Relatório Técnico: Implementação e Análise de Classificação com Redes Convolucionais e o dataset CUFS

Alisson Santos Ribeiro e Irlan Wallace dos Santos Mattos

01/12/2024

Resumo

Este projeto abordou a classificação de imagens por meio de redes neurais convolucionais (CNNs), um dos métodos mais eficazes para problemas de visão computacional. O foco foi treinar, avaliar e ajustar modelos com diferentes combinações de hiperparâmetros, como número de épocas e tamanho do batch, utilizando métricas como F1-Score e AUC-ROC para avaliar desempenho. O treinamento foi complementado com técnicas modernas como Early Stopping e regularização, e as análises destacaram áreas de melhoria relacionadas ao dataset, arquitetura do modelo e seleção de parâmetros.

Introdução:

A classificação de imagens é um dos principais desafios em visão computacional, é uma tarefa de aprendizado de máquina em que o objetivo é determinar a qual categoria ou classe uma imagem pertence, com aplicações que vão desde categorização de fotos e reconhecimento facial até diagnósticos médicos baseados em imagens. Para resolver esse problema é ideal o uso de um modelo de aprendizado de máquina supervisionado, que são modelos que usam um conjunto de dados previamente rotulados para treinar e ajustar o modelo, entre esses modelos as Redes neurais convolucionais (CNNs) se destacam por sua capacidade de aprender padrões visuais de maneira hierárquica, detectando características como bordas, texturas e formas, uma CNN geralmente é composta por três tipos de camadas: camadas convolucionais, camadas de pooling e camadas totalmente conectadas. Essas camadas trabalham em conjunto para extrair características relevantes da imagem e realizar a classificação, elas exploram a estrutura espacial das imagens, levando em consideração a proximidade entre pixels e características locais, o que as torna muito adequadas para a análise de imagens. Este projeto explorou a capacidade das CNNs de classificar imagens em duas categorias, destacando não apenas o potencial do modelo, mas também as armadilhas comuns, como desbalanceamento de classes e desafios com dados reais.

Metodologia:

A metodologia utilizada no projeto foi dividida em diversas etapas estruturadas, cada uma focada em preparar, treinar e avaliar modelos de redes neurais convolucionais (CNN) para a classificação de imagens. Abaixo, os principais componentes e decisões metodológicas são detalhados:

1. Preparação do Dataset

Carregamento das Imagens:

O projeto começou com os autores carregando e visualizando o banco de dados, buscando entender a quantidade, o tipo de imagens que tinha no dataset e como estavam organizadas, uma função foi feita para buscar os arquivos de imagens na pasta especificada e retorna uma lista com os caminhos completos para as imagens encontradas. As imagens foram obtidas de uma pasta específica (photos), sendo carregadas e processadas como arrays NumPy.

Rotulação:

O processo de rotular as imagens para o treinamento do modelo, surgiu a partir da visualização da array de imagens a partir de um **os.listdir** que visava listar o conteudo do diretorio de "photos" e a partir dai foi perceptivel que as imagens ja estavam rotuladas, o rotulo pré definido começava com uma letra e era seguido por numeros, para descobri se essa letra no inicio do rotulo indicava o sexo biologico da pessoa que estava na imagem foi criada uma funçao que rotuluava as imagens a partir da primeira letra do nome do arquivo. Se o nome do arquivo começasse com 'm', o rótulo seria 1 (Masculino). Caso contrário, seria 0 (Feminino), os rótulos binarizados seriam úteis para o problema elencado, de classificação binária.

I

Depois foi feito o processamento dessas imagens, foram feitas outras duas funções, uma para redimensionar as imagens para o formato de 250 x 200 pixels. Isso garantiu que todas as imagens ficassem com dimensões uniformes, o que era necessário para entradas em modelos de aprendizado de máquina. Todas as imagens foram normalizadas para valores entre 0 e 1, garantindo compatibilidade com a entrada da CNN, já que os valores normalizados aceleram o processo de convergência. Com outra função, processam todas as imagens fornecidas e as retornamos em um array NumPy.

Divisão em Conjuntos:

Com as imagens devidamente rotuladas, o dataset foi repartido da maneira que foi solicitada no documento entre: Teste, Validação e Treino, 50% do banco foi reservado para treino, 30% para validação e os 20% restantes para teste. Essa divisão foi feita através de uma função **(def dividir_dataset)** e era apropriada, pois garantia uma maior robustez no treinamento e na validação do modelo, reduzindo as possibilidades de haver um underfitting.

2. Modelagem

CNN simples:

logo após começou a criação do modelo da CNN,começando com o estabelecimento do tamanho das imagens de entrada que o modelo irá aceitar, o tamanho foi definido para o citado anteriormente **(250 x 200 pixels)**. Foi utilizado o **modelo Sequential do Keras**, rodando através do **TensorFlow**. Este modelo é o mais apropriado quando temos uma pilha de camadas simples, com apenas um tensor de entrada e um de saída. inicialmente, usaram 3 camadas convolucionais com função de ativação ReLu A, com filtros de tamanhos variados e em cada camada convolucional, tendo uma camada max Pooling para reduzir a dimensionalidade. Para evitar o overfitting, colocando uma camada dropout A para regularização, após algumas camadas. Finalizaram com camadas densas para mapear as extrações de características aos rótulos binários e uma camada com ativação sigmoidal, para prever uma única probabilidade de classe, que será a saída. No código, a função de custo especificada é a **binary_crossentropy**, que é ideal para problemas de classificação binária (como no caso de diferenciar entre "masculino" e "feminino"). Ela mede a diferença entre as distribuições probabilísticas das saídas previstas e dos valores

reais. Como otimizador, utilizamos o **Adam (Adaptive Moment Estimation)**, um dos algoritmos de otimização mais populares em aprendizado de máquina, especialmente no treinamento de redes neurais. Ele é utilizado para ajustar os pesos da rede com base na função de custo, visando minimizar o erro do modelo de forma eficiente.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 198, 248, 32)	896
max_pooling2d (MaxPooling2D)	(None, 99, 124, 32)	0
conv2d_1 (Conv2D)	(None, 97, 122, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 48, 61, 64)	0
conv2d_2 (Conv2D)	(None, 46, 59, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 23, 29, 128)	0
conv2d_3 (Conv2D)	(None, 7, 9, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 3, 4, 256)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 128)	393,344
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 1)	65

Modelo Aprimorado:

Após os resultados serem iterados sobre uma array, que armazena os principais dados de cada treino por conjunto de hiperparâmetros, analisamos que o modelo precisava de algumas aproximações. As modificações incluíram: Normalização em batch **(BatchNormalization)** após cada camada convolucional. Uso de **LeakyReLU** para melhor propagação de gradientes. **Dropout** adicional para combater o overfitting. Maior número de filtros convolucionais e aumento progressivo da profundidade da rede.

Layer (type)	Output Shape	Param #
conv2d_40 (Conv2D)	(None, 198, 248, 32)	896
max_pooling2d_40 (MaxPooling2D)	(None, 99, 124, 32)	0
batch_normalization (BatchNormalization)	(None, 99, 124, 32)	128
conv2d_41 (Conv2D)	(None, 97, 122, 64)	18,496
max_pooling2d_41 (MaxPooling2D)	(None, 48, 61, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 48, 61, 64)	256
conv2d_42 (Conv2D)	(None, 46, 59, 128)	73,856
max_pooling2d_42 (MaxPooling2D)	(None, 23, 29, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 23, 29, 128)	512
conv2d_43 (Conv2D)	(None, 21, 27, 256)	295,168
max_pooling2d_43 (MaxPooling2D)	(None, 10, 13, 256)	0
dropout_10 (Dropout)	(None, 10, 13, 256)	0
flatten_10 (Flatten)	(None, 33280)	0
dense_30 (Dense)	(None, 128)	4,259,968
leaky_re_lu (LeakyReLU)	(None, 128)	0
dropout_11 (Dropout)	(None, 128)	0

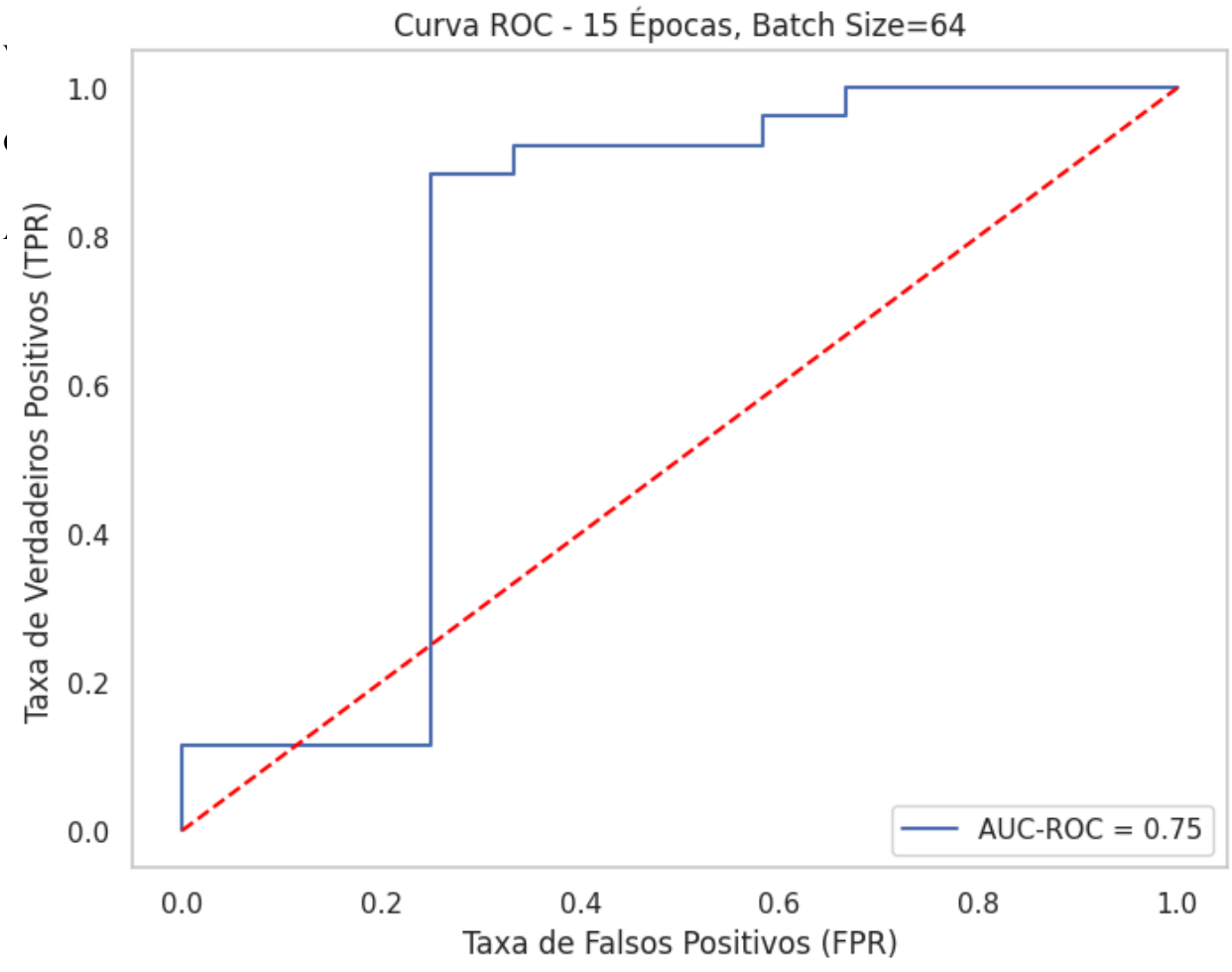
Treinamento

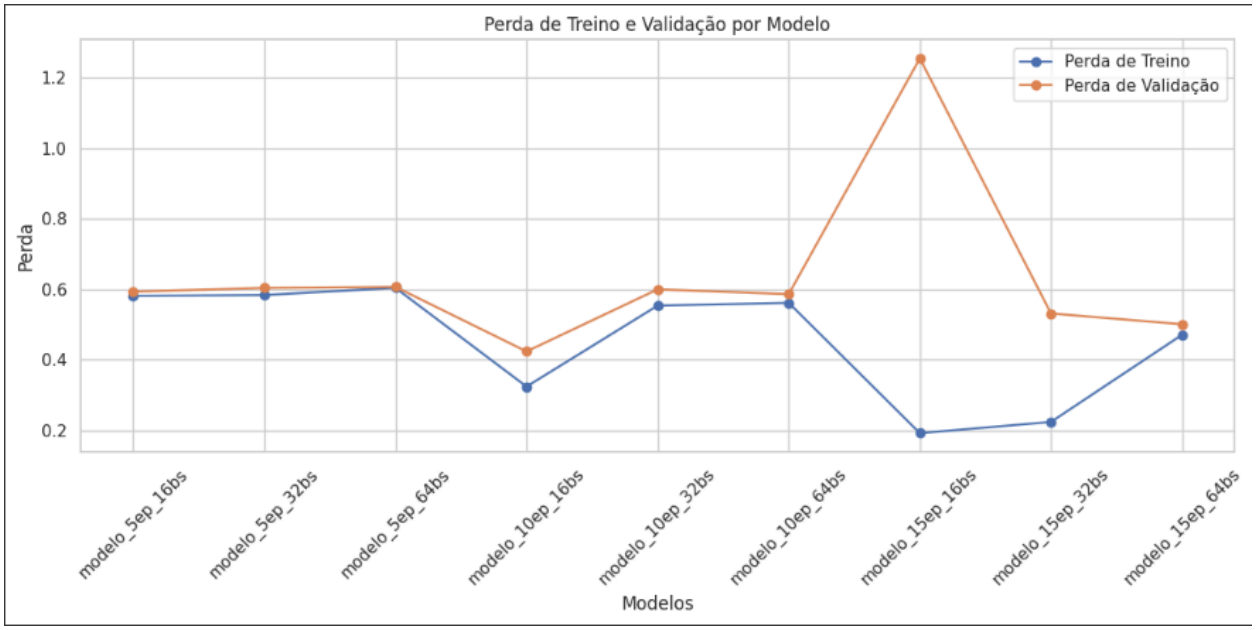
O treinamento do modelo foi feito Estabelecendo uma lista de **hiperparâmetros múltiplos** (Èpocas e Batch Size) a serem testados e criando um modelo a cada teste executado, para mais a frente poderem avaliar as métricas de cada um e estabelecer qual é o melhor número de hiperparâmetros a ser utilizado. No modelo aprimorado, usamos **Validação cruzad**a para monitorar desempenho em diferentes épocas. Implementação de **Early Stopping** para evitar treinamento desnecessário em caso de estagnação na validação. Diferentes tamanhos de batch (16, 32, 64) testados para eficiência e desempenho.

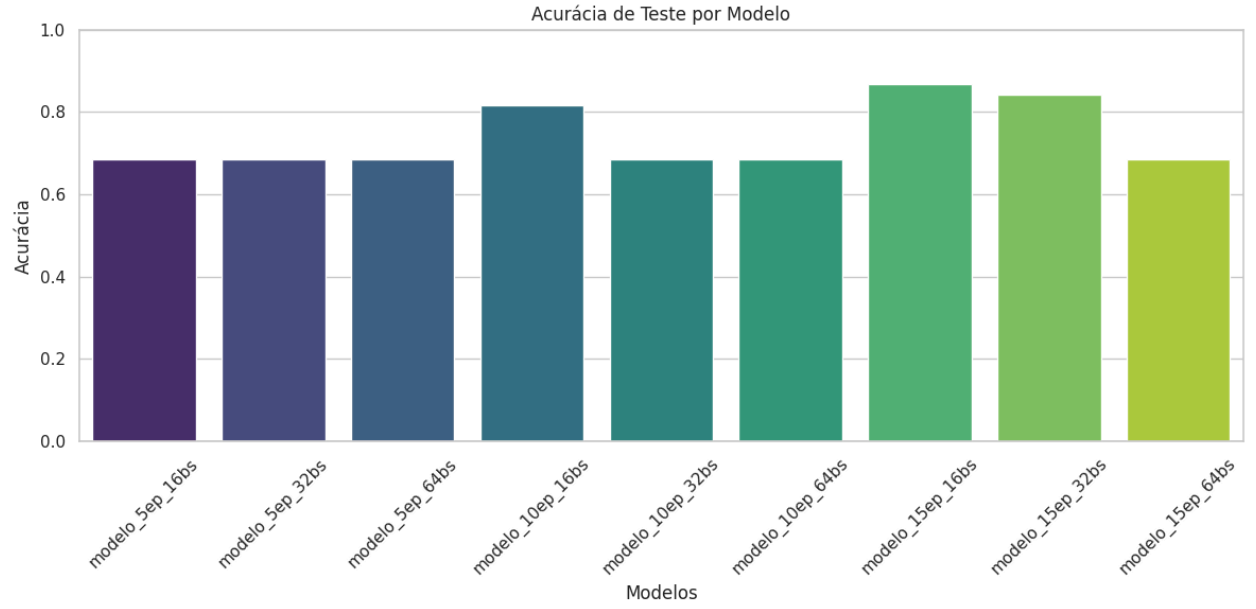
```
Treinando modelo_10ep_32bs...
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 16s 4s/step - accuracy: 0.5848 - loss: 0.6670 - val_accuracy: 0.6964 - val_loss: 0.6172
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.7576 - loss: 0.5929 - val_accuracy: 0.6964 - val_loss: 0.6110
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 13s 4s/step - accuracy: 0.7236 - loss: 0.6018 - val_accuracy: 0.6964 - val_loss: 0.6084
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 22s 6s/step - accuracy: 0.7576 - loss: 0.5739 - val_accuracy: 0.6964 - val_loss: 0.6230
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 19s 5s/step - accuracy: 0.7420 - loss: 0.5620 - val_accuracy: 0.6964 - val_loss: 0.6061
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 18s 4s/step - accuracy: 0.7537 - loss: 0.5516 - val_accuracy: 0.6964 - val_loss: 0.6044
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 23s 5s/step - accuracy: 0.7498 - loss: 0.5684 - val_accuracy: 0.6964 - val_loss: 0.6052
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 13s 5s/step - accuracy: 0.7420 - loss: 0.5776 - val_accuracy: 0.6964 - val_loss: 0.6018
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 15s 6s/step - accuracy: 0.7108 - loss: 0.6194 - val_accuracy: 0.6964 - val_loss: 0.6018
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 18s 5s/step - accuracy: 0.7342 - loss: 0.5608 - val_accuracy: 0.6964 - val_loss: 0.5999
Perda: 0.6130973696708679
Acurácia: 0.6842105388641357
Modelo modelo_10ep_32bs salvo com sucesso!
```

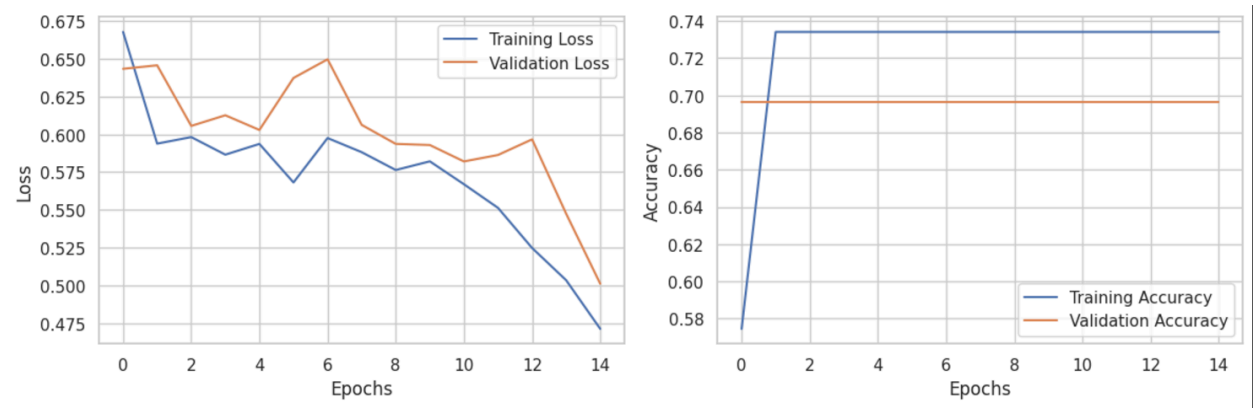
Métricas:

F1-score para avaliar equilíbrio entre precisão e recall. Curva AUC-ROC para medir a separabilidade das classes. A Perda (loss) em treino e validação para verificar o overfitting.




Perda de Treino e Validação por Modelo
Perda de Treino
Perda de Validação
Perda
1.2
1.0
0.8
0.6
0.4
0.2
modelo_5ep_16bs
modelo_5ep_32bs
modelo_5ep_64bs
modelo_10ep_16bs
modelo_10ep_32bs
modelo_10ep_64bs
modelo_15ep_16bs
modelo_15ep_32bs
modelo_15ep_64bs
Modelos


Acurácia de Teste por Modelo
Acurácia
1.0
0.8
0.6
0.4
0.2
0.0
modelo_5ep_16bs
modelo_5ep_32bs
modelo_5ep_64bs
modelo_10ep_16bs
modelo_10ep_32bs
modelo_10ep_64bs
modelo_15ep_16bs
modelo_15ep_32bs
modelo_15ep_64bs
Modelos


Training Loss
Validation Loss
Loss
0.675
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
0
2
4
6
8
10
12
14
Epochs
Training Accuracy
Validation Accuracy
Accuracy
0.74
0.72
0.70
0.68
0.66
0.64
0.62
0.60
0.58
0
2
4
6
8
10
12
14
Epochs

5. Considerações Finais

Ajustes como aumento do dataset com data augmentation e o uso de redes pré-treinadas são mencionados como próximos passos para melhorias no modelo**.**

Discussão:

Após análises dos resultados, o grupo percebeu que a pontuação geral do F1-Score mostrou um equilíbrio aceitável entre as classes, mas houve pequenas variações conforme os hiperparâmetros. Modelos com batch sizes menores e mais épocas tendem a performar melhor, sugerindo que a rede aproveitou melhor os dados ao treinar mais lentamente. Certos tipos de imagens foram frequentemente classificadas incorretamente, como aquelas com baixa iluminação ou ângulos incomuns, foi notado que imagens com muito ruído ou fundos complexos confundiram a rede, indicando que o modelo tinha dificuldade em ignorar informações irrelevantes, o uso de rótulos baseados no nome do arquivo pode ter levado a classes desproporcionais.

O dataset apresentou indícios de desbalanceamento entre as classes, o que pode ter levado o modelo a priorizar a classe majoritária, a qualidade das imagens também variou consideravelmente, e algumas estavam borradas ou pixeladas, dificultando o aprendizado da rede. Após uma discussão o grupo percebeu que se fosse adicionar mais camadas convolucionais ou aumenta-se a largura (número de filtros) poderia permitir à rede aprender padrões mais complexos. Assim como a rotação, espelhamento e ajustes de brilho poderiam ajudar o modelo a generalizar melhor, e que experimentos com otimizadores como RMSprop ou SGD poderiam fornecer insights sobre melhores taxas de aprendizado.

A dupla se surpreendeu quando notou que algumas imagens aparentemente fáceis foram classificadas erroneamente, sugerindo que o modelo aprendeu padrões inadequados durante o treinamento, que o uso de Batch Normalization e Dropout no modelo aprimorado melhorou significativamente a generalização, reduzindo overfitting, e que Batch sizes menores, apesar de consumirem mais tempo, frequentemente levaram a melhores resultados, reforçando a importância de hiperparâmetros bem ajustados.

Conclusão e Trabalhos Futuros:

O projeto demonstrou a viabilidade de utilizar CNNs para resolver problemas de classificação binária de imagens, destacando a importância de um pipeline bem definido, incluindo pré-processamento e validação. Mostrando que Redes convolucionais são incrivelmente poderosas, mas sua eficácia depende diretamente da qualidade e diversidade dos dados. Ficou visível que métricas como F1-Score e AUC-ROC são cruciais para entender o desempenho além da acurácia bruta. Como sugestões para trabalhos futuros, o grupo sugere aumentar o tamanho e a diversidade do dataset para melhorar a generalização, automatizar a busca de parâmetros usando métodos como Grid Search ou Bayesian Optimization e utilizar modelos pré-treinados como VGG16 ou ResNet poderia acelerar o treinamento e melhorar os resultados.

Referências:

https://www.kaggle.com/datasets/arbazkhan971/cuhk-face-sketch-database-cufs

https://lamfo-unb.github.io/2020/10/26/CNN/

https://pt.w3d.community/paulogio/introducao-as-redes-neurais-convolucionais-cnns-para-classificacao-de-imagens-3jah

https://pt.wikipedia.org/wiki/Entropia_cruzada