UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GABRIEL SCHUBERT SILVA ARAÚJO

APRENDIZADO PROFUNDO APLICADO À CLASSIFICAÇÃO DE PEÇAS DE XADREZ

CAMPINA GRANDE - PB

2023

GABRIEL SCHUBERT SILVA ARAÚJO

APRENDIZADO PROFUNDO APLICADO À CLASSIFICAÇÃO DE PEÇAS DE XADREZ

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Herman Martins Gomes.

CAMPINA GRANDE - PB

2023

GABRIEL SCHUBERT SILVA ARAÚJO

APRENDIZADO PROFUNDO APLICADO À CLASSIFICAÇÃO DE PEÇAS DE XADREZ

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Herman Martins Gomes

Orientador – UASC/CEEI/UFCG

Professora Dr. Leandro Balby Marinho

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 14 de fevereiro de 2023.

CAMPINA GRANDE - PB

ABSTRACT

This research presents a solution based on deep learning neural networks for the classification of chess pieces. The goal is to evaluate the applicability of these algorithms in contexts such as robotics, for example. With that in mind, different neural network models were trained, with different architectures and hyperparameters. Later, the metrics accuracy, precision, recall and f1-score were calculated for each trained model and these metrics were compared to define the model with the best performance. The goal of each model was to correctly classify the input image into one of thirteen classes, these classes being 12 chess pieces and one class representing the empty space of the board. In this way, a model with 99.15% accuracy in the recognition of chess pieces was reached, this model was based on the MobileNet architecture and had as the best parameters found: learning rate of 0.00019, 256 neurons in the dense layer and 43 of the first layers frozen. Furthermore, pre-trained weights from ImageNet were used in this model. These results show the effectiveness of deep neural networks in classifying chess piece images.

Aprendizado Profundo Aplicado à Classificação de Peças de Xadrez

Gabriel Schubert Silva Araújo
Unidade Acadêmica de Sistemas e Computação
Campina Grande, Paraíba, Brasil
gabriel.schubert.araujo@ccc.ufcg.edu.br

Herman Martins Gomes
Unidade Acadêmica de Sistemas e Computação
Campina Grande, Paraíba, Brasil
hmg@computacao.ufcg.edu.br

RESUMO

Esta pesquisa apresenta uma solução fundamentada em redes neurais de aprendizado profundo para a classificação de peças de xadrez. O objetivo é avaliar a aplicabilidade destes algoritmos em contextos como na robótica, por exemplo. Pensando nisso, foram treinados diferentes modelos de redes neurais, com arquitetura e hiperparâmetros diferentes. Posteriormente, calculou-se as métricas acurácia, precisão, revocação e f1-score para cada modelo treinado e comparou-se estas métricas para se definir o modelo com melhor desempenho. O objetivo de cada modelo foi classificar corretamente a imagem de entrada em uma de treze classes, sendo estas classes 12 peças de xadrez e uma classe que representa o espaço do tabuleiro vazio. Desta forma, chegou-se a um modelo com 99.15% de acurácia no reconhecimento de peças de xadrez, este modelo foi fundamentado na arquitetura MobileNet e teve como melhores parâmetros encontrados: learning rate de 0.00019, 256 neurônios na camada densa e 43 das primeiras camadas congeladas. Além disso, utilizou-se os pesos pré-treinados da ImageNet neste modelo. Estes resultados mostram a eficácia das redes neurais profundas na classificação de imagens de peças de xadrez.[1]

Palavras-chave

Aprendizagem Profunda, Redes neurais, Processamento digital de imagem, Classificação de objetos em imagens.

1. INTRODUÇÃO

A classificação de imagens é uma das aplicações de inteligência artificial, podendo ser usada em diversos contextos. O uso de inteligência artificial dentro da indústria tem perspectiva de crescimento para os próximos anos, de acordo com a empresa de pesquisa Market and Markets, a IA crescerá para uma indústria de U$ $407 bilhões até 2027 [13].

Nessa perspectiva, diante de um enorme crescimento do uso da Inteligência Artificial, nota-se a relevância da pesquisa na área de classificação de imagens, contribuindo para um melhor uso dessa tecnologia.

Ao observar os séculos passados, percebe-se que a construção de uma máquina que pudesse jogar xadrez de forma autônoma é algo que tem fascinado uma grande quantidade de pessoas. Por décadas, diversos pesquisadores na área de inteligência artificial têm se dedicado ao desenvolvimento de algoritmos que jogam xadrez, pode-se começar citando Alan Turing que fez o primeiro programa de xadrez em 1950, embora tenha sido programado com regras simples devido ao limitado poder computacional da época. Em 1957 o primeiro programa funcional que joga xadrez foi desenvolvido, mas ainda não era capaz de vencer um oponente humano. Posteriormente, em 1996, o Deep Blue da IBM ganhou do mestre de xadrez Garry Kasparov [1].

Pensando-se em continuar este legado histórico, ao se pensar em um robô autônomo de xadrez, que joga em um tabuleiro real e efetivamente veja as peças, as reconheça e as mova para o lugar desejado, nota-se a importância de se obter a maior acurácia possível no reconhecimento das peças de xadrez, para tanto é necessário selecionar o melhor modelo e os parâmetros mais adequados.

Vale ressaltar que são necessários vários passos para se criar um robô autônomo que joga xadrez, dentre eles estão: detecção do tabuleiro, identificação das peças de xadrez, classificação das peças de xadrez, planejamento de movimento controle robótico e outros. Neste estudo tratou-se apenas da classificação das peças de xadrez e sua otimização.

Pode-se apontar as Redes Neurais Convolucionais (CNNs) como muito populares no reconhecimento de imagens. Além disso, esta classe de redes neurais possui vários subtipos diferentes, cada um com suas vantagens e características próprias, dentre as arquiteturas de CNNs mais populares estão: VGGNet, ResNet, MobileNet. Neste estudo, comparou-se o desempenho de modelos fundamentados na arquitetura VGGNet com o desempenho dos fundamentados na arquitetura MobileNet.

Além de diferentes vertentes de modelos, também podemos experimentar diferentes parâmetros para cada modelo, obtendo um ajuste melhor a nosso conjunto de dados.

Portanto, indaga-se: Qual modelo, dentre os escolhidos para serem testados, possui o melhor desempenho para a classificação de peças de xadrez?

Dessa forma, o objetivo da presente pesquisa foi avaliar diferentes modelos de redes neurais e verificar qual possui o melhor desempenho para classificação de peças de xadrez.

Para tanto, foi obtida uma base de dados para realizar o treinamento das redes neurais e logo após, as métricas dos modelos escolhidos foram comparadas para verificar qual teve melhor desempenho.

Partiu-se da suposição de que os modelos que iriam apresentar o melhor desempenho seriam aqueles fundamentados na arquitetura VGG16, porque estes modelos se mostraram superiores para a classificação de peças do jogo digital Progster, como mostrou *Silva* [2].

Neste sentido, foi realizada uma pesquisa de finalidade básica estratégica, objetivo descritivo e

exploratório, com abordagem qualitativa e realizada com procedimentos bibliográficos, documentais e experimentais.

2. PESQUISAS RELACIONADAS

No campo da classificação de peças de tabuleiro, pode-se mencionar *Silva* [2], que utiliza de aprendizagem profunda para classificar as peças e componentes do tabuleiro eletrônico do jogo Progster. No trabalho de Silva[2], a proposta é facilitar o aprendizado de lógica de programação para crianças, ou seja, se tem uma aplicação na área da educação.

Y. Xie, G. Tang e W. Hoff [14] por sua vez, propõem um eficiente reconhecedor de peças de xadrez 3d, tendo como objetivo usar este reconhecedor em um assistente de xadrez para realidade aumentada. Para tanto, usou-se *Oriented Chamfer Matching*, além disso também se fez uma comparação de desempenho com alguns modelos CNN. Sendo assim, testou-se o desempenho do GoogleNet, VGG 16, ResNet50 e do Oriented Chamfer. Dessa forma, notou-se que o modelo com arquitetura GoogleNet obteve a acurácia de 98.14% no reconhecimento de peças de xadrez, valor superior aos outros modelos e métodos testados.

Há também trabalhos como o de *Chen* [15], que propõe a criação de um robô humanoide que possa jogar contra um ser humano sem a necessidade de um ambiente com luz controlada e de modificar as peças ou tabuleiro do xadrez, que são deficiências enfrentadas por outros sistemas robóticos. Para tanto, é necessário um sistema de visão computacional robusto.

3. FUNDAMENTOS

3.1 JOGO DE XADREZ

O Xadrez é um jogo de tabuleiro de 64 casas, cada jogador tem inicialmente 16 peças de 6 tipos, cada qual com sua importância, movimentos e possibilidades de captura específicos [5].

Para se atingir o objetivo do xadrez, o enxadrista deve manipular as suas peças de modo a deixar o rei do adversário sem saída, chama-se isto de xeque-mate. Ao atingir tal condição, o jogo chega ao fim [6].

Devido a questões de escopo, não será exposto aqui em detalhes as regras de movimentação do jogo. Porém, caso o leitor tenha interesse em saber mais sobre as regras, recomenda-se a consulta ao livro *Official Rules Of Chess* [4].

3.2 INTELIGÊNCIA ARTIFICIAL

Definições mais recentes de Inteligência Artificial discorrem sobre "imitar o comportamento humano inteligente". Tal imitação não é tarefa fácil porque um programa de computador deve ser capaz de fazer muitas coisas diferentes para ser chamado de inteligente [16].

Ao invés de procurar uma definição geral de Inteligência Artificial, é possível restringir-se a definição de sistemas inteligentes, dentre as diversas classificações possíveis, a maioria dos sistemas se enquadram em uma das quatro categorias seguintes: sistemas que pensam como seres humanos, sistemas que agem como seres humanos, sistemas que pensam de forma racional e sistemas que agem de forma racional [16].

3.3 APRENDIZAGEM DE MÁQUINA

Monard [7] define aprendizagem de máquina como uma área da IA cujo o objetivo é o desenvolvimento de técnicas computacionais sobre o aprendizado, bem como a construção de sistemas capazes de adquirir conhecimento de forma automática.

Com a Aprendizagem de máquina ou Machine Learning (ML), um computador pode aprender determinada tarefa sem precisar ser programado previamente. A partir da exposição de um modelo de aprendizagem de máquina a um conjunto robusto de dados, esse modelo se utiliza destes dados para extrair as características presentes nele. Existem vários paradigmas de programação nesta área, uma das mais populares hoje em dia são as redes neurais [3].

3.4 REDES NEURAIS

A abordagem convencional da computação é a de usar um conjunto de instruções pré-programadas. Essa abordagem data dos trabalhos de Babbage, Turing e von Neumann. As redes neurais representam o paradigma conexionista da IA, no qual a solução para o problema é aprendida a partir de um conjunto de exemplos. A inspiração para tal modelo veio a partir de estudos a respeito de como acontece o processamento de informações em sistemas nervosos biológicos [8].

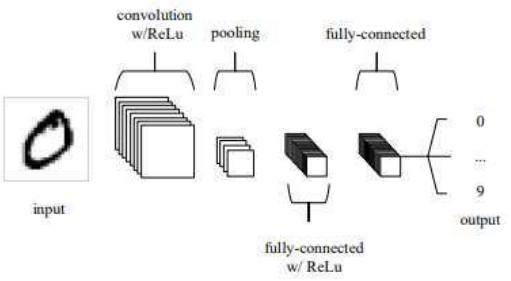
Os principais desafios de uso das redes neurais relacionam-se à necessidade de fornecer um conjunto adequado de amostra de dados para treinar a rede e da possibilidade de erro quando extrapolando para novas regiões do espaço de entrada que são significativamente diferentes daquelas correspondentes aos dados de treinamento [8].

As Redes Neurais Artificiais (RNAs) consistem em diversos nós computacionais interconectados, chamados neurônios, estes trabalham de uma forma distribuída para aprender de uma forma coletiva e otimizar a saída [11].

3.4.1 Redes Neurais Convolucionais

Redes Neurais Convoluionais (CNNs), são um tipo especial de rede neural que possuem uma elevada aptidão à classificação de imagens. Uma das principais diferenças das CNNs em relação às RNAs é a existência de uma camada convolucional em que os neurônios são organizados com compartilhamento de pesos e conectividade local.

Figura 1. Arquitetura simples de uma CNN, possuindo cinco camadas.



Fonte: Figura extraída de [11].

Na figura 1, observamos a arquitetura simples de uma CNN. A camada de convolução tem como objetivo extrair características da imagem, ela trabalha aplicando filtros sobre

cada sub-região da imagem, identificando características como linhas bordas e texturas. Ao longo do treinamento, estes filtros são ajustados para aprender a identificar de forma automática as características importantes de uma imagem. A saída desta camada é um mapa de características que é usado como entrada da próxima camada [11].

A camada de pooling, por sua vez, tem como objetivo diminuir o número de pixels da imagem. Para tanto, pequenos grupos de pixels são selecionados e recebem uma operação matemática, que pode ser de média, por exemplo, A ideia é preservar apenas as características importantes [11].

Já a camada fully connected, ou densa, usa o mapa de características das camadas anteriores para fazer previsões sobre a classificação da imagem. Chama-se *fully connected* (totalmente conectada) porque todos os neurônios desta camada estão conectados a todos os neurônios da camada prévia [11].

Para transformar uma matriz tridimensional em um vetor, utiliza-se a camada de *flatten* ou achatamento. Esta camada geralmente é usada após uma camada de convolução ou *pooling*, preparando a entrada para as camadas totalmente conectadas.

Para se reduzir a dimensionalidade das camadas de convolução sem perder muita informação, usa-se a camada *Global Average Pooling* (GAP), ela funciona mantendo apenas as médias dos valores de ativação de cada camada.

A camada de saída usa as informações obtidas nas camadas anteriores para prever a qual classe a entrada pertence, a saída pode ser um vetor com as probabilidades para cada classe [11].

3.5 Sobreajuste

Em modelos de aprendizagem de máquina, existe um problema que ocorre com frequência. Quando o modelo fica excessivamente ajustado para os dados de treinamento mas não consegue generalizar bem para os dados de teste, temos o que é chamado de sobreajuste. Este problema acontece porque o modelo acaba se ajustando demais ao conjunto de treinamento, aprendendo inclusive a variações aleatórias e ruídos do conjunto [12].

Para lidar com o sobreajuste, foi decidido testar como os modelos selecionados desempenham com e sem Dropout em todos os modelos deste trabalho.

Um modelo com muitos parâmetros livres treinado exaustivamente sobre um conjunto de dados possivelmente sofrerá de sobreajuste pois o excedente de parâmetros fará com que o modelo não se comporte de forma suave fora dos pontos de treinamento, apresentando erro de generalização [12].

3.5.1 Dropout

Dropout é uma técnica efetiva e popular para combater o sobreajuste. A ideia é de aleatoriamente remover unidades e conexões da rede durante o treinamento, isso previne as unidades de se adaptarem demais [12].

3.5.2 Data Augmentation

Em aprendizado de máquina, não é apenas o algoritmo que interfere no desempenho do modelo, a performance pode aumentar ou diminuir dependendo da quantidade de dados de treino, especialmente na área do aprendizado supervisionado [12].

Pensando nisso, usa-se *data augmentation* como uma estratégia para aumentar a generalização dos modelos em várias áreas de aplicação [12].

Aplicando-se *data augmentation*, pode-se gerar novas imagens baseadas no conjunto de dados disponível, rotacionando as imagens, invertendo horizontalmente ou verticalmente dentre várias outras possibilidades. Neste trabalho, foi usado apenas a inversão horizontal como estratégia de *data augmentation*.

3.6 Transferência de Aprendizado

Com o aprendizado de máquina, as características podem ser extraídas dos dados sem a presença de um especialista no campo ao qual os dados pertencem. No entanto, para realizar essa extração automática, são necessários muitos dados. Como resultado, tornou-se popular a reutilização de modelos treinados em domínios com grandes quantidades de dados e adaptá-los a outros domínios que requerem menos dados [10].

3.7 Otimização de hiperparâmetros

Ajuste de parâmetros ou otimização de hiperparâmetros é o processo que busca encontrar os parâmetros que desempenham melhor em relação a um determinado conjunto de dados.

Este processo pode ser feito experimentalmente, testando-se diversas combinações de parâmetros diferentes e verificando-se quais modelos resultaram em melhores métricas de acordo com o objetivo desejado.

Em nosso estudo, visando a obter melhor desempenho, os modelos sofreram ajuste de parâmetros. Para facilitar esta tarefa, utilizou-se a biblioteca Keras Tuner.

3.8 Revocação, Precisão e Acurácia

Para entender estas métricas, é preciso entender alguns conceitos antes. Para facilitar o entendimento, é possível supor que se está treinando um modelo que detecta se tem um gato numa imagem. Para cada tentativa de predição do modelo, se o modelo acertar a classificação de uma imagem que contém um gato é dita como um verdadeiro positivo (VP). Por outro lado, quando um modelo prediz erroneamente a presença de um gato numa imagem que não tem, trata-se de um falso positivo (FP). Por fim, quando o modelo indica que não tem um gato, havendo um na imagem, trata-se de um falso negativo (FN), já se o modelo acerta a inexistência do gato na imagem, tem-se a situação de um verdadeiro negativo (VN).

Entendendo estes conceitos, pode-se agora partir para o conceito de revocação, que é a proporção dos VPs sobre os reais positivos (RP), os RPs são a soma dos VPs com os FNs. Com esta medida, é possível medir o quão completamente um modelo está detectando a classe positiva, que no exemplo acima seria quando existe um gato na imagem [7].

Já a precisão representa a razão entre os VPs e todos os positivos previstos. Para chegar ao número de todos os positivos previstos, basta somar os VPs com os FPs. Com a precisão, pode-se prever a frequência com que um modelo prevê corretamente uma classe positiva quando ela é positiva [7].

Já para se obter a acurácia, é preciso somar os VPs com os VNs, em seguida, se obtém essa soma e se divide pela soma: VP + VN + FP + FN. A partir da acurácia, pode-se medir a capacidade do modelo identificar tanto as classes positivas quanto as negativas [7].

3.9 F1-Score

F1-Score é uma métrica comumente usada para avaliar o desempenho de classificação de modelos de aprendizado de máquina. Apesar de parecer pouco intuitiva, tal métrica combina revocação e precisão em um único valor, na forma de uma média harmônica.

Tal média possui valores similares a uma média normal, porém quando um dos valores está baixo, a média harmônica tende a se aproximar do menor valor.

4. METODOLOGIA

Em relação à obtenção do conjunto de dados, usou-se uma biblioteca de imagens de peças de xadrez pública. Para treinar os modelos, foi usada a plataforma do Google Colab, utilizaram-se as bibliotecas tensorflow e keras. Para carregar a base de dados a partir de um diretório presente no google drive, utilizou-se a função *image dataset from directory*, disponível na biblioteca tf.keras.

Para descobrir os melhores parâmetros para cada cenário de teste, usou-se o framework KerasTuner, por meio deste conjunto de bibliotecas foi possível testar várias combinações de parâmetros diferentes para cada cenário e verificar qual combinação de parâmetros estava tendo um melhor desempenho. Dessa forma, usou-se, no KerasTuner, a acurácia no conjunto de validação como métrica. Sendo assim, em todos os 8 cenários, a combinação de parâmetros que obteve uma maior acurácia no conjunto de validação, foi usada.

Em seguida, para avaliar os modelos em cada um dos oito cenários, já com os parâmetros ajustados, a biblioteca sklearn.metrics foi usada, através dela foi possível calcular as métricas f1-score, precisão, acurácia, e revocação. Ainda sobre avaliação, usou-se a matplotlib para traçar os gráficos de perda e acurácia, e também para gerar a matriz de confusão dos modelos.

4.1 Conjunto de Dados

O Conjunto de imagens usadas para treinar os modelos foi o *Chess ID Public Data* [9], que é uma base de dados de fotos de peças de xadrez pública. O conjunto de treino original possui 10.373 imagens, porém nesse estudo, a fim de manter o número de imagens igual para cada classe, foram mantidas apenas 3536 imagens, com 272 imagens para cada classe.

No conjunto estão presentes 13 classes: peão preto (bp), peão branco(wp), bispo preto(bb), bispo branco(wb), cavalo preto(bn), cavalo branco(wn), torre preta(br), torre branca(wr), rainha preta(bq), rainha branca(wq), rei preto(bk), rei branco(wk) e a classe vazio(empty) que representa a ausência de peça no local.

Destas 3536 imagens, 70% delas foram usadas para treinar cada modelo, 20% ficou como dados de validação e os 10% restantes foram usados para testar os modelos.

Figura 2. Exemplos de imagens do conjunto de dados escolhido.

Fonte: Chess ID Public Data [9].

4.2 Pré-Processamento

Com objetivo de diminuir o tempo de treinamento e aumentar o desempenho dos modelos treinados, foi aplicada uma técnica de normalização de dados simples, na qual o brilho de cada pixel das imagens foi dividido por 256. Dessa forma, transformou-se os valores das imagens, que anteriormente ficavam entre 0 e 255, para que fiquem entre 0 e 1. Sendo assim, os modelos vão se generalizar mais rápido e ter melhores resultados [10].

4.3 Treinamento das arquiteturas de aprendizado profundo

Após os dados serem processados, inicia-se o processo de implementar as arquiteturas escolhidas. Para isso, utilizou-se as bibliotecas Tensorflow e Keras, conforme informado anteriormente.

Além de treinar arquiteturas diferentes, alguns cenários diferentes foram testados, como podemos observar na tabela 1.

Neste estudo, a arquitetura base dos modelos consistia primeiramente de uma CNN carregada sem as camadas densas. Para os modelos utilizando a arquitetura VGG16 foi adicionada uma camada de achatamento ou *Flatten*, uma camada densa com número de neurônios a serem ajustados, em seguida uma camada de dropout, podendo estar presente ou não dependendo do resultado do ajuste, e por fim uma camada densa com 13 neurônios, que corresponde à predição do modelo. O mesmo foi feito nos modelos utilizando a arquitetura MobileNet, porém no lugar da camada *Flatten* foi utilizada uma camada *GlobalAveragePooling2D*.

Em relação ao número de épocas escolhido, para cada treinamento de cada modelo, foram utilizadas 20 épocas.

Nos cenários em que as arquiteturas usaram pesos pré-treinados, testou-se o congelamento de metade das camadas ou de todas. Por outro lado, quando treinou-se os modelos sem usar os pesos pré-treinados, nenhum congelamento de camadas foi feito.

Tabela 1. Cenários de experimentação

Modelo	Cenário
VGG16	Utilizando pesos de modelos pré treinados
MobileNet	Utilizando pesos de modelos pré treinados
VGG16	Utilizando pesos de modelos pré treinados e data augmentation
MobileNet	Utilizando pesos de modelos pré treinados e data augmentation
VGG16	Sem utilizar pesos de modelos pré treinados
MobileNet	Sem utilizar pesos de modelos pré treinados
VGG16	Sem utilizar pesos de modelos pré treinados e utilizando data augmentation
MobileNet	Sem utilizar pesos de modelos pré-treinados utilizando data augmentation

Para cada um dos cenários de experimentação, foram buscados parâmetros que obtivessem um maior desempenho em relação à acurácia na classificação do conjunto de validação, para tanto, utilizou-se a biblioteca Keras Tuner. É possível verificar quais parâmetros foram otimizados na tabela 2.

Tabela 2. Hiperparâmetros que foram otimizados neste estudo

Learning rate	Variando de 1e-6 a 1e-3, os valores foram gerados dentro deste intervalo, mas seguindo uma distribuição logarítmica.
Neurônios na camada densa antes da camada de saída	Possíveis valores: 128,256,512.
Dropout	Podendo estar presente ou não no modelo, estando presente, um *dropout* de 40% era aplicado entre a camada densa e a camada de saída.
Camadas congeladas (no caso de modelos com pesos pré-treinados)	Possíveis valores: congelar metade ou todas as camadas do modelo, seja o VGG 16 ou o Mobile Net.

Sendo assim, se usou o método *Random Search* do Keras Turner. Através deste algoritmo, combinações aleatórias de hiperparâmetros são gerados, considerando os conjuntos de valores previamente definidos para serem testados.

A busca pelos melhores hiperparâmetros durou algo em torno de 10 a 20 minutos para cada cenário, no caso de estar se usando uma GPU *Premium* do Google Colab Pro. Este processo poderia se estender por um dia inteiro ou até mais caso fosse utilizada a GPU gratuita do Google Colab, visto que em testes utilizando tal GPU neste estudo, o treinamento de cada modelo com combinações de hiperparâmetros diferentes chegou a durar 3 horas.

Os experimentos deste estudo foram feitos no ambiente computacional do Google Colab, é importante destacar que, devido à restrição de poder computacional em tal ambiente, utilizou-se o parâmetro *max_trials*=10 do *RandomSearch*. Sendo assim, a nossa busca por combinações aleatórias de hiperparâmetros fez 10 experimentações com parâmetros diferentes para cada cenário apresentado na tabela 1.

Tendo encontrado os parâmetros que resultaram em uma maior acurácia no conjunto de validação, treina-se um modelo com esta configuração novamente, para que se possa plotar os gráficos de perda e de acurácia, gerar métricas de acurácia, *recall* ou revocação, f1-score e precisão, e plotar a matriz de confusão.

4.4 Testes

Tendo-se obtido os modelos com os parâmetros otimizados para cada cenário, cada modelo treinado tenta prever quais eram os rótulos presentes nas imagens dos dados de teste, em seguida, calcula-se a acurácia, revocação, precisão e o f1-score do modelo para o cálculo da precisão, revocação e f1 score, utiliza-se a biblioteca sklearn.metrics, para tanto, utiliza-se o parâmetro "*weighted*". Dessa forma, as métricas são calculadas separadamente para cada classe, em seguida, uma média ponderada pelo número de amostras em cada classe é feita.

5. AVALIAÇÃO EXPERIMENTAL

Neste estudo, considerou-se com melhor desempenho as arquiteturas que obtiveram uma maior acurácia.

Logo após o treinamento de cada modelo e a busca por melhores parâmetros, os modelos passaram por uma etapa de testes, a partir dos resultados destes testes, as métricas foram calculadas.

Além disso, também gerou-se a matriz de confusão de cada modelo, tal matriz foi gerada a partir das predições de cada modelo nos dados de teste comparados com os valores reais, também traçou-se o gráfico relativo a acurácia e perda, tendo uma comparação entre o desempenho do modelo nos dados de teste e nos dados de validação.

Na tabela 3, podemos observar os oito cenários propostos, e o desempenho de cada modelo em cenários diferentes. Percebe-se que o cenário que apontou o melhor desempenho foi o que usa a arquitetura MobileNet com pesos pré-treinados e sem *data augmentation*.

Também é possível notar que a arquitetura VGG16, que está no cenário 3 da tabela 1, também obteve um bom desempenho, porém com menores valores para cada métrica.

É notável a melhora de desempenho dos modelos quando usamos transferência de aprendizado, reutilizando os pesos de um modelo pré-treinado com milhares de imagens.

Nas figuras 3 e 4, pode-se observar o gráfico de perda e acurácia para o modelo VGG16 no cenário 3. Nota-se

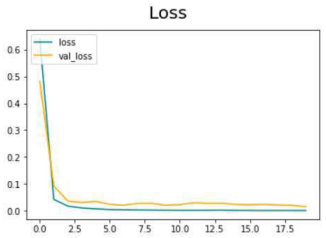
que os valores, tanto para o conjunto de treinamento quanto para o conjunto de validação, tendem a se estabilizar. Apesar do valor de acurácia para o conjunto de treino ser muito alto, o modelo também está desempenhando bem no conjunto de validação, o que indica uma boa capacidade de generalizar o aprendizado.

Na matriz de confusão do cenário 2, que podemos observar na figura 5, notamos que os erros de classificação do modelo são bem pontuais. O modelo tende a confundir, peça preta do rei (bk) com a peça preta da rainha (bq), porém, este foi o único erro neste teste.

Tabela 3. Desempenho dos modelos em diferentes cenários

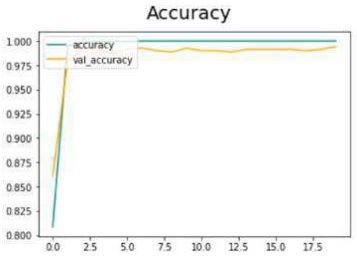
	Precisão	Acurácia	Revocação	F1-Score
VGG16 com pesos pré-treinados	0.9734	0.9716	0.9716	0.9716
MobileNet com pesos pré-treinados	0.9915	0.9915	0.9915	0.9915
VGG16 com pesos pré-treinados e data augmentation	0.9806	0.9801	0.9801	0.9800
MobileNet com pesos pré-treinados e data augmentation	0.9835	0.9830	0.9830	0.9829
VGG16 sem pesos pré-treinados	0.8307	0.8239	0.8239	0.8237
MobileNet sem pesos pré treinados	0.7003	0.6733	0.6733	0.6553
VGG16 sem pesos pré-treinados e com data augmentation	0.8111	0.8011	0.8011	0.8025
MobileNet sem pesos pré-treinados e com data augmentation	0.8853	0.8466	0.8466	0.8478

Figura 3. Gráfico de perda do modelo com arquitetura Mobile Net no cenário 2



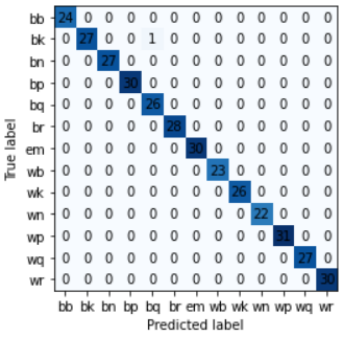
Fonte: Autoria própria.

Figura 4. Gráfico de acurácia do modelo com arquitetura Mobile Net no cenário 2.



Fonte: Autoria própria

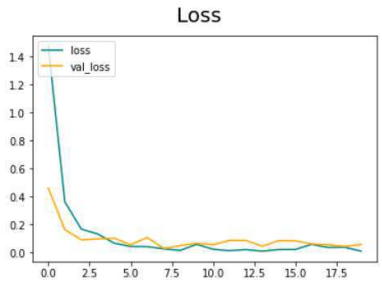
Figura 5. Matriz de confusão do modelo MobileNet no cenário 2



Fonte. Autoria própria.

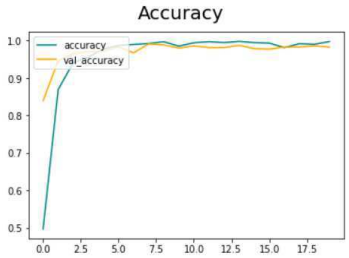
É importante enfatizar que, para implementar um robô como o sugerido por *Urting* [1], é preciso escolher um modelo que tenha a maior acurácia possível, pois um erro de classificação do modelo poderia arruinar o fluxo do jogo.

Figura 6. Gráfico de perda para o modelo com arquitetura VGG16 no cenário 3.



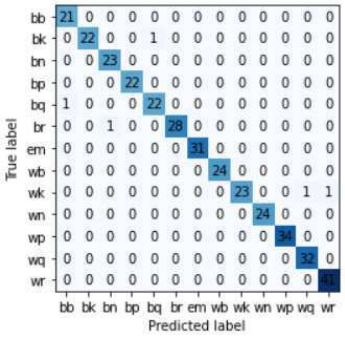
Fonte: Autoria própria

Figura 7. Gráfico de acurácia para o modelo com arquitetura VGG16 no cenário 3



Fonte. Autoria própria

Figura 8. Matriz de confusão para o modelo VGG16 no cenário 3



Fonte. Autoria própria

Nas figuras 6 e 7, referentes ao modelo VGG16, cenário 3, conseguimos observar uma tendência de estabilidade nas linhas de acurácia e perda do conjunto de treinamento e validação,

Na figura 8, notamos que na matriz de confusão do cenário 3 (VGG16 com pesos pré-treinados e data augmentation) o modelo está confundindo a peça preta da rainha (bq) com a peça branca do bispo (bb), além disso, também confunde a torre preta (br) com a peça preta do cavalo (bn), a peça preta do rei (bk) com a bq, a peça branca do rei (wk) com a peça branca da rainha (wq) e o wk com a peça branca da torre (wr).

Pode-se notar que o desempenho indica ser bem inferior ao do modelo do cenário dois, que só cometeu 1 erro no teste.

O modelo mobile Net do cenário 2 ainda poderia errar em algumas situações caso fosse usado em um robô autônomo de xadrez como o de *Urting* [1]. Tem-se a hipótese de que os modelos não foram ainda mais precisos devido à similaridade das classes de imagens, além da necessidade de mais imagens para que os modelos aprendam as características específicas da imagem.

Silva [2] apontou que a alta precisão do melhor modelo encontrado por ele, com precisão de 99.35%, se deve a várias características divergentes nas classes de imagens, e que tal resultado já era esperado.

A partir dos experimentos apresentados, verificou-se que modelos com a arquitetura Mobile Net teriam o desempenho superior neste conjunto de dados específico. Apesar deste fato, as diferenças percentuais das métricas de classificação entre o 1º e 2º colocados na tabela 3 foram pequenas. Um teste estatístico provavelmente indicaria equivalência entre os modelos MobileNet e VGG, mas esta atividade foi deixada como trabalho futuro.

6. CONSIDERAÇÕES FINAIS

Os resultados que foram obtidos nos modelos de aprendizagem profunda foram relativamente satisfatórios, tendo sido obtida uma acurácia de 99.15% no modelo com o melhor desempenho.

Pode-se citar como contribuição a comparação de diferentes modelos com diversos parâmetros. Além disso, a comparação de diferentes arquiteturas (incluindo busca automática dos melhores parâmetros) e como elas desempenharam a partir de uma mesma base de dados.

6.1 Trabalhos Futuros

Para trabalhos posteriores, seria adequado testar uma amplitude maior de parâmetros, em busca de maior precisão na escolha destes parâmetros. Para tanto, seria necessário um ambiente local com uma maior disponibilidade de poder computacional, ou mais investimento em unidades computacionais no google colab, por exemplo.

Além disso, uma implementação dos modelos treinados neste trabalho com um sistema robótico para identificação e movimentação das peças poderia ser desenvolvido, validando o uso destes modelos em outros contextos.

Ainda seria possível também, estudar os casos em que os modelos continuam classificando incorretamente, dessa forma seria possível desenvolver estratégias para suprir essa dificuldade e ter um modelo mais robusto.

Outra possibilidade seria a realização de testes estatísticos para comparar os modelos com um maior rigor matemático.

6.2 Limitações

Uma das maiores limitações deste trabalho foi a restrição de uso do google colab. Inicialmente, uma assinatura do Colab Pro foi adquirida, porém as unidades computacionais disponíveis foram rapidamente esgotadas no desenvolver do projeto. Como a comparação do desempenho de diversos

modelos exige que estes modelos sejam treinados um por um, e posteriormente comparados, a limitação de velocidade do google colab e a limitação de horas de uso por dia fez com que a pesquisa fosse limitada.

7. REFERÊNCIAS

[1] Urting, David e Yolande Berbers. MarineBlue: A Low-Cost Chess Robot (2003).

[2] Silva, A. A. N. da. "Reconhecimento de peças do jogo Progster utilizando aprendizado profundo." 2019, PDF, páginas não numeradas.

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press. (in ACM SIG format)

[4] SCHILLER, Eric. Official Rules Of Chess. Cardoza Publishing, 2003.

[5] Castro, Celso. "Uma história cultural do xadrez." (1994).

[6] Santos, P.S. O que é Xadrez. Brasiliense, 2017.

[7] Monard, M. C., & Baranauskas, J. A. (2003). Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, *1*(1), 32.

[8] BISHOP, Chris M. Neural networks and their applications. Review of Scientific Instruments, v. 65, n. 6, p. 1803-1832, 1994.

[9] Towards Data Science. Board Game Image Recognition Using Neural Networks. [Online]. Towards Data Science. https://towardsdatascience.com/board-game-image-recognition-using-neural-networks-116fc876dafa.

[10] Homem, W. L., & Ufes, P. E. M. (2020). Apostila de Machine Learning. PET Engenharia Mecânica, UFES.

[11] O'Shea, Keiron, and Ryan Nash. "An Introduction to Convolutional Neural Networks." In: arXiv preprint arXiv:1511.08458 (2015).

[12] Ying, X. (2019, February). An overview of overfitting and its solutions. In Journal of physics: Conference series (Vol. 1168, No. 2, p. 022022). IOP Publishing.

[13] MarketsandMarkets Research. (2021). Artificial Intelligence Market. Disponível em: https://www.marketsandmarkets.com/Market-Reports/artificial-intelligence-market-74851580.html.

[14] Y. Xie, G. Tang and W. Hoff, "Chess Piece Recognition Using Oriented Chamfer Matching with a Comparison to CNN," 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 2018, pp. 2001-2009, doi: 10.1109/WACV.2018.00221.

[15] Chen, A. T. Y., & Wang, K. I. K. (2019). Robust computer vision chess analysis and interaction with a humanoid robot. Computers, 8(1), 14.

[16] Kok, J. N., Boers, E. J., Kosters, W. A., Van der Putten, P., & Poel, M. (2009). Artificial intelligence: definition, trends, techniques, and cases. Artificial Intelligence, 1, 270-299.

Sobre os autores:

Gabriel Schubert Silva Araújo. Graduando em Ciência da Computação.

Herman Martins Gomes. Professor orientador