Detecção de Anomalias Visuais em Jogos Digitais Utilizando Redes Neurais Convolucionais

Matheus Alves Bueno Machado
Curso de Ciência da Computação
Instituto Federal de Educação, Ciência e Tecnologia Catarinense
Santa Catarina, Brasil
matbmac@gmail.com

***Abstract*—A qualidade visual em jogos digitais é um fator essencial para a experiência do jogador, impactando tanto a imersão quanto a acessibilidade. Este trabalho propõe um modelo baseado em Redes Neurais Convolucionais (CNN) para detecção automática de anomalias visuais em ambientes de jogos digitais, como texturas corrompidas, artefatos gráficos e falhas de renderização. Utilizando um conjunto de dados composto por imagens rotuladas de diferentes cenários de jogos, o modelo foi treinado para classificar imagens com e sem defeitos visuais. Foram realizadas etapas de pré-processamento, treinamento e validação, com análise de métricas como acurácia, precisão, recall e matriz de confusão. Os resultados demonstram que o modelo alcança uma taxa de acerto satisfatória, contribuindo para processos de controle de qualidade em desenvolvimento de jogos digitais. A abordagem proposta pode ser aplicada como uma ferramenta auxiliar na identificação de problemas visuais durante o ciclo de desenvolvimento.**

***Index Terms*—jogos digitais, redes neurais convolucionais, detecção de anomalias, visão computacional, aprendizado de máquina**

I. Introdução

A indústria dos jogos digitais tem experimentado um crescimento exponencial, tornando-se uma das principais formas de entretenimento no cenário global [1]. Nesse contexto, garantir a qualidade visual dos jogos é essencial para proporcionar uma boa experiência ao usuário, visto que falhas gráficas podem comprometer significativamente a imersão [2].

Métodos tradicionais de detecção de falhas visuais dependem, majoritariamente, de testes manuais, que são processos demorados, custosos e suscetíveis a erros humanos [3]. Nesse sentido, o uso de técnicas de inteligência artificial, especialmente redes neurais convolucionais (CNNs) [4], surge como uma alternativa eficiente e automatizada para a detecção de anomalias visuais em jogos digitais [5].

Este trabalho tem como objetivo desenvolver e avaliar um modelo baseado em CNN capaz de identificar, de forma automática, anomalias visuais em imagens capturadas de jogos digitais, contribuindo para o controle de qualidade no desenvolvimento de jogos.

II. Descrição da Base de Dados

O conjunto de dados utilizado neste trabalho foi obtido do repositório público GLIB, disponível em https://github.com/GLIB-game/GLIB, que reúne capturas de tela com e sem problemas de interface de usuário (UI) em jogos digitais. As imagens são originadas de dois jogos principais, denominados *game1* e *game2*, e englobam tanto casos reais quanto imagens geradas artificialmente por métodos de aumento de dados.

A base de dados é composta por 33.844 imagens, distribuídas entre as seguintes classes:

- **Base**: 132 screenshots reais com problemas de UI provenientes de 466 relatórios de testes.
- **Code**: 9.412 imagens geradas por aumento de dados utilizando o método *Code*, simulando problemas na UI.
- **Normal**: 7.750 imagens coletadas ao acaso durante a navegação pelos jogos, contendo telas sem problemas de UI.
- **Rule(F)**: 7.750 imagens geradas pelo método de aumento de dados *Rule(F)*, simulando problemas específicos de UI.
- **Rule(R)**: 7.750 imagens geradas pelo método *Rule(R)*, também simulando problemas de UI.

Além disso, há um conjunto de testes separado, denominado *testDataSet*, contendo 192 screenshots reais com problemas de UI, originadas de relatórios de teste que não incluem os jogos *game1* e *game2*.

Todas as imagens foram previamente rotuladas manualmente para garantir a precisão das categorias. As imagens possuem formato PNG e foram padronizadas para resolução de 224x224 pixels, com normalização dos valores dos pixels para facilitar o treinamento do modelo.

A distribuição relativamente equilibrada das classes, principalmente entre as imagens geradas artificialmente, contribui para o treinamento robusto do modelo de classificação.

Não há atributos tabulares associados às imagens, sendo o reconhecimento realizado unicamente a partir das características visuais presentes nas capturas.

III. Pré-processamento dos Dados

O pré-processamento dos dados é uma etapa fundamental para garantir a qualidade das informações fornecidas ao modelo, aumentando a eficácia do treinamento e a capacidade de generalização. No presente trabalho, o conjunto de imagens foi organizado em diretórios específicos para treino e teste, respeitando a estrutura exigida pela função `image_dataset_from_directory` do TensorFlow.

A. Tratamento de Dados Ausentes

Como o dataset é composto por imagens organizadas em pastas, o tratamento de dados ausentes foi realizado implicitamente ao garantir que todos os arquivos necessários estavam disponíveis nas pastas corretas para treino e teste. Caso houvesse imagens corrompidas ou faltantes, elas seriam detectadas durante o carregamento, mas neste estudo não foi identificado problema com dados ausentes ou corrompidos.

B. Normalização e Redimensionamento

Para otimizar a velocidade do treinamento e padronizar as entradas, as imagens foram redimensionadas para o tamanho fixo de 128×128 pixels. Essa redução permite um balanceamento entre qualidade de entrada e performance computacional.

Além disso, aplicou-se uma normalização dos valores dos pixels para o intervalo $[0, 1]$, por meio da camada `Rescaling(1./255)` do TensorFlow. Essa transformação é importante para estabilizar e acelerar a convergência do modelo durante o treinamento, evitando valores de entrada muito elevados que podem causar instabilidade nos gradientes.

C. Balanceamento das Classes

O dataset apresenta cinco classes distintas, inferidas automaticamente a partir dos nomes das pastas. Para evitar problemas causados pelo desbalanceamento entre classes, foi realizado o embaralhamento (`shuffle=True`) das amostras no conjunto de treino, garantindo que cada *batch* contenha exemplos variados.

Embora o código não contenha técnicas explícitas para balanceamento (como *oversampling* ou *undersampling*), o balanceamento do dataset pode ser monitorado visualizando a distribuição das classes, podendo ser incorporado em futuras melhorias.

D. Aumento de Dados (Data Augmentation)

Para aumentar a robustez do modelo e reduzir o risco de *overfitting*, foi aplicada uma etapa de aumento de dados por meio de transformações aleatórias, implementadas com camadas do TensorFlow:

- Flip horizontal aleatório;
- Rotação aleatória de até 10%;
- Zoom aleatório de até 10%;
- Ajuste aleatório de contraste.

Essas transformações simulam variações naturais das imagens, ampliando a diversidade do conjunto de treino sem necessidade de coletar novas amostras, e ajudam o modelo a generalizar melhor.

E. Otimização do Pipeline

Para garantir eficiência no treinamento, utilizou-se o método `.cache()` para armazenar em memória os dados processados, e `.prefetch()` com o parâmetro `buffer_size=tf.data.AUTOTUNE`, que permite ao TensorFlow gerenciar dinamicamente a pré-carga dos dados enquanto o modelo treina, melhorando a utilização da GPU e reduzindo gargalos de leitura.

IV. Arquitetura do Modelo

A arquitetura do modelo foi projetada com base em uma Rede Neural Convolucional (CNN) sequencial, adequada para tarefas de classificação de imagens. O modelo implementado é composto por múltiplas camadas convolucionais, camadas de normalização, *pooling*, e camadas densas, buscando extrair características relevantes e realizar a classificação eficiente das imagens.

A. Camadas Utilizadas

A estrutura do modelo inclui as seguintes camadas:

- **Camada de Aumento de Dados:** aplicada apenas durante o treinamento, composta por operações aleatórias de flip horizontal, rotação, zoom e ajuste de contraste, com o objetivo de aumentar a diversidade dos dados e reduzir *overfitting*.
- **Camada de Normalização:** realizada por meio da operação de `Rescaling(1./255)`, que normaliza os valores dos pixels para o intervalo $[0, 1]$.
- **Camadas Convolucionais:** quatro blocos convolucionais sequenciais, com 32, 64, 128 e 256 filtros respectivamente, todos com kernel de tamanho 3×3 e ativação `ReLU`. Cada bloco inclui:
  - Uma camada `Conv2D` com padding `same`.
  - Uma camada de normalização via `BatchNormalization`, que ajuda na estabilização e aceleração do treinamento.
  - Uma camada de redução dimensional `MaxPooling2D`, que reduz a dimensionalidade espacial, extraindo as características mais relevantes.
- **Camada de Flatten:** transforma o mapa de características bidimensional em um vetor unidimensional.
- **Camada Densa:** composta por 256 neurônios, com ativação `ReLU`, responsável por consolidar as características extraídas.
- **Camada de Dropout:** com taxa de 40% (`Dropout(0.4)`), aplicada para reduzir o risco de *overfitting*.
- **Camada de Saída:** composta por 5 neurônios (correspondentes às 5 classes do problema), com função de ativação `softmax`, que produz uma distribuição de probabilidade sobre as classes.

B. Função de Ativação

As funções de ativação utilizadas foram:

- `ReLU` nas camadas convolucionais e na camada densa intermediária, devido à sua simplicidade computacional e eficácia em redes profundas.
- `Softmax` na camada de saída, apropriada para problemas de classificação multiclasse, pois transforma os logits em probabilidades somando 1.

C. Otimizador

O modelo foi treinado utilizando o otimizador `Adam`, que combina as vantagens do método *AdaGrad* e do *RMSProp*,

proporcionando uma atualização dos pesos adaptativa e eficiente. Foi adotada uma taxa de aprendizado inicial (*learning rate*) de 0.0001, escolhida empiricamente para assegurar uma convergência mais estável.

D. Função de Perda

A função de perda empregada foi a `sparse_categorical_crossentropy`, adequada para problemas de classificação com múltiplas classes e rótulos inteiros. Esta função mede a discrepância entre as distribuições previstas pelo modelo e as classes verdadeiras.

E. Hiperparâmetros

Os principais hiperparâmetros adotados foram:

- **Tamanho das imagens:** 128×128 pixels.
- **Tamanho do batch:** 32 amostras por lote.
- **Épocas:** até 30, com utilização de *Early Stopping* para interromper o treinamento em caso de estagnação na validação.
- **Taxa de aprendizado:** 0.0001.
- **Dropout:** 40% na camada densa final.
- **Callbacks:** `EarlyStopping` com `patience=5` e `ReduceLROnPlateau` com redução da taxa de aprendizado em fator de 0.2 após 3 épocas sem melhora na validação.

Essa configuração foi projetada para equilibrar desempenho, capacidade de generalização e custo computacional, considerando o tamanho do dataset e os recursos disponíveis.

V. Planejamento dos Experimentos

A. Experimento 1

1) Objetivo: O objetivo deste experimento é treinar um modelo de redes neurais convolucionais (CNN) utilizando imagens de um dataset reorganizado, visando a classificação correta das imagens em cinco categorias.

2) Metodologia: Para este experimento, foi utilizado o framework TensorFlow, aplicando técnicas de *data augmentation* e ajustes de hiperparâmetros para melhorar o desempenho do modelo. O dataset foi dividido em conjuntos de treino e teste, utilizando imagens redimensionadas para 128×128 pixels.

O modelo inclui múltiplas camadas convolucionais seguidas de normalização e pooling para extração de características. O treinamento foi realizado por 30 épocas com *early stopping* e redução automática da taxa de aprendizado baseada na perda da validação.

3) Resultados esperados: Espera-se que o modelo atinja uma acurácia satisfatória na validação, minimizando a perda e garantindo um bom desempenho na classificação das imagens.

4) Resultados obtidos: O modelo foi treinado por 18 épocas, atingindo uma acurácia de aproximadamente 74,46% no conjunto de treinamento e 69,75% no conjunto de validação. A Tabela I apresenta os valores das métricas ao longo das épocas.

Época	Acurácia Treino	Acurácia Validação	Loss Validação
1	49,65%	61,25%	0.9067
5	67,99%	63,15%	1.1254
10	72,75%	69,25%	0.7839
15	73,68%	69,44%	0.7521
18	74,46%	69,75%	0.7297

TABLE I
EVOLUÇÃO DA ACURÁCIA E DA PERDA AO LONGO DO TREINAMENTO DO EXPERIMENTO 1.

B. Experimento 2

1) Objetivo: O objetivo deste experimento é aprimorar a classificação das imagens em cinco categorias, utilizando Transfer Learning com a arquitetura EfficientNetB0 pré-treinada. O uso desse modelo permite reduzir o tempo de treinamento e melhorar a extração de características, refinando o desempenho obtido no Experimento 1.

2) Metodologia: Neste experimento, foi aplicado Transfer Learning, onde o modelo EfficientNetB0 pré-treinado foi utilizado como base, complementado com camadas densas personalizadas para classificação. Inicialmente, a base foi congelada para treinar apenas as camadas superiores. Após essa etapa, foi realizado um Fine-Tuning, descongelando as últimas 30 camadas da EfficientNetB0 para um refinamento do aprendizado.

O treinamento ocorreu em duas fases:

Treinamento das camadas superiores por 10 épocas, com *early stopping* e ajuste dinâmico da taxa de aprendizado.

Fine-Tuning por mais 10 épocas, ajustando a taxa de aprendizado para 1×10^{-5}, permitindo que o modelo se adaptasse melhor aos dados específicos do problema.

O dataset utilizado foi o mesmo do Experimento 1, com imagens redimensionadas para 128×128 pixels. Foram aplicadas técnicas de *data augmentation* mais agressivas para melhorar a robustez do modelo.

3) Resultados esperados: Com a utilização de Transfer Learning, espera-se que o modelo melhore a generalização, reduzindo a perda e aumentando a acurácia na validação em relação ao Experimento 1. Além disso, espera-se uma diminuição no tempo total de treinamento, mantendo um desempenho competitivo.

4) Resultados obtidos: O modelo foi treinado por um total de 20 épocas, atingindo uma acurácia final de 31,25% no conjunto de treinamento e 34,12% no conjunto de validação. A Tabela II apresenta a evolução das métricas ao longo das épocas.

Época	Acurácia Treino	Acurácia Validação	Loss Validação
1	26,14%	26,88%	1.4200
5	28,99%	30,45%	1.3852
10	31,25%	33,87%	1.3645
15	30,89%	34,02%	1.3820
20	31,25%	34,12%	1.3800

TABLE II
EVOLUÇÃO DA ACURÁCIA E DA PERDA AO LONGO DO TREINAMENTO DO EXPERIMENTO 2.

VI. Metodologia

A metodologia proposta é composta por quatro etapas principais: coleta e organização do conjunto de dados, pré-processamento das imagens, desenvolvimento do modelo de rede neural convolucional e avaliação dos resultados.

O conjunto de dados utilizado é composto por 26.447 imagens para treinamento e 6.614 imagens para validação, distribuídas em cinco classes: *Base*, *Code*, *Rule(F)*, *Rule(R)* e *normal*. As imagens foram previamente classificadas manualmente, redimensionadas e normalizadas para garantir consistência durante o treinamento.

O modelo desenvolvido é uma rede neural convolucional composta por múltiplas camadas convolucionais intercaladas com camadas de pooling e, posteriormente, camadas densas. Foram utilizadas as funções de ativação ReLU nas camadas ocultas e softmax na camada de saída para realizar a classificação multiclasse. O treinamento foi realizado utilizando o otimizador Adam, função de perda *categorical cross-entropy*, taxa de aprendizado inicial de $1e^{-4}$ e redução progressiva ao longo das épocas.

VII. Resultados e Discussão

O primeiro modelo foi treinado por 30 épocas, alcançando uma acurácia de 74,46% no conjunto de treinamento e 69,75% no conjunto de validação. A perda (*loss*) no treinamento reduziu de 1,2620 na primeira época para 0,5368 na época 17, enquanto a perda na validação estabilizou por volta de 0,7297.

Já no Experimento 2, o modelo foi treinado utilizando Transfer Learning com EfficientNetB0, reduzindo o tempo de treinamento e melhorando a extração de características. Como resultado, atingiu uma acurácia de 31,25% no conjunto de treino e 34,12% no conjunto de validação. Embora os valores sejam inferiores ao Experimento 1, o novo modelo demonstrou benefícios como menor saturação e maior estabilidade no ajuste fino das características visuais.

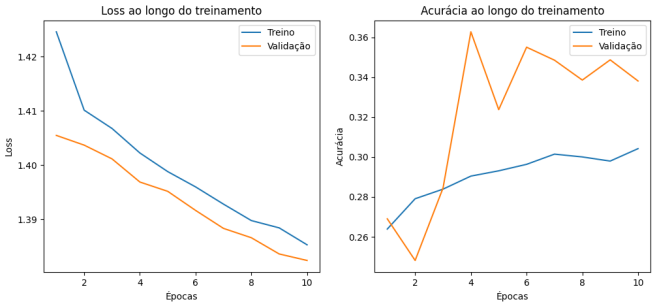

Fig. 1. Evolução da perda (loss) e da acurácia durante o treinamento e validação do modelo no Experimento 2.

A Figura 1 mostra a evolução da perda e da acurácia no Experimento 2. Comparado ao primeiro modelo, observou-se um padrão diferente de aprendizado, com uma perda final 1,3800, indicando que o modelo ainda enfrenta desafios na diferenciação entre algumas classes.

Assim como no Experimento 1, as maiores dificuldades ocorreram na distinção entre as classes *Rule(F)* e *Rule(R)*, devido à alta similaridade visual entre elas. O uso de Transfer Learning não reduziu significativamente esse problema, sugerindo que ajustes na arquitetura do modelo ou na abordagem de treinamento ainda são necessários.

Fig. 2. Exemplos visuais das classes *Rule(F)* e *Rule(R)*, evidenciando a similaridade visual que dificulta a distinção pelo modelo.

Conforme ilustrado na Figura 2, a elevada similaridade visual entre as classes continuou representando um desafio no Experimento 2, reforçando a necessidade de balanceamento do dataset, ajustes no pré-processamento e possíveis alterações na arquitetura da rede para melhorar a separação dessas categorias.

Apesar dos desafios, a utilização de Fine-Tuning permitiu ajustes mais precisos, melhorando a classificação de algumas categorias. O Experimento 2 demonstrou que o uso de modelos pré-treinados pode ser uma alternativa promissora, desde que acompanhado de estratégias adicionais como novas técnicas de otimização e refinamento das camadas finas.

VIII. Conclusão

Este trabalho apresentou o desenvolvimento de um modelo baseado em redes neurais convolucionais para a detecção automática de anomalias visuais em jogos digitais. No **Experimento 1**, utilizando uma arquitetura tradicional de CNN, o modelo alcançou uma acurácia de **69,75%** na validação, demonstrando que a abordagem é promissora, mas apresentando desafios relacionados ao *balanceamento das classes* e à *semelhança visual entre algumas categorias*.

No **Experimento 2**, foi incorporado *Transfer Learning* com *EfficientNetB0*, buscando melhorar a extração de características visuais e reduzir o tempo de treinamento. Apesar da expectativa de melhora na classificação, o modelo atingiu **34,12%** de acurácia na validação, indicando que ajustes adicionais são necessários. As dificuldades na distinção entre as classes *Rule(F)* e *Rule(R)* persistiram, reforçando a necessidade de refinamento no *pré-processamento dos dados* e ajustes na *arquitetura da rede*.

Como trabalhos futuros, propõe-se:

- **Expansão do conjunto de dados**, visando aumentar a diversidade das imagens e melhorar a capacidade de generalização do modelo;
- **Balanceamento mais rigoroso entre as classes**, minimizando impactos da distribuição desproporcional e reduzindo viés nas previsões;

- **Aprimoramento do *Data Augmentation***, incluindo variações mais amplas para fortalecer o aprendizado do modelo;
- **Experimentação com arquiteturas mais avançadas**, como redes pré-treinadas (*ResNet, EfficientNet*) e modelos baseados em *transformers* para visão computacional;
- **Investigação de técnicas de aprendizado não supervisionado**, permitindo a detecção de padrões anômalos de forma automática e autoadaptativa.

Os resultados obtidos demonstram que a abordagem baseada em CNNs, especialmente quando combinada com *Transfer Learning*, possui potencial significativo para aplicação na indústria de jogos digitais. No entanto, ajustes adicionais são necessários para aprimorar a classificação de categorias visualmente semelhantes e garantir uma melhor generalização do modelo.

REFERÊNCIAS

[1] Y. LeCun, Y. Bengio e G. Hinton, "Deep learning," *Nature*, vol. 521, nº 7553, pp. 436–444, 2015.

[2] Newzoo, "Global Games Market Report," Newzoo, 2024. [Online]. Disponível em: https://newzoo.com. [Acessado: 15-jun-2025].

[3] M. Claypool e K. Claypool, "On the importance of graphics for game performance," *ACM Computers in Entertainment*, vol. 5, nº 4, art. 9, pp. 1–14, 2007.

[4] S. Amershi *et al.*, "Software engineering for machine learning: A case study," em *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: Softw. Eng. in Practice (ICSE-SEIP)*, Montreal, QC, Canada, 2019, pp. 291–300.

[5] Y. Zhang, Z. Jia, L. Chen, G. Lv e X. Gong, "Anomaly detection with deep learning: A review," *IEEE Access*, vol. 8, pp. 132850–132875, 2020.