UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

Predição de Dificuldade em Jogos Match-Three Utilizando Redes Neurais Convolucionais

Juan Burtet

Pelotas, 2022

Juan Burtet

Predição de Dificuldade em Jogos Match-Three Utilizando Redes Neurais Convolucionais

Dissertação apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Ricardo Matsumura de Araújo

Pelotas, 2022

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

B973p Burtet, Juan

Predição de dificuldade em jogos match-three utilizando redes neurais convolucionais / Juan Burtet ; Ricardo Matsumura Araújo, orientador. — Pelotas, 2022.

45 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2022.

1. Redes neurais convolucionais. 2. Aprendizado profundo. 3. Predição de dificuldade. 4. Match-three. I. Araújo, Ricardo Matsumura, orient. II. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Juan Burtet

Predição de Dificuldade em Jogos Match-Three
Utilizando Redes Neurais Convolucionais

Dissertação aprovada, como requisito parcial, para obtenção do grau de Mestre em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 08 de abril de 2022

Banca Examinadora:

Prof. Dr. Ricardo Matsumura Araújo (orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Tiago Thompsen Primo
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Frederico Schmitt Kremer
Doutor em Biotecnologia pela Universidade Federal de Pelotas.

Dedico este trabalho a todas as pessoas que me ajudaram de alguma maneira e fizeram com que esta etapa fosse finalizada.

If a machine is expected to be infallible,
it cannot also be intelligent.
— ALAN TURING

RESUMO

BURTET, Juan. **Predição de Dificuldade em Jogos Match-Three Utilizando Redes Neurais Convolucionais**. Orientador: Ricardo Matsumura de Araújo. 2022. 45 f. Dissertação (Mestrado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2022.

Desde o surgimento do gênero *Match-Three* em 1994, novos jogos com essa temática foram criados, chegando ao seu ápice em 2012 com o lançamento de *Candy Crush Saga*, que utilizou novos conceitos como mapas de diferentes formatos e modos de jogo diversos. Com o aumento da popularidade de jogos deste estilo, surgiu a necessidade de atualizações constantes de conteúdo. Um dos pontos-chave da geração de conteúdo é a avaliação da dificuldade, que envolve uma grande quantidade de tempo e testes com diferentes usuários. Devido a estes fatores, a predição de dificuldade em jogos obteve grande foco na área de desenvolvimento de jogos, removendo a necessidade de testar o conteúdo criado com jogadores reais. No contexto de jogos *Match-Three*, uma das métricas usadas para estimar a dificuldade é feita através do desempenho médio humano, que é calculado verificando a porcentagem média de objetivos completados do mapa. Atualmente, existe a possibilidade de predizer este desempenho através de simulações do estado do jogo, o que faz este processo lento e custoso computacionalmente. Este trabalho apresenta uma solução para resolver o problema da necessidade de múltiplas simulações do jogo, utilizando Redes Neurais Convolucionais. Desta maneira, sendo possível classificar a dificuldade de um mapa apenas observando seus aspectos visuais e seus objetivos. Ao final do trabalho, foi possível alcançar um modelo de Redes Neurais Convolucionais capaz de predizer a dificuldade corretamente de 75,6% dos mapas.

Palavras-chave: Redes Neurais Convolucionais. Aprendizado Profundo. Predição de Dificuldade. Match-Three.

ABSTRACT

BURTET, Juan. **Difficulty Prediction in Match-Three Games Using Convolutional Neural Networks**. Advisor: Ricardo Matsumura de Araújo. 2022. 45 f. Dissertation (Masters in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2022.

Since the emergence of the Match-Three genre in 1994, new games with this theme have been created, reaching its peak in 2012 with the launch of Candy Crush Saga, wich utilized new concepts such as maps with different shapes and diverse game modes. With the increasing popularity of games with this style, the need for constant content updates has arisen. One of the key points of content generation is the difficulty evaluation, wich envolves a great amount of time and tests with different users. Due to these factors, the difficulty prediction in games has gained a big focus on the game development area, removing the need of testing the created content with real players. In the context of Match-Three games, one of the metrics used to estimate the difficulty is made through the average human performance, wich is calculated by verifying the percentage of the average map objectives completed. Currently, there is the possibility of predicting this performance through simulations of the game state, making this process slow and computationally costly. This work presents a solution to the problem of the need for multiple game simulations, using Convolutional Neural Networks. In this way, it is possible to classify the difficulty of a map by just observing its visual aspects and its objectives. At the end of this work, it was possible to achieve a model of Convolutional Neural Networks capable of predicting the difficulty of 75,6% of the maps correctly.

Keywords: Convolutional Neural Network. Deep Learning. Difficulty Prediction. Match-Three.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MCTS	Monte Carlo Tree Search
RNC	Redes Neurais Convolucionais

SUMÁRIO

1 INTRODUÇÃO

Desde o surgimento de Tetris, cada vez mais jogos com ideias simples são propostos para tentar atrair o maior número de pessoas (JUUL, 2007). Com isso, surgiu o estilo *Tile-Matching*, um modelo de jogos de quebra-cabeça com a necessidade de criar padrões para completar um determinado desafio. Uma das categorias deste gênero é o *Match-Three*, onde é necessário fazer combinações de três ou mais peças iguais para prosseguir pelo jogo.

Com a chegada do século XXI, os jogos *Match-Three* tiveram um aumento de popularidade com o surgimento do jogo *Bejeweled* (JUUL, 2007), que pode ser observado na Figura 1, e alcançando seu auge com o lançamento de *Candy Crush* pela Empresa King para o Facebook em 2012. No jogo, o objetivo principal é trocar as posições das peças do tabuleiro, que são representadas como doces, para assim, elimina-las do campo e continuar adicionando novas peças.

Figura 1 – Captura de tela do jogo Bejeweled (Fonte: Bejeweled Wiki | Fandom (2021))

Em 2013, Candy Crush já tinha sido instalado mais de 500 milhões de vezes pelo *Facebook* e em aparelhos *iOS* e *Android* (WEBSTER, 2013), com pelo menos 6,7 milhões de jogadores ativos diariamente (MUSIL, 2013). Com uma grande quantidade de jogadores, foi percebida a necessidade de atualizações constantes de conteúdo do jogo e, por isso, novos níveis são adicionados semanalmente.

Essas necessidades de atualizações constantes levantam alguns problemas relacionados a criação de mapas. A criação manual de mapas precisa especificar o formato da fase, bem como os objetivos necessários para completar uma partida. Dessa forma, fica evidente o problema de que cada nível é especificado por *designers*, que precisam testar o conteúdo diversas vezes para saber se ele é possível de ser concluído, se parece ser acessível e atrativo visualmente (STEVEN, 2018).

Atualmente, a produção manual de conteúdo em um jogo é geralmente custosa economicamente (IOSUP, 2011), sendo necessário o trabalho de muitas pessoas ou uma grande quantidade de tempo. Um dos principais problemas neste processo é a necessidade de se avaliar um mapa novo com diferentes usuários, para ser possível fazer uma avaliação da dificuldade deste conteúdo. Sendo assim, a dificuldade de um jogo acaba sendo um fator de grande importância na percepção de qualidade para um jogador (DENISOVA; GUCKELSBERGER; ZENDLE, 2017), o que faz a predição dessa métrica um dos pontos chaves para um conteúdo criado.

A predição de dificuldades em jogos tem como foco principal a criação de um agente que seja capaz de jogar o jogo, tendo a possibilidade de avaliar sua dificuldade através de múltiplas jogadas. Uma das técnicas muito utilizadas nesta área são as *Monte Carlo Tree Search* (MCTS), que trabalham como um algoritmo de busca heurística, onde simulações dos estados futuros são usados para decidir qual a melhor ação do estado atual (BROWNE et al., 2012). Além disso, as *MCTS* tiveram destaque após serem capazes de obter um desempenho melhor que um humano no jogo *Go*, o que nenhum outro modelo de Inteligência Artificial foi capaz até o momento (SILVER et al., 2017).

O foco das *MCTS* está na análise dos movimentos mais promissores, expandindo a árvore de busca em amostragem aleatória no espaço de busca. A sua aplicação em jogos é baseada em múltiplas jogadas, onde o jogo é percorrido até o final selecionando movimentos aleatórios. Desta maneira, o resultado final de cada jogada é utilizado para ponderar os nós da árvore, para que os caminhos com melhores resultados tenham maior probabilidade de serem escolhidos em jogadas futuras.

No contexto de dificuldade em jogos *Match-Three*, os trabalhos de Poromaa (2017) e Burtet (2019) utilizaram as *MCTS* para replicar a avaliação do desempenho humano médio, sem a necessidade de utilizar jogadores reais. Em ambos os trabalhos, os testes foram avaliados através de uma comparação do desempenho médio recuperado dos dados de jogadores reais, contra os resultados de múltiplas simulações utilizando *MCTS*.

Apesar de existir muitos trabalhos que focam na avaliação de conteúdo de jogos de maneira automática, poucos estão trabalhando diretamente na solução do problema em jogos *Match-Three*. Além dos trabalhos mencionados no parágrafo anterior, o trabalho de Gudmundsson et al. (2018) foi capaz de predizer a jogada mais “humana”

para um determinado estado do jogo *Candy Crush*. Isto foi feito utilizando um modelo de Redes Neurais Convolucionais (RNC) utilizando dados recuperados de milhões de jogadores. Este trabalho possuí um grande ganho no tempo de processamento em comparação aos que utilizaram as *MCTS*, pois o modelo é capaz de predizer a próxima jogada visualizando o conteúdo do mapa sem a necessidade de simulação de estados futuros. Entretanto, a etapa de repetir diversas partidas para se recuperar o desempenho médio continua presente.

Sendo assim, este trabalho seguirá com o que foi desenvolvido em Burtet (2019), que apresentava um gerador de mapas para jogos *Match-Three*, utilizando as *MCTS* para simulação do estado do jogo. Como atualização para este trabalho, é proposto a utilização de Redes Neurais Convolucionais para uma nova solução do problema de predição de dificuldade de um mapa, utilizando apenas os padrões visuais do conteúdo, como seu formato e objetivos, sem a necessidade de múltiplas simulações.

2 TRABALHOS RELACIONADOS

Apesar de existir muitos trabalhos que focam na avaliação de conteúdo de jogos de maneira automática, poucos estão trabalhando diretamente na solução do problema em jogos *Match-Three*. Neste capítulo será explorado trabalhos que possuam foco na predição de dificuldade, além da geração de conteúdo artificial que será utilizado para o desenvolvimento desta dissertação.

2.1 Predição do desempenho humano médio

No trabalho "*Crushing Candy Crush: Predicting Human Success Rate in a Mobile Game using Monte-Carlo Tree Search*" de Poromaa (2017), as MCTS foram utilizadas com sucesso em jogos Match-Three. O método de busca das árvores de Monte Carlo foram utilizados para replicar a avaliação de desempenho humano médio, retirando a necessidade de jogadores humanos.

O jogo *Candy Crush* foi utilizado como base para o desenvolvimento deste trabalho, onde foi verificado a porcentagem de vitórias de jogadores reais em comparação aos resultados de um *bot* utilizando MCTS. A Figura 2 apresenta uma visão geral do processo aplicado pelo *bot* para cada jogada.

Este trabalho criou a possibilidade de avaliar o desempenho de mapas de forma automática, sendo necessário cerca de 200 partidas para obter o resultado de vitórias. Entretanto, este processo possuí um custo computacional elevado, devido as centenas de simulações em cada estado do jogo para que o mesmo possua um resultado semelhante a média humana.

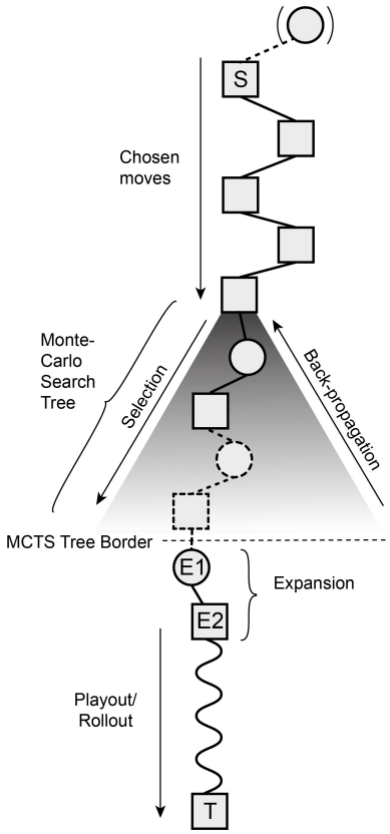

Figura 2 – Visão Geral do bot implementado para Candy Crush (Fonte: Poromaa (2017))

2.2 Geração de Mapas para Jogos Match-Three

Devido ao resultados alcançados pelo trabalho de Poromaa (2017), foi averiguado a possibilidade da geração de mapas para este gênero de jogo feito de forma automática. Em “Geração Procedural de Mapas para Jogos Match-Three” de Burtet (2019), algoritmos genéticos são utilizados para gerar mapas com um certo desempenho médio desejado.

Para o desenvolvimento deste trabalho, um jogo base foi criado para que os testes pudessem ser aplicados. Este jogo possuia recursos básicos semelhantes ao *Candy Crush*, com a presença de peças especiais, pontuações e modos de jogo a serem finalizados. Diferente do trabalho anterior, a métrica de desempenho médio humano utilizada foi modificada para a porcentagem de objetivos médios finalizados. Esta métrica possuía uma variação menor através de múltiplos testes em comparação a porcentagem de vitórias média.

No contexto de algoritmos genéticos, o objetivo do mapa criado é obter um desempenho médio próximo ao requisitado pelo usuário. Sendo assim, a avaliação de um conteúdo gerado é a diferença absoluta entre o desempenho médio do mapa e o desempenho médio desejado. O fluxograma sobre o processo de geração de mapas é apresentado na Figura 3.

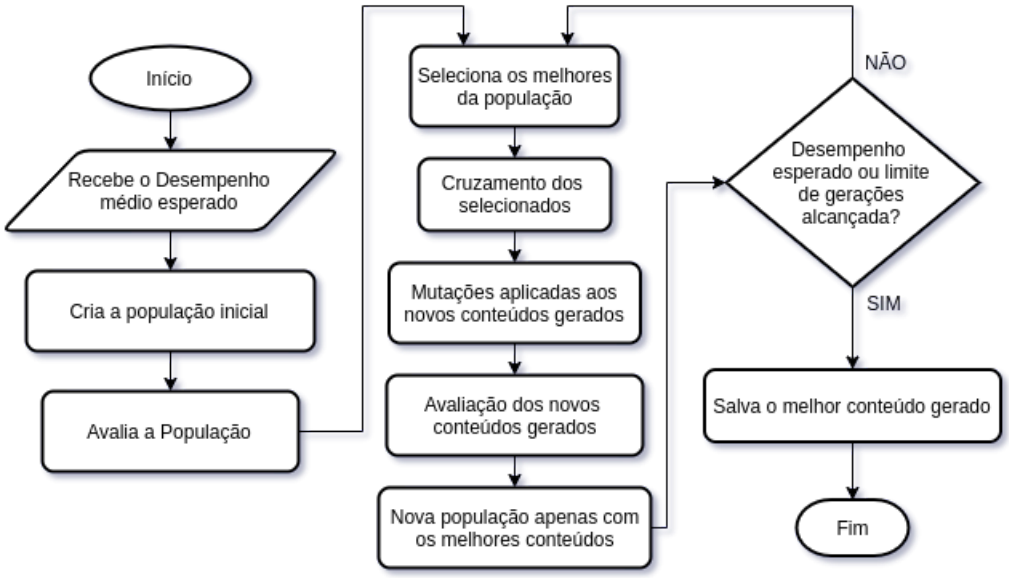

Figura 3 – Fluxograma do Gerador de Mapas (BURTET, 2019)

2.3 Predição da jogada mais humana

Um dos grandes problemas de utilizar as MCTS para predizer o desempenho médio humano é a necessidade de simular o estado atual do jogo inúmeras vezes, para saber qual o melhor movimento a ser feito. Somado a isso, existe a necessidade de repetir o processo de uma partida centenas de vezes para calcular a métrica desejada. Para tentar solucionar isso, o trabalho "*Human-Like Playtesting with Deep Learning*" de Gudmundsson et al. (2018) propõe a utilização de Redes Neurais Convolucionais para predizer a jogada com comportamento mais próximo do humano, retirando a necessidade de simulações do estado atual do jogo.

Com a cooperação da Empresa King [1], foi recuperado milhões de dados de usuários do jogo *Candy Crush* contendo um certo estado de um mapa do jogo, junto da jogada feito pelo jogador. Com estes dados, foi desenvolvido uma RNC que recebe um estado do jogo e indica as probabilidades de um humano fazer uma certa jogada. Este processo é exemplificado na Figura 4.

Ao fim do trabalho, é feito uma comparação entre os resultados da RNC contra a utilização das MCTS. Foi observado que as Redes Neurais Convolucionais possuem um desempenho superior na solução do problema de predição de dificuldade, pois conseguem replicar um comportamento humano com mais precisão do que o trabalho anterior produzido por Poromaa (2017). Entretanto, o ponto de maior destaque está no tempo de processamento, pois este modelo retira a necessidade de simular estados futuros do jogo para fazer uma única jogada.

[1] Desenvolvedora do jogo *Candy Crush*

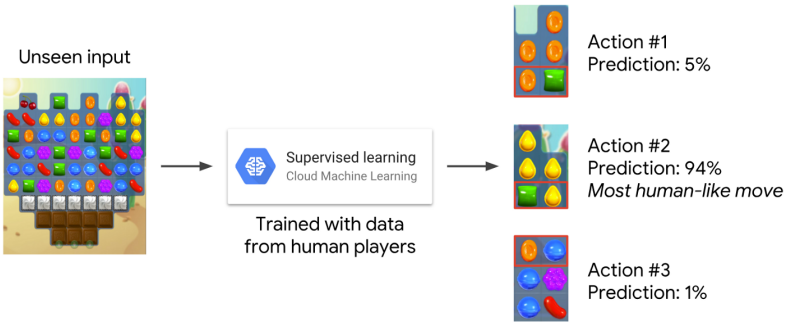

Figura 4 – Exemplo de um resultado do modelo de predição da jogada mais humana (Fonte: Gudmundsson et al. (2018))

3 OBJETIVO

Este trabalho tem como objetivo criar um modelo de Redes Neurais Convolucionais, que seja capaz de classificar a dificuldade de um mapa de jogo *Match-Three*, utilizando apenas padrões visuais sobre seu formato e objetivos, sem a necessidade de simulação de conteúdo.

3.1 Objetivo Específicos

Para que o objetivo geral deste trabalho seja alcançado, é necessário concluir os objetivos específicos. Eles são divididos em: criar um dataset de mapas com suas respectivas dificuldades, desenvolver modelos capazes de predizer a dificuldade de um mapa e analisar os resultados para definir sua assertividade neste problema.

3.1.1 Dataset de mapas

Criação de um dataset de mapas de jogo *Match-Three* utilizando o gerador de mapas desenvolvido em Burtet (2019), que será utilizado para treinar modelos de Aprendizado de Máquina. Estes mapas serão rotulados com suas respectivas dificuldades através do cálculo do desempenho médio humano.

3.1.2 Desenvolvimento de modelos

Implementação de diferentes modelos de Redes Neurais Convolucionais com inspiração do modelo desenvolvido em Gudmundsson et al. (2018). Estes modelos serão capazes de predizer a dificuldade de um mapa de jogo Match-Three apenas observando seu aspectos visuais e objetivos.

3.1.3 Comparação e análise dos resultados

Comparação dos resultados de acurácia e F1-Score dos modelos desenvolvidos com modelos clássicos de Aprendizado de Máquina, como: Árvores de Decisão, Random Forest e XGBoost. Além disso, será avaliado seu desempenho com os modelos de visão computacional AlexNet e ResNet-12. Ao fim, será analisado de forma mais

detalhada os resultados do modelo de melhor acurácia, observando seu desempenho em dificuldades específicas. Isto será feito utilizando a biblioteca Scikit-Learn (PEDREGOSA et al., 2011), que é capaz de retornar métricas de precisão e recall, além da visualização da matriz de confusão das predições.

4 METODOLOGIA

Esta capítulo irá apresentar a metodologia utilizada para o desenvolvimento deste trabalho, que foi dividida em cinco seções: geração dos exemplos, rotulamento, pré-processamento, desenvolvimento de modelos e seus treinamentos.

4.1 Geração dos Exemplos

A primeira etapa para desenvolvimento deste trabalho é a criação do *dataset* contendo as informações sobre o mapa, junto do seu desempenho médio calculado. Para isso, foi utilizado o trabalho desenvolvido por Burtet (2019), que utiliza técnicas de *MCTS* para criar um mapa que tenha uma taxa de desempenho médio próxima da desejada pelo usuário. A representação de um mapa gerado é apresentado na Figura 5, que contém as informações utilizadas para inicializar uma partida dentro do jogo.

	A	B	C	D	E	F	G	H	I
1	35	0	4						
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0
6	2	2	2	2	2	2	2	2	2
7	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1

Figura 5 – Representação de um mapa criado pelo gerador

Essas informações estão presentes em um arquivo *csv*, que possuí os seguintes dados: **A1** indica o limite de movimentos que o jogador pode fazer para completar o mapa; **B1** indica a pontuação necessária para completar o objetivo de pontos; **C1**

representa quantas peças diferentes estarão presentes no campo; O intervalo de **A2** a **I10** representa cada posição do mapa, e o que será preenchido nesta localização. Em cada posição do mapa, é possível ter quatro valores distintos:

- **0** - uma área vazia, indicando que não pode haver peças neste local.
- **1** - uma peça padrão, gerada aleatoriamente pela quantidade de peças diferentes que estão presentes no campo.
- **2** - uma peça de bloqueio, que precisa ser destruída para completar a partida.
- **3** - uma peça de objetivo, que precisa ser retirada do campo para completar a partida.

Para este trabalho, as informações sobre limite de movimentos, pontuação e a quantidade de peças diferentes serão iguais para todos os dados gerados. Isto foi decidido para que os modelos de Aprendizado de Máquina testados trabalhem apenas com a informação visual do mapa, verificando se mudanças na arquitetura podem indicar a dificuldade média a ser encontrada pelos jogadores.

A métrica utilizada pelo gerador é calculada verificando a porcentagem média de objetivos completados em cada uma das partidas avaliadas. Ao fim de uma partida, é avaliado a porcentagem de objetivos completados: caso o mapa tenha sido finalizado completamente, retorna o valor 1; Em caso contrário, é calculado a quantidade de objetivos finalizados e retornado o quão próximo o jogador estava de ganhar o mapa. O resultado final é calculado fazendo a média dos resultados de 100 partidas diferentes.

Através dos resultados do trabalho de Burtet (2019), foi constatado que mapas com desempenho médio abaixo de 75% possuem uma dificuldade próxima do impossível, onde jogadores reais possuem chances muito baixas de finalizar o mapa com sucesso. Sendo assim, os dados gerados foram limitados a ter desempenhos médios entre 75% e 100%, para que os mapas possuam uma diferença de dificuldade mais perceptiva entre os usuários.

Para geração dos dados, foi feito um processo de selecionar um valor de desempenho médio aleatório no intervalo de 75% até 100%, para tentar gerar mapas com desempenhos mais diversificados. Durante a busca do mapa com a métrica desejada, inúmeros mapas são criados durante o processo do algoritmo genético. Sendo assim, os mapas gerados durante a busca são adicionados a base de dados caso estejam dentro do intervalo de desempenho médio definido.

Ao fim do processo de geração, foi gerado um total de 13791 mapas distintos, onde o mapa de menor de desempenho médio possui um valor exato de 75%, enquanto o maior possui 99,9%. Algumas informações estatísticas sobre os mapas podem ser encontradas na Tabela 1, além do gráfico do histograma dos dados na Figura 6.

Tabela 1 – Estatísticas sobre o conjunto de dados gerados

Estatística	**Valor**
Quantidade	13,791
Média	0.887
Desvio Padrão	0.065
Mínimo	0.750
Primeiro Quartil	0.837
Mediana	0.899
Terceiro Quartil	0.943
Máximo	0.999

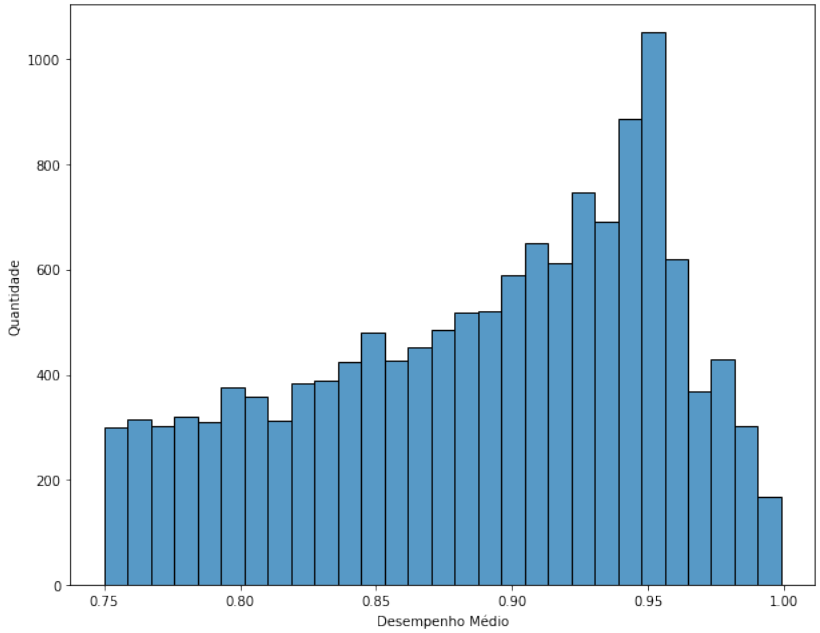

Figura 6 – Histograma do Desempenho Médio dos dados gerados

4.2 Rotulamento

Os exemplos gerados na seção anterior possuem um valor real como desempenho médio humano. Para este trabalho, iremos trabalhar como um problema de classificação, onde o modelo terá que indicar em qual classe de dificuldade o mapa avaliado está. Esta mudança é feita por dois motivos: diminuir a quantidade de resultados possíveis a serem preditos, pois um problema de classificação é uma tarefa mais simples de ser realizada e verificar se é possível diferenciar dificuldades apenas utilizando os aspectos visuais de um mapa.

Para transformar os dados de desempenho médio em classes de dificuldade, são definido intervalos de valores, onde cada um desses intervalos representam uma dificuldade diferente. Para este trabalho, é definido cinco classes de dificuldade: Muito Fácil, Fácil, Médio, Difícil e Muito Difícil. Os intervalos utilizados para cada uma das classes pode ser verificado na Tabela 2.

Tabela 2 – Intervalos de classificação de dificuldade

Dificuldade	**Limites**	
	Inferior	**Superior**
Muito Fácil	0,95	1,00
Fácil	0,90	0,95
Médio	0,85	0,90
Difícil	0,80	0,85
Muito Difícil	0,75	0,80

Após a definição da dificuldade para cada um dos mapas, ficamos com os dados divididos em cinco intervalos diferentes. As quantidades de mapas em cada uma das classes pode ser verificado na Tabela 3.

Tabela 3 – Divisão das dificuldades por mapa

Dificuldade	**Quantidade**
Muito Fácil	1833
Fácil	2211
Médio	2890
Difícil	4155
Muito Difícil	2702

Com os dados rotulados em suas respectivas dificuldades, é preciso separá-los em 2 conjuntos distintos: Treino e Teste. Este processo é feito utilizando a biblioteca *Scikit-Learn* (PEDREGOSA et al., 2011), onde é possível definir uma porcentagem de divisão entre os dois grupos. Neste trabalho, foi utilizado a separação de 80%

dos dados para treinamento e os 20% restantes para avaliar os desempenhos dos modelos. As classes de dificuldade, junto das suas respectivas quantidades de treino e teste podem ser verificados na Tabela 4.

Tabela 4 – Divisão dos dados de treino e teste

Dificuldade	**Quantidade**	
	Treino	**Teste**
Muito Fácil	1466	367
Fácil	1380	831
Médio	2312	578
Difícil	3713	442
Muito Difícil	2335	367

4.3 Pré-processamento

Apesar de termos todos os dados gerados e rotulados, ainda é necessário modificar a informação dos mapas para o formato de entrada dos modelos a serem avaliados. Como não foi encontrado outros trabalhos que trabalhem com a predição de dificuldade utilizando RNCs, é proposto verificar o desempenho de diferentes modelos de Aprendizado de Máquina para comparar o possível ganho de desempenho do nosso modelo contra opções já estabelecidas. Sendo assim, iremos dividir os pré-processamentos em três seções diferentes: Modelos clássicos, Modelos de Visão Computacional e Modelos desenvolvidos.

4.3.1 Modelos Clássicos

Os modelos clássicos utilizados utilizam os dados de formas tabulares, onde se faz necessário modificar o formato de matriz que os mapas gerados possuem. Sendo assim, é preciso aplicar um método de *reshape* nos dados, onde o mapa que possuí um formato de 9x9 irá resultar em um vetor de 81 posições. A representação visual desta operação é apresentada na Figura 7.

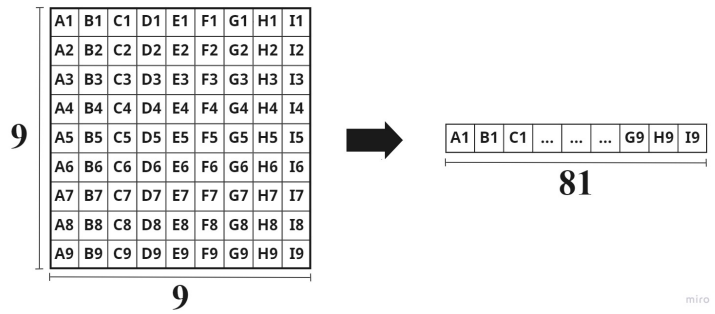

Figura 7 – Pré-processamento necessário para os modelos clássicos

4.3.2 Modelos de Visão Computacional

Os modelos de visão computacional a serem avaliados neste trabalho foram as redes convolucionais *AlexNet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) e *ResNet-18* (HE et al., 2015). Ambos os modelos já estão implementados através da biblioteca *PyTorch* (PASZKE et al., 2019) e recebem como entrada uma imagem RGB de tamanho 224x224x3.

Para o pré-processamento, três operações são necessárias para que os dados dos mapas estejam no formato correto: Normalizar a informação de cada posição do mapa, expandir a matriz 9x9 para o formato 224x224 e por fim, replicar a matriz expandida como três canais de cores diferentes.

Para a primeira etapa, é preciso modificar cada posição para que possuam valores normalizados de 0 até 1, ao invés da atual formatação de 0 até 3. Este resultado é alcançado com facilidade, apenas aplicando a operação de divisão em cada uma das posições pelo maior valor possível, que neste caso é 3.

Em seguida, é necessário expandir a matriz que representa o mapa para o tamanho de entrada dos modelos. Este processo é feito utilizando o método de interpolação *nearest-neighbor* (BOVIK, 2009) da biblioteca *PIL* (UMESH, 2012). O resultado desta operação é que nenhuma informação original é perdida ou modificada, apenas replicada para ser representada em uma resolução maior.

Por fim, é necessário replicar a matriz 224x224 em três camadas diferentes. Neste processo, os mapas seriam representados como uma imagem em tons de cinza, onde as diferenças de tom indicam os diferentes valores representados em cada posição do mapa.

4.3.3 Modelos Desenvolvidos

O ultimo pré-processamento necessários nos dados é os que serão utilizados nos modelos desenvolvidos especificamente para este trabalho. Este processo irá possuir dois pré-processamentos distintos, para que possamos avaliar qual representação dos mapas retorna um melhor resultado dos modelos.

4.3.3.1 Camada Única

No formato que iremos chamar de "camada única", a única operação necessária nos dados é a normalização da informação. Este processo é exatamente o mesmo do apresentado na seção dos modelos de visão computacional, onde cada posição do mapa irá possuir um valor de 0 até 1.

4.3.3.2 Informação por Camada

Este formato possuí a representação de "cada informação em uma camada", onde ao invés de trabalhar com os dados como uma matriz 9x9, a informação são divididas

em quatro camadas booleanas, resultando em uma matriz 9x9x4. Neste processo, cada camada teria a função de indicar a presença de tal valor. Sendo assim, a primeira camada indicaria a posição do valor 0 no mapa, a segunda camada a posição do valor 1 e assim por diante. Este pré-processamento da representação do mapa em diferentes camadas foi inspirado do trabalho de Gudmundsson et al. (2018), onde o mesmo processo é aplicado. Um exemplo desta operação pode ser visualizado na Figura 8.

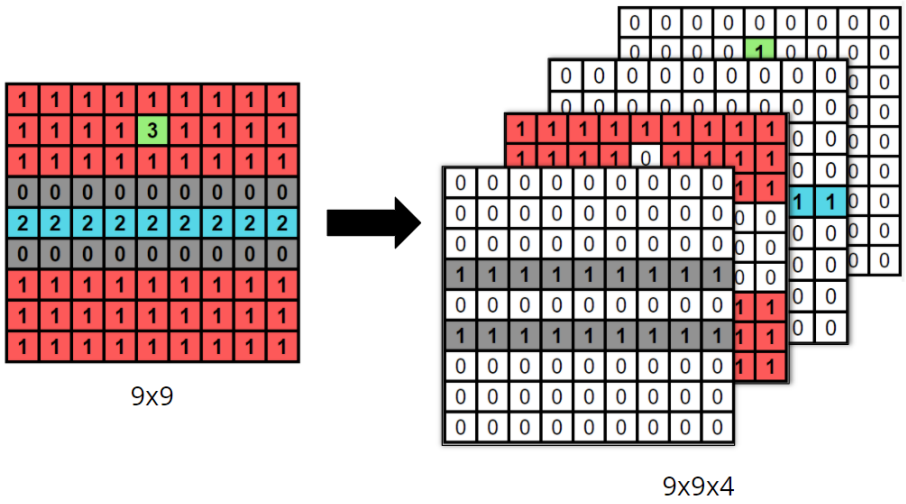

Figura 8 – Pré-processamento dos valores em diferentes camadas

4.4 Arquiteturas dos Modelos Desenvolvidos

Com os dados gerados e processados, é preciso definir a arquitetura dos nossos modelos que serão avaliados para a solução do problema de predição de dificuldade. Como foi definido na seção anterior que os dados para os modelos desenvolvidos são divididos em dois grupos diferentes, o mesmo precisa ser feito para os modelos.

A arquitetura definida para este projeto foi baseada no trabalho de Gudmundsson et al. (2018), pois o mesmo já trabalhou com uma entrada para representar um jogo *Match-Three*, com a diferença de predizer a jogada mais humana do que a dificuldade do mapa. Entretanto, a sua saída resultava em qual dos 144 movimentos possíveis mais se parecia com o de um humano, o que faz ser necessário trabalhar com uma rede muito mais profunda.

A nossa rede é definida iniciando com uma camada de convolução de 5 filtros de tamanho 3, com *padding* e *stride* igual a 1. Este formato foi decidido para que sua saída se mantenha no formato de 9x9 dos mapas, para que cada canal resultante encontre possíveis padrões para cada uma das dificuldades possíveis. Esta camada de convolução definida permite que ela seja replicada diversas vezes, onde a sua saída se conecta em outra camada de mesmo formato, apenas modificando a quantidade de

canais de entrada. Após a saída das camadas de convolução, é feito uma ligação em sequência de 3 módulos lineares FC *(Fully Connected)*. O primeiro módulo FC possuí um tamanho 405 (9x9x5), seguindo para um declínio de 256 até o último módulo de 128 entradas, resultando nas 5 saídas de dificuldade possíveis. Todas as camadas de convolução e lineares contém a função de ativação *ReLU* (AGARAP, 2018), além da adição de *Dropout* (p=0.5) (SRIVASTAVA et al., 2014) nas duas primeiras camadas lineares. A ilustração da arquitetura da rede utilizada pode ser encontrada na Figura 9.

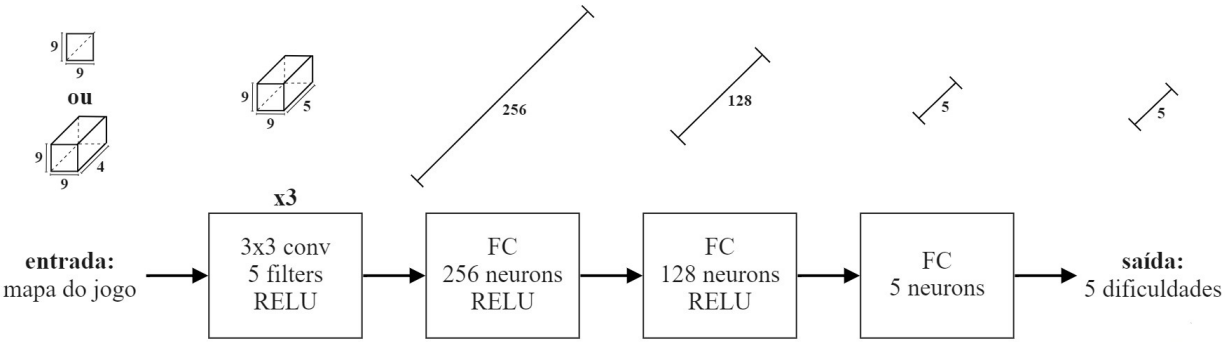

Figura 9 – Ilustração da arquitetura da rede desenvolvida

Para este trabalho, será testado modelos contendo de uma à três camadas de convolução, onde cada uma desses versões terão testes feitos com as duas entradas de pré-processamento apresentadas.

4.5 Treinamento dos Modelos

A etapa final da metodologia deste trabalho é o treinamento dos modelos. Neste processo, será organizado os treinamentos dos três grupos de modelos diferentes: clássicos, visão computacional e os desenvolvidos especificamente para esta tarefa. Os dois primeiros grupos são focados para termos uma certa estimativa do desempenho desta tarefa, onde será feito múltiplas tentativas para encontrar os melhores resultados nos nossos próprios modelos. A comparação dos modelos será feita avaliando a acurácia no conjunto de teste, além do seu F1-Score.

4.5.1 Modelos Clássicos

Os modelos clássicos a serem treinados são: Árvores de Decisão, *Random Forest* e *XGBoost*. Os dois primeiros modelos estão desenvolvidos na biblioteca *Scikit-Learn* (PEDREGOSA et al., 2011), enquanto o *XGBoost* possuí uma biblioteca própria para sua utilização (CHEN; GUESTRIN, 2016).

Os treinamentos dos modelos clássicos foram feitos de forma simples, utilizando as configurações padrões das bibliotecas, apenas modificando para “balanceado” a op-

ção *class_weight*. Esta opção foi escolhida para garantir que todas as classes tenham o mesmo peso, desconsiderando o desbalanceamento dos dados. Os resultados desses testes serão considerados como um limite inferior de desempenho para este problema, já que o formato do mapa é desconsiderado ao fazer o pré-processamento necessário.

4.5.2 Modelos de Visão Computacional

Este grupo de testes consiste nos modelos: *AlexNet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) e *ResNet-18* (HE et al., 2015), onde iremos verificar o desempenho de modelos de Redes Neurais Convolucionais conhecidos em tarefas de classificação de imagens. Os resultados destas arquiteturas são avaliadas para verificar se a presença de camadas de convolução causam uma melhora no desempenho em comparação aos modelos clássicos.

Os treinamentos dos modelos de visão computacional foram feitos por 50 épocas, usando um *Learning Rate* de 0,01 e tamanho de *batch* igual a 8. Além disso, é verificado se a utilização de modelos pré-treinados retornam melhores resultados neste problema. Todos os testes destes modelos foram feitos utilizando a biblioteca *PyTorch* (PASZKE et al., 2019), que já possuí todos esses recursos desenvolvidos.

4.5.3 Modelos Desenvolvidos

A etapa final de testes consiste nos modelos desenvolvidos para este trabalho, onde será avaliado qual pré-processamento obtém o melhor resultado, além de qual quantidade de camadas de convolução é melhorar para solucionar o problema da predição de dificuldade.

A grande maioria dos testes será feito na tentativa de obter os melhores resultados de acurácia e F1-Score nos nossos modelos. Para isso, será avaliado os resultados com diversos parâmetros diferentes, testando diferentes *Learning Rates* e quantidade de épocas. Além disso, será verificado as diferenças no desempenho entre dados não balanceados, utilizando técnicas de *oversample* e balanceando os pesos das classes.

Por fim, o melhor modelo encontrado terá uma análise mais completa, avaliando as predições de dificuldade de cada classe individualmente. Isto será feito com a utilização da biblioteca *Scikit-Learn* (PEDREGOSA et al., 2011), onde podemos recuperar informações sobre precisão e recall nos dados de teste (HOSSIN; SULAIMAN, 2015), além da impressão da matriz de confusão dos dados (TING, 2010).

5 RESULTADOS

Neste capítulo será apresentado os resultados dos testes apresentados na metodologia, sendo dividido em três seções: testes dos modelos clássicos, modelos de visão computacional e modelos desenvolvidos.

5.1 Modelos Clássicos

Os primeiros resultados obtidos para predição de dificuldade foram utilizando os modelos clássicos. Os modelos de Árvores de Decisão e *XGBoost* obtiveram resultados bem próximos, tanto de acurácia quanto de F1-Score, onde o segundo modelo teve uma pequena vantagem em ambas as métricas. Entretanto, o modelo de *Random Forest* conseguiu cerca de 6% mais de acurácia e F1-Score, obtendo 70,9% e 69,8%, respectivamente. Os modelos de *Random Forest* e *XGBoost* utilizam técnicas de *ensemble*, onde os resultados de múltiplos modelos gerados são utilizados para se chegar ao resultado final. Apesar de o *XGBoost* possuir uma capacidade de ter melhores resultados, o *Random Forest* consegue generalizar com mais facilidade quando utilizados os parâmetros padrões das bibliotecas (BENTÉJAC; CSÖRGO; MARTÍNEZ-MUÑOZ, 2019). Os resultados das métricas para cada um dos modelos pode ser encontrado na Tabela 5.

Tabela 5 – Resultados de acurácia e *F1-Score* dos modelos clássicos

Modelo	Acurácia (%)	F1-Score (%)
Árvores de Decisão	64.4	62.6
Random Forest	**70.9**	**69.8**
XGBoost	64.6	63.2

5.2 Modelos de Visão Computacional

Com os resultados preliminares dos modelos clássicos, já podemos ter uma base de valores a serem batidos pelos modelos de visão computacional. Nesta seção,

será avaliado os modelos *AlexNet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) e *ResNet-12* (HE et al., 2015).

5.2.1 AlexNet

Para os testes iniciais da AlexNet, foi treinando o modelo por 50 épocas utilizando um *Learning Rate* de 0.01, avaliando seu desempenho com e sem pré-treinamento. A Tabela 6 contém os resultados destes testes e pode ser verificado que o modelo sem pré-treinamento obteve um desempenho superior ao modelo pré-treinado. Isto pode ocorrer pois o pré-treinamento é feito em cima de dados com domínio muito diferentes do que o nosso trabalho propõem (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Tabela 6 – Resultados de acurácia e *F1-Score* do modelos AlexNet com pré-treino e sem pré-treino.

AlexNet	Acurácia (%)	F1-Score (%)
Com pré-treino	71.9	70.8
Sem pré-treino	**75.8**	**74.2**

5.2.2 ResNet-12

Devido ao ganho dos resultados do modelo *AlexNet* sem o pré-treinamento, foi decidido fazer o treinamento do modelo *ResNet-12* apenas desta maneira. O modelo ResNet-12 obteve resultados inferiores aos testes anteriores, adquirindo uma acurácia de 73,6% e um F1-Score de 72.4%. Estes resultados podem indicar o mesmo comportamento dos modelos clássicos, onde um modelo mais simples possuí uma capacidade de generalização melhor.

5.3 Modelos Desenvolvidos

Os primeiros testes feitos nos modelos desenvolvidos tem como objetivo verificar qual pré-processamento garante os melhores resultados. Sendo assim, foi avaliado seis treinamentos diferentes, que podem ser observados na Tabela 7. Todos os modelos foram treinados por 500 épocas, utilizando um *learning rate* de 0,01 e tamanho de *batch* igual a 8. Nestes testes, foi verificado a acurácia das combinações de pré-processamento de “camada única” e “informação por camada”, com as três quantidades de camadas de convolução.

Tabela 7 – Comparação dos resultados de acurácia dos modelos desenvolvidos treinados por 500 épocas, com *learning rate* de 0.01, com os dois pré-processamentos e quantidade diferentes de camadas de convolução.

Pré-Processamento	**Camadas de Convolução**	**Acurácia (%)**
Camada única	1	64.3
Camada única	2	71.5
Camada única	3	69.1
Informação por camada	1	74.5
Informação por camada	2	74.8
Informação por camada	**3**	**75.6**

Neste teste, foi observado que os dados pré-processados por “informação por camada” obtiveram os melhores resultados de acurácia em comparação a “camada única” em qualquer uma das quantidades de camadas de convolução. Na avaliação de apenas uma camada de convolução, o pré-processamento de “informação por camada” teve a maior diferença de acurácia, chegando a mais de 10% de ganho. Devido a esses resultados, todos os testes posteriores utilizarão apenas o pré-processamento de “informação por camada”.

Já nesta segunda rodada de testes, é feito o treinamento pelo mesmo número de épocas e mesmo learning rate, só que verificando a diferença dos resultados utilizando técnicas de balanceamento dos dados. Os resultados desses testes pode ser encontrado na Tabela 8. Como pode ser observado, a utilização do balanceamento dos dados não garantiu resultados melhores que os testes anteriores, onde todos os seus testes resultaram em uma acurácia próxima de 73%. Apesar de não ficar com resultados muito abaixo, é possível verificar que não aplicar balanceamento nos dados garante modelos mais acurados em todas as arquiteturas avaliadas. Entretanto, todos os dados balanceados obtiveram resultados muito próximos, o que permite mais uma rodada de testes para avaliação.

Como os resultados dos balanceamento de dados não tiveram uma grande variação de acurácia, os mesmos testes serão refeitos com o aumento do learning rate para 0,1. Esta informação pode ser encontrada na Tabela 9, que apresenta dados sem nenhuma melhora em relação aos testes anteriores. Aqui podemos observar que os modelos tiveram resultados mais variados e piores que utilizando um learning rate menor, o que pode indicar que o modelo esteja começando a dar “overfit” ao conjunto de treinamento ou que este learning rate seja alto demais para essas arquiteturas. Vale notar que o teste de melhor resultado foi utilizando apenas uma camada de convolução e balanceando pelo peso nas classes, alcançando 72.6% de acurácia. Entretanto, o modelo com o pior resultado em acurácia nos testes utilizando learning rate de 0,01 continua sendo superior a este, o que leva a conclusão que o balanceamento dos

Tabela 8 – Comparação dos resultados de acurácia dos modelos desenvolvidos treinados por 500 épocas, com *learning rate* de 0.01, com pré-processamento de “informação por camada”, quantidade diferentes de camadas de convolução e diferentes métodos de balanceamento dos dados.

Camadas de Convolução	Balanceamento dos Dados	Acurácia (%)
1	Nenhum	74.5
1	Peso nas classes	73.1
1	Oversample	73.3
2	Nenhum	74.8
2	Peso nas classes	73.1
2	Oversample	73.5
3	**Nenhum**	**75.6**
3	Peso nas classes	72.9
3	Oversample	73.3

Tabela 9 – Comparação dos resultados de acurácia dos modelos desenvolvidos treinados por 500 épocas, com *learning rate* de 0.1, com pré-processamento de “informação por camada”, quantidade diferentes de camadas de convolução e diferentes métodos de balanceamento dos dados.

Camadas de Convolução	Balanceamento dos Dados	Acurácia (%)
1	Nenhum	72.0
1	**Peso nas classes**	**72.6**
1	Oversample	72.2
2	Nenhum	71.4
2	Peso nas classes	68.6
2	Oversample	71.7
3	Nenhum	70.0
3	Peso nas classes	71.9
3	Oversample	72.1

dados nestas arquiteturas não é vantajoso.

Como tentativa para melhorar os resultados dos melhores modelos, será feito uma última rodada de testes, retornando o learning rate para 0.01 mas aumentando o número de épocas para 1000. Além disso, como agora estamos comparando apenas três modelos, é adicionado a métrica de F1-Score para possuirmos mais informações do desempenho desses modelos. Os resultados deste último teste podem ser encontrados na Tabela 10. Aqui podemos conferir que o aumento de épocas não causaram retorno de acurácia aos modelos avaliados, podendo ser observado uma perda de desempenho em todas as arquiteturas. Estes resultados podem indicar novamente que os modelos estão caminhando em direção ao "overfit" dos dados, devido a perda de acurácia no conjunto de teste.

Tabela 10 – Comparação dos resultados de acurácia e F1-Score dos modelos desenvolvidos por 1000 épocas, com *learning rate* de 0.1, com pré-processamento de "informação por camada", quantidade diferentes de camadas de convolução e diferentes métodos de balanceamento dos dados.

Camadas de Convolução	**Acurácia (%)**	**F1-Score (%)**
1	73.5	72.2
2	**74.2**	**72.8**
3	73.2	72.2

Com os resultados obtidos neste trabalho, podemos verificar que o teste com melhores resultados de acurácia foi utilizando uma arquitetura de três camadas de convolução, sem balanceamento dos dados, sendo treinado por 500 épocas e learning rate de 0,01. Considerando este modelo como a melhor arquitetura para solucionar o problema de predição de dificuldade, será feito uma análise mais profunda sobre seus resultados.

Uma maneira simples de avaliar o desempenho de um modelo de aprendizado de máquina em problemas de classificação é utilizando o método chamado "Relatório de Classificação" da biblioteca *Scikit-Learn* (PEDREGOSA et al., 2011). Este método retorna a avaliação de algumas métricas, comparando os resultados preditos pelo modelo em comparação aos resultados reais. As métricas utilizadas são as seguintes (HOSSIN; SULAIMAN, 2015):

- **Precisão** - Habilidade do classificador identificar apenas as instâncias corretas de cada classe.
- **Recall** - Habilidade do classificador em encontrar todas as respostas corretas de cada classe.
- **F1-Score** - média harmônica da precisão e sensibilidade.

- **Quantidade** - número de ocorrências da classe no conjunto de teste.

A impressão do relatório de classificação do melhor modelo desenvolvido pode ser verificado na Tabela 11. Com estas informações, podemos ter uma percepção mais completa do desempenho do modelo. Como pode ser observado, a classe de dificuldade que o modelo possuí maior facilidade de detecção é a “Muito Fácil”, onde 91% desses mapas foram classificados corretamente. Outro ponto de destaque é a dificuldade “Fácil”, onde o modelo obteve 78% de precisão nos acertos, o que pode ser causado por ser a classe com maior quantidade de exemplos de treino e teste. Além disso, as classes “Médio” e “Difícil” tiveram os piores resultados, obtendo um F1-Score de 68% e 62%, respectivamente. Entretanto, apesar da dificuldade “Muito Difícil” ser a com menor quantidade de exemplos para testes, ela conseguiu uma precisão próxima das duas melhores classes. Com estes resultados, podemos supor que o modelo tem mais facilidade em classificar as dificuldades em cada extremo e tendo problemas para decidir as dificuldades próximas de “Médio”.

Tabela 11 – Relatório de Classificação do melhor modelo desenvolvido.

Dificuldade	**Precisão (%)**	**Recall (%)**	**F1-Score (%)**	**Quantidade**
Muito Fácil	77	**91**	**83**	541
Fácil	**78**	76	77	831
Médio	70	67	68	578
Difícil	64	61	62	442
Muito Difícil	77	70	73	367

Uma maneira de poder verificar como estão distribuídos os erros e acertos do modelo é através de uma Matriz de Confusão (TING, 2010). Esta representação dos resultados permite avaliar como ficou cada uma das predições do modelo, indicando onde o mesmo errou e como foi este erro. A ilustração da Matriz de Confusão do melhor modelo desenvolvido pode ser encontrada na Figura 10. Aqui podemos observar o que foi suposto pelo Relatório de Classificação, onde pode ser verificado que as classes nos extremos tem mais facilidade de acertar, principalmente a dificuldade “Muito Fácil”. Além disso, é possível verificar que os erros nas outras classes se encontram principalmente nas dificuldades adjacentes, o que é uma boa indicação da capacidade do modelo em predizer a dificuldade de um mapa.

Por fim, é válido ressaltar que o modelo *AlexNet* obteve uma acurácia superior ao melhor modelo desenvolvido, o que é uma indicação que esta rede seria melhor para solucionar o nosso problema. Entretanto, a diferença é de apenas 0.2% de ganho no conjunto de teste e levando um tempo de treinamento superior aos nossos modelos desenvolvidos. Ao fim deste trabalho, no capítulo de Apêndice, é possível verificar

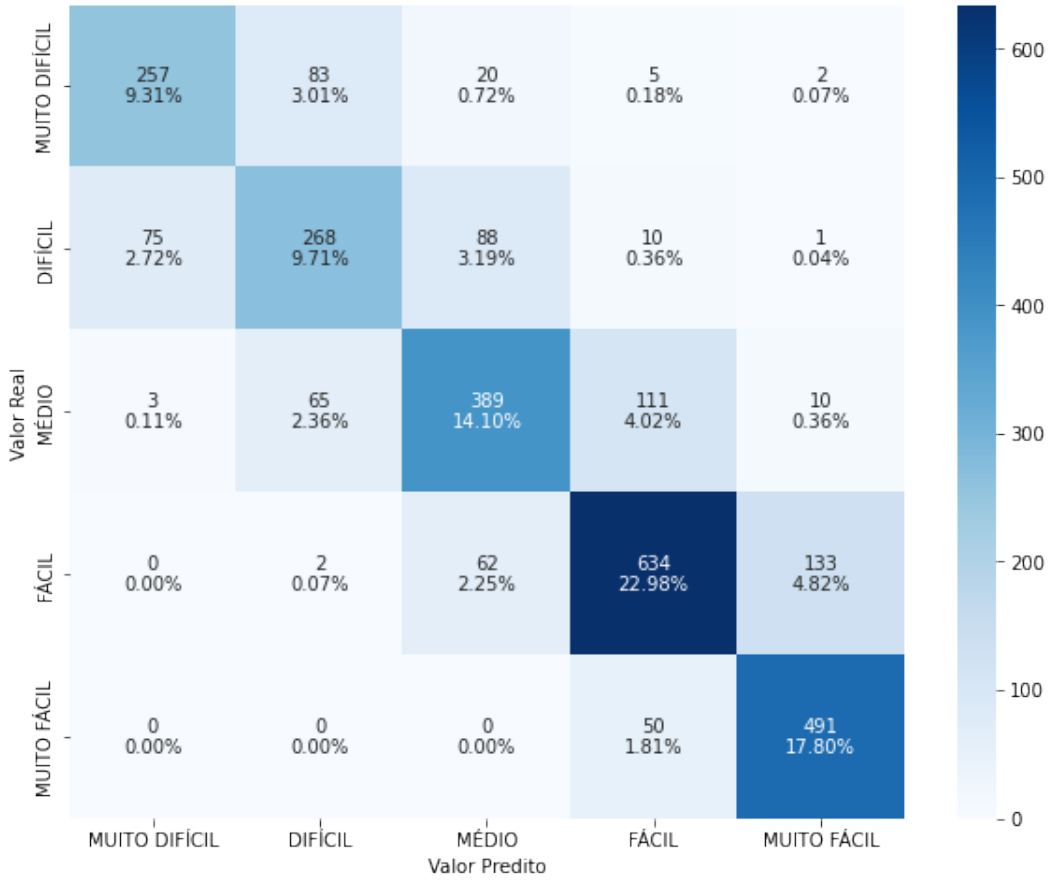

Figura 10 – Matriz de confusão do melhor modelo desenvolvido.

o Relatório de Classificação na Tabela 12 e a matriz de confusão dos resultados da AlexNet na Figura 11.

6 CONCLUSÃO

Esse trabalho apresentou uma maneira de avaliar a dificuldade de mapas de jogos *Match-Three* utilizando Redes Neurais Convolucionais, sem a necessidade de simulação de múltiplas jogadas. Com os diversos testes dos modelos criados, foi possível fazer uma análise da dificuldade de uma fase apenas observando seu formato e seus objetivos, alcançando uma acurácia de 75,6%. O melhor resultado do modelo foi em prever mapas de dificuldade “Muito Fácil”, obtendo uma Precisão de 77%, Recall de 91% e F1-Score de 83%.

Em comparação aos trabalhos relacionados, o modelo proposto possui a vantagem em tempo de processamento, pois retira a necessidade de simular o mapa inúmeras vezes para recuperar seu desempenho médio. Entretanto, analisando a acurácia dos resultados, esta comparação não é possível, pois outros trabalhos tentam predizer o exato valor de desempenho médio humano, enquanto este trabalho ataca o problema avaliando a dificuldade em diferentes categorias. Além disso, a métrica usada para dificuldade não é equivalente aos outros trabalhos, que utilizam a porcentagem de vitórias enquanto este trabalho utilizou a porcentagem de objetivos completados.

Analisando os resultados do melhor modelo desenvolvido, foi possível avaliar que o mesmo possuí uma boa lógica para predizer as dificuldades, onde a grande maioria dos seus erros se encontram nas dificuldades adjacentes da correta. Este erro pode ser causado pelos motivos que a métrica de desempenho médio não é um valor completamente confiável, pois seu resultado é gerado em um número de cem simulações, podendo causar variações que podem levar a uma dificuldade classificada diferente. Para tentar sanar tal problema, uma nova leva de testes podem ser feitos ao desempenho médio humano, calculando esta métrica através de um número maior de simulações. Além disso, a divisão em um número menor de dificuldades poderia facilitar o modelo em agrupar os mapas nas classes corretamente ou até mesmo trabalhar com intervalos com distâncias distintas.

Por fim, a possibilidade de se avaliar mapas com dados de jogadores reais causaria resultados mais confiáveis, o que permite um melhor estudo de como melhorar o desempenho dos modelos nesta tarefa. Além disso, uma etapa seguinte para este

projeto seria a predição do valor real da dificuldade, trabalhando como um problema de regressão. Desta maneira, seria possível aplicar uma modificação no próprio gerador de mapas desenvolvido em Burtet (2019), possibilitando a criação de conteúdo de forma mais rápida e dinâmica, além de uma possível validação do conteúdo gerado manualmente, sem a necessidade de testes com jogadores reais.

REFERÊNCIAS

AGARAP, A. F. Deep learning using rectified linear units (relu). **arXiv preprint arXiv:1803.08375**, [S.l.], 2018.

Bejeweled Wiki | Fandom. **Captura de tela de Bejeweled**. [Online; acessado 10 de março, 2021]. Disponível em: <https://bejeweled.fandom.com/wiki/Bejeweled>.

BENTÉJAC, C.; CSÖRGO, A.; MARTÍNEZ-MUÑOZ, G. A Comparative Analysis of XGBoost. **CoRR**, [S.l.], v.abs/1911.01914, 2019.

BOVIK, A. C. Chapter 3 - Basic Gray Level Image Processing. In: BOVIK, A. (Ed.). **The Essential Guide to Image Processing**. Boston: Academic Press, 2009. p.43–68.

BROWNE, C. B. et al. A Survey of Monte Carlo Tree Search Methods. **IEEE Transactions on Computational Intelligence and AI in Games**, [S.l.], v.4, n.1, p.1–43, mar 2012.

BURTET, J. **Geração Procedural de Mapas para Jogos Match-Three**. 2019. Bachelor's thesis — Universidade Federal de Pelotas (UFPEL).

CHEN, T.; GUESTRIN, C. XGBoost: A Scalable Tree Boosting System. In: ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 22., 2016, New York, NY, USA. **Proceedings...** ACM, 2016. p.785–794. (KDD '16).

DENISOVA, A.; GUCKELSBERGER, C.; ZENDLE, D. Challenge in Digital Games: Towards Developing a Measurement Tool. In: CHI CONFERENCE EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2017., 2017, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2017. p.2511–2519. (CHI EA '17).

Gudmundsson, S. F. et al. Human-Like Playtesting with Deep Learning. In: IEEE CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND GAMES (CIG), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–8.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep Residual Learning for Image Recognition. **CoRR**, [S.l.], v.abs/1512.03385, 2015.

HOSSIN, M.; SULAIMAN, M. A review on evaluation metrics for data classification evaluations. **International Journal of Data Mining & Knowledge Management Process**, [S.l.], v.5, n.2, p.1, 2015.

IOSUP, A. POGGI: generating puzzle instances for online games on grid infrastructures. **Concurrency and Computation: Practice and Experience**, [S.l.], v.23, n.2, p.158–171, 2011.

JUUL, J. SWAP ADJACENT GEMS TO MAKE SETS OF THREE: A HISTORY OF MATCHING TILE GAMES. **ARTIFACT**, [S.l.], v.1, p.205–216, 12 2007.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). **Advances in Neural Information Processing Systems 25**. [S.l.]: Curran Associates, Inc., 2012. p.1097–1105.

MUSIL, S. **Candy Crush Saga tops iTunes app download list for 2013 - CNET**. Disponível em: <https://www.cnet.com/news/candy-crush-saga-tops-itunes-app-download-list-for-2013/>. Acesso em: 2019-04-23.

PASZKE, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: WALLACH, H. et al. (Ed.). **Advances in Neural Information Processing Systems 32**. [S.l.]: Curran Associates, Inc., 2019. p.8024–8035.

PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, [S.l.], v.12, p.2825–2830, 2011.

POROMAA, E. R. **Crushing Candy Crush** : Predicting Human Success Rate in a Mobile Game using Monte-Carlo Tree Search. 2017. Dissertação (Mestrado em Ciência da Computação) — KTH, School of Computer Science and Communication (CSC).

SILVER, D. et al. Mastering the game of Go without human knowledge. **Nature**, [S.l.], v.550, p.354–, Oct. 2017.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, [S.l.], v.15, n.1, p.1929–1958, 2014.

STEVEN, R. **Creating Candy Crush: Behind the scenes at King's Stockholm studio**. Disponível em: <https://www.creativereview.co.uk/creating-candy-crush/?fbclid=IwAR3JLA5qTCFqxi9ycGDCgn6zVwhXCXwCNMUrInX KORLzznCJvpMg1UhDXNk>. Acesso em: 2019-04-23.

TING, K. M. Confusion Matrix. In: SAMMUT, C.; WEBB, G. I. (Ed.). **Encyclopedia of Machine Learning**. [S.l.]: Springer, 2010. p.209.

UMESH, P. Image Processing in Python. **CSI Communications**, [S.l.], v.23, 2012.

WEBSTER, A. **Half a billion people have installed 'Candy Crush Saga' - The Verge**. Disponível em: <https://www.theverge.com/2013/11/15/5107794/candy-crush-saga-500-million-downloads>. Acesso em: 2019-04-23.

Apêndices

APÊNDICE A – Resultados completos do modelo AlexNet

Tabela 12 – Relatório de Classificação da AlexNet

Dificuldade	Precisão (%)	Recall (%)	F1-Score (%)	Quantidade
Muito Fácil	**84**	**90**	**87**	541
Fácil	81	81	81	831
Médio	71	69	70	578
Difícil	61	61	61	442
Muito Difícil	75	70	72	367

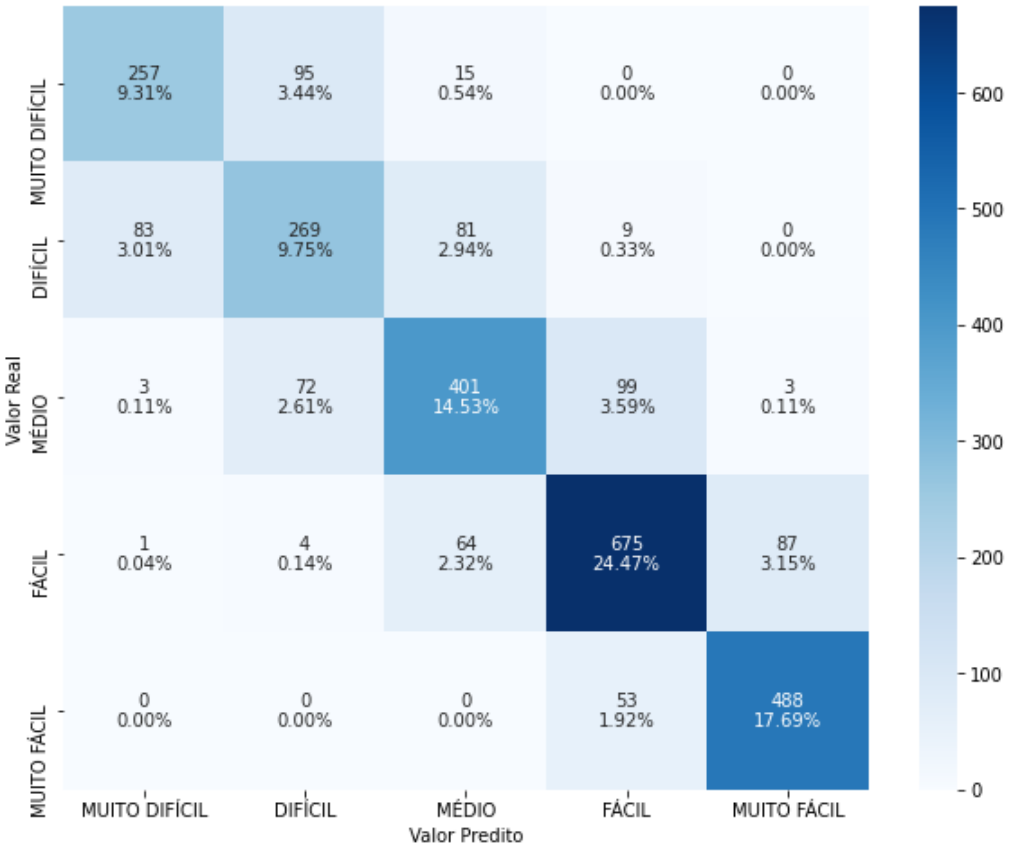

Figura 11 – Matriz de confusão do modelo AlexNet