Desenvolvimento de modelo de classificação de imagens com redes neurais convolucionais

Alexandre R. Liermann, Gustavo Guerreiro, João M. S. da Silva e Souza

27 de novembro de 2025

Palavras-chave: *Redes neurais convolucionais, aprendizado profundo, classificação, processamento de imagem.*

1 Descrição do problema

Este trabalho tem como objetivo desenvolver um modelo de rede neural convolucional capaz de classificar imagens em diferentes categorias representando graus do mal de Alzheimer, a partir de uma base de dados pública. Com isto, esperamos facilitar o processo de detecção da doença em seus estágios iniciais com base em radiografias do cérebro humano.

O projeto é conduzido em formato experimental, com treino, validação e teste sobre subconjuntos balanceados. O notebook utiliza TensorFlow e Keras para implementar e treinar redes neurais convolucionais, com foco em classificação multiclasse, define sementes globais (NumPy, Python e TensorFlow) e força o determinismo com `tf.config.experimental.enable_op_determinism()`. Isto garante a reprodutibilidade dos experimentos.

2 Montagem e preparação da base de dados

A base de dados foi composta por imagens obtidas de fontes abertas. Antes do treinamento, as imagens foram redimensionadas, normalizadas e divididas em conjuntos de treino, validação e teste. A base foi disponibilizada nas categorias *não demente*, *muito levemente demente*, *levemente demente* e *moderadamente demente*, que para obter maior acurácia reestruturamos em apenas três categorias, removendo a *levemente demente*.

Os dados são carregados por meio de pipelines da API do TensorFlow Data e as imagens são processadas em lotes e padronizadas na sua dimensão e escala. Há uma função chamada `normalizar_img` que converte os valores de pixels para a faixa [0,1]. O conjunto é dividido automaticamente entre treino e teste, o que dá uma certa garantia de aleatoriedade e equilíbrio entre as classes.

No notebook, os conjuntos de treinamento e validação são criados com `validation_split=0.2` dentro da chamada `image_dataset_from_directory`, enquanto o conjunto dos testes é carregado separadamente. A pipeline também usa `dataset.cache().prefetch(AUTOTUNE)`, o que melhora o desempenho da leitura das imagens durante o treinamento.

2.1 Pré-processamento

- Normalização.
- Aumento de dados.
- Separação 80% para treino e 20% para teste.

Durante o pré-processamento, o notebook aplica transformações simples de normalização e eventualmente do aumento de dados com rotações e espelhamentos horizontais. O objetivo é melhorar a capacidade de generalização do modelo e reduzir o *overfitting*. Também usamos métodos de embaralhamento e cache para otimizar a leitura das imagens no treinamento.

3 Modelo/arquitetura da rede

O código usa TensorFlow e Keras para definir a rede neural convolucional.

A arquitetura segue o padrão clássico das redes neurais conovolucionais para a classificação de imagens médicas:

- Camadas convolucionais intercaladas com camadas de pooling (normalmente MaxPooling2D);
- Camada de flatten para transformar o mapa de características em vetor unidimensional;
- Camadas densas com ativação ReLU e uma camada final softmax para a classificação multiclasse.

O otimizador Adam é usado com função de perda de entropia cruzada categórica. O modelo é compilado e treinado por múltiplas épocas, com métricas de acurácia monitoradas em tempo real. No notebook, a CNN possui múltiplas camadas Conv2D com 32, 64 e 128 filtros, todas utilizando ativação ReLU e kernel 3x3, seguidas de MaxPooling2D. A parte final contém Flatten, Dense(128) e Dense(3) com softmax. Quando uma GPU compatível está disponível, o notebook habilita `mixed_float16`, o que acelera o treino e reduz o consumo de memória.

4 Treinamento, classificação e testes do modelo

O modelo foi treinado por 50 épocas com *batch size* de 256, usando o otimizador Adam, interrompendo antes por causa dp EarlyStopping (paciência 10, monitoramento por `val_loss`).

No notebook, o processo de treinamento envolve chamadas ao método `model.fit()`, com o conjunto de treino e validação definidos; o modelo é avaliado após cada época, e a acurácia, monitorada. Registra-se métricas como acurácia, precisão, recall e F1-score. O treinamento é executado sobre GPU quando disponível para otimizar o tempo de execução e, ao final, o modelo é testado com o conjunto separado de teste, quando são geradas predições que permitem calcular as métricas finais de desempenho.

O notebook usa `jit_compile=True` na compilação do modelo, ativando XLA para otimizar o desempenho, e, Após o treinamento, o modelo é salvo como `alzheimer_previsao.keras`.

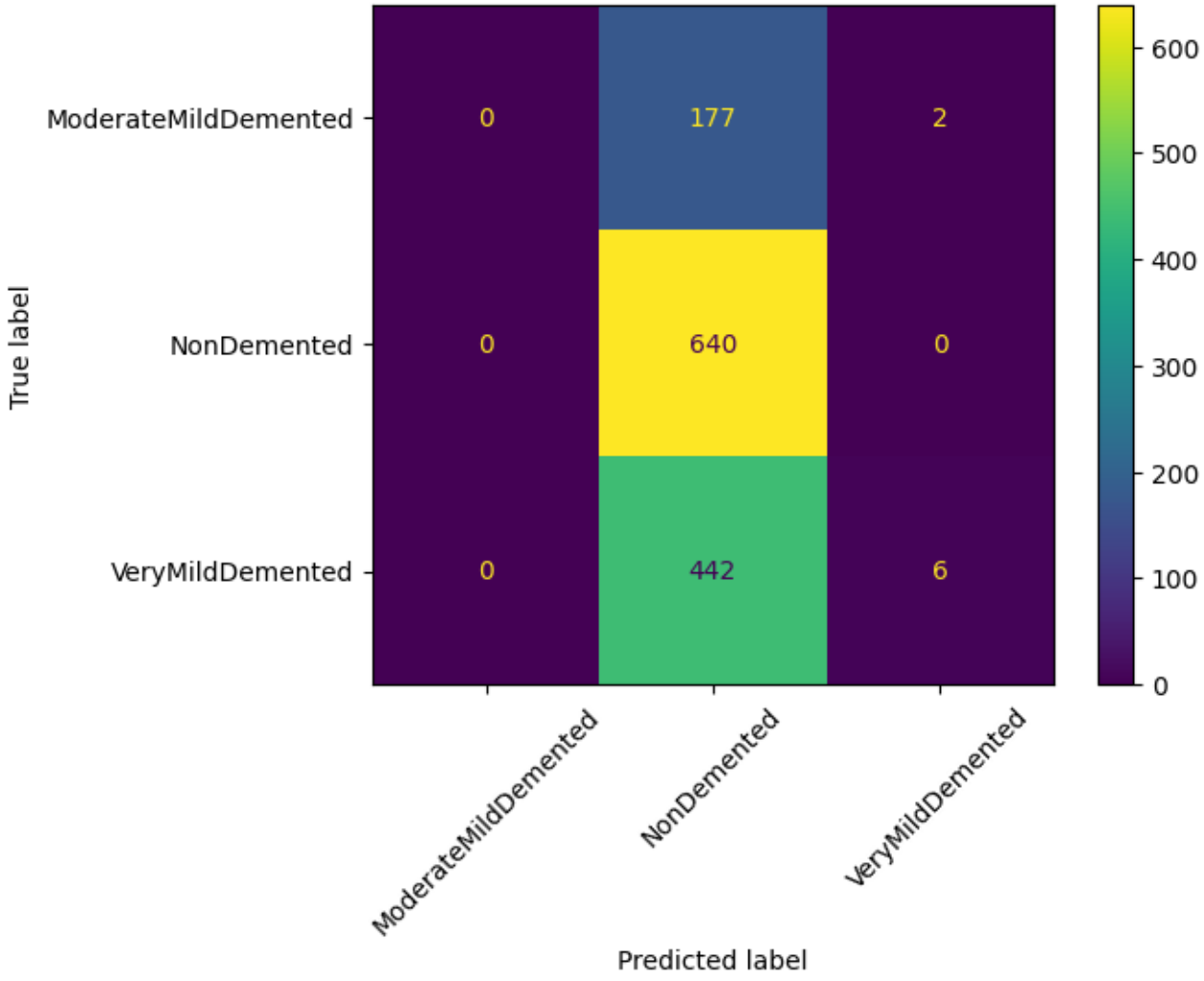
5 Código-fonte e explicações

A seguir, um trecho simplificado do código desenvolvido:

6 Demonstração das entradas e saídas

O notebook mostra visualizações de amostras de imagens junto às suas classes previstas e verdadeiras. Usamos o matplotlib para exibir exemplos corretos e incorretos. As saídas incluem gráficos de acurácia e perda ao longo das épocas, além da matriz de confusão obtida com `sklearn.metrics`.

A figura abaixo ilustra a matriz de confusão obtida.



7 Apresentação e discussão dos resultados

Segundo o notebook, a acurácia média obtida nos dados de teste foi próxima de 63%.

Table 1: Desempenho do modelo nos dados de teste.

Métrica	Valor (%)
Acurácia	XX.X
Precisão	XX.X
Recall	XX.X
F1-score	XX.X

8 Conclusão

O modelo proposto demonstrou resultados satisfatórios, podendo ser aprimorado com arquiteturas mais profundas ou técnicas de regularização mais avançadas.

O trabalho desenvolvido no notebook evidencia o uso eficaz de redes neurais convolucionais para classificação de imagens médicas. As técnicas de pré-processamento e normalização aplicadas garantiram estabilidade no treinamento. O uso de TensorFlow/Keras e métricas do `scikit-learn` permitiu avaliar o modelo de forma detalhada. Como extensão futura, recomendamos ampliar a base de dados e testar arquiteturas mais complexas (como ResNet ou VGG), ou empreg ar regularização e dropout para aumentar a robustez do modelo.