UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ARTHUR ANTUNES NOGUEIRA DA SILVA

RECONHECIMENTO DE PEÇAS DO JOGO PROGSTER UTILIZANDO APRENDIZADO PROFUNDO

QUIXADÁ

2019

ARTHUR ANTUNES NOGUEIRA DA SILVA

RECONHECIMENTO DE PEÇAS DO JOGO PROGSTER UTILIZANDO APRENDIZADO PROFUNDO

Monografia apresentada no curso de Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Ciência da Computação. Área de concentração: Computação.

Orientador: Prof. Dr. Criston Pereira de Souza

Coorientador: Prof. Bel. Pedro Henrique Lopes Torres

QUIXADÁ

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S578r Silva, Arthur Antunes Nogueira da.
Reconhecimento de peças do jogo Progster utilizando aprendizado profundo / Arthur Antunes Nogueira da Silva. – 2019.
64 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2019.
Orientação: Prof. Dr. Criston Pereira de Souza.
Coorientação: Prof. Bel. Pedro Henrique Lopes Torres.

1. Aprendizagem Profunda. 2. Visão Computacional. 3. Jogos de Tabuleiro. I. Título.

CDD 004

ARTHUR ANTUNES NOGUEIRA DA SILVA

RECONHECIMENTO DE PEÇAS DO JOGO PROGSTER UTILIZANDO APRENDIZADO PROFUNDO

Monografia apresentada no curso de Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Ciência da Computação. Área de concentração: Computação.

Aprovada em: ___/___/___

BANCA EXAMINADORA

Prof. Dr. Criston Pereira de Souza (Orientador)
Universidade Federal do Ceará – UFC

Prof. Bel. Pedro Henrique Lopes Torres (Coorientador)
Pontifícia Universidade Católica - PUC

Prof. Dr. Paulo de Tarso Guerra Oliveira
Universidade Federal do Ceará - UFC

Prof. Dr. Regis Pires Magalhães
Universidade Federal do Ceará - UFC

À minha extraordinária mãe, Mazinha Nogueira.

AGRADECIMENTOS

Agradeço á minha mãe, Mazinha Nogueira, por sempre fazer tudo que estava ao seu alcance para que eu pudesse chegar até aqui. Reconheço todo seu amor e seu esforço, mãe.

Agradeço ao meu irmão, Henrique, e ao meu pai, Pádua, por fazerem parte da minha formação como pessoa.

Agradeço a toda minha família, tios, tias e avós, por sempre me incentivarem a seguir em frente nos estudos meio as contravenções que a vida proporciona.

Agradeço ao Prof. Dr. Criston Pereira de Souza, por aceitar o desafio de me guiar neste trabalho e pela excelente orientação exercida.

Agradeço ao meu co-orientador e amigo Pedro Henrique, por me auxiliar na reta final da formação, com suas orientações acadêmicas e pessoais. Que esteve comigo em todas as dimensões possíveis até o presente instante. R15.

Agradeço a todos colegas que contribuíram na minha formação de alguma forma, em especial os colegas de turma, Décio, Leonardo, Ronildo, Marcelo, Raul, Dieinison, Daiane, João Vitor, e Salathiel.

Agradeço aos professores que tive durante minha vida escolar e acadêmica, em especial, Joel, Carlos, Rogério e Rodinei.

Por último, agradeço à minha namorada, Cíntia Beatriz, por acreditar mais em mim do que eu mesmo em alguns momentos e me motivar seguir em frente independente das circunstancias. Obrigado meu amor, por tudo.

“Nossas maiores realizações não podem ficar para trás porque nosso destino está acima de nós.”

(Joseph Cooper)

RESUMO

Este trabalho propõe uma nova abordagem em um dos processos do jogo Progster, fazendo a utilização de Aprendizagem Profunda para a classificação das regiões de interesse de seu tabuleiro. O objetivo deste trabalho é criar modelos de Aprendizagem Profunda que classificam as regiões de interesse do tabuleiro e possam ser utilizados em uma aplicação que descarta o tabuleiro eletrônico proposto anteriormente, tornando o jogo mais acessível. Para isto, foram utilizadas técnicas de Visão Computacional na etapa de criação e coleta de dados das regiões de interesse do tabuleiro. A partir dos dados coletados, foram aplicadas técnicas de Aprendizagem Profunda para classificar cada região de interesse do tabuleiro. Os resultados mostraram que no melhor cenário de experimentação a acurácia da classificação chegou a 99,37%, evidenciando a atual eficácia dos métodos de Aprendizagem Profunda em problemas de classificação de imagem.

Palavras-chave: Aprendizagem Profunda. Visão Computacional. Progster. Jogos de Tabuleiro.

ABSTRACT

This work proposes a new approach in one of the processes of the game Progster, making the use of Deep Learning to the classification of the regions of interest of its board. The objective of this work is to create Deep Learning models that classify the regions of interest of the board and can be used in an application that discards the previously proposed electronic board, making the game more accessible. For this, we used the techniques of Computational Vision in the stage of creation and data collection of regions of interest of the board. From the data collected, Deep Learning techniques were applied to classify each region of interest of the board. The results showed that in the best experimental scenario the accuracy of the classification reached 99.37 %, evidencing the current effectiveness of the Deep Learning methods in image classification problems.

Keywords: Deep Learning. Computer vision. Progster. Board games.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

VC	Visão Computacional
DL	*Deep Learning*
CNN	*Convolutional Neural Network*

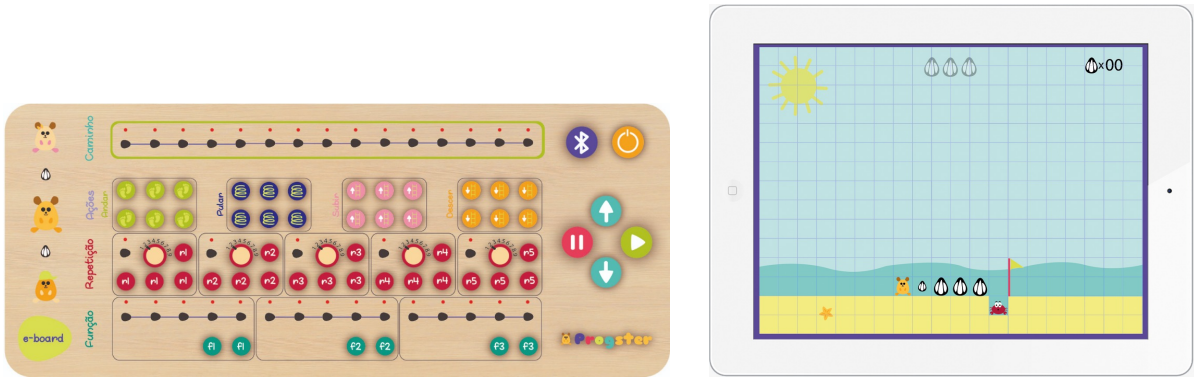
SUMÁRIO

1 INTRODUÇÃO

Em (LEMOS et al., 2017) é proposto um brinquedo, denominado *Progster*, com o objetivo de auxiliar o desenvolvimento lógico, ensinando de forma didática lógica de programação para crianças entre 7 e 9 anos. Além do desenvolvimento de raciocínio lógico das crianças, outra motivação do brinquedo é instigar o aprendizado na área de Tecnologia da Informação para conseguir aumentar a oferta de profissionais na área e conseguir suprir o mercado existente (LEMOS et al., 2017). A utilização de uma abordagem como essa é mais atrativa para o público infantil, a tornando uma boa estratégia para realizar o desenvolvimento lógico neste público.

Originalmente, o brinquedo proposto possui um tabuleiro eletrônico onde são indicadas as instruções lógicas do jogo por meio das peças do tabuleiro, então, o tabuleiro se conecta a um *tablet* ou *smartphone* via *bluetooth* para realizar a execução das instruções passadas pelo jogador no tabuleiro, em uma interface de jogo lúdica. Na Figura 1 são ilustrados o tabuleiro eletrônico e a interface de execução das instruções em um *tablet*.

Figura 1 – Tabuleiro eletrônico e cenário lúdico da versão original do Progster

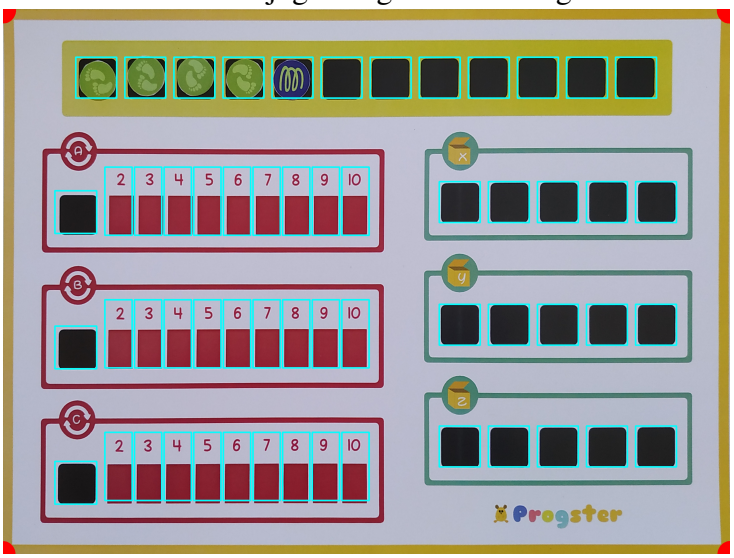


Fonte: (LEMOS et al., 2017)

Previamente ao trabalho proposto, a aluna Raíssa Barros de Oliveira Lemos do curso de Design Digital, uma das autoras do *Progster*, realizou uma análise de custo da implementação do *Progster* e como alternativa para reduzir o custo, sugere a utilização de um tabuleiro e peças de papel com a configuração do tabuleiro captada pela câmera do *tablet* ou *smartphone*. A estratégia para o novo funcionamento é captar imagem do tabuleiro com as peças posicionadas, reconhecer a configuração do tabuleiro e seguir a execução do cenário lúdico proposto.

Diante desta situação, este trabalho propõe a aplicação de Visão Computacional e Aprendizado Profundo para realizar a recognição do tabuleiro. O processo vai desde a criação de uma base de dados com imagens do tabuleiro e de suas peças, utilizando métodos de Visão Computacional, à classificação dessas imagens utilizando métodos de Aprendizado Profundo, que de forma eficiente realiza a classificação de cada elemento do tabuleiro, gerando uma saída que seja utilizada no restante da execução do aplicativo. A principal motivação da aplicação deste trabalho é tornar dispensável a utilização do tabuleiro eletrônico do *Progster* e torná-lo mais acessível para o público a qual de destina, com a utilização do tabuleiro de papel. A Figura 2 mostra o tabuleiro do jogo *Progster* adaptado com as regiões de interesse destacadas. A demarcação dessas regiões será realizada utilizando Visão Computacional, e com as regiões obtidas será feita a classificação utilizando Aprendizado Profundo.

Figura 2 – Tabuleiro do jogo Progster e suas regiões de interesse



Fonte: Elaborada pelo autor

Corroborando com a proposta desse trabalho foi criado um ambiente, utilizando técnicas de Visão Computacional de projeção perspectiva e de seleção de regiões de interesse, para gerar os dados de forma otimizada. Os dados gerados foram utilizados na elaboração de um conjunto de treinamento, dividido em um conjunto de treino e validação, para realizar a criação de 8 modelos de Aprendizado Profundo. Além disso foi criado um conjunto de teste, para avaliar o modelo que apresentou a maior acurácia na etapa de treinamento. É importante destacar que o problema deste trabalho exige uma alta acurácia por parte dos modelos criados, uma vez que, caso haja um erro de classificação de uma peça do tabuleiro, todo o restante do funcionamento do jogo é prejudicado.

Os resultados dos modelos criados foram satisfatórios no conjunto de teste, com o melhor modelo apresentando uma acurácia de 99.37%. Certamente os resultados das acurácias dos modelos criados poderiam ser melhores, caso houvesse um número maior de imagens criadas para gerar os dados de treinamento dos modelos. Apesar disso, o resultado foi satisfatório mostrando como técnicas de Aprendizado Profundo podem ser aplicados em problemas reais.

Para entender como funciona Visão Computacional (VC) é importante interpretar a visão humana. O sistema visual humano permite captar e identificar qualquer objeto que reflete luz em um abrangente processo cognitivo (TORTORA; DERRICKSON, 2016). Todo esse processamento de dados é facultado pelos receptores sensoriais indo ao encontro das informações prévias que o cérebro já tem armazenado. VC tenta replicar esse processo de forma a fornecer recursos para que as máquinas sejam capazes de visualizar e interpretar imagens, conseguindo diferenciar os objetos de um cenário específico a fim de assemelhar-se ao sistema de visão humano. Dentre o conjunto de subáreas que fazem parte da VC, o Processamento Digital de Imagens (PDI) tem como um de seus objetivos realizar o processamento dos dados das imagens para percepção automática das máquinas (GONZALEZ; WOODS, 2008), (SZELISKI, 2010).

Atualmente, alguns problemas matemáticos são difíceis ou trabalhosos demais para pessoas comuns resolverem, porém, computadores os executam facilmente. De forma análoga, problemas como classificar, atribuir sentido ou detectar objetos em uma imagem qualquer, são difíceis para os computadores justamente por não possuir uma entrada e parâmetros bem definidos, já que a representação do mundo real para as máquinas é bastante abstrata. São nesses cenários problemáticos de recognição e conhecimento onde se propagam as aplicações dos métodos de Aprendizado de Máquina (*Machine Learning*) e, mais atualmente, Aprendizado Profundo (do inglês *Deep Learning*) (CHOLLET, 2018). O volume de dados que existe atualmente e a facilidade de acesso e criação desses dados, torna possível transformar esses dados em conhecimento, através dessas áreas com o fim de prever, categorizar ou encontrar padrões (RASCHKA; MIRJALILI, 2017).

A utilização de Aprendizado Profundo reflete o atual estado da arte na classificação, detecção e segmentação de objetos em imagens. Parte da ideia de usar *Deep Learning* visa criar modelos robustos para conseguir identificar padrões em um conjunto de dados e realizar alguma tarefa considerada complexa, dispensando o pré-processamento tradicional e utilizando propriedades invariantes (GARCIA-GARCIA et al., 2017) em suas camadas hierárquicas de representação. Os principais desafios abordados são os de detecção e segmentação de objetos

em imagens. Para resolvê-los, é feita a utilização de Redes Neurais Convolucionais (CNN — do inglês *Convolutional Neural Network*). Em geral, as CNNs utilizam modelos treinados com várias camadas, aprendendo eficientemente características individuais e utilizando-as no restante do processo de aprendizagem (RASCHKA; MIRJALILI, 2017).

As aplicações que utilizam a combinação dos métodos tradicionais de VC com as técnicas robustas de DL estão presentes cada vez mais no nosso dia a dia. Problemas de indexação semântica, ou seja, agrupar imagens por características incomuns, são postos constantemente em aplicações conhecidas como o Google Fotos[1] que consegue automaticamente criar álbuns personalizados com imagens semelhantes em cada um deles (NAJAFABADI et al., 2015). Aplicações usadas por várias empresas que produzem *smartphones* é o reconhecimento facial que autoriza ou não o usuário a fazer certas atividades com o aparelho (NAJAFABADI et al., 2015). Problemas específicos, como os da área da saúde, que vão desde detecção de anomalias em radiografias à contagem de tipos de células no sangue, são resolvidos através da classificação utilizando DL e bastante disseminadas atualmente (LEE et al., 2019; SHEN; WU; SUK, 2017).

Portanto, o trabalho presente se estende por duas etapas principais onde, a etapa inicial consiste em coletar imagens das posições que são regiões de interesse do tabuleiro e das peças. Para realização desta etapa, são utilizadas técnicas tradicionais de VC que utilizam propriedades bem definidas de cada imagem, como a localização de uma região da imagem. A próxima etapa aplica técnicas de DL e é realizada utilizando arquiteturas de CNNs para criação de modelos que classificam cada peça e cada região do tabuleiro a partir de uma imagem.

Os demais capítulos estão ordenados do seguinte modo: no Capítulo 2 são apresentados os trabalhos relacionados a este, que em geral aplicam Visão Computacional à problemas de recognição de tabuleiros; no Capítulo 3 está descrita toda a fundamentação teórica essencial para o entendimento deste trabalho; no Capítulo 4 é exposto detalhadamente os procedimentos metodológicos deste trabalho e seus devidos arranjos; por fim, o Capítulo 5 apresenta as considerações finais deste trabalho.

[1] https://www.google.com/photos/about/

2 TRABALHOS RELACIONADOS

Neste capitulo são apresentados trabalhos relacionados que contribuem para o presente trabalho. Essa contribuição ocorre em relação ao escopo dos trabalhos que serão apresentados, como em (CALDERON-CORDOVA et al., 2016), ou das tecnologias empregadas no trabalho, como em (MOLLA; LEPETIT, 2010) e (CHEN; WANG, 2019).

Em (CALDERON-CORDOVA et al., 2016) é proposta a utilização de uma ferramenta de Visão Computacional, denominada *LabVIEW* (JOHNSON, 1997), para criação de um *software* que automatiza a identificação de um tabuleiro matemático projetado para crianças que tem deficiência visual. Originalmente para a utilização do tabuleiro era necessário que um tutor supervisionasse todo o processo de execução da criança sobre o tabuleiro. O *software* proposto em (CALDERON-CORDOVA et al., 2016) necessita de um ambiente com uma câmera, para a leitura do tabuleiro, um teclado, para receber as operações da criança, e um alto-falante ou fone de ouvido, para reportar a saída da execução. As peças do tabuleiro são símbolos em braille de 0 à 9, operadores básicos e um símbolo de identificação para a entrada dos números. Os resultados apresentados em cima do reconhecimento do tabuleiro foram satisfatórios, apresentando 94.6% de sucesso na identificação.

Em (MOLLA; LEPETIT, 2010) é proposto um mapeamento de jogos de tabuleiro para Realidade Aumentada utilizando Visão Computacional. O método proposto se chama *MonopolyTM*1 por ter sido testado e motivado inicialmente pelo jogo *Monopoly* (Banco Imobiliário) mas que é facilmente adaptado por outros jogos de tabuleiro que tenham elementos a ser projetados. Para aplicar esse método é necessário uma *webcam* registrando o cenário em tempo real com o tabuleiro centralizado. O método inicialmente detecta as bordas do tabuleiro, depois detecta as regiões de interesse com elementos que podem vir a serem projetados e então aplica a realidade aumentada de cada elemento encontrado. Todo o desenvolvimento do método utiliza a biblioteca de Visão Computacional *OpenCV* (BRADSKI; KAEHLER, 2008), desde a detecção do tabuleiro até a detecção dos peões e das regiões de interesse.

Em (CHEN; WANG, 2019) é proposto um robô humanoide que tem capacidade de jogar xadrez. Alega-se que com o desenvolvimento tecnológico a maquina de um jogo de xadrez virtual consegue jogar e competir com pessoas reais, porém na área da robótica replicar esse processo não é tão trivial já que obter as informações do tabuleiro de xadrez no mundo real requer a identificação do tabuleiro, das peças e da organização das peças. O robô humanoide Baxter que é proposto, utiliza Visão Computacional através da biblioteca *OpenCV* para todo o processo,

desde a identificação do tabuleiro, identificação das peças e da recognição do estado do tabuleiro em tempo real. O método é realizado capturando uma imagem do tabuleiro de xadrez com uma câmera embutida no Baxter a cada movimento do adversário durante uma partida. Logo após é feita uma comparação com o estado anterior do tabuleiro e em cima disso é processada a análise do próximo movimento a ser feito.

Este trabalho é responsável por desenvolver a tecnologia de Visão Computacional para recognição de cada região de interesse do tabuleiro do ambiente de aprendizagem Progster proposto em (LEMOS et al., 2017). O processo deste trabalho vai desde a criação do banco de imagens de cada região de interesse do tabuleiro com o uso do *OpenCV* à aplicação de métodos de *Deep Learning* com o auxilio da biblioteca *Keras* (CHOLLET, 2018) onde são criados modelos de classificação de imagens para indicar o reconhecimento de cada região de interesse do tabuleiro do jogo.

O Quadro 1 evidência as diferenças e semelhanças entre os trabalhos relacionados e o trabalho proposto. São citados três métricas para que a comparação dos trabalhos fique mais visível. Inicialmente foi feita uma comparação da área de aplicação a qual cada trabalho se propõe a resolver, indo do âmbito educacional ao de inovação. Em seguida é diferenciado o principal objetivo de cada trabalho. E por final, as bibliotecas utilizadas para realizar a execução de cada trabalho respectivamente.

Quadro 1 – Comparações deste trabalho com os trabalhos relacionados.

	Trabalho Proposto	(CALDERON-CORDOVA et al., 2016)	(MOLLA; LEPETIT, 2010)	CHEN; WANG, 2019)
Área de aplicação	Educação: aprendizagem de lógica de programação para crianças	Educação: aprendizagem de matemática de pessoas cegas	Criação de cenários 3D a partir de imagens 2D	Robótica
Objetivos	Regonição das Regiões de interesse e das peças do tabuleiro	Identificação das peças do tabuleiro	Aplicar realidade aumentada em ROIs de tabuleiros de jogos	Identificação do tabuleiro em diversos ambientes
Bibliotecas utilizadas	*OpenCV e Keras*	*LabVIEW*	*OpenCV*	*OpenCV*

Fonte: Elaborado pelo autor.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos aplicados neste trabalho. Na Seção 3.1 é apresentado um breve histórico sobre Visão Computacional e suas aplicações. Na Seção 3.2 são expostos os conceitos de projeção perspectiva e regiões de interesse. Na Seção 3.3 é apresentada toda a parte de Aprendizagem Profunda, desde o funcionamento de Redes Neurais às arquiteturas que utilizam Redes Neurais Convolucionais.

3.1 Visão Computacional

A Visão Computacional é a área que estuda a extração de informações sobre imagens digitais através de técnicas que conseguem captar características únicas de cada imagem. Essas características variam de acordo com a finalidade de cada método aplicado. De modo geral, as técnicas buscam representar a formação da imagem, processar os dados que cada imagem já dispõe, detectar características de objetos dispostos na imagem, segmentar os objetos e realizar o reconhecimento sobre os objetos na imagem (SZELISKI, 2010).

Desde a década de 1960 a Visão Computacional é estudada pela comunidade acadêmica para conseguir fazer que as máquinas pudessem interpretar de forma direta o que tinha nas imagens (SZELISKI, 2010). Nos primeiros passos da exploração dessa área, o Processamento Digital de Imagens se destacou, pois de maneira mais simples buscava apenas encontrar propriedades matemáticas que as imagens ofereciam e, diferente do que acreditava-se na época, fazer a interpretação do contexto de cada imagem era bastante difícil (SZELISKI, 2010). Assim, de forma a abranger todo o processo, a pretensão da Visão Computacional não se limita a extrair dados das imagens, mas sim reconhecer e representar o que é visto nas imagens (SZELISKI, 2010).

Atualmente, fazer essa interpretação do mundo através de imagens é alcançado com a combinação de Visão Computacional e Aprendizagem de Máquina, mais especificamente Aprendizagem Profunda (CHOLLET, 2018). Na última década, a eficiência alcançada dos algoritmos e a robustez atingida dos *hardwares*, tornou possível fazer com que as máquinas aprendessem a interpretar de forma mais clara, conseguindo reconhecer os objetos dispostos em cada imagem e suas características.

O uso da Visão Computacional é empregado atualmente em várias áreas e no dia a dia. O avanço na precisão de diagnósticos é notável quando trata-se de imagens médicas, pois

a precisão no registro e classificação de imagens pré-operatórias e operatórias, ou, realizando estudos de longo prazo que utilizam imagem para avaliação de um diagnóstico apresenta uma boa acurácia. Outra área que mostra bastante proveito do uso de Visão Computacional é o reconhecimento de impressões digitais e biometria para autenticação de acesso automático e aplicações forenses (GARCIA-GARCIA et al., 2017). Mais aplicações atuais que utilizam propriedades e técnicas de Visão estão presentes na Figura 3

Figura 3 – Aplicações de Visão Computacional.

Fonte: Elaborada pelo autor.

Neste trabalho a aplicação de métodos tradicionais de Visão Computacional ocorre na coleta e no pré-processamento dos dados, fazendo a projeção perspectiva do tabuleiro das imagens criadas e na seleção de regiões de interesse do tabuleiro.

3.2 Projeção Perspectiva e Regiões de Interesse

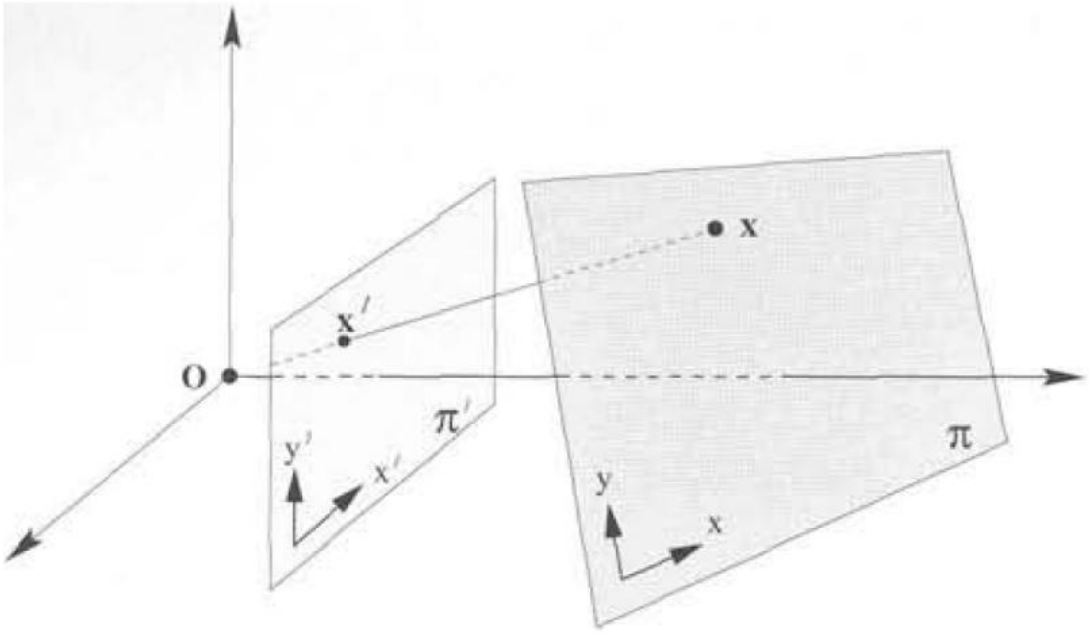
Uma projeção perspectiva consiste em fazer um deslocamento em uma imagem de uma região bidimensional, que é selecionada por pontos da extremidade dessa região e a preservação do ponto de referência do observador em orientação ao centro da imagem (SZELISKI, 2010). Para que a aplicação dessa projeção funcione de forma efetiva é necessário primeiramente uma transformação projetiva planar de homografia que mapeia os pontos de um plano para outro (SANTOS; ROCHA, 2012).

Considere um conjunto de pontos $p_i = (x_i, y_i, w_i)^T$ sobre um plano π e outro conjunto de pontos correspondentes $p'_i = (x'_i, y'_i, w'_i)^T$ em um plano π', a transformação projetiva que gera a matriz H e que mapeia os pontos de π para π' é descrita como:

$$\begin{bmatrix} x'_i \\ y'_i \\ w'_i \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}$$

ou como $p'_i = H * p_i$, em que H é uma matriz 3 x 3, e os pontos 2D são expressos em coordenadas homogêneas (SANTOS; ROCHA, 2012). A figura 4 exemplifica a projeção com relação aos planos e ao ponto O da origem.

Figura 4 – Projeção perspectiva de um plano π para um plano π'.

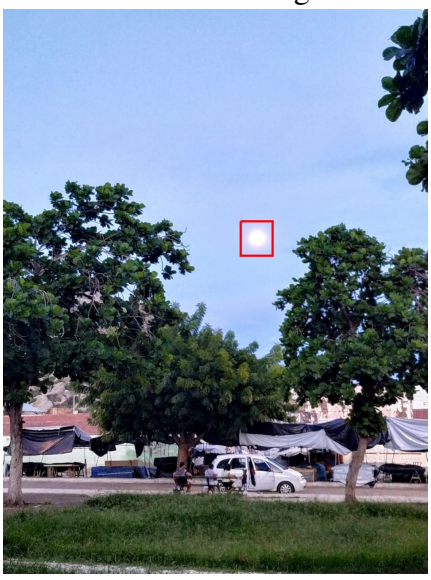


Fonte: (SANTOS; ROCHA, 2012).

Atualmente a aplicação da projeção em Visão Computacional utiliza técnicas mais sofisticadas que otimizam o processo de transformação, onde, dados os pontos que formam o plano a ser projetado numa imagem, a projeção é calculada baseada no plano da imagem original, e existem técnicas que calculam a transformação de forma bastante eficiente. Uma dessas técnicas, e que é bastante utilizada, é o método *RANSAC* (SZELISKI, 2010) (BRADSKI; KAEHLER, 2008).

O conceito de região de interesse numa imagem digital se dá por uma área de *pixels* que esteja inclusa na distribuição original de *pixels* desta imagem e que tenha algum valor semântico para um problema específico (SZELISKI, 2010). Não existe forma geométrica preestabelecida para uma região de interesse, podendo apresentar um caráter diverso. Para este trabalho é dito que, dado quatro pontos sob o plano da imagem, a área interna entre esses pontos é identificado como região de interesse. Na prática selecionar uma região de interesse em uma imagem, exige apenas as coordenadas x e y do plano da imagem 2d (SZELISKI, 2010) (BRADSKI; KAEHLER, 2008). Na figura 5 é vista a seleção de uma região de interesse qualquer da imagem.

Figura 5 – A Lua é selecionada como Região de Interesse na imagem.



Fonte: Elaborada pelo autor.

Na etapa de pré-processamento dos dados, as imagens coletadas inicialmente passam pela fase de projeção perspectiva, com o objetivo de expandir a imagem somente do tabuleiro e logo após, é selecionada e salva imagem de cada região de interesse do tabuleiro para criação da base de dados de cada peça e região do tabuleiro.

3.3 Aprendizagem Profunda

Aprendizagem profunda, ou do inglês *Deep Learning*, é uma subárea específica de aprendizado de máquina (*Machine Learning*) que aborda de uma nova forma as representações de aprendizado com a característica da utilização de camadas sucessivas de representação e cada vez mais expressivas (CHOLLET, 2018). A ideia de profundidade nessa abordagem não é uma compreensão a qualquer tipo de entendimento mais expressivo alcançado pela abordagem, significa na verdade, a utilização de múltiplas camadas de representações sucessivas no aprendizado (CHOLLET, 2018).

Outros nomes apropriados para esta abordagem poderiam ser aprendizado de representação de camadas ou aprendizado de representações hierárquicas (CHOLLET, 2018). No geral, o aprendizado profundo é uma estrutura matemática que aprende representações diversas a partir de uma base expressiva de dados (CHOLLET, 2018). Para compreender o processo de aprendizagem de um modelo de Aprendizagem Profunda é necessário dar alguns passos atrás e entender inicialmente o funcionamento das Redes Neurais Artificiais (RNA).

3.3.1 Redes Neurais

Uma Rede Neural é uma estrutura de aprendizagem concebida na área de Aprendizado de Máquina que foi inspirada no funcionamento cognitivo de um neurônio cerebral (RASCHKA; MIRJALILI, 2017). A principal motivação para a exploração dessa área é se aproximar da forma de aprendizagem de recognição que o cérebro realiza através da sua experiência adquirida. O conjunto de estruturas biológicas e os processos do cérebro já vem desde o nascimento de cada ser, formando um sistema robusto com a capacidade de construir suas próprias regras de comportamento através da experiência (HAYKIN et al., 2009). Essa experiência é construída ao longo do tempo. Inicialmente o cérebro faz associações simples de sensações, mas ao longo do tempo o desenvolvimento continua bem além desse estágio (HAYKIN et al., 2009).

De forma prática, uma Rede Neural é um processador constituído de diversas unidades de processamento simples e paralelas, que se designam a armazenar conhecimento experiencial com uso assegurado (HAYKIN et al., 2009).

As principais características de uma Rede Neural são o conhecimento adquirido pela rede a partir de seu ambiente, por meio de um processo de aprendizado, e a sua estrutura de conexão interna, conhecidas como pesos sinápticos, são usadas para armazenar o conhecimento baseado na experiencia de uma iteração. Esse processo de aprendizagem é chamado de algoritmo de aprendizado, que tem a função de modificar os pesos da rede de forma ordenada para atingir o objetivo de cada rede (HAYKIN et al., 2009).

3.3.1.1 Processo de Aprendizado

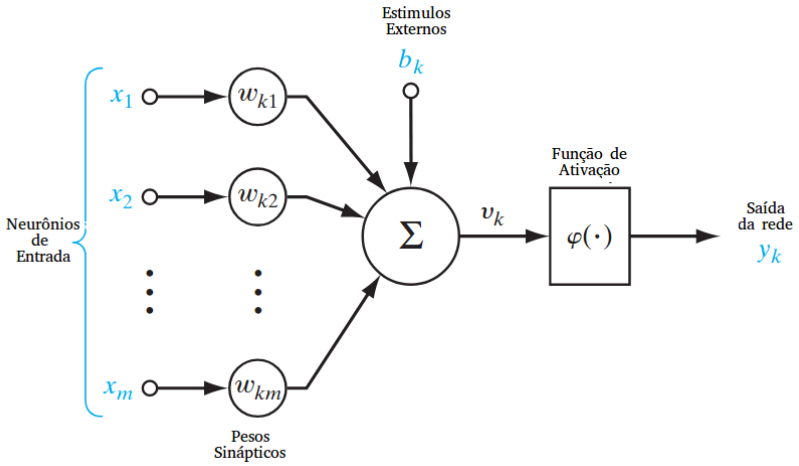
O processo de aprendizado consiste em um conjunto de regras bem definidas para a solução de um problema e a maneira pela qual uma rede neural se relaciona com o ambiente. Com isso, o processo de aprendizagem é responsável por adaptar os pesos das conexões de acordo com os estímulos recebidos do ambiente. Portanto, é notável que a aprendizagem do processo está representada nos pesos sinápticos (HAYKIN et al., 2009).

O comportamento do processo de aprendizagem se distingue pela forma de interação com o ambiente, portanto, a mudança na etapa de adequação dos pesos sinápticos são diferentes de acordo com a abordagem que são empregadas às RNA. Nesse contexto os paradigmas de aprendizado são:

- Aprendizado Supervisionado, quando há a utilização de um agente externo que aponta à rede a resposta esperada dado o padrão de entrada. Os problemas de classificação são mais comuns neste tipo de abordagem;
- Aprendizado Não Supervisionado, quando não existe um agente externo fornecendo a resposta desejada para os padrões de entrada e a rede se adéqua naturalmente;
- Aprendizado por Reforço, quando existe uma estrutura externa que avalia a resposta fornecida pela rede e a notifica.

Este trabalho aborda o cenário de Aprendizado Supervisionado, já que as arquiteturas de RNAs de aprendizagem profunda empregadas aqui se destinam a classificação de múltiplas classes, com utilização de métricas para verificar o comportamento da rede durante sua execução e os resultados obtidos. Na Figura 6 é apresentado o esboço da estrutura de uma Rede Neural que apresenta algumas estruturas importantes para seu funcionamento (HAYKIN et al., 2009).

Figura 6 – Esboço de uma Rede Neural.



Fonte: adaptado de (HAYKIN et al., 2009)

Os neurônios de entrada formam a camada inicial da rede, onde são distribuídos todas as entradas de um problema específico tal que cada uma dessas entradas representa um neurônio dessa camada (HAYKIN et al., 2009).

Os pesos sinápticos são as conexões internas da rede. Cada neurônio k tem seu peso correspondente w_k que influencia a rede de alguma forma mas que pode vir a ser atualizado de acordo com os estímulos externos da rede, ou seja, é atualizado caso em algum momento o algoritmo aprenda que diminuir ou aumentar o peso de um certo neurônio aumente a eficacia do processo de aprendizagem. Os pesos de uma Rede Neural podem estar em uma faixa que inclui tanto valores negativos quanto positivos (HAYKIN et al., 2009).

O agente designado ao estimulo externo da rede b_k tem o papel de aumentar ou diminuir a influência do valor da entrada do sinal do neurônio correspondente, para a auxiliar na ativação do neurônio k (HAYKIN et al., 2009).

A função de ativação exerce uma limitação a amplitude da saída de um neurônio, ou seja, ela influencia na decisão se um neurônio deve ser ativado ou não e se a informação que o neurônio está produzindo é relevante para o progresso do processo de aprendizagem da rede (HAYKIN et al., 2009). De modo geral sua utilização tem dois propósitos: limitar a saída do neurônio e permitir que um modelo tenha múltiplas saídas.

A saída da rede representa o resultado de todas as etapas da rede, desde a obtenção do sinal de cada neurônio até a ativação de cada neurônio através da função de ativação da rede. De modo geral, a saída de um neurônio k pode ser representada como $y_k = \sum_{j=1}^{m} w_{kj} * x_j + b_k$ (HAYKIN et al., 2009).

3.3.1.2 Funções de Ativação

As funções de ativação das RNAs são responsáveis pela a transformação não-linear aplicada ao sinal de entrada. Caso não haja uma função de ativação, os pesos sinápticos e o fator externo de cada neurônio fazem uma aplicação linear na saída da rede, que apesar de simples resolução, apresenta limitações na resolução de problemas com mais robustez. Uma RNA sem função de ativação torna-se um modelo mais simples de regressão linear. Portanto, a função de ativação tem o papel de fazer a transformação não-linear nos dados e sinais de entrada de um modelo, o habilitando a aprender e processar tarefas mais complexas (GOODFELLOW; BENGIO; COURVILLE, 2016).

A função de ativação mais simples é denominada Função de Etapa Binária *(Binary Step Function)*, que de forma clara determina se um neurônio de uma camada deve ser ou não ativado, dada a influência daquele neurônio em toda a RNA. Essa função atua como um classificador binário, resolvendo problemas mais simples.

A *Sigmóide* é uma função de ativação que já foi bastante utilizada e que tem como característica ser não linear. Isto implica que quando há vários neurônios em uma camada que a utilizam, a saída não é linear. A função essencialmente tenta impulsionar os valores da saída de cada neurônio para os extremos. Pode ser definida pela equação 3.1:

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.1}$$

Devido a natureza de sua equação, um problema comum da *sigmóide* está ligado ao gradiente de sua função. O gradiente gerado é muito limitado, pois satura para valores acima de 5 e abaixo de -5, se aproximando de zero. Isso causa uma resistência no aprendizado da rede.

ReLU é a função de ativação mais utilizada atualmente ao projetar modelos de RNA. Assim como a *sigmóide* ela é não linear, isso significa que facilmente copia os erros para as camadas de trás auxiliando o algoritmo de treinamento em RNAs com mais camadas. A vantagem de usa-la em relação as demais é que ela não ativa todos os neurônios simultaneamente, pois se o sinal de algum neurônio for negativo, ela será convertida em zero e o neurônio não será ativado e isso a torna mais barata computacionalmente. Logo apenas alguns neurônios são ativados, tornando a rede mais esparsa. Pode ser definida pela equação 3.2:

$$f(x) = max(0,x) \tag{3.2}$$

Outra função de ativação importante é a *Softmax* que é uma variação da função *Sigmóide*. A *Sigmóide* foi desenvolvida visando problemas de classificação binária, ou seja, o neurônio que passa pela função de ativação *Sigmóide* tem a saída como 1 ou 0, representando se aquele neurônio corresponde ou não a classe que o problema verifica. Já a *Softmax* generaliza essa classificação podendo ser utilizada em cenários onde o problema apresenta mais que duas classificações para os neurônios da rede. Nesses problemas com muitas classes, classificar "sim ou não" para uma única classe não funciona. A *Softmax* então faz uma transformação das saídas para calcular a probabilidade da entrada testada estar em uma determinada classe. A *Softmax* pode ser definida pela equação 3.3:

$$S(y^i) = \frac{e^{y^i}}{\sum_j^N e^{y^i}} \tag{3.3}$$

As funções de ativação são utilizadas em problemas que se propõem a realizar classificação de imagens utilizando Redes Neurais. Elas realizam a predição em várias etapas do processo de aprendizagem e serão utilizadas na criação dos modelos de classificação do presente trabalho.

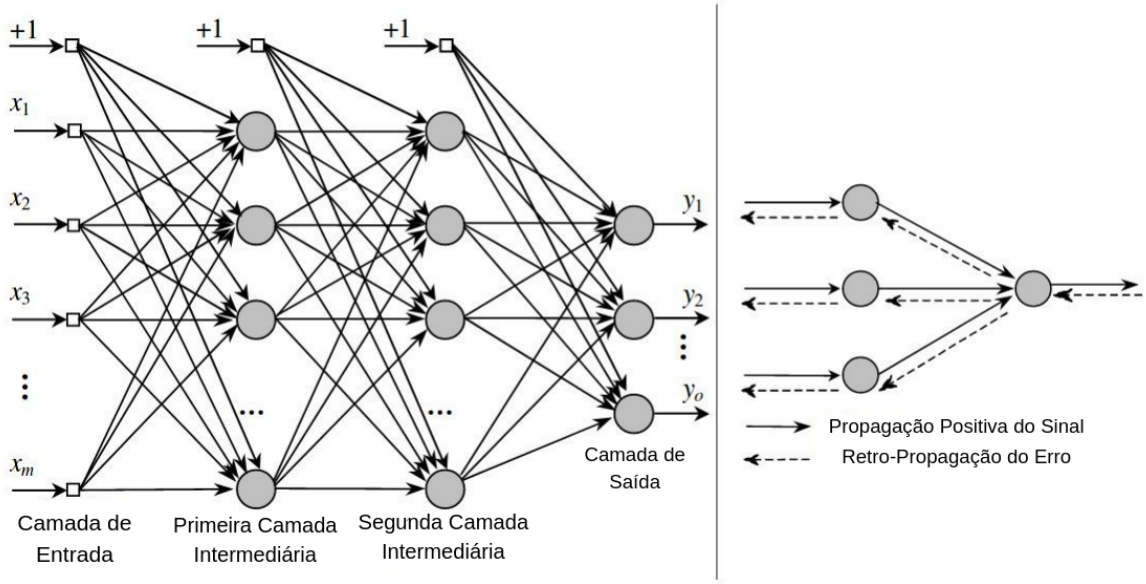
3.3.2 Redes Neurais com Múltiplas Camadas

Em RNAs de uma só camada, os padrões do processo de treinamento são apresentados à entrada e mapeados diretamente em um conjunto de padrões de saída da rede, ou seja, não é possível a formação de uma representação de aprendizagem interna pois não possui camadas intermediarias e torna essas estruturas incapacitadas de aprender mapeamentos importantes dentro de um problema (HAYKIN et al., 2009).

Com isso, é proposto *perceptron* de múltiplas camadas *(multi-layer perceptron, MLP)* que é uma RNA com pelo menos uma camada intermediária, com alto grau de conectividade entre as camadas e seus neurônios. As camadas intermediárias facilitam o aumento de iterações do algoritmo de aprendizagem e ocorrem mais correções nos pesos sinápicos da rede para que eles representem melhor o aprendizado e um problema através do treinamento da MLP (RASCHKA; MIRJALILI, 2017). A Figura 7 mostra que nesse caso podem haver camadas intermediárias na rede e o alto grau de conectividade que essa estrutura apresenta.

Todos os fundamentos aplicados a RNAs com múltiplas camadas estão englobados e formam a denominação de Redes Neurais Profundas, que usualmente são chamados de modelos de Aprendizagem Profunda. A palavra profunda utilizada na nomenclatura do termo é tida pela quantidade de camadas intermediárias utilizadas nos modelos (CHOLLET, 2018).

Figura 7 – Esboço de um *perceptron* de múltiplas camadas, MLP.



Fonte: adaptado de (HAYKIN et al., 2009)

3.3.2.1 Backpropagation

Na etapa de treinamento do processo de aprendizado de uma MLP, ocorre a adaptação dos pesos sinápticos da rede para um balanceamento que resulta em uma melhor desempenho da rede. Para isto, é implementado um algoritmo designado como retro propagação de erro ou *backpropagation* (HAYKIN et al., 2009). Este algoritmo consiste basicamente de dois passos:

- Propagação positiva do sinal funcional: é emitido na camada de entrada passando por toda a rede até alcançar a saída. Durante esta etapa os pesos sinápticos de cada neurônio são fixados;
- Retro-propagação do erro: é propagação em sentido oposto ao de propagação do sinal funcional. Durante esta etapa os pesos sinápticos são definidos baseados no erro.

O erro do *backpropagation* é calculado no segundo passo do algoritmo, onde a saída encontrada na propagação positiva do sinal é comparada à saída esperada para o padrão de entrada da MLP (HAYKIN et al., 2009). A partir disso, o erro é retro propagado partindo da camada de saída à camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo ajustados conforme o erro é retro propagado.

A criação de um modelo baseado em uma RNA de múltiplas camadas visa um problema especifico bem definido. Este problema contém um conjunto de dados a qual a rede irá utilizar para treinar, realizando o balanceamento dos pesos e depois realizar sua validação. Há também um outro conjunto de dados separado que será utilizado para testar o modelo treinado. Esse conjunto de teste é necessário quando há a necessidade de aumentar a segurança da eficiência do resultado obtido de uma rede (HAYKIN et al., 2009).

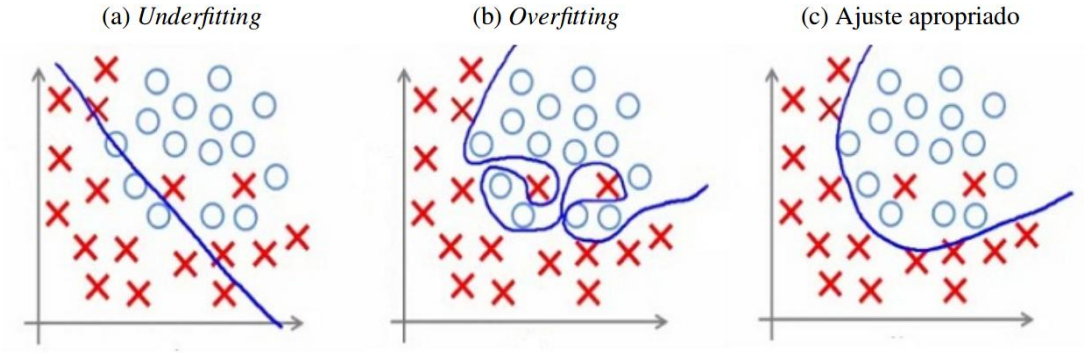
Quando ocorre a retro propagação do erro por toda a rede é definida uma *época*. Comumente, é feita utilização aleatória das amostras do conjunto de treino entre as épocas. Esta aleatoriedade faz que o ajuste dos pesos sinápticos tenha uma natureza estocástica ao longo das épocas de treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016). No processo de aprendizagem há a realização de um número definido de épocas, ou, até um critério de parada ser alcançado. O critério de parada é baseado no erro da rede. Como a cada iteração de treinamento os pesos se ajustam, é esperado que o valor de erro diminua até atender o critério de parada (GOODFELLOW; BENGIO; COURVILLE, 2016). Na Figura 7 é ilustrada a propagação positiva e a retro propagação do sinal.

3.3.2.2 Overfitting e Underfitting

Durante a implementação de modelos de Aprendizagem Profunda, podem ocorrer discrepâncias sobre o conjunto de dados utilizado, que de alguma forma mascare o comportamento do modelo alterando o resultado que aquele modelo fornece. Esses casos podem ser denominados *overfitting* e *underfitting* (GOODFELLOW; BENGIO; COURVILLE, 2016).

É denominado *overfitting* quando ocorre um sobreajuste do modelo aos dados do conjunto de treino. Acontece sempre que o modelo é experimentado com os dados do conjunto de treino e os resultados aparentam ser satisfatórios, mas na verdade está enviesado com as características daquele conjunto e, por consequência, não atende o requisito de generalizar a solução encontrada para os demais conjuntos de dados (GOODFELLOW; BENGIO; COURVILLE, 2016). No caso do *underfitting* ocorre um subajuste, ou seja, o modelo não reage a etapa de treinamento e não consegue ajustar os pesos da rede para que o erro diminua (GOODFELLOW; BENGIO; COURVILLE, 2016). Geralmente isso acontece pela simplicidade da rede ou por erro na parametrização do modelo criado.

Figura 8 – Representação de Overfitting e Underfitting .



Fonte: adaptado de (HAYKIN et al., 2009)

A Figura 8 demonstra como funciona cada abordagem em um conjunto de dados hipotético de um dado problema. Os elementos circulares do gráfico representam os dados de treino, os "x", os dados de teste e a linha azul é a classificação dada do modelo já treinado. Na Figura 8 a a linha se mostra inoperante sobre os dados e representa um problema de *underfitting*. A Figura 8 b mostra que o modelo consegue ajustar a linha do modelo com muita restrição em cima dos dados do conjunto de teste e representa o *overfitting*. Na Figura 8 c mostra o comportamento da linha de classificação do modelo de forma adequada, que não é tão restrita mas reage bem ao conjunto de dados.

Este trabalho busca classificar as peças e as regiões de interesse do tabuleiro, e nas abordagens de classificação de imagens pode ocorrer *overfitting* ou *underfitting*. Aumentar a quantidade de dados presentes no conjunto de treino é uma estratégia para solucionar esse problema, já que a rede utilizará mais dados para adequar seus pesos sinápticos, porém não é sempre que essa estratégia pode ser feita ou surte resultado (GOODFELLOW; BENGIO; COURVILLE, 2016). Desse modo, existem outras formas de mitigar esse problema.

A primeira abordagem é denominada *Regularização* e sua estratégia é acrescentar um termo a função de custo da rede, chamado termo de regularização. Esse termo de regularização faz que a rede prefira escolher pesos menores e penaliza pesos de coeficientes maiores, a não ser que a rede consiga distinguir e julgar que esses pesos maiores contribuam positivamente para a rede, tornando o erro menor (GOODFELLOW; BENGIO; COURVILLE, 2016).

A função de custo da rede sempre é atrelada a um fator que corresponde ao seu ambiente externo e no caso de uma rede com várias camadas e neurônios, esse fator é interpretado como ruídos que torna o aprendizado enviesado, se ajustando demais ao conjunto de treino (GOODFELLOW; BENGIO; COURVILLE, 2016). É aí que ocorre *overfitting* no modelo. Por esse motivo utilizar técnicas de regularização é funcional, pois mantém os pesos menores da rede e faz com que os ruídos mais irregulares sejam descartados (GOODFELLOW; BENGIO; COURVILLE, 2016). As principais técnicas de regularização que recalculam a função de custo da rede são chamadas *L1* e *L2*. As equações da *L1* e *L2* são definidas em 3.4 e 3.5.

$$C = c_0 + \lambda * \sum |w| \tag{3.4}$$

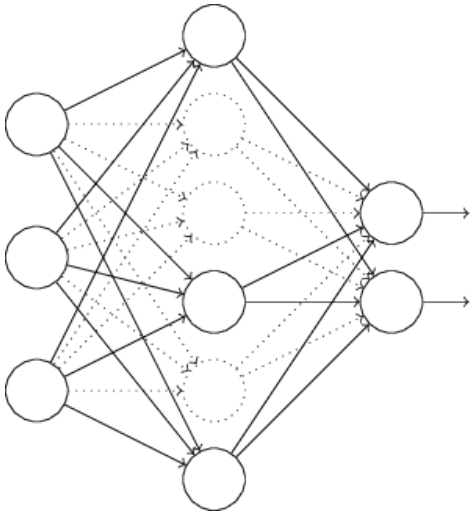
$$C = c_0 + \lambda * \sum w^2 \tag{3.5}$$

Na L1, a nova função de custo, C, representa a soma da função de custo anterior da rede, c_0, com o coeficiente de regularização, λ, multiplicado pelo somatório do modulo de todos os pesos da rede, onde estes tendem a diminuir em valor constante para 0. Já na L2, a multiplicação ocorre entre λ e somatório do modulo de todos os pesos. Estes diminuem proporcionalmente aos pesos da rede. Portanto se um peso tem grande magnitude, a L1 diminui esse peso muito menos do que a L2. Porém se o módulo do peso é pequeno, L1 reduz o novo custo mais que L2.

Outra abordagem para evitar *overfitting* é denominada de *Dropout*. Diferente da regularização, não é buscada alteração da função de custo, mas sim da estrutura de treinamento da rede. O comportamento dessa abordagem funciona selecionando neurônios das camadas intermediárias aleatoriamente e eliminando-os do processo de treinamento da rede (*backpropagation*), ou seja, nem o sinal de propagação positivo nem a retro propagação do sinal irá passar por aquele neurônio em uma determinada época (GOODFELLOW; BENGIO; COURVILLE, 2016).

Nessa abordagem é necessária a utilização dos *batchs* no conjunto de dados, que são lotes com um número reduzido de amostras do conjunto. A cada iteração do *backpropagation* um novo *batch* de neurônios é selecionado e os que tinham sido removidos anteriormente são restaurados. Esse processo faz com que os pesos que representam a rede sejam atualizados em função de cada *batch* (GOODFELLOW; BENGIO; COURVILLE, 2016). Para escolher qual conjunto vai representar o modelo, é verificado qual resultado é o mais frequente dentre os elementos e então, esse será utilizado. Apesar da robustez que essa abordagem demanda, a mitigação do *overfitting* é alcançada pois ao gerar vários elementos para esse subconjunto, a rede é treinada por diversas configuração de estrutura das camadas e dos pesos dos neurônios (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 9 mostra a rede com um *batch*, onde neurônios da camada intermediária são removidos em uma iteração do *dropout*.

Figura 9 – *Dropout* em um *batch* de neurônios.



Fonte: adaptado de (GOODFELLOW; BENGIO; COURVILLE, 2016)

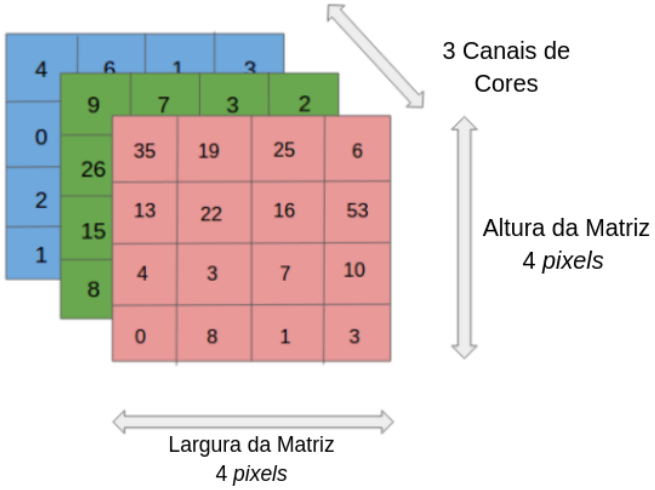
3.3.3 Redes Neurais Convolucionais

No contexto de classificação de imagens, RNAs com múltiplas camadas recebem nos seus neurônios de entrada cada pixel da imagem, atribuindo para cada neurônio a intensidade daquele pixel. O sucesso de aprendizado desse tipo de rede vai depender da diferença da intensidade dos pixeis das imagens, do conjunto de dados e da quantidade de processamento disponível para computar essas entradas (GOODFELLOW; BENGIO; COURVILLE, 2016). Com isso surge a Rede Neural Convolucional (Convolutional Neural Network - CNN) que utiliza atributos da estrutura espacial local da imagem para aprender classificar imagens (CHOLLET, 2018).

Uma CNN é uma estrutura de Aprendizado Profundo que permite captar uma imagem de entrada e atribuir valor a vários elementos dispostos na imagem sendo capaz de diferencia-los. As RNAs com múltiplas camadas realizam a aprendizagem extraindo informações globais da imagem, já as CNN aprendem a respeito de trechos locais utilizando filtros que deslizam sobre a matriz dos *pixels* da imagem extraindo informações desse trecho (CHOLLET, 2018).

Em CNNs reconhecer ou classificar imagens necessita realizar um processo onde suas entradas são interpretadas como matrizes tridimensionais com altura e largura, sendo o tamanho da imagem, e profundidade, determinada pela quantidade de canais de cores. Em geral as imagens utilizam três canais, RGB, com os valores de cada pixel (GOODFELLOW; BENGIO; COURVILLE, 2016). Na Figura 10 são ilustradas as matrizes que representam uma imagem 4x4 de altura e largura e os três canais de cores.

Figura 10 – Matrizes de representação de imagem utilizadas por uma CNN



Fonte: adaptado de (ACADEMY, 2019)

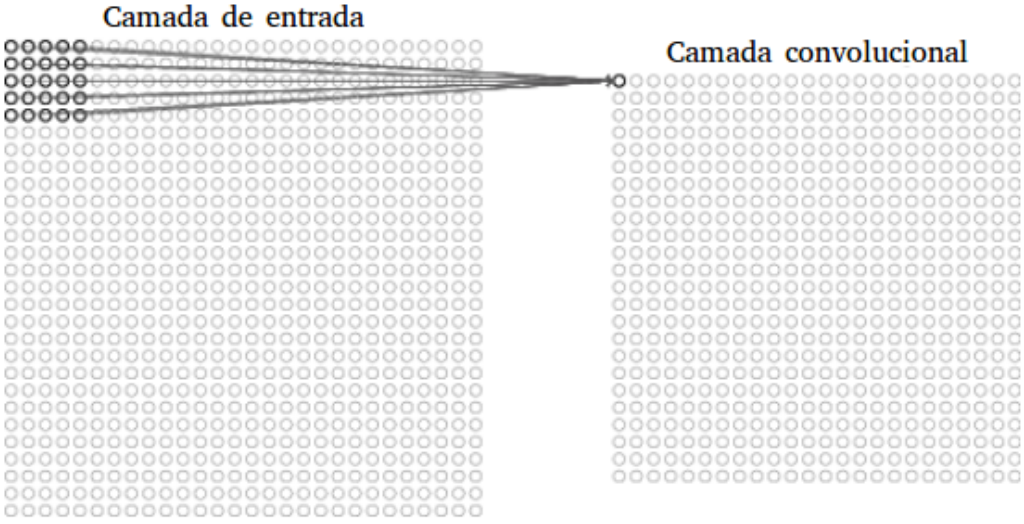
3.3.3.1 Camadas de uma Rede Neural Convolucional

O aprendizado das camadas de convolução das CNNs é mais flexível pois permite que a rede aprenda um certo padrão em partes específicas da imagem e depois reconhecê-lo o em qualquer região da imagem. Esse é um ponto positivo de aprendizagem das CNNs, tendo em vista a grande variedade de formas e padrões que as imagens do mundo real apresentam. Com isso, a rede precisa de menos amostras no conjunto de treinamento para aprender representações que tenham poder de generalização (CHOLLET, 2018).

Intuitivamente, a estrutura de uma CNN pode se comportar de forma que a primeira camada de convolução aprende pequenos padrões locais, uma segunda camada de convolução aprende padrões maiores feitos com os recursos das primeiras camadas, e assim por diante. Isso dá poder para que as CNNs aprendam eficientemente conceitos visuais cada vez mais complexos e abstratos (CHOLLET, 2018).

Na Figura 11 é vista a camada de entrada que representa uma imagem 28x28 *pixels* sendo mapeada pra uma camada convolucional 24x24. A região filtrada pelo neurônio receptor da camada convolucional extrai as características da região selecionada da camada inicial, e é feito progressivamente esse deslize da janela até atribuir todas as regiões 5x5 da camada de entrada para a nova camada(GOODFELLOW; BENGIO; COURVILLE, 2016).

Figura 11 – Mapeamento da camada de convolução



Fonte: adaptado de (ACADEMY, 2019)

Existem outros tipos de camadas, além das convolucionais, que são utilizadas por arquiteturas de Aprendizado Profundo buscando melhorar a performance de um modelo. As camadas em aprendizado profundo podem ser utilizadas na etapa de treinamento do modelo ou na etapa de classificação de imagens (CHOLLET, 2018). Algumas camadas que são utilizadas pelas arquiteturas usadas nesse trabalho são:

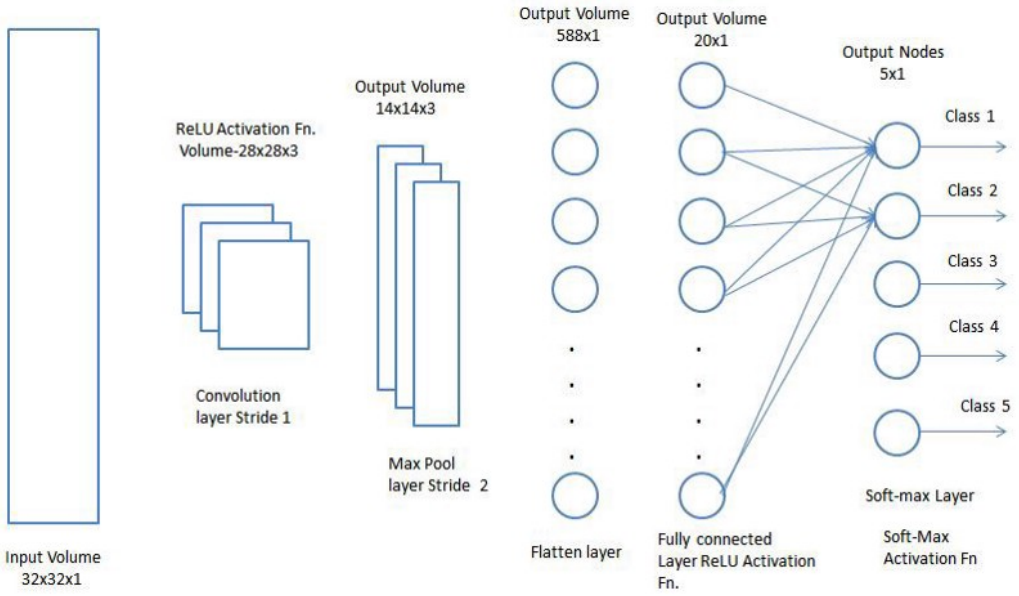
- Camada de *Pooling*
- Camada de *Flatten*
- Camada *Dense* ou *Fully Connected*

A camada de *Pooling* ou *Max-Pooling* recebe o resultado da camada convolucional e diminui ainda mais a dimensão desta camada, mas ao invés de extrair as características da região, é feita a extração dos principais valores representantes daquela região mapeada. Isso auxilia na redução de ocorrência de um possível *overfitting* do modelo, que poderia ocorrer caso a CNN recebesse muita informação não relevante para classificar a imagem (CHOLLET, 2018).

Depois que a camada de *pooling* cria a nova estrutura de valores que representam a camada convolucional, a camada de *Flatten* realiza um "achatamento" esses valores. Esse processo envolve a transformação de toda a matriz gerada pela camada anterior em um vetor unidimensional de neurônios, utilizada pela rede neural para processamento na próxima camada (CHOLLET, 2018).

A Camada *Dense* ou *Fully Connected* apresenta semelhanças às camadas intermediárias das RNAs, mas neste caso está totalmente conectada. Normalmente é a camada de saída, ou próxima a ela, onde obtemos as classes previstas. Ao final de uma época é verificado o resultado que o modelo produziu baseado em uma métrica, então o *backpropagation* é novamente chamado com o erro calculado pela rede. Nestas camadas são utilizadas outras funções que funcionam classificando cada parte aprendida da imagem (GOODFELLOW; BENGIO; COURVILLE, 2016). Na Figura 12 mostra a representação das camadas. Primeiro a imagem recebida originalmente tem um tamanho 32x32x1 e depois da convolução ela vai para 28x28x3 pelo mapeamento 5x5 que a camada de convolução realiza e pelos canais de cores; Após isso é mapeada para uma *Max-Pooling* que à redimensiona para um tamanho 14x14x3 mapeando e escolhendo seus representantes 2x2; A aplicação de camada de *flatten* vem em seguida, achatando a imagem para que na camada *fully-connected* seja mais fácil realizar a comunicação direta com os neurônios da rede, possibilitando a classificação.

Figura 12 – Camadas de representação da estrutura de uma CNN



Fonte: (ACADEMY, 2019)

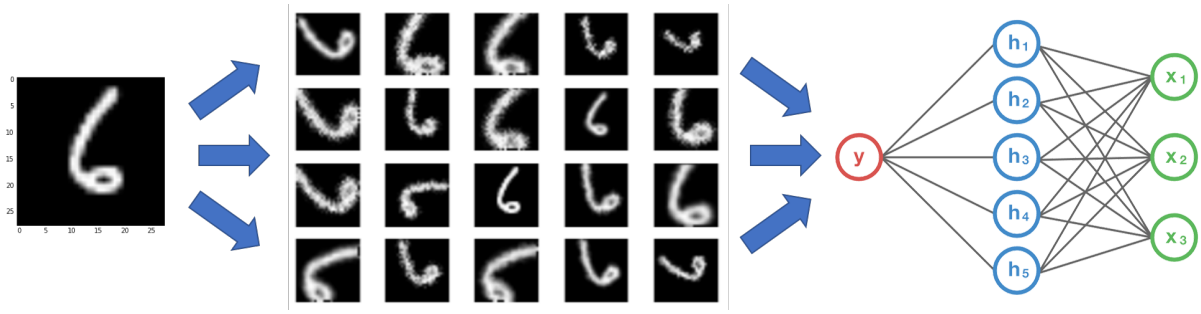
3.3.3.2 Data Augmentation

O *Data Augmentation* é uma técnica utilizada para gerar novas amostras de dados no conjunto de treino, com o objetivo de aumentar a generalidade do modelo. Com a aplicação dessa técnica é comum que a acurácia do conjunto de treino piore, a medida que a acurácia do conjunto de teste melhore, melhorando a generalização do modelo. A aplicação dessa técnica visa aumentar a robustez dos modelos, pois as amostras no conjunto de treino passam a mostrar mais variedade de características. Em alguns casos também é válido aplicar o *Data Augmentation* para tentar mitigar um problema de *overfitting* que o modelo venha a apresentar.

A técnica consiste em fazer alterações diretas nas imagens passadas para rede na tentativa de distorcer as características únicas de cada imagem e aumentar a capacidade de identificação de uma classe mesmo com uma imagem distorcida no mundo real. Na Figura 13 é mostrado como a técnica é aplicada a uma imagem genérica. As variações recorrentes reforçam o aprendizado da rede sobre características imutáveis de uma classe de imagens.

Neste trabalho os cenários de experimentação onde ocorrem a sua utilização, cada amostra do conjunto de treino passa por uma variação de orientação em torno de si mesma de 15º graus e uma variação de luminosidade de 0.2 à 1.0 em cada amostra do conjunto. Esses valores foram definidos pensando nas rotações e na luminosidade que o caso de uso real do tabuleiro pode apresentar.

Figura 13 – Exemplo da aplicação de *Data Augmentation*



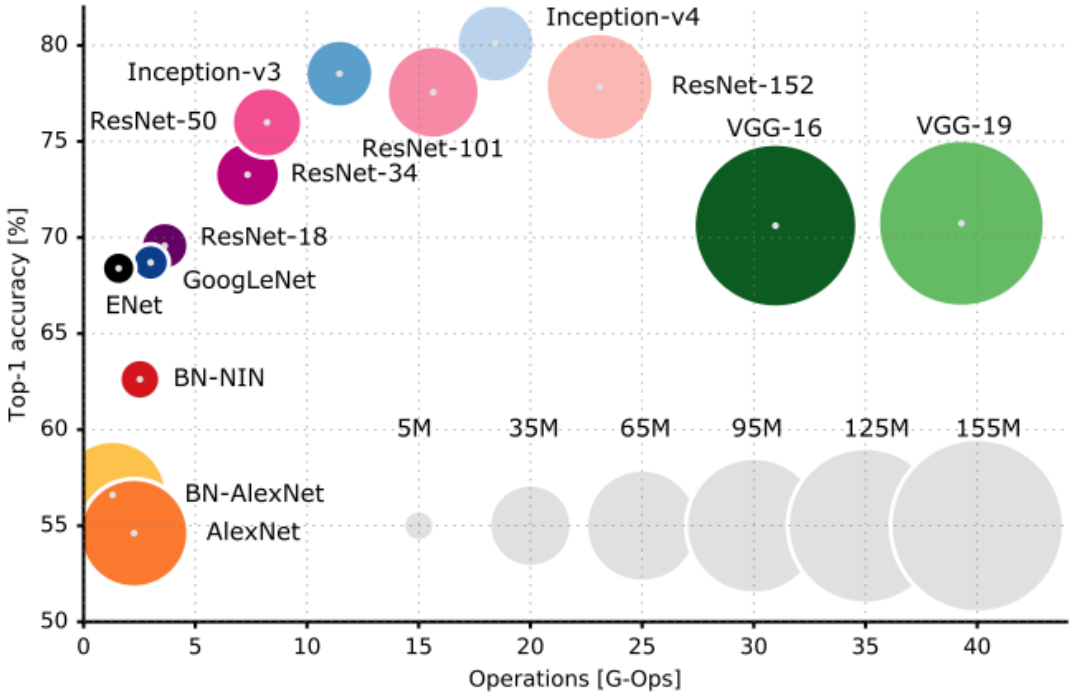
Fonte: Adaptado de (CHOLLET, 2018)

3.3.3.3 Arquiteturas de Aprendizado Profundo

A partir dessas estruturas de camadas alguns laboratórios de pesquisas começaram a propor seus próprios modelos de CNNs combinando várias camadas e modelagens de técnicas para conseguir aumentar suas acurácias. Nos últimos 10 anos, o desenvolvimento desses modelos de aprendizagem profunda tiveram bastante êxito em problemas de recognição de imagens, e suas principais motivações foram desafios como *ImageNet Large Scale Visual Recognition Challenge* (RUSSAKOVSKY et al., 2015). Nestes desafios os principais aspectos a serem resolvidos eram de detecção de objetos, localização de objetos e classificação de imagens. Eles propõem uma base de dados e os laboratórios testam seus modelos para cada categoria das bases e verificam o sucesso em geral dos modelos.

Em (CANZIANI; PASZKE; CULURCIELLO, 2016) é feita uma comparação das arquiteturas de Aprendizado Profundo mais conhecidas atualmente, realizando a avaliação não só do desempenho, mas da robustez necessária para alcançar uma boa performance sobre os dados utilizados nos modelos projetados. Na figura 14 podem ser vistas as mais variadas arquiteturas com seus respectivos desempenhos verificados em (CANZIANI; PASZKE; CULURCIELLO, 2016). Dentre essas, o trabalho presente utiliza duas específicas: **VGG-16** (SIMONYAN; ZISSERMAN, 2014) proposta pelo laboratório de pesquisa *Visual Geometry Group* e a **Inception V3** (SILVA, 2018) proposta pelo *Google*.

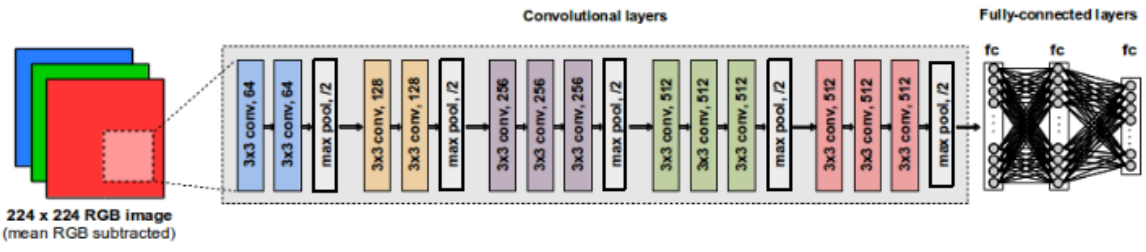
Figura 14 – Arquiteturas de CNN para recognição de imagens



Fonte: (CANZIANI; PASZKE; CULURCIELLO, 2016)

A **VGG-16** é bastante utilizada e atraente por causa de sua arquitetura muito uniforme com convoluções 3x3, muitos filtros e camadas de *pooling* entre as camadas de convolução, utilizadas para aumentar a dimensionalidade do mapa de características. Atualmente, é a escolha preferida na comunidade para extrair recursos das imagens (CANZIANI; PASZKE; CULURCIELLO, 2016). Os pesos desta arquitetura em todas as competições estão disponíveis publicamente e tem sido usada em muitas aplicações. Um problema na organização de suas camadas é que exige muita robustez computacional de processamento (QASSIM; VERMA; FEINZIMER, 2018). Na Figura 15 é vista a organização das camadas da VGG-16.

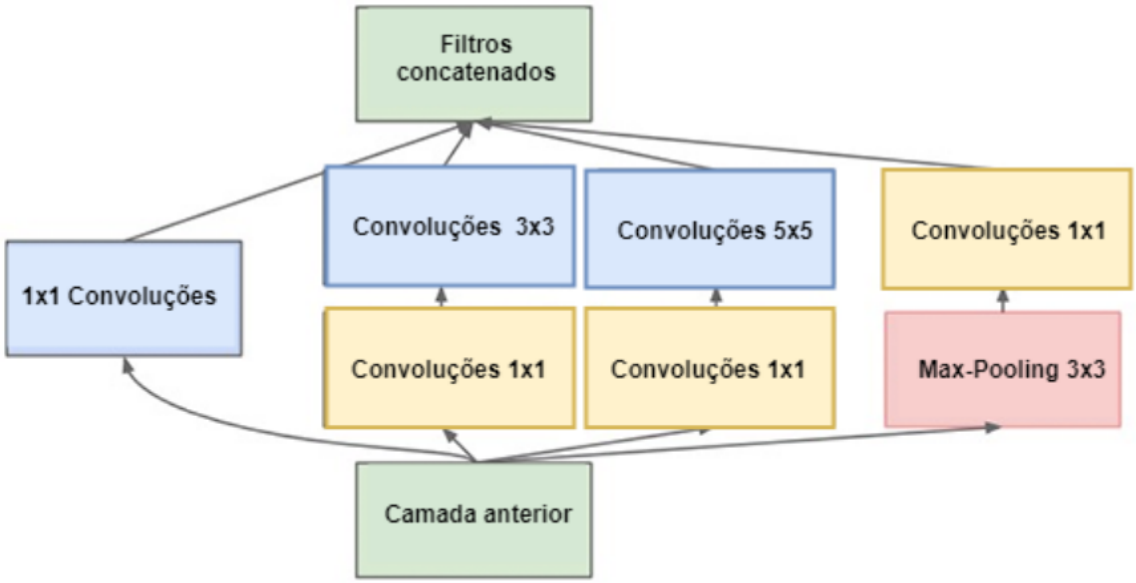
Figura 15 – Estrutura de Camadas VGG-16



Fonte: (REZENDE et al., 2018)

Já a **Inception V3** foi desenvolvida visando os problemas de classificação de imagens, foi treinada inicialmente nos conjuntos de dados do desafio ImageNet. O seu desempenho foi aceitável comparado a utilização de parâmetros desse modelo com os demais, resultando num baixo custo computacional (SZEGEDY et al., 2016). Na Figura 29 é mostrado como o módulo foi desenvolvido e como sua estrutura sem camadas *Denses* contribuem para essa redução de parâmetros.

Figura 16 – Módulo Inception V3



Fonte: (SILVA, 2018)

Neste trabalho, ambas arquiteturas são utilizadas e comparadas na classificação das regiões de interesse e das peças do tabuleiro. Os modelos criados a partir dessas arquiteturas serão experimentados com variações em alguns aspectos, como: Criar modelos a partir de pesos de modelos pré-treinados; Criar modelos sem pesos pré treinados, fazendo que esses modelos adéquem seus pesos desde a primeira iteração do processo de aprendizagem; Utilização de técnicas para aumentar a base de dados e realizar variações em cada imagens.

3.3.3.4 Métricas de avaliação

Para realizar a avaliação da eficiência de uma determinada atividade a qual um modelo de Aprendizado Profundo é submetido, é necessário utilizar métricas para poder medir o desempenho de tal modelo. As métricas utilizadas para mensurar os resultados alcançados de um modelo treinado, dependem da atividade a qual esse modelo se propõem a resolver.

Os modelos projetados ao decorrer deste trabalho realizam a atividade de classificar uma imagem para conseguir determinar a qual tipo de peça ou região de interesse aquela imagem é pertencente. Para realizar classificação de imagens as métricas mais comuns e que melhor avaliam o modelo são:

- Acurácia
- Perda de log (do inglês *Log Loss*)
- Precisão (do inglês *Precision*)
- Cobertura (do inglês *Recall*)
- *F1 Score*

A Acurácia de um modelo consiste em avaliar de forma objetiva a quantidade de imagens que foram classificadas corretamente dado o número total de imagens testadas em um determinado cenário. Ela é a principal métrica de avaliação de problemas de classificação e é esperado que ao decorrer das épocas do processo de treinamento do modelo, sua acurácia aumente a medida que o *log loss* diminua (RASCHKA; MIRJALILI, 2017). Na equação 3.6 é visto como funciona a acurácia. O número de classificações corretas do modelo (ClfTrue) dividido pelo número total de imagens que o modelo apresenta (NumTotal).

$$ACC = \frac{ClfTrue}{NumTotal} \tag{3.6}$$

O *log loss* pode ser considerada a métrica mais importante para avaliação do treinamento dos modelos. Ele indica a taxa de incerteza da previsão do modelo, se baseando no quanto a classificação do modelo varia do rótulo real. Com o *log loss*, o desempenho do modelo é avaliado mensurando o grau de imprecisão que um modelo aponta, ou seja, quão menor o valor de *log loss* maior a acurácia que o modelo apresenta. Na equação 3.7 é definido o *log loss*. Quando se trata de classificação de múltiplas classes, o *log loss* calcula o somatório da multiplicação das amostras preditas do modelo $(y_{o,c})$ pela probabilidade de validação que aquela amostra produziu, utilizando uma função logarítmica $(log(p_{o,c}))$.

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{3.7}$$

A classificação da classe de uma imagem que um modelo treinado realiza, ou deixa de realizar, pode divergir da classe real dessa imagem. O restante das métricas utiliza notações mais comuns na área de probabilidade, porém, que se relacionam diretamente a problemas de classificação. Quando o modelo classifica corretamente é um caso de Verdadeiro Positivo (do inglês *True Positive*, (TP)). Quando o modelo classifica uma imagem a uma classe que na verdade é outra, é uma caso de Falso Positivo (do inglês *False Positive*, (FP)). Quando há ausência de classificação do modelo a uma classe, que deveria ser classificada, é um caso de Falso Negativo (do inglês *False Negative*, (FN)).

A Precisão de um modelo pode ser considerada como sua métrica da exatidão. Caso haja uma baixa precisão para uma classe, isso pode indicar um grande número de falsos positivos. De forma objetiva, a Precisão é a razão de previsões corretas de uma classe (TP) para o total de previsões que o corretas que o modelo fez daquela classe (TP+FP). Na equação 3.8 é visto como é calculada a Precisão.

$$PRECISION = \frac{TP}{TP + FP} \tag{3.8}$$

A Cobertura, ou, *recall*, mede a sensibilidade de classificação do modelo. É feito o cálculo realizando a razão do número de amostras que foram classificadas corretamente a uma classe, que realmente são daquela classe (TP), para o total de classificações que ocorreram dessa classe (TP+FN), inclusive as que deixaram de ser preditas. Na equação 3.9 é visto como é calculada a Cobertura.

$$RECALL = \frac{TP}{TP + FN} \tag{3.9}$$

O *F1 Score* realiza a média harmônica da Precisão e de Cobertura. O resultado dessa métrica leva em conta tanto os as classificações que foram feitas de forma equivocada, quanto as que deveriam ter sido feitas. Quando se tem uma distribuição de classes desiguais, a *F1 Score* costuma se sobressair de forma negativa em relação as outras métricas. Na equação 3.10 é visto como é calculado o *F1 Score*.

$$F1Score = \frac{2 * (Recall * Precision)}{(Recall + Precision)} \tag{3.10}$$

4 RECONHECIMENTO DE PEÇAS DO JOGO PROGSTER

Neste capítulo são apresentados todos os passos da execução deste trabalho. Na Figura 17 pode ser visto todo o fluxo de trabalho que foi empregado. Inicialmente foi realizada a criação e coleta dos dados (Seção 4.1), logo após foi feito um pré processamento desses dados (Seção 4.2), para que pudesse ser feita a implementação das Redes Neurais Convolucionais (Seção 4.3). Após isso foram feitos uma análise dos resultados obtidos na implementação e nos cenários de experimentação (Seção 4.4).

Figura 17 – Passos Metodológicos.

Fonte: Elaborada pelo autor.

4.1 Criação e Coleta de Dados

O problema deste trabalho envolve um cenário controlado, o tabuleiro adaptado do jogo *Progster* (LEMOS et al., 2017) e suas regiões de interesse. A fase de coleta de dados foi efetivada através da criação de todas as imagens com configurações específicas definidas através da análise do problema. Estas imagens foram criadas visando fornecer uma amostra representativa das peças que são encontradas no jogo, que servirão de entrada para o treinamento das Redes Neurais.

4.1.1 Criação dos Dados

A criação inicial das imagens foi feita pela aluna Raíssa Barros de Oliveira Lemos, autora do *Progster* (LEMOS et al., 2017), e para essa atividade foram apresentados três exigências para que cada cenário de registro tivesse configurações específicas de distribuição dos elementos do tabuleiro, que são:

1. Cada peça deveria estar presente nos locais em que estaria num caso de uso real, em pelo menos cinco imagens, em diversos ângulos definidos aleatoriamente;
2. Cada espaço vazio também deveria estar presente em pelo menos cinco imagens;
3. A posição, em *pixels*, das extremidades do tabuleiro seguindo sempre a mesma ordem;

Para o primeiro ponto foi proposta a realização do registro do cenário em cinco ângulos com distâncias diferentes e aleatórias em relação ao tabuleiro, e que cada peça do tabuleiro em que haverá um caso de uso real fosse alocada nesses locais do tabuleiro com variação de rotação em torno da própria peça com o objetivo de diversificar o máximo possível a classe em que a aquela peça pertence.

Em seguida foi feito um aproveitamento de cenários, ou seja, na mesma imagem em que era feita a captação de um tipo de peça, parte do tabuleiro se encontrava vazio mas com a região de interesse do espaço vazio sendo captada para criação da classe desses espaços vazios.

Por último, todos os registros tiveram a identificação dos pontos de extremidade do tabuleiro realizada manualmente utilizando uma ferramenta de visualização e edição de imagens. A precisão da localização dos pontos extremos do tabuleiro é necessária para que na fase de coleta de dados a projeção perspectiva do tabuleiro seja feita de forma correta. Na continuidade deste trabalho, essa etapa deve ser automatizada para facilitar a criação de novos dados.

4.1.2 Coleta dos Dados

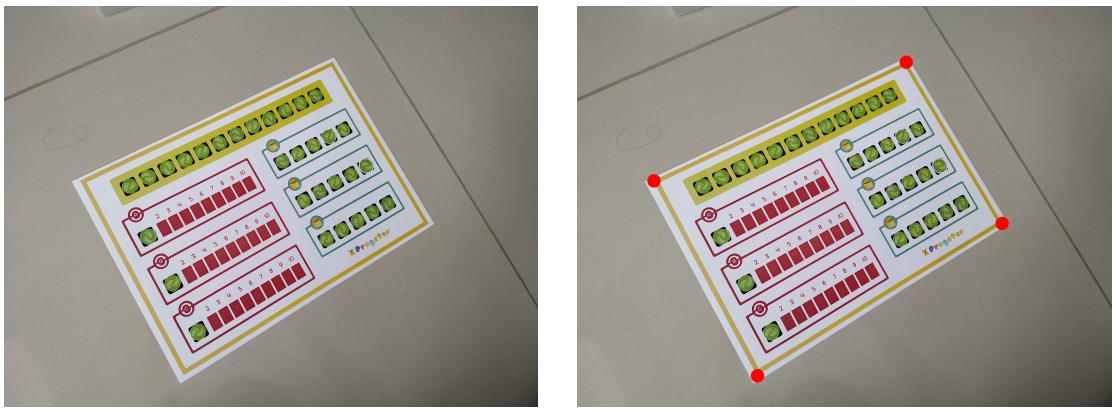
Após a criação dos registro de cenários de caso de uso do tabuleiro, a etapa de coletar os dados para ajusta-los às configurações dos métodos de aprendizagem profundo contém dois passos:

- Projeção perspectiva do tabuleiro, utilizando os pontos de extremidades;
- Localização dos pontos das regiões de interesse do tabuleiro, ou seja, o mapeamento de todos os locais possíveis onde uma peça ou um espaço vazio seja identificado;

Os pontos citados acima foram feitos utilizando a biblioteca de Visão Computacional *OpenCV*[1], com auxílio do ambiente *Jupyter Notebook*[2]. A biblioteca *OpenCV* foi desenvolvida no começo dos anos 2000 e é livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos e pesquisas na área de Visão Computacional. Já o *Jupyter Notebook* é um ambiente computacional web que funciona através de células interativas de execução independentes, fazendo com que a organização e análise dos experimentos com grandes volume de dados seja ágil.

A projeção perspectiva do tabuleiro consiste em, dado os quatro pontos extremos do tabuleiro, é feita a transformação do plano formado por esses pontos para a visão perspectiva, fixando os pontos extremos do tabuleiro nos extremos da nova imagem. Nas imagens da Figura 18 está presente o registro original do cenário e os pontos de extremidade do tabuleiro.

Figura 18 – Exemplo de registro de um cenário e identificação dos pontos de extremidade do tabuleiro.

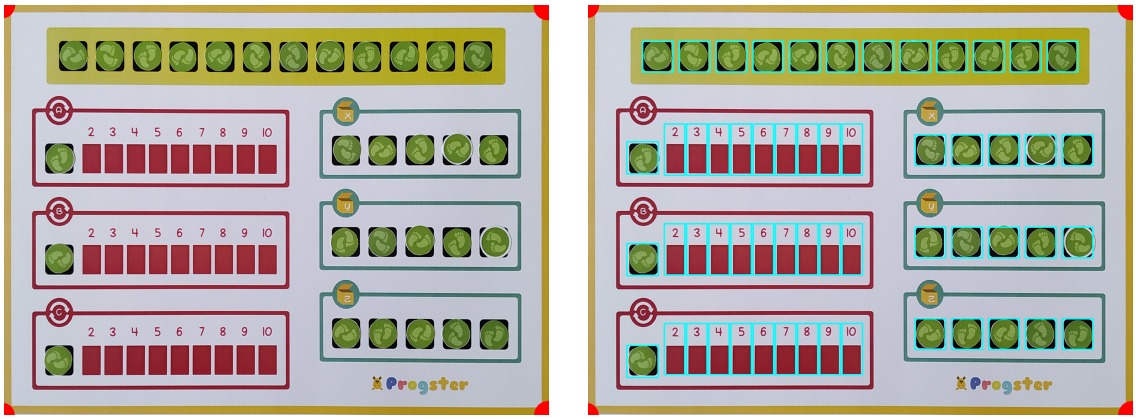


Fonte: Elaborado pelo autor.

Com os pontos bem definidos, há a projeção do plano e o resultado do exemplo da Figura 18 é mostrado na Figura 19, assim são selecionadas as regiões de interesse que são todas as regiões onde pode ter uma peça do tabuleiro, seja peça de instrução, função ou a seta indicando quantidade de repetições. Depois da transformação perspectiva houve o trabalho manual de identificar os pontos das regiões de interesse e na Figura 19 são revelados todas as regiões de interesse.

[1] https://opencv.org/
[2] https://jupyter.org/

Figura 19 – Transformação perspectiva do tabuleiro e seleção de regiões de interesse.



Fonte: Elaborado pelo autor.

A região de interesse do tabuleiro de cada imagem criada é coletada seguindo os passos descritos anteriormente, e a distribuição das imagens é idealizada com o objetivo de criar o maior número de imagens para cada tipo de peça a ser coletada do tabuleiro. Na Figura 27 são vistos alguns tipos de classes de imagens do tabuleiro. No Apêndice A podem ser vistas as imagens do restante das classes.

Figura 20 – Exemplos de peças do tabuleiro postas em regiões de interesse.

Fonte: Elaborado pelo autor.

O tabuleiro é composto por 29 classes de tipos de peças diferentes, que são: andar, pular, subir, descer, função x, função y, função z, repetição um, repetição dois, repetição três, o vazio, que é catalogado por ausência de alguma peça, 9 classes de números de repetições verdadeiros e 9 falsos dentro da função de repetição. Todas as imagens mostradas acima já são resultados obtidos com auxilio da biblioteca *OpenCV*, utilizada para realizar a projeção perspectiva do tabuleiro e o recorte das regiões de interesse.

4.2 Análise e Pré-processamento dos Dados

Com a criação e coleta de todos os dados, esta etapa consiste em analisar os dados e organizá-los para que as arquiteturas de aprendizagem profunda possam ser implementadas. Cada arquitetura que trata problemas de aprendizagem profunda tem suas características individuais e de parametrização. As arquiteturas utilizadas nesse trabalho são a *VGG-16* (**??**) e a *InceptionV3* (SZEGEDY et al., 2016), ambas escolhidas pela robustez, escalabilidade e eficácia, apontada em (CANZIANI; PASZKE; CULURCIELLO, 2016). Cada imagem de uma classe é salva em um diretório específico, pois é através dessa configuração que as funções de pré-processamento da biblioteca *Keras*[3] trabalha, para que seja feito o pré-processamento individual de cada arquitetura.

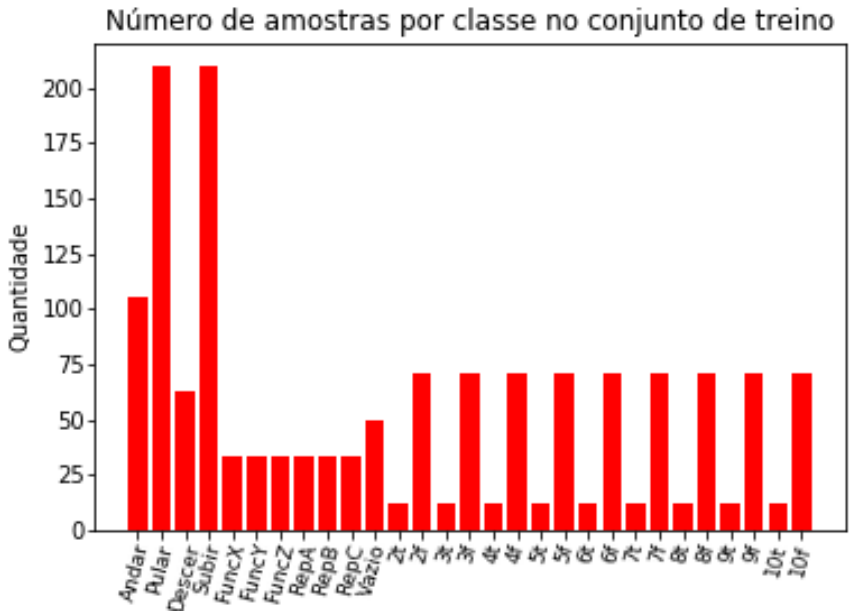
Com cada classe definida, é feita a seleção e separação dos dados de cada classe aleatoriamente para um conjunto de treino com 70% das imagens e um conjunto de validação com 30% das imagens. Esses conjuntos são utilizados na etapa de treinamento do modelo. Enquanto os dados do conjunto de treino são utilizados para de fato realizar o treinamento, o conjunto de validação é utilizado para avaliar a evolução do modelo ao decorrer do processo de aprendizagem. Esse tipo de abordagem é comum em problemas de aprendizagem profundo para garantir uma boa avaliação de acurácia de um modelo treinado. No total, foram alocados ao conjunto de treino 1608 imagens e ao conjunto de validação 672 imagens.

O conjunto de dados de teste foi criado separadamente dos anteriores e tem o total de 549 imagens. Esse conjunto é importante para avaliar a acurácia do modelo em imagens que ele não utilizou para realizar a etapa de treinamento. O resultado de acurácia desse conjunto é o que nos revela realmente como seria a aplicação de um modelo em um cenário real.

Ambos conjuntos estão distribuídos entre 29 classes que representam as regiões de interesse do tabuleiro. Nas Figuras 21, 22 e 23 são fornecidos gráficos da distribuição de amostras de cada classe nos conjuntos utilizados.

[3] https://keras.io

Figura 21 – Distribuição de imagens no conjunto de Treino

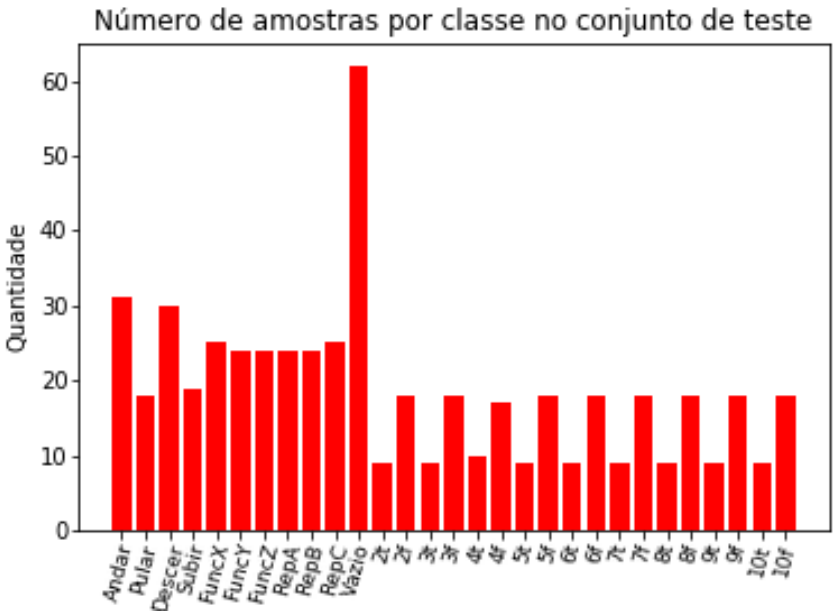


Fonte: Elaborado pelo autor.

Figura 22 – Distribuição de imagens no conjunto de Validação

Fonte: elaborado pelo autor.

Figura 23 – Distribuição de imagens no conjunto de Teste



Fonte: elaborado pelo autor.

A desproporção da quantidade de elementos existentes em cada classe de imagens nos conjuntos de dados, vista nos gráficos das Figuras 21, 22 e 23, se deve ao fato que na etapa de criação de dados não foi realizado o registro recomendado da quantidade de imagens planejada para cada tipo de peça. Na continuidade deste trabalho, deverá ser feito uma redistribuição na quantidade de amostras de cada classe do problema.

Além da configuração disposta para a organização dos diretórios das classe, existem outras etapas de pré-processamento. Tanto na *VGG-16* quanto na *InceptionV3* foi feito um redimensionamento de cada imagem para 128x128 *pixels* de forma que estas se adéquem à camada de entrada das arquiteturas, e a partir disso seja feita a extração de características de cada classe.

Tanto a etapa de análise e pré-processamento dos dados quanto a etapa de implementações dos modelos, foram feitas na plataforma *Kaggle Kernels*[4], um ambiente com grande desempenho computacional em nuvem. Por meio desse ambiente é possível realizar a implementação desse tipo de modelo, pois com a alta demanda de processamento que as Redes Neurais Convolucionais demandam, realizar esses experimentos em maquinas comum se torna inviável. Atualmente o ambiente dispõe de um processador Intel Xeon 2.30GHz, 16 Gb de memória RAM e uma placa de vídeo NVIDIA-SMI 390.25, Tesla K80.

[4] https://www.kaggle.com/kernels

4.3 Implementação das arquiteturas de Aprendizado Profundo

Com os dados pré-processados e ajustados para utilização na etapa de treino do modelo, implementamos as arquiteturas escolhidas. Para isso foi utilizada a biblioteca *Keras* com a linguagem *Python* no ambiente de desenvolvimento *Kaggle Kernels*. Para a implementação das arquiteturas, cada etapa é bem definida:

- Inicialmente a arquitetura é carregada utilizando três hiper-parâmetros: uma variável booleana para indicar a utilização da última camada densa padrão, as dimensões das imagens do conjunto de dados e os pesos utilizados na inicialização da rede;
- Com o modelo carregado, é feita uma adaptação na distribuição das camadas com a inserção de três camadas na saída dos modelos: uma camada *flatten* para criar o vetor de características extraídas de cada imagem utilizada, uma camada *dense* com função de ativação *Relu* para filtrar os neurônios que correspondem melhor o aprendizado de cada classe, e uma última camada *dense* com função de ativação *softmax* e 29 neurônios, indicando cada classe do problema;
- Por fim, o modelo é compilado utilizando alguns hiper-parâmetros: o otimizador *Adam*, uma versão melhorada do gradiente descendente que serve para auxiliar no ajuste dos pesos da etapa de treinamento do modelo; função objetivo *log loss* que indica a taxa de erro de cada época mostrando o comportamento de aprendizagem da rede; e a métrica de avaliação do modelo (acurácia);

Além da escolha das arquiteturas, foram propostos oito cenários de experimentação para avaliar o comportamento das redes dado o conjunto de dados do problema, os cenários são:

1. *Inception V3*: com pesos de modelos pré treinados;
2. *VGG 16*: com pesos de modelos pré treinados;
3. *Inception V3*: com pesos de modelos pré treinados e *data augmentation*;
4. *VGG 16*: com pesos de modelos pré treinados e *data augmentation*;
5. *Inception V3*: sem pesos de modelos pré treinados;
6. *VGG 16*: sem pesos de modelos pré treinados;
7. *Inception V3*: sem pesos de modelos pré treinados e *data augmentation*;
8. *VGG 16*: sem pesos de modelos pré treinados e *data augmentation*;

Nos cenários de experimentos com a arquitetura *Inception V3* com pesos de modelos pré-treinados, foi aplicado um congelamento nas 32 primeiras camadas pois são camadas que buscam extrair características muito básicas, e como as imagens do conjunto de dados foram bem pré-processadas não houve a necessidade de retreinar esses pesos. Para os cenários onde a rede utilizaria pesos pré-treinados foram utilizadas 25 épocas de treinamento, e nos cenários que a rede ajustaria os pesos somente com as imagens dadas do conjunto de dados, foram utilizadas 50 épocas de treinamento. Esta quantidade de épocas foi suficiente para observar a convergência da acurácia.

Com as arquiteturas projetadas, é iniciado o processo de treinamento de cada arquitetura. Para isso, são passados como parâmetro:

- O número de épocas, tal que uma época consiste em processar uma quantidade de *batches* que corresponda a processar todo o conjunto de treino;
- O *batch size*, que corresponde ao número de imagens processadas simultaneamente no algoritmo de treinamento (*backpropagation*) até que se termine uma época;
- Dados de treino e validação;

Para coletar os resultados do treinamento, é utilizada a função *history* que realiza as predições de cada cenário nos conjuntos de dados, mostrando a cada época, o valor da acurácia e o *log loss*, verificando a confiança de aprendizado da rede.

Além disso foram implementados relatórios de classificação para cada modelo, avaliando os modelo dos cenários de experimentação nas métricas de Precisão, Cobertura e *F1 Score*. Neste relatório de classificação, também foram feitas as avaliações das acurácias dos modelos criados sobre o conjunto de dados de teste.

4.4 Análise dos Resultados

Nesta etapa é feita a análise e verificação da acurácia dos modelos criados. Inicialmente, a acurácia é verificada tanto no conjunto de treino quanto no conjunto de validação do modelo. Outra métrica analisada é o *log loss*, que mede a incerteza da previsão do modelo com base em quanto ela varia do rótulo real. O conjunto de teste foi utilizado para realizar a avaliação dos modelos criados dos cenários de experimentação, verificando as métricas de Acurácia, Precisão, Cobertura e *F1 Score*.

O mais importante a ser avaliado é o como essas arquiteturas irão se comportar perante o conjunto de dados que foi criado neste trabalho. Uma vez que essas arquiteturas são propostas para enfrentar desafios mais robustos, um problema num ambiente controlado onde as regiões de interesse são bem captadas e estáticas, com variação apenas nas angulações e na luminosidade que um caso de uso pode trazer, é esperado que a performance seja boa, em todas as configurações propostas.

No Quadro 2 são apresentados os valores de acurácia e *log loss* de cada modelo dos cenários experimentados na etapa de treinamento. Os valores mostrados correspondem ao resultado de cada cenário de experimentação ao final das épocas de treinamento que cada modelo foi submetido.

Quadro 2 – Taxa de *log loss* e da acurácia de todos os cenários, ao fim das épocas de treinamento.

	Log Loss (treino)	*Log Loss (val)*	*Acurácia (treino)*	*Acurácia (val)*
Inception V3 com pesos	0.0114	0.0606	0.9962	0.9743
VGG 16 com pesos	9.0955e-06	0.0028	1.0000	0.9985
Inception V3 com pesos e D. A	0.0045	0.0298	0.9994	0.9955
VGG 16 com pesos e D. A	2.3082e-05	0.0056	1.0000	0.9985
Inception V3 sem pesos	0.0771	0.2263	0.9737	0.9365
VGG 16 sem pesos	8.1773e-05	0.0619	1.0000	0.9849
Inception V3 sem pesos e D. A	0.0478	0.1388	0.9845	0.9667
VGG 16 sem pesos e D. A	0.0066	0.0938	0.9981	0.9894

Fonte: Elaborado pelo autor.

É evidenciado no Quadro 2 que os menores valores de *log loss* do conjunto de treino estão presentes nos cenários 2, 4 e 6, todos utilizando arquiteturas VGG-16. No conjunto de validação, os menores valores de *log loss* encontrados estão presentes nos cenários 2 e 4, com a utilização da VGG-16, com pesos pré treinados e com *data augmentation*. Em relação às acurácias, no conjunto de treino os melhores cenários também foram o 2, 4 e 6, e no conjunto de validação foram os cenários 2 e 4, mostrando a relação que o *log loss* do aprendizado da rede está diretamente ligada com o acurácia do modelo. Fica evidente então que para o problema deste trabalho a arquitetura VGG-16 se destacou na etapa de treinamento do modelo.

Caso o valor de acurácia da validação não fosse tão alto quanto o da acurácia de treino, poderia indicar um problema de *overfitting*, porém, o valor de acurácia no conjunto de validação também foi satisfatório em todos os cenários de experimentação.

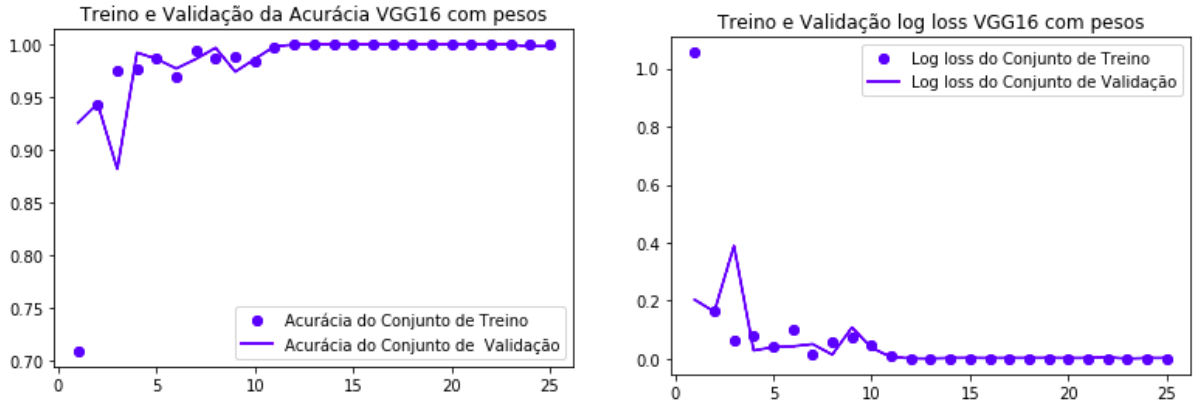
Após a etapa de treinamento, os modelos 2 e 4 ,que apresentaram melhores resultados na etapa de treinamento, foram submetidos a realizar a predição nas imagens do conjunto de teste utilizando os pesos aprendidos em cada cenário de experimentação. Para verificar o desempenho dos modelos treinados no conjunto de teste, foi realizada a verificação da acurácia de cada classe identificando o número de classificações certas dos modelos dado o total de imagens que cada classe continha. Para determinar a acurácia total do modelo, foi calculada a média das acurácias de todas as classes de cada modelo. O modelo que apresentou a melhor acurácia durante essa etapa foi o do cenário 4, que utiliza VGG-16 com pesos de modelos pré treinados e aplicação de *Data Augmentation*, apresentando uma acurácia de 99.37%.

Sendo condizente com a etapa de treinamento, os cenários que mostraram os melhores valores de acurácia sobre o conjunto de teste foram os que utilizaram a arquitetura VGG 16. Também foi notável que os cenários que utilizam pesos pré-treinados se mostraram bem superiores aos cenários que não fizeram a utilização desses pesos. Isso ocorre devido ao número insatisfatório de imagens de cada classe no conjunto de dados utilizados na etapa de treinamento. Quando são utilizados esses pesos de modelos pré-treinados, o aprendizado sobre características mais genéricas de uma imagem já estão preconcebidos, enquanto quando não se usa esses pesos a rede tenta realizar esse aprendizado ao longo do treinamento, deixando de lado parte das características que diferenciam cada classe do conjunto de dados deste trabalho.

Nas Figuras 24, 25 e 26 são vistos os comportamentos da acurácia e *log loss*, ao longo das épocas de treinamento pré estabelecidas dos três melhores cenários de experimentação na etapa de treinamento. Nesses gráficos o eixo horizontal representa o número das épocas do modelo e o eixo vertical a acurácia e o *log loss*. O restante das figuras que mostram esse comportamento para os demais cenários são encontrados no Apêndice B.

Na Figura 24 o gráfico da acurácia mostra que a partir da quinta época, as acurácias do conjunto de treino e validação já estavam acima de 95%, e a partir da época 11, as acurácias do conjunto de treino já se mostravam muito próximas de 100% de acurácia. O gráfico do comportamento de *log loss* desse cenário mostra que o valor de *log loss* diminui rapidamente, chegando a valores muito baixos à medida que as épocas passam. Houveram pequenas discrepâncias nas primeiras épocas do treinamento do modelo.

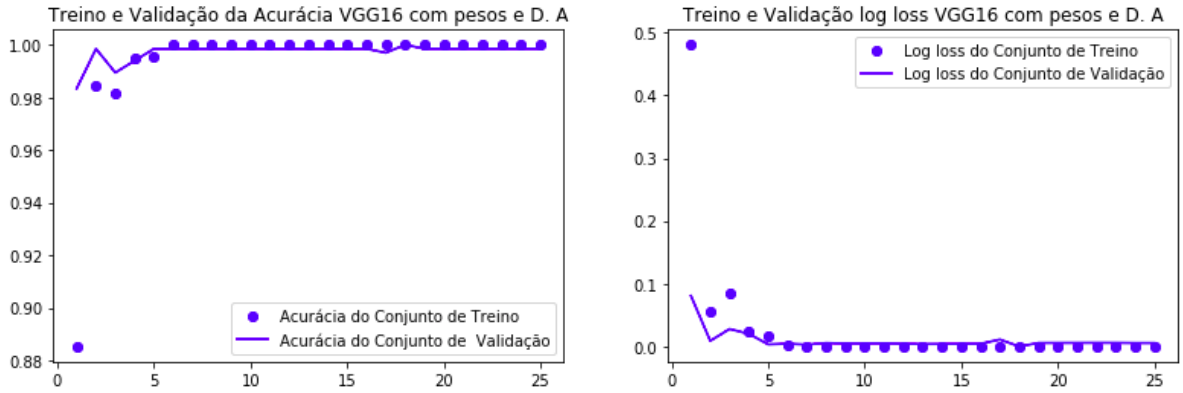
Figura 24 – Gráfico de comportamento VGG16 com pesos pré treinados.



Fonte: Elaborado pelo autor.

A Figura 25 mostra que durante quase toda distribuição das épocas de aprendizagem o conjunto de validação teve valores muito próximos comparado aos valores do conjunto de treino. No gráfico do *log loss* a maior discrepância ocorreu na primeira época no conjunto de treino, após isso ficou estável.

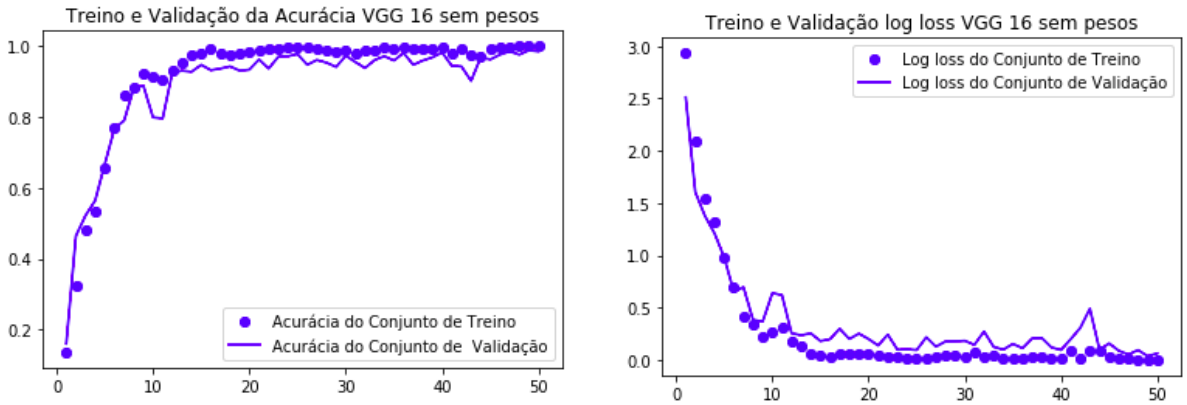
Figura 25 – Gráfico de comportamento VGG 16 com pesos pré treinados e *data augmentation*.



Fonte: Elaborado pelo autor.

A Figura 26 mostra o gráfico do cenário que utiliza a arquitetura VGG 16 sem pesos pré treinados. Comparando com as figuras anteriores, fica visível que quando não é utilizado pesos de modelos pré treinados há uma certa demora para alcançar valores maiores que 90% de acurácia. Fica nítido também que não há uma regularidade tão grande durante as épocas de treinamento, apresentando pequenas discrepâncias durante as épocas. O gráfico comportamento de *log loss* mostra que nas primeiras épocas os valores são bastante altos comparados dos cenários anteriores, reforçando ainda mais a ideia de aprendizado de características genéricas que os pesos pré-treinados contém.

Figura 26 – Gráfico de comportamento VGG16 sem pesos pré treinados.



Fonte: Elaborado pelo autor.

Nos Quadros 3 e 4 são vistos os resultados das métricas de Precisão, Cobertura e *F1 Score* dos cenários de experimentação 2 e 4 sobre o conjunto de teste. Os cenários de experimentação 2 e 4 apresentaram os melhores resultados na etapa de treinamento dos modelos .

Quadro 3 – Precisão, Cobertura e F1-Score do modelo com a VGG 16 e pesos pré treinados sobre o conjunto de teste.

	Precisão	**Cobertura**	**F1-Score**	**Nº de IMG**
10f	1.00	0.83	0.91	18
10t	1.00	1.00	1.00	9
2f	1.00	1.00	1.00	18
2t	1.00	1.00	1.00	9
3f	1.00	1.00	1.00	18
3t	1.00	1.00	1.00	9
4f	1.00	1.00	1.00	17
4t	1.00	1.00	1.00	10
5f	1.00	1.00	1.00	18
5t	1.00	1.00	1.00	9
6f	1.00	1.00	1.00	18
6t	1.00	1.00	1.00	9
7f	1.00	1.00	1.00	18
7t	1.00	1.00	1.00	9
8f	1.00	1.00	1.00	18
8t	1.00	1.00	1.00	9
9f	0.86	1.00	0.92	18
9t	1.00	1.00	1.00	9
andar	1.00	1.00	1.00	31
descer	1.00	1.00	1.00	30
fX	1.00	1.00	1.00	25
fY	0.83	1.00	0.91	24
fZ	1.00	0.79	0.88	24
pular	1.00	1.00	1.00	18
repet_a	0.96	1.00	0.98	24
repet_b	1.00	1.00	1.00	24
repet_c	1.00	0.96	0.98	25
subir	1.00	1.00	1.00	19
vazio	1.00	1.00	1.00	62

Fonte: Elaborado pelo autor.

Quadro 4 – Precisão, Cobertura e F1-Score do a VGG 16, pesos pré treinados e Data Augmentation sobre o conjunto de teste.

	Precisão	**Cobertura**	**F1-Score**	**Nº de IMG**
10f	1.00	0.94	0.97	18
10t	1.00	1.00	1.00	9
2f	1.00	1.00	1.00	18
2t	1.00	1.00	1.00	9
3f	1.00	1.00	1.00	18
3t	1.00	1.00	1.00	9
4f	1.00	1.00	1.00	17
4t	1.00	1.00	1.00	10
5f	1.00	1.00	1.00	18
5t	1.00	1.00	1.00	9
6f	1.00	1.00	1.00	18
6t	1.00	1.00	1.00	9
7f	1.00	1.00	1.00	18
7t	1.00	1.00	1.00	9
8f	1.00	1.00	1.00	18
8t	1.00	1.00	1.00	9
9f	0.95	1.00	0.97	18
9t	1.00	1.00	1.00	9
andar	1.00	0.97	0.98	31
descer	0.94	1.00	0.97	30
fX	1.00	1.00	1.00	25
fY	0.86	1.00	0.92	24
fZ	1.00	0.79	0.88	24
pular	1.00	1.00	1.00	18
repet_a	1.00	1.00	0.98	24
repet_b	1.00	1.00	1.00	24
repet_c	1.00	1.00	1.00	25
subir	1.00	1.00	1.00	19
vazio	1.00	1.00	1.00	62

Fonte: Elaborado pelo autor.

Comparando os quadros fica visível que o quadro 4 tem melhores resultados em algumas classes. Em ambos cenários, quase todas as classes tiveram resultados da Precisão, Cobertura e F1-Scores positivos, tendo algumas discrepâncias principalmente nas classes onde no conjunto de treinamento tiveram menos imagens para treinar.

É importante enfatizar que quando uma imagem não é classificada corretamente, a execução do jogo Progster pode ser afetada por não interpretar corretamente o cenário do tabuleiro. Com isso, a métrica de acurácia deve ser o mais próximo possível de 100%.

A partir da análise dos experimentos realizados é possível perceber que o cenário de experimentação que mais obteve êxito foi o cenário 4, que conta com a arquitetura VGG-16, com pesos de modelos pré-treinados e *Data Augmentation*. Além de se sobressair na etapa de treinamento do modelo, apresentado os melhores valores de acurácias e *log loss* no conjunto de treino e validação, também apresentou melhor desempenho quando lidou com um conjunto de

dados totalmente desconhecido para os modelos, apresentando 99.37% de acurácia.

5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo criar modelos de Aprendizagem Profunda para realizar a classificação das regiões de interesse do tabuleiro do jogo Progster (LEMOS et al., 2017). A motivação da execução deste trabalho é fazer parte de um processo da adaptação do jogo Progster, que está em desenvolvimento em paralelo a este trabalho pela aluna Raíssa Barros de Oliveira. Esta adaptação foi idealizada pensando na acessibilidade do jogo, uma vez que, originalmente o jogo proposto requer a utilização de um tabuleiro eletrônico que aumenta os custos de produção e nem todo o público teria acesso.

Parte dos trabalhos acadêmicos realizados na área de Aprendizagem Profundo, utilizam banco de imagens coletados por terceiros e de desafios, e são focados na aplicação das técnicas, buscando resultados mais expressivos. Porém, este trabalho envolvia um ambiente mais específico, onde o cenário utilizado nunca havia se colocado para realização de testes nessas área. Por isso, além das técnicas de Aprendizagem Profunda para classificação das imagens, foram utilizadas técnicas de Visão Computacional para criação da base de dados.

A etapa da criação dos dados mostrou que, utilizando técnicas de Visão Computacional, foi dispensada a realização manual da captação de cada região de interesse. Realizar essa atividade de forma manual certamente demandaria mais tempo e esforço do que criar um ambiente que realiza essa captação automaticamente, inclusive para aumento da base de dados em algum momento futuro. No total, foram criadas 2759 imagens distribuídas entre 29 classes diferentes.

Os resultados obtidos pelos modelos de Aprendizagem Profunda foram realmente satisfatórios, pois, dentre os cenários de experimentação a melhor acurácia foi de 99.37%. Era esperado um bom resultado já que na etapa de criação dos dados foi observado que cada classe de imagens tinham várias características divergentes e todas regiões de interesse foram fixadas no centro da imagem para que os modelos de Aprendizagem Profunda, com sua robustez, conseguissem diferenciá-las satisfatoriamente.

Fica visível que a utilização de Aprendizado Profundo, como processo em um projeto maior, também pode ser aplicado. Graças aos laboratórios de pesquisa que realizam a criação das arquiteturas de redes convolucionais e ao François Chollet, este trabalho obteve sucesso e irá se integrar a um projeto relevante. Espero que futuramente este trabalho sirva de inspiração para que pessoas visem a aplicação desses métodos em problemas diversos que contribuam de alguma forma com o desenvolvimento de uma área ou auxilie a população.

REFERÊNCIAS

ACADEMY, D. S. **Deep Learning Book**. Data Science Academy, 2019. Disponível em: <http://deeplearningbook.com.br/>. Acesso em: 23 abr. 2019.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV: Computer vision with the OpenCV library**. [S.l.]: "O'Reilly Media, Inc.", 2008.

CALDERON-CORDOVA, C.; GUAJALA-MICHAY, M.; BARBA-GUAMAN, R.; QUEZADA-SARMIENTO, P. Design of a machine vision applied to educational board game. In: **2016 6th International Workshop on Computer Science and Engineering, WCSE 2016**. [S.l.: s.n.], 2016. p. 808–811.

CANZIANI, A.; PASZKE, A.; CULURCIELLO, E. An analysis of deep neural network models for practical applications. **arXiv preprint arXiv:1605.07678**, 2016.

CHEN, A. T.-Y.; WANG, K. I.-K. Robust computer vision chess analysis and interaction with a humanoid robot. **Computers**, Multidisciplinary Digital Publishing Institute, v. 8, n. 1, p. 14, 2019.

CHOLLET, F. **Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek**. [S.l.]: MITP-Verlags GmbH & Co. KG, 2018.

GARCIA-GARCIA, A.; ORTS-ESCOLANO, S.; OPREA, S.; VILLENA-MARTINEZ, V.; GARCIA-RODRIGUEZ, J. A review on deep learning techniques applied to semantic segmentation. **arXiv preprint arXiv:1704.06857**, 2017.

GONZALEZ, R. C.; WOODS, R. E. Object recognition. **Digital image processing, 3rd ed. Pearson**, p. 861–909, 2008.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016.

HAYKIN, S. S. et al. **Neural networks and learning machines/Simon Haykin.** [S.l.]: New York: Prentice Hall,, 2009.

JOHNSON, G. W. **LabVIEW graphical programming**. [S.l.]: Tata McGraw-Hill Education, 1997.

LEE, S. M.; SEO, J. B.; YUN, J.; CHO, Y.-H.; VOGEL-CLAUSSEN, J.; SCHIEBLER, M. L.; GEFTER, W. B.; BEEK, E. J. van; GOO, J. M.; LEE, K. S. et al. Deep learning applications in chest radiography and computed tomography: Current state of the art. **Journal of Thoracic Imaging**, LWW, 2019.

LEMOS, R.; ALVES, A.; BARROSO, G.; MONTEIRO, I. Progster: aprendendo lógica de programação com um tabuleiro eletrônico. **XVI Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais - IHC 2017**, 2017.

MOLLA, E.; LEPETIT, V. Augmented reality for board games. In: IEEE. **2010 IEEE International Symposium on Mixed and Augmented Reality**. [S.l.], 2010. p. 253–254.

NAJAFABADI, M. M.; VILLANUSTRE, F.; KHOSHGOFTAAR, T. M.; SELIYA, N.; WALD, R.; MUHAREMAGIC, E. Deep learning applications and challenges in big data analytics. **Journal of Big Data**, Nature Publishing Group, v. 2, n. 1, p. 1, 2015.

QASSIM, H.; VERMA, A.; FEINZIMER, D. Compressed residual-vgg16 cnn model for big data places image recognition. In: IEEE. **2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)**. [S.l.], 2018. p. 169–175.

RASCHKA, S.; MIRJALILI, V. **Python machine learning**. [S.l.]: Packt Publishing Ltd, 2017.

REZENDE, E.; RUPPERT, G.; CARVALHO, T.; THEOPHILO, A.; RAMOS, F.; GEUS, P. de. Malicious software classification using vgg16 deep neural network's bottleneck features. In: **Information Technology-New Generations**. [S.l.]: Springer, 2018. p. 51–59.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M. et al. Imagenet large scale visual recognition challenge. **International journal of computer vision**, Springer, v. 115, n. 3, p. 211–252, 2015.

SANTOS, M. C. d.; ROCHA, A. Revisão de conceitos em projeção, homografia, calibração de câmera, geometria epipolar, mapas de profundidade e varredura de planos. 2012.

SHEN, D.; WU, G.; SUK, H.-I. Deep learning in medical image analysis. **Annual review of biomedical engineering**, Annual Reviews, v. 19, p. 221–248, 2017.

SILVA, R. E. V. D. **Um estudo comparativo entre redes neurais convolucionais para classificação de imagens**. 2018.

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

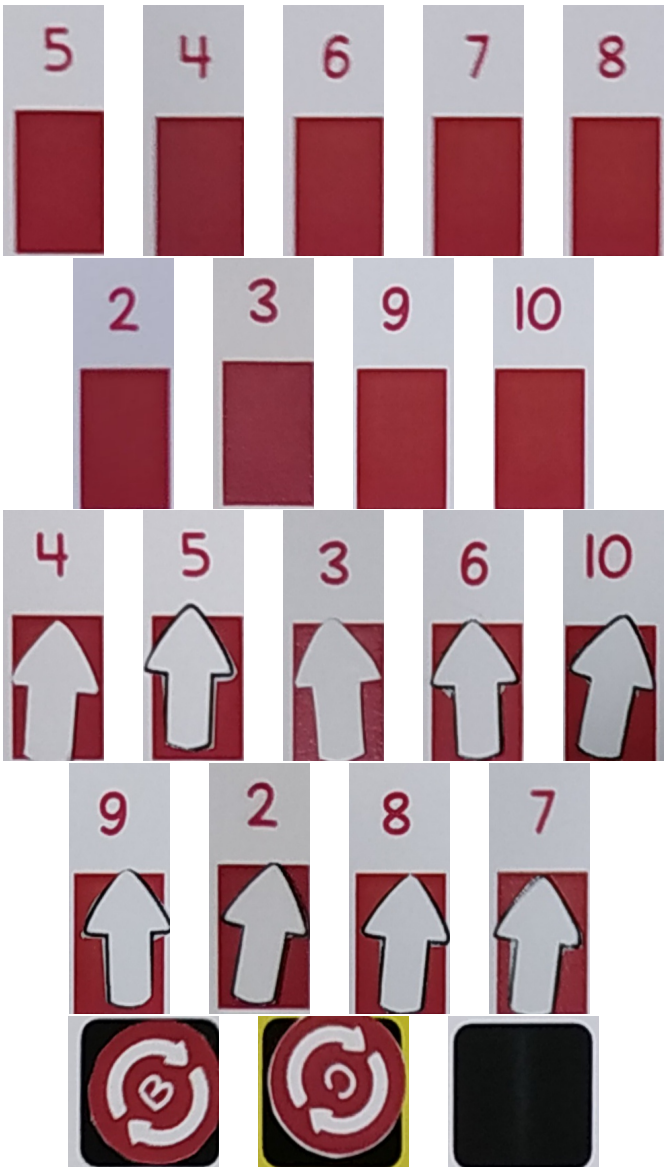
SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J.; WOJNA, Z. Rethinking the inception architecture for computer vision. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 2818–2826.

SZELISKI, R. **Computer vision: algorithms and applications**. [S.l.]: Springer Science & Business Media, 2010.

TORTORA, G. J.; DERRICKSON, B. **Corpo Humano-: Fundamentos de Anatomia e Fisiologia**. [S.l.]: Artmed Editora, 2016.

APÊNDICE A – Demais classes de imagens do tabuleiro

Figura 27 – Exemplos de peças do tabuleiro postas em regiões de interesse.

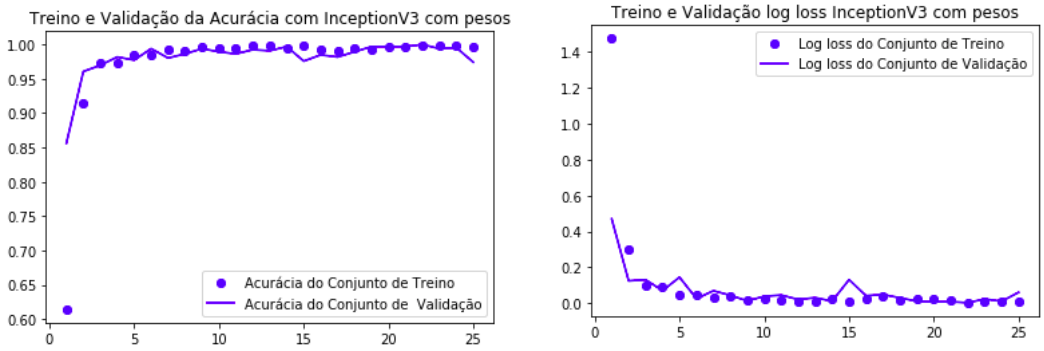


Fonte: Elaborado pelo autor.

APÊNDICE B – Acurácia e *Log Loss* dos demais cenários de experimentação

Na Figura 28 o gráfico da acurácia mostra que antes de 5 épocas as acurácias do conjunto de treino e validação já estavam em torno de 95%, destacando que utilizando os pesos pré-treinados o modelo aprendeu rapidamente sobre o nosso conjunto de dados. O *log loss* do cenário também desceu rapidamente, ficando abaixo de 0.2 antes da quinta época de treinamento.

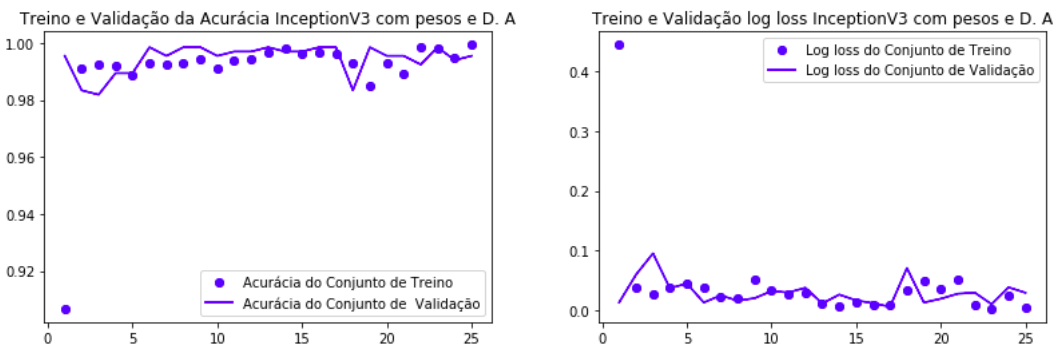
Figura 28 – Gráfico de comportamento Inception V3 com pesos pré treinados.



Fonte: Elaborado pelo autor.

Na Figura 29 fica evidente que com o uso do *data augmentation* os valores de acurácia e *log loss* do conjunto de validação são superiores aos valores do conjunto de treino em alguns momentos. Tanto a acurácia quanto o *log loss* desse cenário convergiram para valores satisfatórios rapidamente. Houveram alguns ruídos nos resultados, principalmente se aproximando ao fim das épocas de treinamento.

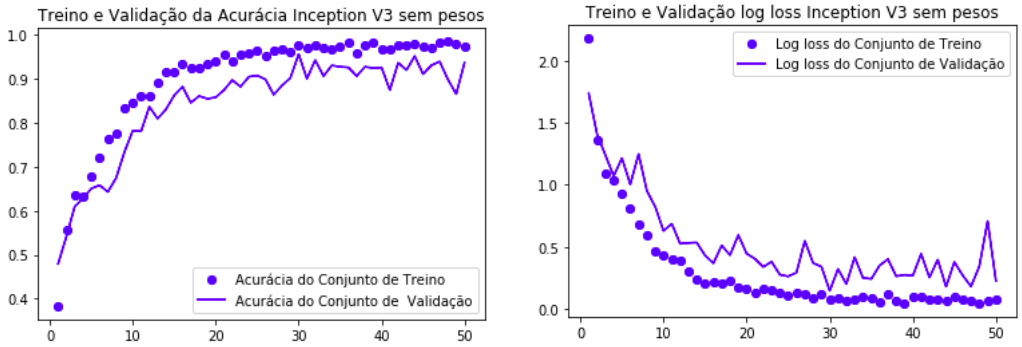
Figura 29 – Gráfico de comportamento Inception V3 com pesos pré treinados e *data augmentation*.



Fonte: Elaborado pelo autor.

A Figura 30 revela a importância das camadas iniciais da Inception V3 quando não há pesos pré treinados no modelo. Como previsto, o cenário sem pesos pré-treinados não convergiu rapidamente pra um alto valor de acurácia. O conjunto de treino levou aproximadamente 20 épocas de treinamento para alcançar valor superior a 90% de acurácia e 25 épocas para valores maiores que 95%. Já o conjunto de validação demorou 30 épocas para valores maiores que 90% de acurácia e só alcançou valores maiores que 95% na época 31, com o valor final abaixo disso. O gráfico do *log loss* nesse cenário também evidência a dificuldade que a rede apresentou de aprender os pesos da rede sem utilizar pesos pré treinados. Importante destacar que nesses cenários sem pesos pré treinados foram utilizados 50 épocas de treinamento.

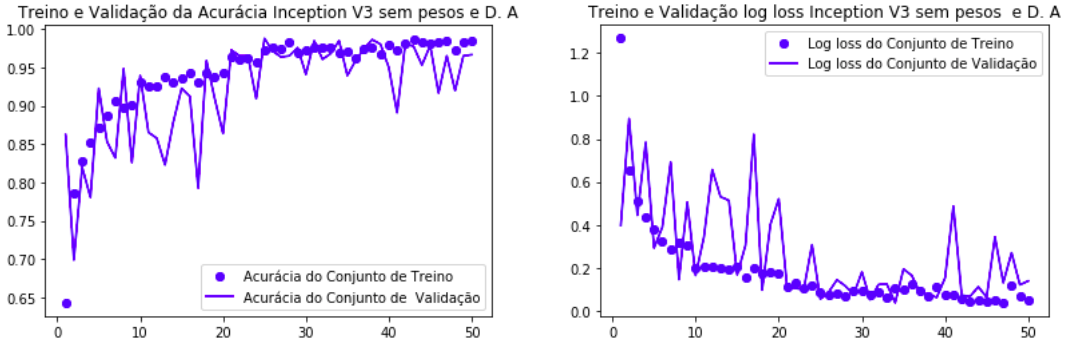
Figura 30 – Gráfico de comportamento Inception V3 sem pesos pré treinados.



Fonte: Elaborado pelo autor.

A Figura 31 não teve exito em destacar que o conjunto de validação teve valores melhores que o conjunto de treino utilizando o *data augmentation*. Houveram muitas discrepância de valores da acurácia e *log loss* do conjunto de validação durante quase todas as épocas de treinamento. Para minimizar isso, certamente o número de imagens para todas as classes deveriam ser maiores.

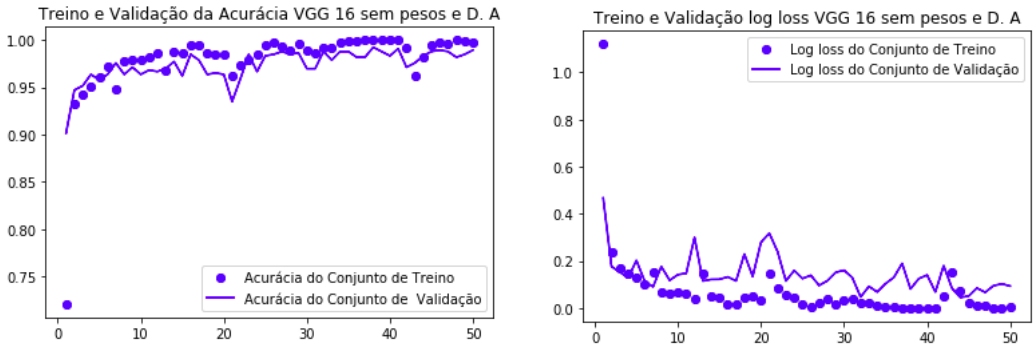
Figura 31 – Gráfico de comportamento Inception V3 sem pesos pré treinados e *data augmentation*.



Fonte: Elaborado pelo autor.

Por fim, a Figura 32 mostra o quão próximo os valores do conjunto de treino e validação se apresentaram num cenário que utiliza *data augmentation* sem pesos pré treinados. O modelo levou 7 épocas para ter o conjunto de validação com valores superiores a 90% na acurácia e 10 épocas para apresentar o mesmo resultado com o conjunto de validação. Durante a execução de todas as épocas os valores de *log loss* e acurácia dos conjuntos de treino e validação estiveram próximos. Fica claro, comparado a figura anterior, que a arquitetura VGG-16 se sobressaiu em relação à Inception V3 nesses cenários onde não houve a utilização de pesos de modelos pré-treinados.

Figura 32 – Gráfico de comportamento VGG16 sem pesos pré treinados e *data augmentation*.



Fonte: Elaborado pelo autor.