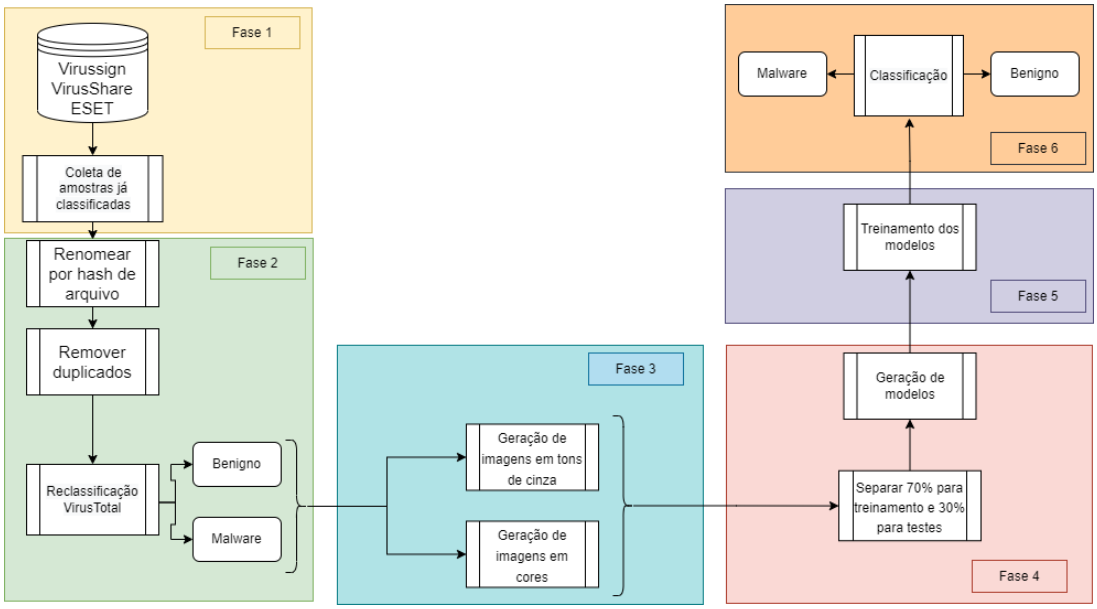

Figura 21 – Diagrama do método proposto. Autoria própria

Durante a fase 1 (coleta de amostras) amostras de *malware* foram obtidas de duas fontes diferentes: VirusShare (114) e VirusSign (115). Por outro lado, as amostras benignas foram doadas pela empresa ESET (116). Como as amostras vieram de diferentes origens, para eliminar duplicatas, foi realizada a fase 2 (pré-processamento das amostras), nesta fase os arquivos foram renomeados, as duplicatas foram removidas. Posteriormente, durante a fase 3 (geração de imagens), foram geradas imagens de todas as amostras resultantes da fase anterior, foram criadas imagens em tons de cinza e imagens coloridas de cada amostra.

Tabela 3 – Modelos propostos

Modelo	Arquitetura
M 1	ResNet-18
M 2	ResNet-34
M 3	ResNet-50
M 4	ResNet-152
M 5	VGG16
M 6	DenseNet-101
M 7	DenseNet-201
M 8	AlexNet

Durante a fase 4 (geração do modelo), as amostras foram separadas em dois conjuntos: *treinamento* e *teste* e posteriormente foram criados 8 modelos de redes neurais pré-treinados, como pode ser visto na Tabela 3 . A fase que exigiu mais tempo e esforço foi a fase 5 (treinamento do modelo), onde foram experimentados os 8 modelos de redes neurais pré-treinados com diferentes hiperparâmetros. O produto final da fase 6 (classificação da amostra) foi uma tabela comparativa com os valores de precisão correspondentes para

cada uma das diferentes arquiteturas avaliadas.

5.2 Conjunto de dados

As amostras utilizadas neste trabalho foram obtidas de três fontes diferentes. O primeiro conjunto de dados contém 14.972 amostras de *malware* de diferentes famílias, das quais 7.328 foram obtidas do site VirusShare (114) e 7.644 amostras foram obtidas do site VirusSign (115) a seleção foi feita aleatoriamente. Por fim, 14.500 amostras de arquivos benignos, mas com algum comportamento semelhante a arquivos maliciosos, foram doados pela empresa ESET (116), essas amostras foram analisadas por um analista humano porque exibiram alguma funcionalidade ou comportamento semelhante ao malware.

A Tabela 4 resume as informações sobre o conjunto de dados resultante da combinação dos três conjuntos de dados mencionados acima.

Tabela 4 – Conjunto de amostras de malware e benigno

Tipo	Amostras	Fonte
Malware	7.328	VirusShare
Malware	7.644	VirusSign
Benigno	14.500	ESET

Ao analisar as amostras de *malware* e amostras de arquivos benignos, observou-se que muitas das amostras benignas foram identificadas como *malware* por algumas empresas de antivírus. Esta observação foi feita durante a descompressão das amostras.

Como as amostras benignas são provenientes de um conjunto de amostras que por algum motivo foram analisadas por um analista humano, que classificou a amostra como benigna, decidiu-se realizar uma redefinição de arquivo malicioso e arquivo benigno.

Para evitar esse problema, uma amostra foi determinada como *Malware* se 3% ou mais dos antivírus no site VirusTotal (117) detectam a amostra como "*malware*". Por outro lado, se a amostra for detectada em até 3%, ela é classificada como "benigna". O valor de 3% foi definido de forma a gerar um conjunto de artefatos benignos com comportamento anômalo.

O conjunto de dados foi dividido de forma estratificada na proporção mais usada de 70:30 (70% de conjuntos de treinamento e 30% de teste), a escolha foi aleatoriamente. No total, o conjunto de treinamento é composto por 20.630 amostras pertencentes a 2 classes (benigno e *malware*), enquanto o conjunto de teste conjunto possui 8.842 amostras também pertencentes às mesmas duas classes (benigno e *malware*).

5.3 Conversão de amostras em imagens

Como todas as amostras obtidas estão no formato de arquivo binário Portable Executable (PE) e nossos modelos são treinados para classificar imagens, devemos converter essas amostras em imagens. Neste trabalho, todas as amostras foram convertidas para imagens em tons de cinza e imagens coloridas. As imagens em tons de cinza na faixa [0, 255] (0: preto, 255: branco). A largura da imagem é fixa e a altura pode variar dependendo do tamanho do arquivo. Para as imagens coloridas, um arquivo RGB de largura fixa e altura variável é criado a partir de dados binários de 24 bits: vermelho de 8 bits, verde de 8 bits e azul de 8 bits. As imagens em tons de cinza e coloridas foram geradas a partir da mesma amostra. Para determinar a largura das imagens, seguiu-se o critério utilizado por (118) que pode ser visto na Tabela 5.

Tabela 5 – Largura da imagem para vários tamanhos de arquivo

Faixa de tamanho do arquivo	Largura da imagem
<10 kB	32
10 kB - 30 kB	64
30 kB - 60 kB	128
60 kB - 100 kB	256
100 kB - 200 kB	384
200 kB - 500 kB	512
500 kB - 1000 kB	768
>1000 kB	1024

A Figura 22 mostra o resultado da conversão de uma amostra em uma imagem em tons de cinza e uma imagem colorida de uma amostra de **malware**.

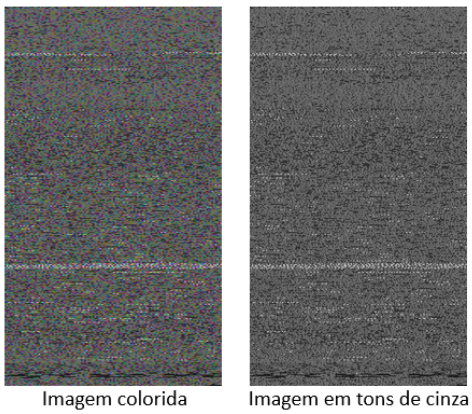

Figura 22 – Exemplo de imagem de um arquivo **malware** em cores e tons de cinza. Autoria própria

A Figura 23 mostra o resultado da conversão da amostra em uma imagem em tons de cinza e uma imagem colorida de uma amostra **benigna**.

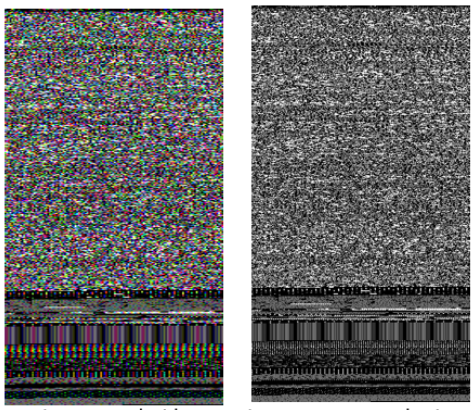

Figura 23 – Exemplo de imagem de um arquivo **benigno** em cores e tons de cinza. Autoria própria

5.3.1 Redimensionamento de imagem

Os tamanhos de imagem de entrada para cada uma das arquiteturas dos modelos propostos têm o mesmo tamanho de imagem que pode ser visto na Tabela (6):

Tabela 6 – Tamanho de imagem sugerido para cada arquitetura

Modelo	Arquitetura	Tamanho de imagem
M1	ResNet-18	224x224
M2	ResNet-34	224x224
M3	ResNet-50	224x224
M4	ResNet-152	224x224
M5	VGG16	224x224
M6	DenseNet-121	224x224
M7	DenseNet-201	224x224
M8	AlexNet	224x224

Como um dos objetivos deste trabalho é experimentar a modificação de parâmetros das CNNs, experimentou-se modificando o tamanho das imagens de entrada, para determinar o comportamento do modelo diante dessas mudanças.

Quando ao usar reamostragem ou redimensionamento de imagens, talvez pode ser necessário realizar uma interpolação. Os operadores de redimensionamento mais comuns são *interpolação bilinear*, *convolução cúbica* e *vizinho mais próximo* (119).

interpolação bilinear foi escolhido como operador de redimensionamento, pois é o que faz uma melhor estimativa nos casos comuns. Os outros dois métodos, *convolução cúbica* e *vizinho mais próximo*, são mais rápidos, mas alguns autores mostram que eles não são tão precisos (119).

Todas as imagens foram redimensionadas para 64x64, 128x128, 224x224 e 255x255

pixels usando *interpolação bilinear*. O redimensionamento permite que cada imagem seja adaptada aos dados de entrada das arquiteturas CNN utilizadas neste trabalho.

5.3.1.1 Interpolação bilinear

Interpolação bilinear consiste em inserir um número entre dois números. O diagrama principal de interpolação linear pode ser visto na Figura 24.

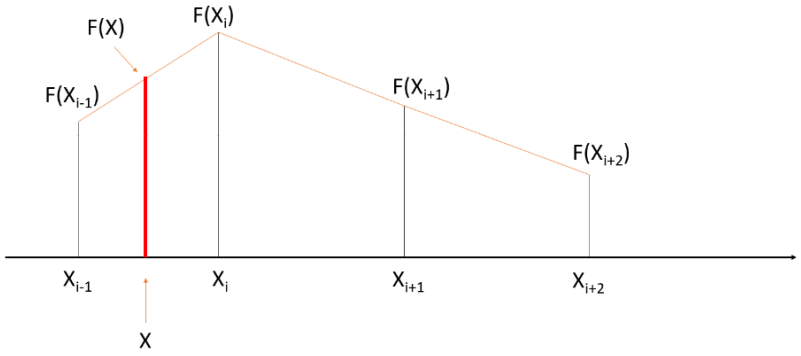

Figura 24 – Diagrama de interpolação bilinear geral. Adaptação de Cao et al.(120)

Interpolação bilinear usa duas linhas retas paralelas ao eixo de coordenadas para decompor a coordenada decimal em quatro pontos de coordenadas inteiras adjacentes. O peso é inversamente proporcional à distância.

No diagrama esquemático da Figura 25 do algoritmo *interpolação bilinear*, pode-se ver que as coordenadas decimais de P são alteradas para a soma ponderada das quatro coordenadas inteiras de Q_{11}, Q_{12}, Q_{21} e Q_{22}.

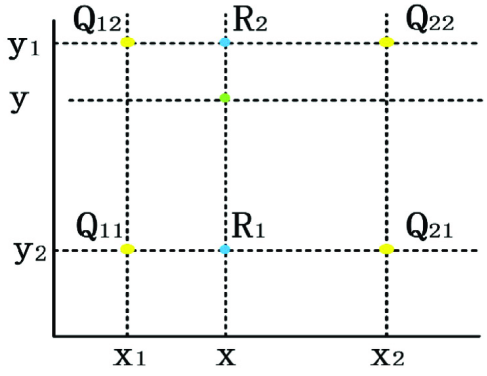

Figura 25 – Esquema de interpolação bilinear. Fonte: Cao et al.(120)

No Apêndice A e B pode-se ver dois exemplos de implementação Python de interpolação bilinear para imagens em tons de cinza e imagens coloridas, respectivamente.

5.4 Bibliotecas e *framework* de aprendizado profundo

Os modelos CNN testados ao longo deste trabalho foram treinados usando o *framework* FastAI (121). Em vez de construir e treinar um modelo "do zero", o aprendizado de transferência foi usado para economizar tempo e recursos, evitando treinar vários modelos de aprendizado de máquina do zero para concluir tarefas semelhantes, sendo este um dos objetivos deste trabalho.

O objetivo do *framework FastAI* é tornar o treinamento de redes neurais profundas o mais fácil possível, tornando-o rápido e preciso usando as práticas recomendadas modernas. A biblioteca *FastAI* é escrita em *Python*, é de código aberto e é construída em cima do *PyTorch*, uma das principais bibliotecas de aprendizado profundo modernas e flexíveis. Ele foi criado com um objetivo principal, tornar a IA fácil e acessível a todos, especialmente pessoas de diferentes origens, habilidades, conhecimentos e recursos, além de cientistas e especialistas em aprendizado de máquina. *FastAI* inclui suporte pronto para uso para tarefas de visão computacional, texto, processamento de linguagem natural, classificação ou regressão de dados tabulares/estruturados e modelos de filtragem colaborativa.

Neste trabalho, o *FastAI* foi escolhido por sua simplicidade e rapidez, pois permite criar modelos de redes neurais com pouquíssimas linhas de código.

5.5 Armazenamento de amostra

Como as amostras de *malware* são altamente perigosas, foi decidido armazená-las e manipulá-las através de um *RaspberryPi* com o sistema operacional *Debian 10*. Como este sistema operacional é baseado em *Linux* e as amostras de *malware* são Executáveis Portáteis para *Windows*, não há maior risco ao manipular as amostras. Uma unidade de estado sólido externa de 250 GB foi adicionada para armazenamento de amostras.

Posteriormente, uma vez convertidos em imagens, os arquivos são movidos para o *Google Drive*, de modo que a integração com o *Google Colab* é completa e permite trabalhar em conjunto com os arquivos não hospedados no *GoogleDrive* . Na Figura 26 pode-se ver a estrutura de diretórios utilizada.

5.6 Framework de trabalho

O *Google Colab* (122) é um aplicativo online de *Jupyter Notebook* que permite aos usuários executarem códigos *Python* em um navegador da Web. O *Colab* fornece um ambiente de computação gratuito e irrestrito e é compatível com muitos dos pacotes *Python* de código aberto mais populares. O *Colab* também facilita o acesso a *GPUs* e *TPUs*, permitindo que os usuários executem código de treinamento de aprendizado de

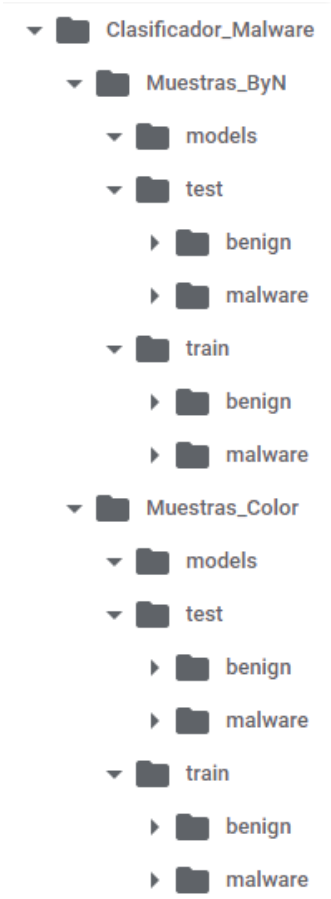

Figura 26 – Estrutura de armazenamento no Google Drive. Autoria própria

máquina gratuitamente.

Um ***GPU*** é uma unidade de processamento gráfico usada para acelerar tarefas relacionadas à criação e manipulação de imagens em computadores. Ele é usado em aprendizado de máquina para acelerar o treinamento de modelos de aprendizado de máquina. *GPUs* possuem um grande número de unidades de processamento paralelo, o que as torna muito eficientes para computar tarefas repetitivas. No aprendizado de máquina, isso se traduz em treinamento de modelos mais rápido.

Por outro lado, ***TPUs*** são unidades de processamento de redes neurais especializadas que são usadas para acelerar o treinamento e o desempenho das redes neurais. As *TPUs* podem ser usadas para aumentar a velocidade de treinamento, melhorar o desempenho e reduzir o custo de treinamento de redes neurais. Além disso, *TPUs* foram originalmente projetadas para melhorar o desempenho de redes neurais em algoritmos de aprendizado de máquina. as *TPUs* têm sido usados em aplicativos de reconhecimento de objetos, detecção de rosto, tradução de idiomas e análise de texto, entre outros.

5.7 Metodologia de avaliação

Como o conjunto de dados está razoavelmente balanceado, optou-se por utilizar a ***acurácia*** como métrica para avaliar o desempenho da classificação e cuja equação é a seguinte (Equação 5.1):

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN}, \tag{5.1}$$

onde: VP (Verdadeiros Positivos) é o número de exemplos em que o modelo indicou que foram positivos e estavam corretos; VN (Verdadeiros Negativos) é o número de exemplos que o modelo indicou foram negativos e estavam corretos; FP (Falsos Positivos) é o número de exemplos em que o modelo indicou que eles eram positivos, mas eram negativos; e FN (Falsos Negativos) é o número de instâncias em que o modelo indicou que elas eram negativas quando na verdade eram positivas.

A acurácia é uma métrica que geralmente descreve o desempenho do modelo em todas as classes globalmente. É útil quando todas as classes têm a mesma importância e é calculada como a razão entre o número de previsões corretas e o número total de previsões.

5.8 Comparação de arquiteturas

Esta seção visa analisar os modelos implementados um a um para mostrar, no próximo capítulo, uma comparação dos resultados obtidos.

O código a seguir mostra o uso da classe *ImageDataLoaders*, que é uma classe de dados FastAI usada para carregar e processar imagens para treinamento de modelos de aprendizado profundo. Fornece funcionalidades para manipulação de imagens, tais como redimensionamento, corte e aumento de imagem.

Em todos os modelos foi definido da mesma forma, variando os parâmetros *resize, bs e num_workers* para alguns testes.

```
data = ImageDataLoaders.from_folder(
            path,
            train='train',
            valid='test',
            item_tfms = resize(224),
            bs=64,
            num_workers=8
            )
```

ImageDataLoaders é um dos tipos de classe que usam-se para carregar conjuntos de dados para problemas de visão computacional. Em geral, os conjuntos de dados de visão computacional são estruturados de tal forma que o rótulo de uma imagem é o nome da pasta na qual a imagem está presente. Como nosso conjunto de dados está estruturado dessa maneira, usam-se um método *from_folder* para carregar as imagens das pastas no *path* fornecido.

Especifica-se o caminho do conjunto de dados de onde as imagens são carregadas em lotes, além do nome das pastas que consistem em dados de treinamento e validação a serem usados.

O código a seguir usa a classe *VisionLearner*, que é um módulo FastAI que fornece uma interface para aprender tarefas de visão computacional usando uma rede neural convolucional. Ele fornece funções para treinar e avaliar modelos de redes neurais convolucionais, bem como para prever as saídas de novas imagens.

```
learn = cnn_learner(data, MODEL, metrics=accuracy)
```

O módulo *VisionLearner* é baseado na biblioteca de aprendizado profundo *PyTorch* e foi projetado para simplificar o processo de treinamento e avaliação de modelos de redes neurais convolucionais. Ele fornece uma interface fácil de usar para carregar dados, selecionar hiperparâmetros, treinar modelos e avaliá-los, além de fornecer funcionalidades para prever a saída de novas imagens usando os modelos treinados. Neste caso usa-se uma técnica chamada *transfer learning* para treinar o modelo que utiliza um modelo pré-treinado, ou seja, uma arquitetura padrão já treinada para uma finalidade diferente. Apresentaremos, então, os oito modelos testados.

O **modelo 1** usa a ***ResNet-18*** que é uma rede neural de 18 camadas que foi introduzida por He(123) em seu artigo "Deep Residual Learning for Image Recognition" (Aprendizado Residual Profundo para o Reconhecimento da Imagem). Esta rede é construída a partir de uma série de blocos "*residual*" que permitem que a rede aprenda de forma mais eficiente representações de recursos mais abstratos dos dados de entrada.

Para usar o Res-Net-18, definimos a variável *learn* da seguinte forma:

```
learn = vision_learner(data, resnet18, metrics=accuracy)
```

O **modelo 2** usa a ***ResNet-34*** que é uma rede neural convolucional de 34 camadas que foi introduzida pela primeira vez no artigo de 2015 "Deep Residual Learning for Image Recognition". É uma variante menor do ResNet-50, que possui 50 camadas. Ambas as redes usam o mesmo princípio de aprendizado residual, mas *ResNet-34* é mais fácil de treinar e requer menos memória.

Para usar o ResNet-34 define-se a variável *learn* da seguinte forma:

```
learn = vision_learner(data, resnet34, metrics=accuracy)
```

O **modelo 3** usa ***ResNet-50*** que é uma arquitetura de rede neural convolucional que foi treinada para realizar a classificação de imagens no conjunto de dados ImageNet. A arquitetura ResNet-50 é composta por cinco blocos de resíduos, cada um dos quais é composto por três camadas de convolução. A primeira camada de cada bloco residual é uma camada de convolução 3 x 3, seguida por duas camadas de Normalização em Lote e ReLU. As duas camadas restantes são 1 x 1 e 3 x 3 camadas, seguidas de Normalização em lote e ReLU. A arquitetura ResNet-50 também inclui uma camada de *pooling* 2 x 2 após a primeira e terceira camadas residuais, e uma camada *Global Average Pooling* após a última camada residual.

Para usar o ResNet-50 define-se a variável *learn* da seguinte forma:

```
learn = vision_learner(data, resnet50, metrics=accuracy)
```

O **modelo 4** usa ***ResNet-152*** que é uma rede neural muito profunda e permite aprender representações de alta qualidade para classificação de imagens. As camadas adicionais da rede também permitem que você treine a rede mais rapidamente do que as redes mais rasas (menos camadas). A arquitetura ResNet-152 é baseada na arquitetura ResNet original, que foi desenvolvida por He(123). Na arquitetura original do ResNet, as camadas são empilhadas umas sobre as outras e conectadas por saltos (saltos de identidade). Isso permite que a rede treine mais rápido e evita o problema do gradiente de desvanecimento.

Para usar o ResNet-152 define-se a variável *learn* da seguinte forma:

```
learn = vision_learner(data, resnet152, metrics=accuracy)
```

O **modelo 5** utiliza a arquitetura ***VGG16*** que se caracteriza por sua simetria, pois todas as suas camadas possuem o mesmo tamanho e o mesmo número de neurônios. Também é notável pelo uso de filtros convolucionais 3x3 em todas as suas camadas, o que o torna mais compacto do que outras arquiteturas de redes neurais convolucionais.

Para usar o VGG16 define-se a variável *learn* da seguinte forma:

```
learn = vision_learner(data, vgg16, metrics=accuracy)
```

O **modelo 6** usa ***DenseNet-101***, que é uma rede neural do tipo *DenseNet* que foi treinada para reconhecer objetos em imagens. É composto por uma série de camadas

densamente conectadas, o que permite que as informações se espalhem de forma mais eficiente pela rede. A rede foi treinada usando o conjunto de dados ImageNet, que contém mais de 14 milhões de imagens rotuladas.

Para usar o DenseNet-101 define-se a variável *learn* da seguinte forma:

```
learn = vision_learner(data, densenet101, metrics=accuracy)
```

O **modelo 7** usa ***DenseNet-201***, que é uma rede neural convolucional densa desenvolvida pelo *Facebook*. É uma melhoria na rede original *DenseNet*, aumentando o número de camadas e reduzindo o tamanho dos filtros. Isso melhora a capacidade da rede de aprender recursos e o desempenho em tarefas de classificação de imagens.

Para usar o DenseNet-201 define-se a variável *learn* da seguinte forma:

```
learn = vision_learner(data, densenet201, metrics=accuracy)
```

Finalmente, o **modelo 8** usa ***AlexNet***, que é uma rede neural convolucional de cinco camadas. A primeira camada da rede, de entrada, consiste em uma camada de convolução seguida por uma camada de agrupamento. As camadas 2 a 5 são camadas de convolução seguidas de camadas de agrupamento. A última camada da rede é uma camada de nivelamento seguida por uma camada de saída softmax.

Para usar o AlexNet, define-se a variável *learn* da seguinte forma:

```
learn = vision_learner(data, AlexNet, metrics=accuracy)
```

Depois de criar os modelos com cada arquitetura, procede-se a treinar cada um deles. Para treinar os modelos, é utilizada a classe *fine_tune*. *Fine_tune* é um método de treinamento de rede neural usado para ajustar os pesos de uma rede neural pré-treinada para melhor se adequar a um novo conjunto de dados. Esse método geralmente é usado quando se tem um pequeno conjunto de dados para o qual se deseja treinar um modelo. O método consiste em primeiro treinar a rede neural em um conjunto de dados maior e mais diversificado (neste caso o *Imagenet*), e depois ajustar os pesos da rede neural usando o novo conjunto de dados (imagens de *malware* e benignos). O objetivo do ajuste fino é melhorar a precisão do modelo, adaptando-se melhor ao novo conjunto de dados.

Os modelos foram treinados com 25 épocas cada, como pode ser visto no código a seguir:

```
learn.fine_tune(25)
```

Na Figura 27 pode-se ver um exemplo da saída do comando anterior, por questões de espaço apenas as primeiras 9 linhas são mostradas.

```
# Entrenamos el modelo en 25 épocas de datos con tasa de aprendizaje predeterminada
learn.fit_one_cycle(25)
```

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.560919	0.364464	0.850082	0.149918	10:36
1	0.471284	0.331920	0.868204	0.131796	05:45
2	0.372477	0.319226	0.875000	0.125000	05:48
3	0.345932	0.301246	0.880972	0.119028	05:51
4	0.300012	0.263532	0.892504	0.107496	05:52
5	0.298401	0.280891	0.879530	0.120470	05:49
6	0.273650	0.228608	0.906507	0.093493	05:53
7	0.241965	0.223676	0.912479	0.087521	05:52
8	0.233315	0.200437	0.929983	0.070016	05:53
9	0.218659	0.184056	0.932455	0.067545	05:52

Figura 27 – Exemplo de treinamento de um modelo. Autoria própria

Modelos pré-treinados são basicamente arquiteturas que já são treinadas em um conjunto de dados diferente e para uma finalidade diferente. Por exemplo, pode se empelar o ResNet-34 como uma de nossas redes pré-treinadas. Também conhecido como redes residuais, o ResNet-34 consiste em 34 camadas e é treinado em mais de um milhão de imagens do conjunto de dados *ImageNet*. Esta rede pré-treinada pode facilmente classificar imagens em 1000 classes, como livros, lápis, animais, etc. Portanto, esse modelo conhece vários objetos e traços antes mesmo de ser treinado em nosso conjunto de dados. Por isso, é chamada de rede pré-treinada.

Agora, o aprendizado de transferência é a técnica que permite usar um modelo previamente treinado para uma nova tarefa e conjunto de dados. O aprendizado de transferência é basicamente o processo de usar um modelo previamente treinado para uma tarefa diferente daquela em que foi treinado originalmente, ou seja, neste caso estamos usando o ResNet-34 para treinar com imagens de arquivos executáveis portáteis (*malware* e benigno).

Isso é possível graças a um passo fundamental chamado *fine tuning*. Quando utiliza-se um modelo pré-treinado, é possível usar esta etapa para atualizar o modelo pré-treinado de acordo com as necessidades da tarefa/dados utilizados. O ajuste fino é basicamente uma técnica de aprendizado de transferência que atualiza os pesos do modelo pré-treinado treinando algumas épocas no novo conjunto de dados. Portanto, usando esta técnica é possível obter resultados de última geração em nossa tarefa, ou seja, a classificação de arquivos Executáveis Portáteis (*malware* e benigno).

Outras classes importantes que foram usadas neste trabalho são *unfreeze*, *freeze* e *freeze_to*.

As classes *freeze* e *unfreeze* nos permitem decidir quais camadas específicas do

nosso modelo querem-se treinar a qualquer momento. Isso é feito porque muitas vezes usa-se o aprendizado de transferência, e as primeiras camadas do modelo já estarão bem treinadas para fazer o que fazem, reconhecendo traços básicos, padrões, gradientes, etc., mas as últimas (que são mais específicas para a nossa tarefa exata, como identificar o *malware*) precisarão de mais treinamento.

O *Unfreeze* irá descongelar todas as camadas do modelo, então ele estará treinando a primeira e a última camada, embora ainda possa estar treinando os diferentes grupos de camadas em diferentes velocidades de aprendizado. Isso é chamado de *taxa de aprendizado discriminativa* ou *treinamento de camada discriminativa.*

Por outro lado, *freeze* fará com que todos os seus grupos de camadas, exceto o último, não possam ser treinados, isso significa que congelam-se o primeiro grupo de camadas (o que vem do aprendizado de transferência) e descongelam-se o segundo grupo (também o último), para continuar treinando.

Também neste trabalho experimenta-se a classe *freeze_to (n)* onde especifica-se quais grupos de camadas queremos congelar e quais queremos treinar. Os primeiros grupos de camadas n serão congelados e os últimos grupos de camadas n serão descongelados.

Abaixo pode-se ver um exemplo do uso dessas três classes:

```
learn.unfreeze()
```

```
learn.freeze()
```

```
learn.freeze_to (30)
```

Outro método utilizado neste trabalho foi o *learn.lr_find*. Este método é usado para encontrar a melhor taxa de aprendizado para um modelo. Este método é executado antes de iniciar o treinamento do modelo e usa uma pesquisa para encontrar a *taxa de aprendizado* ideal. Ele também é executado em um lote de dados de treinamento e avalia o modelo em cada etapa. O método retorna um objeto *LRFinder* com os resultados da pesquisa da taxa de aprendizado. O objeto *LRFinder* possui um método *plot_loss* que pode ser usado para exibir os resultados da pesquisa. A taxa de aprendizado é uma função usada para ajustar o tamanho da etapa durante o treinamento de um modelo de aprendizado de máquina. Isso é feito dinamicamente, o que significa que a taxa de aprendizado é ajustada com base no desempenho do modelo em cada iteração. Isso torna o treinamento mais eficiente e permite que o modelo convirja para um bom resultado em menos tempo.

Já uma taxa de aprendizado ruim pode significar um desempenho ruim. Existem 2 maneiras de isso acontecer:

1. **Aprendizado muito lento** se a taxa de aprendizado for muito pequena, levará muito tempo para treinar o modelo. Isso pode significar que para obter um modelo com a mesma precisão, precisa-se empelar mais tempo. Dito de outra forma, levará mais tempo para treinar o modelo usando o mesmo hardware ou será necessário de um hardware com maior poder computacional (ou alguma combinação dos dois); e
2. **Aprendizado muito rápido**: Se a taxa de aprendizado for muito alta, os passos serão tão grandes que excederão o que é um modelo ideal. Muito simplesmente, sua precisão vai saltar em vez de melhorar constantemente.

Para encontrar a taxa de aprendizado ideal, o seguinte comando foi usado:

```
learn.lr_find()
```

Pode-se então traçar essa taxa de aprendizado e treinar novamente o modelo. O comando a seguir gera o gráfico com a taxa de aprendizado ideal sugerida (Figura 28):

```
learn.recorder.plot(suggestion=True)
```

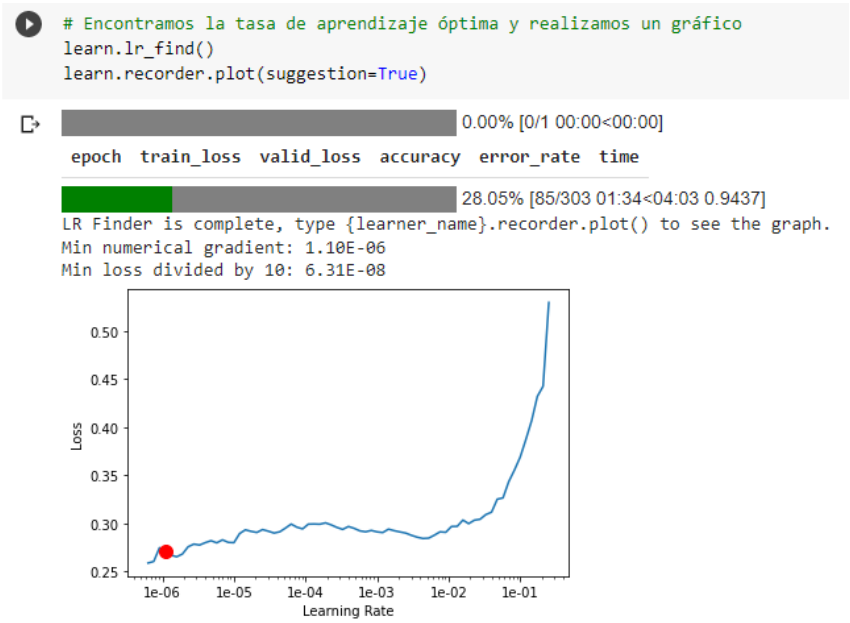

Figura 28 – Exemplo de tasa de aprendizaje óptima. Autoria própria

O que procura-se é um lugar lógico no gráfico onde a perda está diminuindo. O ponto no gráfico indica o valor mínimo no gráfico dividido por 10, bem como o ponto mais íngreme no gráfico.

Finalmente ajusta-se o modelo como primeiro passo de treinamento, com o código mostrado abaixo, então pode-se continuar buscando melhores taxas de aprendizado ou encerrar o aprendizado.

```
learn.fine_tune(1, base_lr = 2e-6)
```

6 ANÁLISE DOS RESULTADOS

Neste capítulo são apresentados os principais resultados dos experimentos realizados. Para uma melhor compreensão, eles foram divididos em: Tempos de Treinamento, Acurácia dos Modelos, Variação em Tamanho e Cor e Comparação com Trabalhos Relacionados. Em cada seção, são apresentados os principais resultados dos experimentos realizados, bem como algumas conclusões antecipadas.

6.1 Tempos de Treinamento

Do ponto de vista de uma rede neural convolucional, o tempo de treinamento é dividido em duas fases: a fase de aprendizado e a fase de teste. Na fase de aprendizado, a rede neural aprende a reconhecer padrões nos dados de entrada. Isso é feito por meio de um processo de ajuste de pesos, no qual são modificados os pesos das conexões entre os neurônios da rede. Na fase de teste, a rede neural é colocada à prova com novos dados, permitindo avaliar seu desempenho. O tempo de treinamento de uma rede neural convolucional depende de vários fatores, como o tamanho da rede, o número de camadas na rede, o número de neurônios por camada, o número de dados de treinamento, a complexidade dos padrões a serem aprendidos, etc. Em geral, é necessário mais tempo para treinar redes neurais maiores e mais complexas. Como todos os modelos já foram previamente treinados e o que fizemos foi utilizar técnicas de transferência de aprendizado, isso nos permitiu experimentar e testar novas ideias mais rapidamente.

A Tabela 7 mostra a demanda de tempo de cada modelo, discriminando o tamanho e as características da imagem. Os valores correspondem ao tempo de treinamento com 25 épocas usando ***GPU*** e sem realizar *freeze* ou *unfreeze* do modelo. Os tempos são expressados em **horas:minutos** [1].

Observando a Tabela 7 podemos concluir que o tempo de treinamento aumenta à medida que o tamanho da imagem e o número de camadas da rede neural aumentam.

Por outro lado, a Tabela 8, mostra a demanda de tempo de cada modelo, discriminando o tamanho e as características da imagem. Os valores correspondem ao tempo de treinamento com 25 épocas usando ***TPU*** e sem realizar *freeze* ou *unfreeze* do modelo. Os tempos também são expressados em **horas:minutos**.

A comparação entre as tabelas nos faz notar que mantendo o conjunto de dados, o tamanho das imagens, a característica de cor e o número de épocas fixos, não há diferença

[1] Os tempos são aproximados, pois as diferenças de horário foram observadas para o mesmo modelo/parâmetros, dependendo da hora do dia. O *Google Colab* pode alocar mais recursos na medida em que não há demanda por eles, por isso os horários podem variar de acordo com a hora do dia.

Tabla 7 – Tabela comparativa de tempos das diferentes arquiteturas utilizando GPU

Arquitetura	64x64 Cinza	64x64 Cor	128x128 Cinza	128x128 Cor	224x224 Cinza	224x224 Cor	256x256 Cinza	256x256 Cor
ResNet-18	02:18	02:02	03:01	02:00	04:20	02:19	04:19	02:46
ResNet-34	02:31	02:01	02:59	02:01	04:17	02:45	04:16	02:48
ResNet-50	02:37	02:01	02:59	02:03	04:19	02:47	04:18	02:50
ResNet-152	02:45	02:09	03:06	02:12	04:29	03:01	04:29	03:10
VGG16	02:36	02:01	03:03	02:03	04:24	02:54	04:24	03:06
DenseNet-121	02:44	02:07	03:10	02:09	04:27	02:58	04:25	03:10
DenseNet-201	02:51	02:15	03:15	02:17	04:34	03:20	04:28	03:23
AlexNet	02:36	02:01	03:02	02:01	04:17	02:43	04:10	02:49

Tabla 8 – Tabela comparativa de tempos das diferentes arquiteturas utilizando TPU

Arquitetura	64x64 Cinza	64x64 Cor	128x128 Cinza	128x128 Cor	224x224 Cinza	224x224 Cor	256x256 Cinza	256x256 Cor
ResNet-18	03:49	02:13	02:51	02:10	04:58	02:58	04:47	03:20
ResNet-34	03:38	02:20	02:45	01:53	05:30	02:34	04:26	04:23
ResNet-50	02:38	02:41	02:48	02:27	04:34	03:48	04:35	03:18
ResNet-152	03:36	02:41	04:06	02:45	05:27	04:26	04:59	03:49
VGG16	02:46	03:14	04:10	01:49	04:44	03:49	04:34	03:55
DenseNet-121	03:00	02:09	04:14	01:58	05:31	04:34	05:51	04:09
DenseNet-201	04:21	03:06	03:53	02:02	05:14	04:27	05:19	04:22
AlexNet	03:31	03:29	03:34	02:25	05:09	04:17	04:08	02:49

significativa no tempo ao usar uma instância do *Google Colab Pro* com GPU e TPU.

Quanto ao tempo total de treinamento de cada modelo (com *freeze* ou *unfreeze*), foi de aproximadamente 10 horas em média, por modelo, utilizando um ambiente acelerador de *Graphic Processing Unit* (GPU), sem observar qualquer melhoria em relação à aceleração da *Tensor Processing Unit* (TPU) fornecida pelo Google Colab Pro, conforme observado na Seção 6.1 (Tempos de Treinamento).

6.2 Acurácia dos Modelos

A acurácia é uma métrica que geralmente descreve o desempenho do modelo em todas as classes. É útil quando todas as classes têm a mesma importância e é calculada como a razão entre o número de previsões corretas e o número total de previsões. Como nosso conjunto de dados (amostras benignas e de *malware*) é balanceado, optamos por usar a ***acurácia*** como métrica para avaliar o desempenho da classificação.

Nas subseções seguintes são mostradas as Tabelas 9, 10 e 11, a primeira mostra os resultados dos experimentos quando um modelo com 25 épocas é treinado e não é

usado *unfreeze* ou *freeze*, ou seja, todos os pesos permanecem invariantes com respeito à arquitetura original. Posteriormente, experimentamos retreinar cada modelo usando as classes *freeze* e *unfreeze*. Todos os resultados são apresentados nas Tabelas 9, 10 e 11.

6.2.1 Acurácia dos modelos sem utilizar *unfreeze* ou *freeze*

Neste experimento, os modelos foram treinados com a chamada *learn.fit_one_cycle()*, que nos permite treinar facilmente uma rede usando a política de 1 ciclo de Leslie Smith (124). Essa chamada apenas descongela a camada final, que é a que realmente classifica as imagens. Na Tabela 9 podemos ver a melhor precisão de cada modelo treinado com 25 épocas sem usar *unfreeze* ou *freeze*.

Como pode ser visto, os valores em negrito destacam os melhores resultados obtidos, que correspondem a arquiteturas com mais camadas e tamanhos de imagem maiores. As redes ResNet-152, DenseNet-121 e DenseNet-201 com tamanho de imagem acima de 128x128 são as que obtiveram os melhores resultados.

Tabla 9 – Tabela comparativa das diferentes arquiteturas com sua correspondente acurácia sem usar *unfreeze* ou *freeze*

Arquitetura	64x64 Cinza	64x64 Cor	128x128 Cinza	128x128 Cor	224x224 Cinza	224x224 Cor	256x256 Cinza	256x256 Cor
ResNet-18	85,17%	86,45%	87,05%	90,12%	91,09	93,26%	89,88%	91,61%
ResNet-34	84,41%	87,97%	89,99%	88,10%	90,96%	91,37%	91,69%	92,81%
ResNet-50	86,84%	86,18%	90,36%	91,03%	90,44%	93,18%	91,48%	91,63%
ResNet-152	88,33%	87,63%	91,88%	**92,07%**	93,50%	91,15%	**94,33%**	**93,07%**
VGG16	87,27%	87,15%	88,18%	87,47%	91,42%	89,97%	91,47%	90,53%
DenseNet-121	86,91%	87,77%	**92,30%**	89,57%	93,47%	90,47%	**92,66%**	**92,69%**
DenseNet-201	89,49%	88,26%	90,87%	91,37%	**92,51%**	**94,56%**	**93,68%**	**94,55%**
AlexNet	83,09%	83,97%	83,44%	88,63%	87,04%	87,11%	84,50%	87,39%

6.2.2 Acurácia dos modelos utilizando *freeze*

Outro experimento foi feito usando a chamada *freeze* que congela todos os grupos de camadas exceto o último, isso significa que nenhum grupo de camadas exceto o último pode ser treinado. Na Tabela 10 podemos ver a melhor acurácia de cada modelo treinado com 25 épocas usando a chamada *freeze*.

Ao analisar a Tabela 10 notamos que algumas acurácias melhoraram em relação ao experimento anterior, aumentando em alguns casos em 1% , embora não seja uma melhoria substancial, é uma melhoria para tais modelos.

Tabla 10 – Tabela comparativa das diferentes arquiteturas com sua correspondente acurácia usando *freeze*

Arquitetura	64x64 Cinza	64x64 Cor	128x128 Cinza	128x128 Cor	224x224 Cinza	224x224 Cor	256x256 Cinza	256x256 Cor
ResNet-18	86,44%	86,90%	87,69%	91,24%	91,55%	**93,50%**	90,55%	**93,04%**
ResNet-34	85,62%	86,64%	91,41%	88,95%	91,14%	91,62%	92,16%	**93,28%**
ResNet-50	88,12%	87,48%	91,70%	91,33%	91,41%	**93,45%**	92,24%	92,80%
ResNet-152	89,08%	87,94%	92,88%	92,99%	**93,85%**	91,29%	**94,67%**	**93,22%**
VGG16	88,59%	88,01%	88,38%	88,70%	92,66%	91,03%	92,52%	91,60%
DenseNet-121	88,20%	89,23%	**93,06%**	90,58%	**94,08%**	91,85%	**93,73%**	92,97%
DenseNet-201	90,53%	88,63%	91,14%	92,69%	**93,00%**	**94,94%**	**94,24%**	**94,94%**
AlexNet	83,67%	84,71%	84,82%	88,83%	87,58%	87,88%	85,15%	87,85%

6.2.3 Acurácia dos modelos utilizando *unfreeze*

O último experimento foi feito usando a chamada *unfreeze* que descongela todas as camadas do modelo, então treinamos todas as camadas. Na Tabela 11 podemos ver a melhor acurácia de cada modelo treinado com 25 épocas usando a chamada *unfreeze*.

Tabla 11 – Tabela comparativa das diferentes arquiteturas com sua correspondente acurácia usando *unfreeze*

Arquitetura	64x64 Cinza	64x64 Cor	128x128 Cinza	128x128 Cor	224x224 Cinza	224x224 Cor	256x256 Cinza	256x256 Cor
ResNet-18	88,69%	88,67%	91,26%	92,25%	94,15%	95,63%	93,57%	94,15%
ResNet-34	88,83%	88,38%	92,60%	92,23%	94,52%	94,54%	94,23%	94,76%
ResNet-50	91,28%	90,03%	93,86%	93,96%	95,07%	95,44%	95,51%	95,24%
ResNet-152	91,41%	90,93%	94,25%	94,54%	95,81%	95,28%	**96,23%**	**96,12%**
VGG16	90,38%	89,39%	92,25%	92,31%	94,81%	94,25%	94,50%	94,66%
DenseNet-121	91,53%	91,53%	94,06%	94,37%	95,40%	95,49%	95,77%	95,44%
DenseNet-201	93,43%	91,96%	94,23%	94,99%	**96,08%**	**96,23%**	***96,35%***	**96,33%**
AlexNet	86,61%	86,20%	88,13%	90,17%	90,89%	91,55%	89,02%	90,56%

Esta tabela representa os melhores resultados obtidos neste trabalho, nela é apresentado o desempenho de cada arquitetura, tomando a acurácia como valor de comparação. Os valores abaixo dos trabalhos relacionados são devidos a diferença entre arquivos benignos e arquivos de *malware* ser muito superficial, como já explicado. Os arquivos benignos em nosso conjunto de dados possuem alguma função semelhante à de um *malware*.

Além disso, na tabela podemos ver que a arquitetura **DenseNet-201** obteve os melhores resultados para os tamanhos de imagem: 224x224 e 256x256, cujas precisões foram 96,26% e 96,35% respectivamente, sem denotar uma diferença importante para imagens Coloridas ou em tons de cinza. A outra arquitetura que obteve bons resultados foi **ResNet-152** obtendo 96,23% (tons de cinza) e 96,12% (Colorida) no tamanho da imagem 255x255.

6.3 Variação em Tamanho e Cor

Em relação às características das imagens relacionadas à cor, não há diferença significativa se a imagem está em tons de cinza ou em cores, e em trabalhos futuros será possível ignorar as imagens coloridas, pois o tempo de treinamento é significativamente maior e não há ganho demasiado na acurácia.

Em relação ao tamanho das imagens, como pode ser visto nos resultados expressos nas seções anteriores, há melhorias muito significativas, da ordem de 3% a 5% em alguns casos. Embora as imagens recomendadas das arquiteturas experimentadas sejam 224x224 bits, em algumas arquiteturas obtivemos melhores resultados com tamanhos 255x255 bits, não sendo possível experimentar tamanhos maiores devido ao alto consumo de memória dos modelos.

Nas Figuras 29, 30, 31 e 32 podemos ver a melhor acurácia de cada modelo usando a chamada *unfreeze* e com 25 épocas de treinamento.

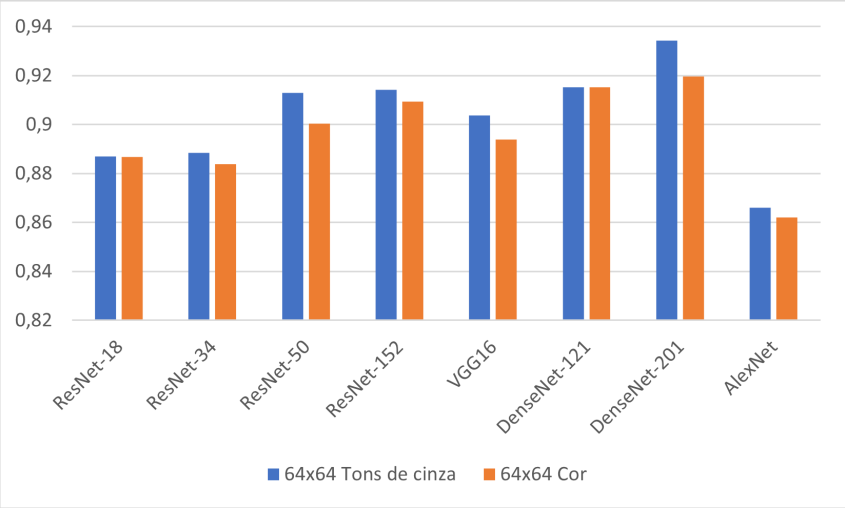

Figura 29 – Acurácia dos modelos para imagens de 64x64 bits, em tons de cinza e coloridas.

6.4 Comparação com Trabalhos Relacionados

Embora não seja possível fazer uma comparação direta da medida de desempenho com os trabalhos relacionados, pois são utilizados conjuntos de dados diferentes, podemos tomar esses valores como uma base. A diferença de acurácia com relação aos trabalhos relacionados se deve, em grande parte, ao fato de que a maioria dos trabalhos analisados utiliza redes neurais pré-treinadas em combinação com outros métodos, tornando-os mais eficientes, enquanto outros trabalhos utilizam redes neurais criadas “a partir de zero” para abordar o problema.

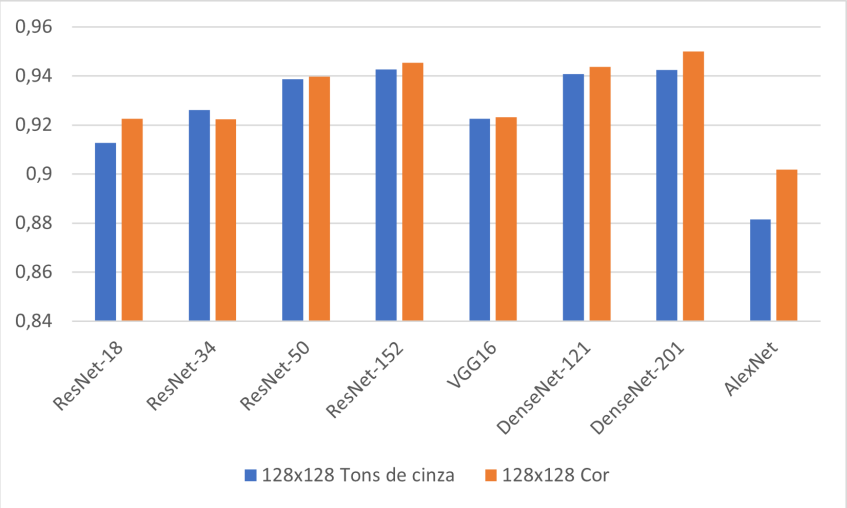

Figura 30 – Acurácia dos modelos para imagens de 128x128 bits, em tons de cinza e coloridas.

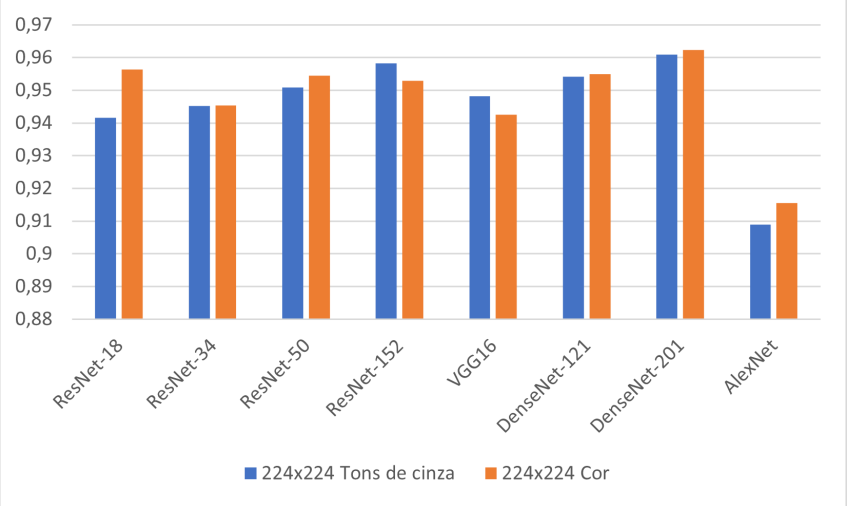

Figura 31 – Acurácia dos modelos para imagens de 224x224 bits, em tons de cinza e Coloridas.

Além disso, como já dito, ao usar um conjunto de dados diferente dos trabalhos relacionados, e tendo em vista que as amostras benignas deste trabalho apresentam alguma função semelhante à do *malware*, a acurácia das arquiteturas avaliadas tem um desempenho inferior em relação aos trabalhos investigados. Pode-se observar que comparado ao trabalho feito por Roseline et al.(100) que foi de 99,97%, há uma diferença de 3,62%; Por outro lado, ao comparar com o trabalho de Naeem et al.(112) cuja acurácia foi de 98,4%, podemos notar uma diferença de 2,05%. Embora a ideia não seja comparar uma vez que os outros trabalhos utilizam outro conjunto de dados diferentes, é feita uma comparação para se ter uma ideia comparativa genérica.

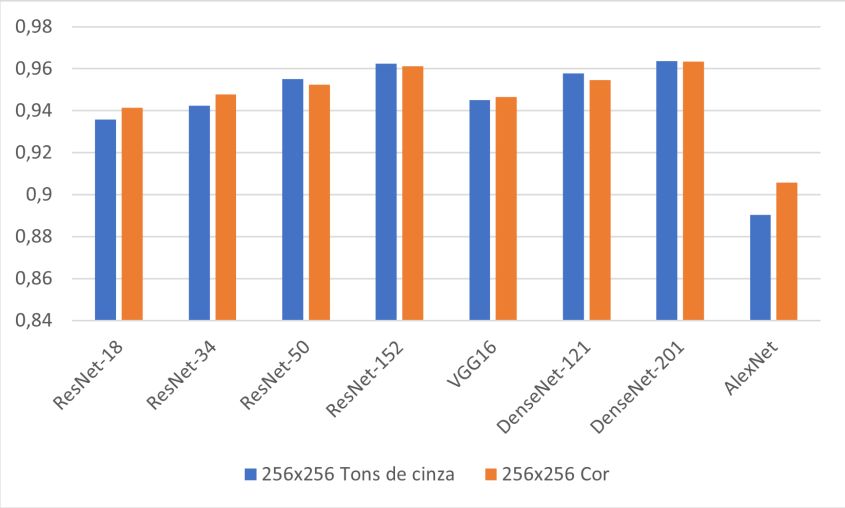

Figura 32 – Acurácia dos modelos para imagens de 256x256 bits, em tons de cinza e Coloridas.

Na Tabela 12 podemos ver a diferença de acurácias entre os trabalhos relacionados.

Tabela 12 – Tabela comparativa das acurácias dos trabalhos relacionados com o presente trabalho

Autor	Classificador	Arquitetura	Acurácia	Diferença
Hemalatha et al.(6)	Deep Learning	DenseNet	97,5%	1,20 p.p.
Roseline et al.(100)	*Deep Random Forest*	Sliding Window	98,65%	2,30 p.p.
Moussas e Andreatos(7)	ANN	Scratch model	99,14%	2,79 p.p.
Nisa et al.(8)	DCNN, SFTA	AlexNet	99,30%	2,95% p.p.
Roseline et al.(109)	Lightweight CNN	Scratch model	97,68%	1,33 p.p.
Gibert et al.(101)	CNN	Scratch model	98,40%	2,05 p.p.
Vasan et al.(103)	CNN	VGG16, ResNet50	98,82%	2,47 p.p.
Saadat e Raymond(106)	CNN	XG-Boost	97,70%	1,35 p.p.
Xiao et al.(105)	CNN	Scratch model	99,64%	3,29 p.p.
Naeem et al.(111)	DCNN	Scratch model	98,79%	2,44 p.p.
Vasan et al.(107)	CNN	VGG16, ResNet50	98,11%	1,76 p.p.
Roseline et al.(113)	ML	RF, Xgboost, ETC, LR	98,91%	2,56 p.p.
Ouahab et al.(110)	ML	KNN	97,00%	0,65 p.p.
Naeem et al.(112)	ML	LGMP	98,00%	1,65 p.p.

Uma abordagem baseada em visualização foi adotada no trabalho de Hemalatha et al.(6), onde os binários de malware foram representados como imagens bidimensionais e classificados usando um modelo de aprendizado profundo. O sistema usa uma função de perda balanceada de classe reponderada na camada de classificação final do modelo DenseNet para obter melhorias significativas de desempenho na classificação de malware ao lidar com problemas de dados desequilibrados.

7 CONCLUSÃO

A tecnologia é qualquer solução, desenvolvimento ou conhecimento que facilite a vida em sociedade. No último meio século, os avanços tecnológicos foram tão abrangentes que até mudaram a maneira como vivemos, comunicamos e nos relacionamos uns com os outros. Neste sentido, a tecnologia da informação trouxe grandes vantagens para o desenvolvimento social, mas também desvantagens que são expressas individual e coletivamente.

Embora não possamos negar o impacto positivo da tecnologia da informação no desenvolvimento coletivo, ela também trouxe desvantagens que afetam os indivíduos. Uma das desvantagens do impacto da computação no desenvolvimento coletivo e individual é o *malware*. A extensão dos danos causados por *malware* muitas vezes depende se ele infectou um computador doméstico ou uma rede corporativa. As consequências dos danos também podem variar de acordo com o tipo específico de *malware* e o tipo de dispositivo infectado, assim como a natureza dos dados armazenados ou acessados pelo dispositivo. Atualmente, a análise automatizada de *malware* é um componente crítico da postura de segurança de uma organização. Ao automatizar a análise de *malware*, as organizações podem identificar e responder às ameaças de forma mais rápida e eficaz. A automação da análise de *malware* tem muitas vantagens pois, primeiro, permite que os analistas de segurança se concentrem em tarefas mais urgentes. Em segundo lugar, ela acelera o processo de análise, que é crítico quando o tempo é essencial. Finalmente, ela pode ajudar a identificar *malwares* desconhecidos anteriormente, e aqui reside a importância de estudar técnicas de detecção que estejam tão próximas da realidade quanto possível.

Outro ponto importante ao realizar uma análise de *malware* é a seleção de amostras, neste trabalho procuramos otimizar a seleção de amostras que um analista humano deve analisar utilizando amostras de arquivos benignos que tenham alguma funcionalidade similar a *malwares*, em vez de utilizar amostras de arquivos benignos de programas conhecidos ou arquivos do próprio computador, encurtando assim a lacuna entre arquivos maliciosos e benignos, isto causa uma diminuição nas diferenças visuais entre uma imagem *malware* e uma imagem de arquivo benigna.

Neste trabalho, propusemos utilizar o aprendizado por transferência de visão computacional para a classificação estática de *malware*. Utilizando três conjuntos de dados, apresentamos que, embora o desempenho não fosse superior, a técnica proposta é muito eficaz em comparação com o treinamento “do zero” e outros algoritmos clássicos de aprendizado de máquinas.

O objetivo geral definido para este trabalho foi propor o uso de redes neurais pré-treinados para a classificação de arquivos maliciosos, através de um estudo comparativo

entre várias redes pré-treinadas, utilizando um conjunto de dados de arquivos benignos com algum comportamento ou função anômala, testando se o desempenho de tais classificadores é comparável ao dos classificadores criados especificamente para o problema. Como objetivos específicos, foi proposto criar um **benchmark** utilizando uma base de dados de arquivos maliciosos e benignos representativos da tarefa a ser modelada, e realizar uma análise de sensibilidade das redes neurais testadas, verificando seu desempenho e sua reação a alterações em seus parâmetros.

Para alcançar estes objetivos, foi criada uma base de dados, com aproximadamente 15.000 amostras de arquivos benignos que têm algum comportamento ou função similar ao *malware*, isto foi possível graças à doação destas amostras pela empresa *ESET*, que doou todas as amostras benignas. Por outro lado, foram criados 8 modelos com diferentes redes pré-treinadas e experimentamos imagens coloridas e em escala de cinza, além disso experimentamos 4 tamanhos de imagem, a saber: 64x64, 128x128, 224x224 e 255x255 bits. Dando um total de 64 experimentos realizados nos quais pode-se ver que o melhor desempenho **96,35%** foi alcançada com a arquitetura **DenseNet-201** para imagens em escala de cinza e tamanho de entrada de 256x256; a arquitetura **ResNet-152** também apresentou desempenhos similares, **96,23%** para imagens em escala de cinza de tamanho 256x256.

Embora não seja possível comparar diretamente a precisão com o trabalho relacionado porque são utilizados conjuntos de dados diferentes, podemos tomar estes valores como referência. A diferença na precisão com trabalhos relacionados se deve em grande parte ao fato de que a maioria dos trabalhos analisados utiliza redes neurais pré-treinados em combinação com outros métodos, tornando-os mais eficientes, enquanto outros trabalhos utilizam redes neurais criadas "do zero" para resolver o problema. Estes dados experimentais mostram que é possível usar redes neurais pré-treinadas e transferir aprendizado para identificar amostras de *malware*, como era o objetivo geral deste trabalho.

Outra contribuição deste trabalho foi que permitiu a criação de uma base de dados *benchmark* com arquivos maliciosos e benignos representativos da tarefa a ser modelada. Além disso, foi construído um base de dados de arquivos benignos com funções semelhantes às dos arquivos maliciosos, o que permite preencher a lacuna de identificação. Finalmente, foi confirmada a viabilidade do uso de redes neurais pré-treinadas e da transferência de aprendizado para identificar arquivos maliciosos usando um conjunto de dados quase homogêneo com relação às funcionalidades de amostras maliciosas e benignas.

Outro ponto importante é o tempo gasto. Enquanto autores como Halder e Ozdemir(125) ou Parisi(126) afirmam que o tempo gasto para treinar uma rede neural "do zero" pode variar de horas ou dias a semanas, nos experimentos realizados, pode-se ver que os modelos que demoraram mais tempo neste trabalho foram cerca de 4 horas, para um único modelo com uma única característica de imagem (cor ou escala de cinza) e para

um único tamanho de imagem (64x64, 128x128, 224x224 ou 255x255 bits).

Uma particularidade que pode ser observada nos experimentos é que as redes com camadas mais ocultas (ResNet-152 e DenseNet-201) são as que produziram melhores resultados de acurácia, deixando para o trabalho futuro testar outras arquiteturas com mais camadas ocultas. Além das camadas ocultas, pode-se ver que o tamanho da imagem de entrada de 224x224 e 256x256 dá os melhores resultados para todas as arquiteturas, deixando para trabalhos futuros a experimentação de tamanhos de imagem maiores, por exemplo, 512x512.

REFERÊNCIAS

1 GANDOTRA, E.; BANSAL, D.; SOFAT, S. Malware analysis and classification: A survey. *Journal of Information Security*, Scientific Research Publishing, v. 2014, 2014.

2 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.

3 AV-TEST. *Malware | AV-Test*. 2022. <https://www.av-test.org/es/estadisticas/software-malicioso/>. Acessado em 07/08/2022.

4 ALMOMANI, I.; KHAYER, A. Android applications scanning: The guide. In: *2019 International Conference on Computer and Information Sciences (ICCIS)*. [S.l.: s.n.], 2019. p. 1–5.

5 CHAKKARAVARTHY, S. S.; SANGEETHA, D.; VAIDEHI, V. A survey on malware analysis and mitigation techniques. *Computer Science Review*, Elsevier, v. 32, p. 1–23, 2019.

6 HEMALATHA, J.; ROSELINE, S. A.; GEETHA, S.; KADRY, S.; DAMAŠEVIČIUS, R. An efficient densenet-based deep learning model for malware detection. *Entropy*, MDPI, v. 23, n. 3, p. 344, 2021.

7 MOUSSAS, V.; ANDREATOS, A. Malware detection based on code visualization and two-level classification. *Information*, MDPI, v. 12, n. 3, p. 118, 2021.

8 NISA, M.; SHAH, J. H.; KANWAL, S.; RAZA, M.; KHAN, M. A.; DAMAŠEVIČIUS, R.; BLAŽAUSKAS, T. Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features. *Applied Sciences*, MDPI, v. 10, n. 14, p. 4966, 2020.

9 IBM CORPORATION. *Informe sobre el coste de una brecha de seguridad en los datos de 2021*. 2021. <https://www.ibm.com/es-es/security/data-breach>. Acessado em 14/02/2022.

10 SIKORSKI, M.; HONIG, A. *Practical malware analysis : the hands-on guide to dissecting malicious software*. [S.l.]: San Francisco: No Starch Press, 2012.

11 ANDERSON, B. Automating reverse engineering with machine learning techniques. In: *ACM AISec*. [S.l.]: Elsevier, 2014. p. 103–112.

12 Romero, O. E.; Kim, J.-H.; Bárcena, M. A.; Hall, I. R.; Zahn, R.; Schneider, R. R. Alkenone concentration and sea surface temperature of sediment core MD02-2588. In: . PANGAEA, 2018. In supplement to: Romero, OE et al. (2015): High-latitude forcing of diatom productivity in the southern Agulhas Plateau during the past 350 kyr. Paleoceanography, 30(2), 118-132, https://doi.org/10.1002/2014PA002636. Disponível em: <https://doi.org/10.1594/PANGAEA.895560>.

13 UGARTE-PEDRERO, X.; GRAZIANO, M.; BALZAROTTI, D. A close look at a daily dataset of malware samples. *ACM Transactions on Privacy and Security (TOPS)*, ACM New York, NY, USA, v. 22, n. 1, p. 1–30, 2019.

14 MCAFEE. *The Economic Impact of Cybercrime-No Slowing Down.* 2018. <https://www.mcafee.com/enterprise/en-us/assets/executive-summaries/es-economic-impact-cybercrime.pdf>. Acessado em 14/02/2022.

15 LANGNER, R. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, v. 9, p. 49–51, 2011.

16 MINISTERIO DE DEFENSA, PRESIDENCIA DE LA NACIóN Y REPúBLICA ARGENTINA. *Libro Blanco de la Defensa. Argentina: Ministerio de Defensa.* 2015. <http://www.mindef.gob.ar/institucional/pdfs/libro_blanco_2015.pdf>. Acessado em 7/08/2022.

17 GUERRERO-SAADE, J. A. The ethics and perils of apt research: an unexpected transition into intelligence brokerage. In: . [S.l.: s.n.], 2015.

18 SHABTAI, A.; MENAHEM, E.; ELOVICI, Y. F-sign: Automatic, function-based signature generation for malware. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 41, p. 494–508, 2011.

19 SHAUKAT, K.; LUO, S.; VARADHARAJAN, V.; HAMEED, I. A.; XU, M. A survey on machine learning techniques for cyber security in the last decade. *IEEE Access*, v. 8, p. 222310–222354, 2020.

20 SHYAMASUNDAR, R. K.; SHAH, H.; KUMAR, N. V. N. Malware: From modelling to practical detection. In: JANOWSKI, T.; MOHANTY, H. (Ed.). *Distributed Computing and Internet Technology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 21–39. ISBN 978-3-642-11659-9.

21 COHEN, F. Computer viruses: theory and experiments. *Computers & security*, Elsevier, v. 6, n. 1, p. 22–35, 1987.

22 ADLEMAN, L. M. An abstract theory of computer viruses. In: GOLDWASSER, S. (Ed.). *Advances in Cryptology — CRYPTO' 88*. New York, NY: Springer New York, 1990. p. 354–374. ISBN 978-0-387-34799-8.

23 MCGRAW, G.; MORRISETT, G. Attacking malicious code: A report to the infosec research council. *IEEE Software*, v. 17, n. 5, p. 33–41, 2000.

24 SKOUDIS, E.; ZELTSER, L. *Malware: Fighting Malicious Code*. Prentice Hall PTR, 2004. (Prentice Hall series in computer networking and distributed systems). ISBN 9780131014053. Disponível em: <https://books.google.com.br/books?id=TKEAQmQV7O4C>.

25 PARKER, S. *McGraw-Hill Dictionary of Scientific and Technical Terms*. McGraw-Hill, 1993. Disponível em: <https://books.google.com.br/books?id=GBXtngEACAAJ>.

26 ENISA. *Reference Incident Classification Taxonomy*. 2018. <https://www.enisa.europa.eu/publications/reference-incident-classification-taxonomy>. Acessado em 18/02/2022.

27 MISP PROJECT. *MISP Taxonomies and classification as machine tags*. 2022. <https://www.misp-project.org/taxonomies.pdf>. Acessado em 18/02/2022.

28 VINOD, P.; JAIPUR, R.; LAXMI, V.; GAUR, M. Survey on malware detection methods. In: *Proceedings of the 3rd Hackers' Workshop on computer and internet security (IITKHACK'09)*. [S.l.: s.n.], 2009. p. 74–79.

29 FIRDAUSI, I.; ERWIN, A.; NUGROHO, A. S. et al. Analysis of machine learning techniques used in behavior-based malware detection. In: IEEE. *2010 second international conference on advances in computing, control, and telecommunication technologies*. [S.l.], 2010. p. 201–203.

30 MOSER, A.; KRUEGEL, C.; KIRDA, E. Limits of static analysis for malware detection. In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. [S.l.: s.n.], 2007. p. 421–430.

31 HIGUERA, J. B.; ARAMBURU, C. A.; HIGUERA, J.-R. B.; URBAN, M. A. S.; MONTALVO, J. A. S. Systematic approach to malware analysis (sama). *Applied Sciences*, v. 10, n. 4, 2020. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/10/4/1360>.

32 BAYER, U.; MOSER, A.; KRUEGEL, C.; KIRDA, E. Dynamic analysis of malicious code. *Journal in Computer Virology*, v. 2, p. 67–77, 08 2006.

33 JAIN, M.; BAJAJ, P. Techniques in detection and analyzing malware executables: a review. *International Journal of Computer Science and Mobile Computing*, v. 3, n. 5, p. 930–935, 2014.

34 EILAM, E. *Reversing: Secrets of Reverse Engineering*. Wiley, 2008. ISBN 9780470326763. Disponível em: <https://books.google.co.cr/books?id=K8vIf99UBYQC>.

35 GUILLOT, P. Auguste kerckhoffs et la cryptographie militaire. *Bibnum. Textes fondateurs de la science*, FMSH-Fondation Maison des sciences de l'homme, 2013.

36 YOU, I.; YIM, K. Malware obfuscation techniques: A brief survey. In: *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*. [S.l.: s.n.], 2010. p. 297–300.

37 CHRISTODORESCU, M.; JHA, S. Static analysis of executables to detect malicious patterns. In: *12th USENIX Security Symposium (USENIX Security 03)*. [S.l.: s.n.], 2003.

38 BRANCO, R. R.; BARBOSA, G. N.; NETO, P. D. *Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and AntiVM Technologies.* 2012. <https://kernelhacking.com/rodrigo/docs/blackhat2012-paper.pdf>. Acessado em 18/02/2022.

39 CANI, A.; GAUDESI, M.; SANCHEZ, E.; SQUILLERO, G.; TONDA, A. Towards automated malware creation: Code generation and code integration. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2014. (SAC '14), p. 157–160. ISBN 9781450324694. Disponível em: <https://doi.org/10.1145/2554850.2555157>.

40 MITCHELL, T. M. *Machine learning*. [S.l.]: McGraw-hill New York, 1997. v. 1.

41 NILSSON, N. J. *Introduction to Machine Learning.* 1998. <https://ai.stanford.edu/~nilsson/MLBOOK.pdf>. Acessado em 25/02/2022.

42 LU, F.; BAI, Q. A refined weighted k-nearest neighbors algorithm for text categorization. *Proceedings of 2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering, ISKE 2010*, 11 2010.

43 CHAPELLE, O.; SCHOLKOPF, B.; ZIEN, A. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, IEEE, v. 20, n. 3, p. 542–542, 2009.

44 WANJUN, Y.; XIAOGUANG, S. Research on text categorization based on machine learning. In: IEEE. *2010 IEEE International Conference on Advanced Management Science (ICAMS 2010)*. [S.l.], 2010. v. 2, p. 253–255.

45 CHEN, H.; CHAU, M. Web mining: Machine learning for web. *Annual Review of Information Science and Technology 2004*, Information Today, Inc., v. 38, p. 289, 2003.

46 FOGEL, L. J.; OWENS, A. J.; WALSH, M. J. Intelligent decision making through a simulation of evolution. *Behavioral science*, Wiley Online Library, v. 11, n. 4, p. 253–272, 1966.

47 DíEZ, R.; GóMEZ, A.; MARTíNEZ, N. de A. *Introducción a la inteligencia artificial: sistemas expertos, redes neuronales artificiales y computación evolutiva*. Servicio de Publicaciones, Universidad de Oviedo, 2001. ISBN 9788483172490. Disponível em: <https://books.google.com.py/books?id=RKqLMCw3IUkC>.

48 BURKOV, A. *The Hundred-page Machine Learning Book*. Andriy Burkov, 2019. ISBN 9781999579500. Disponível em: <https://books.google.com.br/books?id=ZF3KwQEACAAJ>.

49 MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, v. 5, n. 4, p. 115–133, 1943.

50 ZHANG, L.; ZHANG, B. A geometrical representation of mcculloch-pitts neural model and its applications. *IEEE Transactions on Neural Networks*, v. 10, n. 4, p. 925–929, 1999.

51 HEBB, D. O. *The organization of behavior: A neuropsychological theory*. [S.l.]: Psychology Press, 2005.

52 WIDROW, B.; HOFF, M. E. *Adaptive switching circuits*. [S.l.], 1960.

53 SORIA, E.; BLANCO, A. *Redes neuronales artificiales*. [S.l.: s.n.], 2001. 25-33 p.

54 CUN, Y. L.; JACKEL, L. D.; BOSER, B.; DENKER, J. S.; GRAF, H. P.; GUYON, I.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, IEEE, v. 27, n. 11, p. 41–46, 1989.

55 KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, v. 25, 2012.

56 KURENKOV, A.; DUTTAGUPTA, S.; ZHANG, C.; FUKAMI, S.; HORIO, Y.; OHNO, H. Artificial neuron and synapse realized in an antiferromagnet/ferromagnet heterostructure using dynamics of spin–orbit torque switching. *Advanced Materials*, v. 31, n. 23, p. 1900636, 2019. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.201900636>.

57 KARIć, A.; KONJIC, T.; JAHIć, A.; JAHIć, M. Power system fault detection, classification and location using artificial neural networks. In: . [S.l.: s.n.], 2017.

58 ABDELRAZEK, A.; MOHAMED, E.; EL-FITIANY, F.; ABOUROHIEM, M. Reducing energy consumption in pressurized irrigation networks using neural networks-soms clustering technique. In: . [S.l.: s.n.], 2020.

59 CARLUCCI, S.; GAGLIANO, A.; BABORSKA-NAROZNY, M.; PITTAM, J.; MOSCHOU, C.; ESLAND, R.; DINH, D.-L.; RODRIGUES, F.; COSGROVE, J.; GRUBER, J.; MCGILL, G.; LITTLEWOOD, J. *Smart Energy Control Systems for Sustainable Buildings*. [S.l.: s.n.], 2017. v. 67. ISBN 978-3-319-52074-2.

60 HOWARD, J.; GUGGER, S. *Deep Learning for Coders with fastai and PyTorch*. O'Reilly Media, 2020. ISBN 9781492045496. Disponível em: <https://books.google.com.br/books?id=yATuDwAAQBAJ>.

61 CARELLI, A.; SILANI, S.; STELLA, F. Profiling neural networks for option pricing. *International Journal of Theoretical and Applied Finance*, v. 3, 04 2000.

62 WANG, H.; RAJ, B. A survey: Time travel in deep learning space: An introduction to deep learning models and how deep learning models evolved from the initial ideas. 10 2015.

63 GRUS, J. *Data Science from Scratch: First Principles with Python*. O'Reilly Media, 2019. ISBN 9781492041108. Disponível em: <https://books.google.com.br/books?id=YBKSDwAAQBAJ>.

64 KIMURA, N.; YOSHINAGA, I.; SEKIJIMA, K.; AZECHI, I.; BABA, D. Convolutional neural network coupled with a transfer-learning approach for time-series flood predictions. *Water*, v. 12, p. 96, 12 2019.

65 YANI, M.; IRAWAN, S.; SETIANINGSIH, C. Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, v. 1201, p. 012052, 05 2019.

66 ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)*. [S.l.: s.n.], 2017. p. 1–6.

67 A, T. L.; J, S. Policy transfer via markov logic networks. *International Conference on Inductive Logic Programming*, 2009.

68 RASHID, T. *Make Your Own Neural Network*. 1st. ed. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2016. ISBN 1530826608.

69 STEPHENSON, T. An introduction to bayesian network theory and usage. *Machine Learning*, 01 2000.

70 RICHARDSON, M.; DOMINGOS, P. Markov logic networks. *Machine Learning*, January 2006. Disponível em: <https://www.microsoft.com/en-us/research/publication/markov-logic-networks/>.

71 TENSORFLOW. *TensorFlow*. 2021. <https://www.tensorflow.org/js/tutorials/transfer/what_is_transfer_learning?hl=en>. (Acessado em 25/02/2022).

72 SARKAR, D. A comprehensive hands-on guide to transfer learning with real-world applications in deep learning. *Towards data science*, 2018.

73 PAN, S. J.; YANG, Q. A survey on transfer learning. In: *IEEE Transactions on Knowledge and Data Engineering*. [S.l.]: http://www-edlab.cs.umass.edu/cs689/reading/transfer-learning.pdf, 2010. v. 22.

74 IMAGENET. *ImageNet*. 2021. <https://image-net.org/>. Acessado em 22/02/2022.

75 AL-FALLUJI, R. A.; KATHEETH, Z. D.; ALATHARI, B. Automatic detection of covid-19 using chest x-ray images and modified resnet18-based convolution neural networks. *Computers, Materials, & Continua*, p. 1301–1313, 2021.

76 KUMAR, V.; ARORA, H.; HARSH; SISODIA, J. Resnet-based approach for detection and classification of plant leaf diseases. In: *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*. [S.l.: s.n.], 2020. p. 495–502.

77 DAS, P. P.; ACHARJEE, A.; MARIUM-E-JANNAT. Double coated vgg16 architecture: An enhanced approach for genre classification of spectrographic representation of musical pieces. In: *2019 22nd International Conference on Computer and Information Technology (ICCIT)*. [S.l.: s.n.], 2019. p. 1–5.

78 RYBIAŁEK, A.; JELEŃ, Ł. Application of densenets for classification of breast cancer mammograms. In: SPRINGER. *International Conference on Computer Information Systems and Industrial Management*. [S.l.], 2020. p. 266–277.

79 ALAWI, A. E. B.; MOSLEH, M. A. A.; ALMOHAGRY, Z.; SAEED, A. Y. A. Parasitized cell recognition using alexnet pre-trained model. In: *2021 1st International Conference on Emerging Smart Technologies and Applications (eSmarTA)*. [S.l.: s.n.], 2021. p. 1–4.

80 TARG, S.; ALMEIDA, D.; LYMAN, K. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

81 SRINIVAS, A.; LIN, T.-Y.; PARMAR, N.; SHLENS, J.; ABBEEL, P.; VASWANI, A. Bottleneck transformers for visual recognition. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. [S.l.: s.n.], 2021. p. 16519–16529.

82 MAHMOOD, A.; GIRALDO, A.; BENNAMOUN, M.; AN, S.; SOHEL, F.; BOUSSAID, F.; HOVEY, R.; FISHER, R.; KENDRICK, G. Automatic hierarchical classification of kelps using deep residual features. *Sensors*, v. 20, p. 447, 01 2020.

83 SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

84 IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: PMLR. *International conference on machine learning*. [S.l.], 2015. p. 448–456.

85 FERGUSON, M.; AK, R.; LEE, Y.-T. T.; LAW, K. H. Automatic localization of casting defects with convolutional neural networks. In: IEEE. *2017 IEEE international conference on big data (big data)*. [S.l.], 2017. p. 1726–1735.

86 HUANG, G.; LIU, S.; MAATEN, L. Van der; WEINBERGER, K. Q. Condensenet: An efficient densenet using learned group convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2018. p. 2752–2761.

87 HUANG, G.; LIU, Z.; MAATEN, L. V. D.; WEINBERGER, K. Q. Densely connected convolutional networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 4700–4708.

88 RADWAN, N. Leveraging sparse and dense features for reliable state estimation in urban environments. 06 2019.

89 DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: IEEE. *2009 IEEE conference on computer vision and pattern recognition*. [S.l.], 2009. p. 248–255.

90 STRISCIUGLIO, N.; ANTEQUERA, M. L.; PETKOV, N. Enhanced robustness of convolutional networks with a push–pull inhibition layer. *Neural Computing and Applications*, v. 32, p. 1–15, 12 2020.

91 TAJBAKHSH, N.; SHIN, J. Y.; GURUDU, S. R.; HURST, R. T.; KENDALL, C. B.; GOTWAY, M. B.; LIANG, J. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, IEEE, v. 35, n. 5, p. 1299–1312, 2016.

92 IZADYYAZDANABADI, M.; BELYKH, E.; MOONEY, M.; MARTIROSYAN, N.; ESCHBACHER, J.; NAKAJI, P.; PREUL, M. C.; YANG, Y. Convolutional neural networks: Ensemble modeling, fine-tuning and unsupervised semantic localization for neurosurgical cle images. *Journal of Visual Communication and Image Representation*, Elsevier, v. 54, p. 10–20, 2018.

93 YANG, Y.; DU, X.; YANG, Z.; LIU, X. Android malware detection based on structural features of the function call graph. In: *Electronics*. [S.l.]: CrossRef, 2021. v. 10, p. 186.

94 NAR, M.; KAKISIM, A. G.; YAVUZ, M. N.; SOGUKPINAR, I. Analysis and comparison of disassemblers for opcode based malware analysis. In: *2019 4th International Conference on Computer Science and Engineering (UBMK)*. [S.l.: s.n.], 2019. p. 17–22.

95 ALSOGHYER, S.; ALMOMANI, I. Ransomware detection system for android applications. In: *Electronics*. [S.l.]: CrossRef, 2019. v. 8, p. 868.

96 FARIS, H.; HABIB, M.; ALMOMANI, I.; ESHTAY, M.; ALJARAH, I. Optimizing extreme learning machines using chains of salps for efficient android ransomware detection. *Applied Sciences*, v. 10, n. 11, 2020. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/10/11/3706>.

97 JEON, J.; KIM, J.; JEON, S.; LEE, S.; JEONG, Y. Static analysis for malware detection with tensorflow and gpu. In: *Advances in Computer Science and Ubiquitous Computing*. [S.l.]: Springer, 2021. p. 537–546.

98 MOHAISEN, A.; ALRAWI, O.; MOHAISEN, M. Amal: high-fidelity, behavior-based automated malware analysis and classification. *computers & security*, Elsevier, v. 52, p. 251–266, 2015.

99 SIHWAIL, R.; OMAR, K.; ARIFFIN, K. A. Z.; AFGHANI, S. A. Malware detection approach based on artifacts in memory image and dynamic analysis. *Applied Sciences*, MDPI, v. 9, n. 18, p. 3680, 2019.

100 ROSELINE, S. A.; GEETHA, S.; KADRY, S.; NAM, Y. Intelligent vision-based malware detection and classification using deep random forest paradigm. *IEEE Access*, IEEE, v. 8, p. 206303–206324, 2020.

101 GIBERT, D.; MATEU, C.; PLANES, J.; VICENS, R. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, Springer, v. 15, n. 1, p. 15–28, 2019.

102 JANG, S.; LI, S.; SUNG, Y. Fasttext-based local feature visualization algorithm for merged image-based malware classification framework for cyber security and cyber defense. *Mathematics*, v. 8, p. 460, 03 2020.

103 VASAN, D.; ALAZAB, M.; WASSAN, S.; NAEEM, H.; SAFAEI, B.; ZHENG, Q. Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks*, v. 171, p. 107138, 2020. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128619304736>.

104 NATARAJ, L.; KARTHIKEYAN, S.; JACOB, G.; MANJUNATH, B. S. Malware images: Visualization and automatic classification. In: *Proceedings of the 8th International Symposium on Visualization for Cyber Security*. New York, NY, USA: Association for Computing Machinery, 2011. (VizSec '11). ISBN 9781450306799. Disponível em: <https://doi.org/10.1145/2016904.2016908>.

105 XIAO, G.; LI, J.; CHEN, Y.; LI, K. Malfcs: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. *Journal of Parallel and Distributed Computing*, v. 141, 04 2020.

106 SAADAT, S.; RAYMOND, V. J. Malware classification using cnn-xgboost model. In: *Artificial Intelligence Techniques for Advanced Computing Applications*. [S.l.]: Springer, 2021. p. 191–202.

107 VASAN, D.; ALAZAB, M.; WASSAN, S.; SAFAEI, B.; ZHENG, Q. Image-based malware classification using ensemble of cnn architectures (imcec). *Computers & Security*, Elsevier, v. 92, p. 101748, 2020.

108 NARAYANAN, B. N.; DAVULURU, V. S. P. Ensemble malware classification system using deep neural networks. *Electronics*, MDPI, v. 9, n. 5, p. 721, 2020.

109 ROSELINE, S. A.; SASISRI, A. D.; GEETHA, S.; BALASUBRAMANIAN, C. Towards efficient malware detection and classification using multilayered random forest ensemble technique. In: *2019 International Carnahan Conference on Security Technology (ICCST)*. [S.l.: s.n.], 2019. p. 1–6.

110 OUAHAB, I. B. A.; BOUHORMA, M.; BOUDHIR, A. A.; AACHAK, L. E. Classification of grayscale malware images using the k-nearest neighbor algorithm. In: SPRINGER. *The Proceedings of the Third International Conference on Smart City Applications*. [S.l.], 2019. p. 1038–1050.

111 NAEEM, H.; ULLAH, F.; NAEEM, M. R.; KHALID, S.; VASAN, D.; JABBAR, S.; SAEED, S. Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. *Ad Hoc Networks*, Elsevier, v. 105, p. 102154, 2020.

112 NAEEM, H.; GUO, B.; NAEEM, M. R.; ULLAH, F.; ALDABBAS, H.; JAVED, M. S. Identification of malicious code variants based on image visualization. *Computers & Electrical Engineering*, Elsevier, v. 76, p. 225–237, 2019.

113 ROSELINE, S. A.; HARI, G.; GEETHA, S.; KRISHNAMURTHY, R. Vision-based malware detection and classification using lightweight deep learning paradigm. In: SPRINGER. *International Conference on Computer Vision and Image Processing*. [S.l.], 2019. p. 62–73.

114 VIRUSSHARE. *VirusShare.com - Because Sharing is Caring*. 2022. <https://virusshare.com/>. Acessado em 03/02/2022.

115 VIRUSSIGN. *VirusSign | Malware Research & Data Center, Threat Intelligence*. 2022. <https://www.virussign.com/>. Acessado em 03/02/2022.

116 ESET. *Soluciones antivirus y de seguridad de Internet*. 2022. <https://www.eset.com/ar/>. Acessado em 03/02/2022.

117 VIRUSTOTAL. *VirusTotal*. 2022. <https://www.virustotal.com/>. Acessado em 03/02/2022.

118 NATARAJ, L.; KARTHIKEYAN, S.; JACOB, G.; MANJUNATH, B. S. Malware images: Visualization and automatic classification. In: *Proceedings of the 8th International Symposium on Visualization for Cyber Security*. New York, NY, USA: Association for Computing Machinery, 2011. (VizSec '11). ISBN 9781450306799. Disponível em: <https://doi.org/10.1145/2016904.2016908>.

119 MASTYŁO, M. Bilinear interpolation theorems and applications. *Journal of Functional Analysis*, v. 265, p. 185–207, 07 2013.

120 CAO, C.; WANG, B.; ZHANG, W.; ZENG, X.; YAN, X.; FENG, Z.; LIU, Y.; WU, Z. An improved faster r-cnn for small object detection. *IEEE Access*, v. 7, p. 1–1, 08 2019.

121 FAST.AI. *fast.ai - Making neural nets uncool again*. 2022. <https://www.fast.ai/about/>. Acessado em 03/02/2022.

122 GOOGLE. *Google Colab*. 2021. <https://research.google.com/colaboratory/faq.html>. Acessado em 2/12/2021.

123 HE, K. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *IEEE ICCV*. [S.l.]: Elsevier, 2015. p. 1026–1034.

124 SMITH, L. N. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

125 HALDER, S.; OZDEMIR, S. *Hands-On Machine Learning for Cybersecurity: Safeguard your system by making your machines intelligent using the Python ecosystem*. Packt Publishing, 2018. ISBN 9781788990967. Disponível em: <https://books.google.com.br/books?id=LR2CDwAAQBAJ>.

126 PARISI, A. *Hands-On Artificial Intelligence for Cybersecurity: Implement smart AI systems for preventing cyber attacks and detecting threats and network anomalies*. Packt Publishing, 2019. ISBN 9781789805178. Disponível em: <https://books.google.com.br/books?id=7GOnDwAAQBAJ>.

APÊNDICE A – IMPLEMENTAÇÃO PYTHON DE INTERPOLAÇÃO DE IMAGEM BILINEAR EM ESCALA DE CINZA

```
import numpy as np
import math
import cv2

def double_linear(input_signal, zoom_multiples):

        input_signal_cp = np.copy (input_signal)
        input_row, input_col = input_signal_cp.shape

    output_row = int(input_row * zoom_multiples)
    output_col = int(input_col * zoom_multiples)

        output_signal = np.zeros ((output_row, output_col))

    for i in range(output_row):
        for j in range(output_col):
            temp_x = i / output_row * input_row
            temp_y = j / output_col * input_col

            x1 = int(temp_x)
            y1 = int(temp_y)

            x2 = x1
            y2 = y1 + 1

            x3 = x1 + 1
            y3 = y1

            x4 = x1 + 1
            y4 = y1 + 1

            u = temp_x - x1
```

```
            v = temp_y - y1

        if x4 >= input_row:
            x4 = input_row - 1
            x2 = x4
            x1 = x4 - 1
            x3 = x4 - 1
        if y4 >= input_col:
            y4 = input_col - 1
            y3 = y4
            y1 = y4 - 1
            y2 = y4 - 1

        # interpolation
        output_signal[i, j] = (1-u)*(1-v)*int(input_signal_cp[x1, y1])
        + (1-u)*v*int(input_signal_cp[x2, y2]) +
        u*(1-v)*int(input_signal_cp[x3, y3]) +
        u*v*int(input_signal_cp[x4, y4])
    return output_signal

# Read image
img = cv2.imread("../paojie_g.jpg",0).astype(np.float)
out = double_linear(img,2).astype(np.uint8)

# Save result
cv2.imshow("result", out)
cv2.imwrite("out.jpg", out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

APÊNDICE B – IMPLEMENTAÇÃO PYTHON DE INTERPOLAÇÃO DE IMAGEM BILINEAR A CORES

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import math

def BiLinear_interpolation(img,dstH,dstW):
    scrH,scrW,_=img.shape
    img=np.pad(img,((0,1),(0,1),(0,0)),'constant')
    retimg=np.zeros((dstH,dstW,3),dtype=np.uint8)
    for i in range(dstH):
        for j in range(dstW):
            scrx=(i+1)*(scrH/dstH)-1
            scry=(j+1)*(scrW/dstW)-1
            x=math.floor(scrx)
            y=math.floor(scry)
            u=scrx-x
            v=scry-y
            retimg[i,j]=(1-u)*(1-v)*img[x,y]+
            u*(1-v)*img[x+1,y]+(1-u)*v*img[x,y+1]+
            u*v*img[x+1,y+1]
    return retimg

im_path='../paojie.jpg'
image=np.array(Image.open(im_path))
image2=BiLinear_interpolation(image,image.shape[0]*2,image.shape[1]*2)
image2=Image.fromarray(image2.astype('uint8')).convert('RGB')
```