DETECÇÃO AUTOMÁTICA DE DEEPFAKES COM TÉCNICAS DE PROCESSAMENTO DE IMAGENS E CNNs

Carlos Eduardo Landeira Ribeiro, Vítor Gustavo Hornburg

Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

celribeiro@furb.br, vhornburg@furb.br

1 INTRODUÇÃO

O trabalho tem como objetivo desenvolver um modelo capaz de classificar imagens em duas categorias: reais e *deepfakes*. Técnicas modernas de manipulação digital permitem alterar rostos e expressões com alta fidelidade, tornando difícil a identificação visual dessas alterações. Para lidar com esse desafio, foi utilizada uma Rede Neural Convolucional (CNN), capaz de reconhecer padrões sutis presentes em imagens manipuladas, como artefatos artificiais, texturas inconsistentes e variações anormais de iluminação.

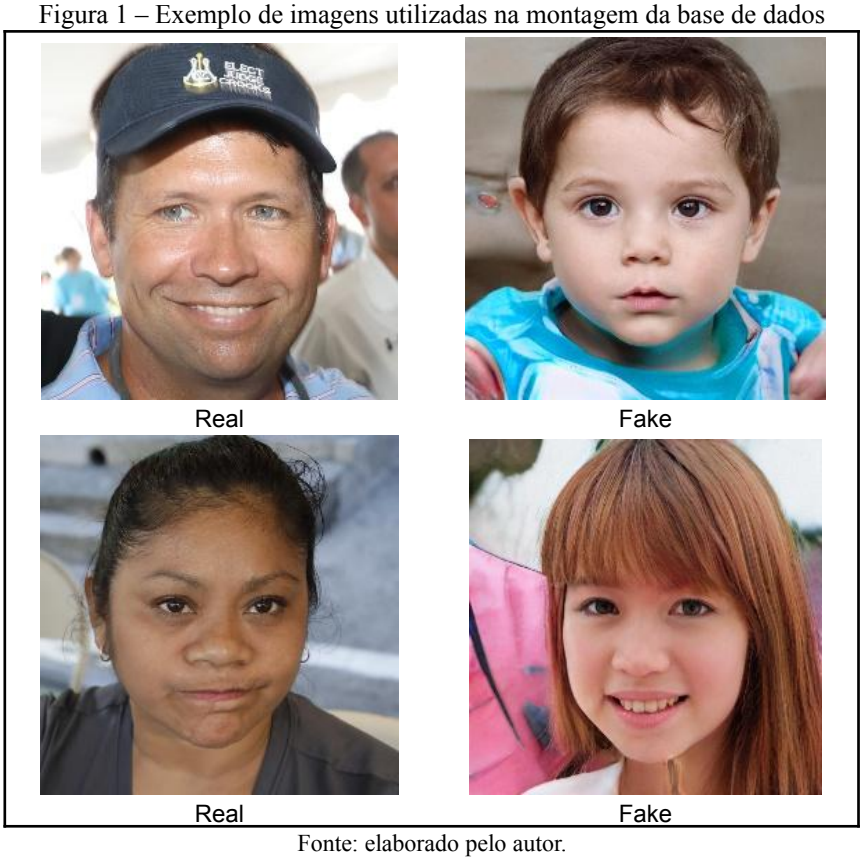
2 DESENVOLVIMENTO

Este capítulo apresenta todas as etapas envolvidas na construção do protótipo de detecção de *deepfakes* desenvolvido neste trabalho. Inicialmente, descreve-se a organização da base de dados utilizada e o processo de preparação das imagens, destacando os passos necessários para garantir consistência e qualidade no treinamento do modelo. Em seguida, é detalhada a arquitetura da rede neural convolucional adotada, bem como os procedimentos de treinamento, validação e ajuste do *threshold*. Por fim, são apresentados os testes realizados, a análise dos resultados numéricos e visuais, além da interpretação das regiões da imagem que mais influenciaram as decisões do modelo.

2.1 Montagem da Base de Dados

Foi utilizado o *dataset* *DeepFake Dataset*, disponível no Kaggle (https://www.kaggle.com/datasets/aryansingh16/deepfake-dataset). A base foi organizada em três subconjuntos independentes: *train*, *valid* e *test*, cada um contendo duas pastas, *real* e *fake*. Essa estrutura garante que o modelo seja treinado, validado e testado em conjuntos distintos, evitando que ele memorize exemplos específicos e permitindo avaliar de forma justa sua capacidade de generalização.

Figura 1 – Exemplo de imagens utilizadas na montagem da base de dados



Fonte: elaborado pelo autor.

2.2 Preparação dos Dados

A preparação das imagens foi realizada no arquivo prepara_dataset.py, seguindo uma sequência de passos que tornam o treinamento mais estável e eficiente. Primeiramente, todas as imagens foram redimensionadas para 224×224 pixels, tamanho padrão em modelos CNN, reduzindo a variação entre exemplos e o custo computacional. Em seguida, cada imagem foi convertida para tensores, estrutura numérica utilizada pelo PyTorch. Os *pixels* também foram normalizados utilizando as médias e desvios padrão do ImageNet, o que ajuda a manter os valores em uma escala consistente entre as entradas. Para aumentar a robustez do modelo, aplicou-se data augmentation apenas no conjunto de treino, incluindo inversão horizontal e ajustes leves de brilho e contraste, ampliando a variedade de exemplos e reduzindo o risco de dependência de padrões específicos. Caso alguma imagem estivesse corrompida, ela era substituída por uma imagem preta, garantindo que o pipeline não fosse interrompido. Após essas etapas, todas as imagens foram organizadas em *batches* de 16 amostras, o que acelera o processamento e otimiza o uso da memória durante o treino.

2.3 Arquitetura da Rede

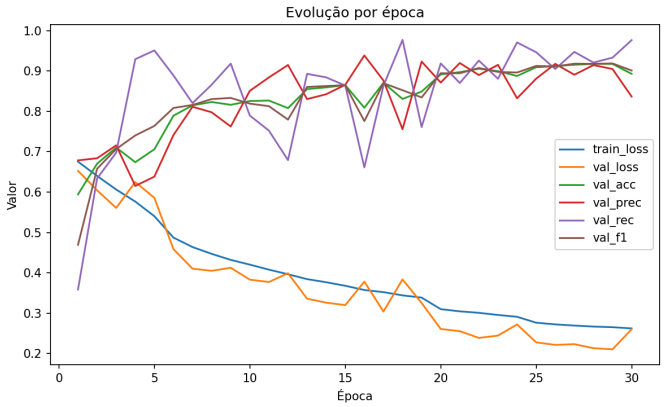
A arquitetura empregada, definida no arquivo model_baseline.py, consiste em uma CNN compacta formada por três blocos convolucionais principais. Cada bloco contém uma convolução 3×3, responsável por detectar padrões como bordas e texturas, seguida de *Batch Normalization* para estabilizar os valores internos da rede, função de ativação *ReLU* e uma camada *MaxPool* 2×2, que reduz pela metade as dimensões espaciais enquanto preserva as informações relevantes. Os blocos utilizam 32, 64 e 128 filtros, permitindo que a rede aprenda padrões progressivamente mais complexos ao longo das camadas. Após a extração de características, aplica-se *Global Average Pooling*, que transforma cada mapa de ativação em um único valor médio, reduzindo substancialmente o número de parâmetros e, consequentemente, o risco de sobreajuste. O classificador final contém um dropout de 0.2, que ajuda a evitar memorização excessiva, seguido por uma camada linear de 128 para 64 neurônios com ativação ReLU, e outra camada linear de 64 para 1 neurônio. A saída produzida é um *logit*, posteriormente convertido em probabilidade via função sigmoid para indicar se a imagem é real ou *deepfake*. Todo esse arranjo foi escolhido para garantir um modelo leve, estável e eficiente, capaz de capturar padrões visuais relevantes sem excesso de complexidade.

2.4 Treinamento do Modelo

O treinamento foi desenvolvido no arquivo main.py utilizando o *PyTorch*. O modelo foi treinado por 30 épocas, com *batches* de 16 imagens. Para atualização dos pesos, foi utilizado o otimizador Adam (*learning rate* inicial de 1×10^{-3} e *weight decay* de 1×10^{-4}), ajudando a evitar que a rede decorasse o conjunto de treino. A função de perda foi a *BCEWithLogitsLoss*, que trabalha diretamente com a saída em logit da rede. Como havia diferença no número de imagens reais e falsas, o parâmetro *pos_weight* foi aplicado para balancear a influência das classes durante o cálculo da perda. A taxa de aprendizado foi ajustada automaticamente pelo *ReduceLROnPlateau*, que reduz o *learning rate* quando a perda de validação deixa de melhorar. Isso torna o ajuste mais fino nas últimas épocas. O conjunto de validação foi usado para acompanhar o desempenho, identificar sinais de sobreajuste e selecionar o melhor modelo salvo (best_model.pth). Ele também serviu para definir o threshold ideal entre real e fake.

Ao final das 30 épocas, a perda de treino e a perda de validação apresentaram queda constante, indicando convergência estável. Essa evolução pode ser observada no gráfico da figura 2.

Figura 2 – Evolução



Fonte: elaborado pelo autor.

2.5 Classificação e Threshold

Após o treinamento, o modelo produz para cada imagem uma probabilidade contínua indicando o quanto ela se aproxima da classe “real”. Para transformar essa probabilidade em uma decisão final, real ou *deepfake*, é necessário definir um valor de corte, o threshold. Thresholds muito baixos aumentam o recall, mas geram mais falsos positivos; *thresholds* altos fazem o oposto, reduzindo falsos positivos, porém elevando o número de falsos negativos. Por isso, a escolha desse valor influencia diretamente o equilíbrio entre precisão e *recall*.

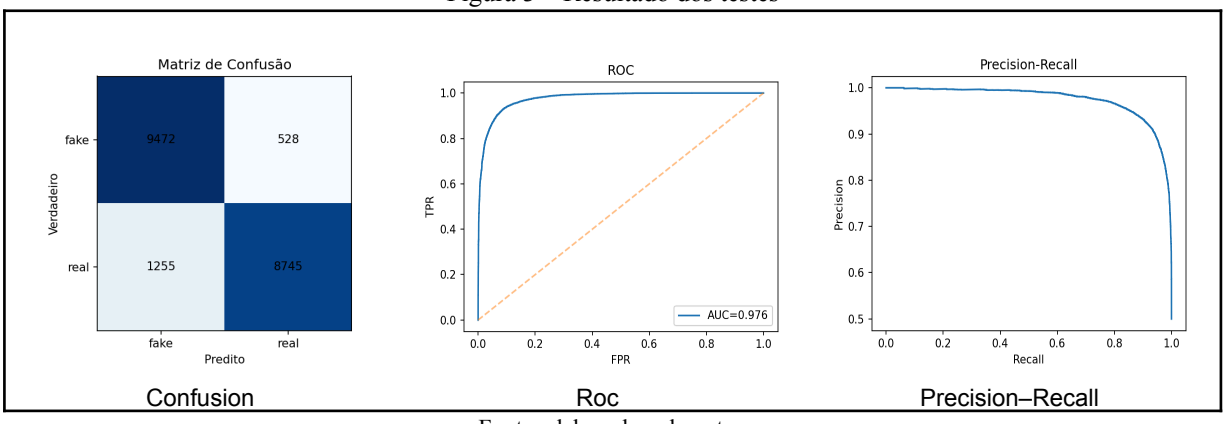
No projeto, foram avaliados diversos thresholds entre 0.30 e 0.70, sempre observando como o *F1-score* variava no conjunto de validação. A métrica F1 foi escolhida por ser mais adequada ao problema, já que combina precisão e *recall* e penaliza situações em que o modelo acerta muito em uma métrica, mas vai mal na outra. Após os testes, o *threshold* de 0.70 apresentou o melhor equilíbrio entre os dois lados: reduziu falsos positivos sem comprometer excessivamente o recall. Além disso, manteve estabilidade entre as classes e evitou que o modelo se tornasse permissivo demais com *deepfakes*. Por esse motivo, o valor 0.70 foi adotado como *threshold* final, sendo utilizado em todas as predições do conjunto de teste e nas avaliações quantitativas apresentadas.

2.6 Testes do Modelo

O conjunto *test* foi mantido completamente isolado das etapas de treinamento e validação, sendo utilizado apenas depois da definição do melhor *checkpoint*. Isso garante que os resultados apresentados reflitam a capacidade real de generalização do modelo, sem influência de exemplos já vistos. Com o *threshold* ajustado para 0.70, o classificador alcançou acurácia de 0.9109, precisão de 0.9431, *recall* de 0.8745 e *F1-score* de 0.9075. Esses números mostram que o modelo conseguiu diferenciar imagens reais e *deepfakes* com um desempenho consistente, preservando um bom equilíbrio entre identificar corretamente a maioria das falsificações e evitar marcações incorretas de imagens genuínas. A matriz de confusão também ajuda a entender esse comportamento: 8745 imagens falsas foram identificadas corretamente, enquanto 9472 imagens reais foram classificadas como autênticas. Mesmo assim, ainda ocorreram alguns erros relevantes, como 528 imagens reais que foram classificadas como falsas e 1255 *deepfakes* que passaram como reais, representando o ponto mais sensível do modelo e indicando onde ainda há espaço para avanço.

Além das métricas básicas, outras visualizações ajudaram a compreender o desempenho geral. A curva ROC apresentou área próxima de 0.97, o que mostra que o modelo separa bem as duas classes ao variar o *threshold*. A curva *Precision Recall* também manteve um comportamento elevado e estável ao longo de praticamente todo o eixo de *recall*, confirmando que o modelo conserva boa precisão mesmo enquanto tenta identificar o maior número possível de *deepfakes*. O gráfico de evolução das perdas ao longo das épocas reforça essa conclusão, já que as curvas de treino e validação diminuíram de maneira constante e sem divergências significativas, o que sinaliza que o modelo não decorou o conjunto de treino e conseguiu aprender padrões úteis. No conjunto, os resultados mostram que a abordagem adotada é funcional, estável e capaz de lidar bem com imagens inéditas, mesmo utilizando uma arquitetura relativamente compacta.

Figura 3 – Resultado dos testes



Fonte: elaborado pelo autor.

2.7 Análise Visual

Além das métricas numéricas, foi realizada também uma análise visual das regiões da imagem que mais influenciaram as decisões do modelo, utilizando o script gradcam.py. Esse script carrega o modelo treinado

(best_model.pth), aplica a técnica *Grad-CAM* sobre a última camada convolucional da CNN e gera um mapa de calor sobreposto à imagem original, evidenciando as áreas com maior contribuição para a classificação.

Foram produzidas visualizações tanto para uma imagem real quanto para uma *deepfake* do conjunto de teste. Na imagem real, observou-se que as ativações se concentram principalmente em regiões estruturadas do rosto, como área dos olhos, boca e contornos, indicando que o modelo utiliza características naturais para confirmar a autenticidade. Já na imagem *fake*, o mapa de calor destaca regiões com artefatos, texturas inconsistentes ou superfícies menos uniformes, sugerindo que a rede aprendeu a identificar sinais sutis de manipulação digital. Esses resultados reforçam a interpretabilidade do modelo e mostram que suas decisões são baseadas em padrões visuais relevantes, e não em ruído aleatório.

Figura 4 – *Heatmaps* Grad-CAM para imagem *real* e imagem *fake*

Real Fake

Fonte: elaborado pelo autor.

3 CONCLUSÕES

O desenvolvimento deste trabalho permitiu construir e avaliar um modelo capaz de distinguir imagens reais de *deepfakes* com desempenho e comportamento consistente ao longo de todas as etapas do processo. Desde a organização da base de dados até a análise visual das decisões internas da rede, cada fase contribuiu para demonstrar que é possível alcançar resultados confiáveis mesmo utilizando uma arquitetura relativamente compacta de CNN.

O treinamento do modelo mostrou boa estabilidade, com queda progressiva da perda de treino e validação, o que indicou que a rede estava aprendendo de forma efetiva sem sinais significativos de sobreajuste. A escolha do threshold, definida a partir do conjunto de validação, também se mostrou apropriada: o valor adotado (0.70) forneceu o melhor equilíbrio entre precisão e recall, evitando que o modelo fosse permissivo demais com *deepfakes* e garantindo um bom nível de acertos para ambas as classes.

Os testes finais foram realizados exclusivamente com imagens que nunca foram vistas pela rede, o modelo apresentou desempenho consistente, alcançando pontuações superiores a 0.90 nas principais métricas de avaliação. A matriz de confusão deixou claro onde o classificador acerta com mais segurança e quais são os casos mais desafiadores. Apesar da existência de falsos negativos, que representam *deepfakes* não identificadas, o desempenho geral ainda se mostrou bastante satisfatório para um protótipo com essa proposta.

A análise visual com *Grad-CAM* foi um complemento importante, pois permitiu verificar que o modelo realmente aprendeu a focar em regiões relevantes das imagens, como áreas do rosto ou regiões onde normalmente aparecem artefatos de manipulação. Essas visualizações dão mais transparência ao processo de decisão da rede e ajudam a entender por que ela acerta ou erra em determinados casos.

De maneira geral, o trabalho atingiu os objetivos propostos: organizar uma base de dados adequada, preparar as imagens corretamente, projetar e treinar uma rede convolucional funcional, ajustar o *threshold* ideal, realizar testes completos e interpretar tanto os resultados numéricos quanto os visuais. O modelo final se mostrou eficaz na detecção de *deepfakes* e apresentou boa capacidade de generalização para dados novos, o que confirma a viabilidade do uso de CNNs para esse tipo de tarefa.

Referências

BHANDARE, Ashwin et al. Applications of Convolutional Neural Networks. International Journal of Computer Science and Information Technologies, [S.l.], v. 7, n. 5, p. 2206-2215, 2016.
https://datascientest.com/en/what-is-the-grad-cam-method

https://poloclub.github.io/cnn-explainer/

https://www.kaggle.com/datasets/aryansingh16/deepfake-dataset