



Universidade Federal do Rio de Janeiro
Instituto de Computação

Detecção de Pneumonia utilizando PCA e Regressão Logística

Arthur de Melo Barbosa
Professor: João Antonio Recio Paixão

Rio de Janeiro, 19 de junho de 2025

Sumário

1	Introdução	3
2	Sobre o Dataset	3
3	Bibliotecas Usadas	4
4	Pré-processamento	4
4.1	Implementação	5
5	Normalização	5
5.1	Implementação	5
6	PCA	6
6.1	Análise de Componentes Principais	7
6.2	Implementação	8
7	Regressão Logística	8
7.1	Implementação	9
8	Avaliação	9
8.1	Implementação	9
8.2	Resultado	10
9	Validação	10
9.1	Implementação	11
9.2	Saídas	11
10	Referências	13
11	Links	13

1 Introdução

Neste projeto final, optei por desenvolver um modelo em uma área de grande interesse pessoal: a computação aplicada à medicina. Após conversar com o professor João Paixão em sala de aula, decidi implementar um modelo de regressão logística para a detecção de doenças por meio de imagens de raio-X. Este relatório apresenta um projeto de classificação voltado à identificação de pneumonia, utilizando técnicas de redução de dimensionalidade (PCA) e modelagem com Regressão Logística. O objetivo principal foi construir um modelo eficiente para classificar imagens de raio-X do tórax de pacientes como "normal" ou "pneumonia".



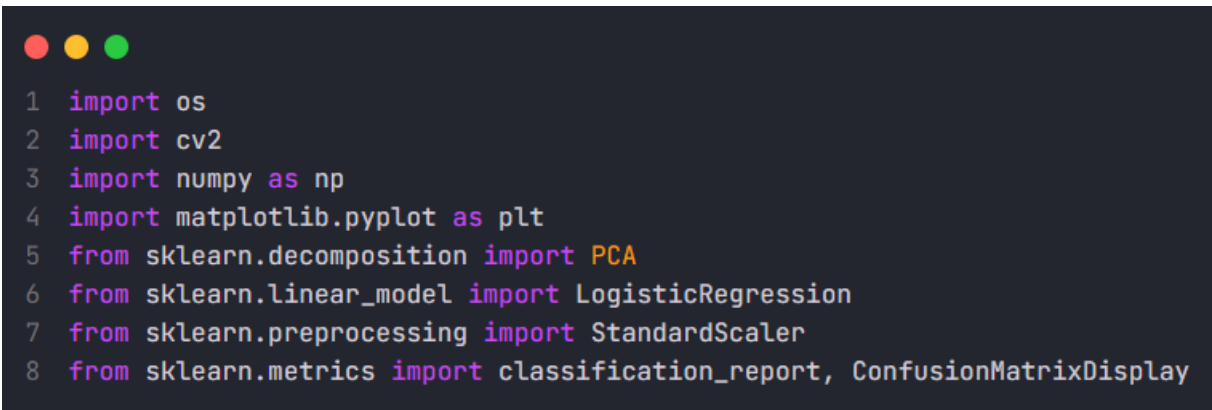
Figura 1: Identificação de Pneumonia

O modelo de identificação automática segue um padrão característico: ele tende a reconhecer pneumonia bacteriana, que geralmente aparece como uma mancha concentrada em uma região específica do pulmão (indicado pelas setas brancas). Já a pneumonia viral (à direita) costuma se espalhar de forma mais difusa pelos dois pulmões, formando um padrão mais esbranquiçado e nebuloso.

2 Sobre o Dataset

Após definir o tema do projeto, busquei um conjunto de dados adequado e encontrei, no site Kaggle, especializado em ciência de dados, um dataset com imagens de raio-X de indivíduos com e sem pneumonia. Esse conjunto contém 5.863 imagens do tórax, com boa diversidade de casos. As imagens estão organizadas em pastas para treino, teste e validação, e cada uma dessas pastas possui subpastas para as categorias "normal" e "pneumonia". Essa estrutura foi fundamental para o treinamento adequado do modelo de regressão logística. O link para o dataset é: <https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia/data>.

3 Bibliotecas Usadas



```
1 import os
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from sklearn.decomposition import PCA
6 from sklearn.linear_model import LogisticRegression
7 from sklearn.preprocessing import StandardScaler
8 from sklearn.metrics import classification_report, ConfusionMatrixDisplay
```

Figura 2: Bibliotecas utilizadas no projeto

4 Pré-processamento

A etapa de pré-processamento é responsável pelo carregamento e tratamento das imagens. Inicialmente, criei a função `carregar_imagens()`, que percorre o dataset, carregando e rotulando automaticamente as imagens de acordo com as categorias "normal" e "pneumonia". Como as imagens já estão separadas por pastas, o processo consiste em identificar os diretórios e aplicar os rótulos adequadamente.

Em seguida, as imagens são convertidas para escala de cinza, o que reduz a complexidade e uniformiza o padrão, mesmo que o raio-X já tenha um padrão monocromático. Também é necessário redimensionar as imagens, neste projeto, para 150x150 pixels, com o objetivo de preservar informação suficiente sem comprometer o desempenho do modelo.

A técnica de "flattening" (achatamento) transforma cada imagem 2D em um vetor 1D com todos os seus pixels. Esse vetor será utilizado como entrada para o PCA e, posteriormente, para a Regressão Logística. Ao final, são gerados os vetores **X** e **y**, representando os dados de entrada e seus respectivos rótulos.

4.1 Implementação

```
1 # Pré-processamento
2 pastas = ['NORMAL', 'PNEUMONIA']
3 tamanho_imagem = 150
4 def carregar_imagens(dataset):
5     X = []
6     y = []
7     for pasta in pastas:
8         caminho_pasta = os.path.join(dataset, pasta)
9         indice = pastas.index(pasta)
10
11         if not os.path.isdir(caminho_pasta):
12             print(f"Pasta {caminho_pasta} não encontrada.")
13             continue
14
15         for img in os.listdir(caminho_pasta):
16             if not img.lower().endswith(('.png', '.jpg', '.jpeg')):
17                 continue
18
19             caminho_imagem = os.path.join(caminho_pasta, img)
20             imagem = cv2.imread(caminho_imagem, cv2.IMREAD_GRAYSCALE)
21             if imagem is None:
22                 print(f"Falha ao carregar {caminho_imagem}")
23                 continue
24
25             imagem_redimensionada = cv2.resize(imagem, (tamanho_imagem, tamanho_imagem))
26             X.append(imagem_redimensionada.flatten())
27             y.append(indice)
28
29     return np.array(X), np.array(y)
30
31 X_treino, y_treino = carregar_imagens(treino_dataset)
32 X_teste, y_teste = carregar_imagens(teste_dataset)
```

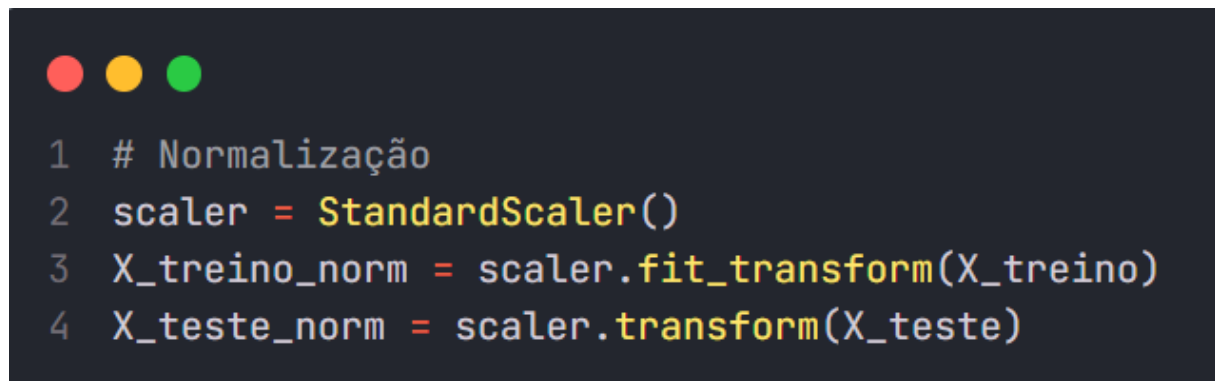
Figura 3: Código de pré-processamento das imagens do dataset

5 Normalização

Antes da aplicação do PCA, é fundamental normalizar os dados, isto é, colocar todas as variáveis (nesse caso, os pixels) na mesma escala. Isso é importante porque o PCA trabalha com variâncias, e sem a normalização, ele pode acabar dando mais peso a pixels com valores maiores apenas por estarem em uma escala diferente.

5.1 Implementação

No código, utilizamos a classe `StandardScaler()` da biblioteca `sklearn`, que transforma os dados para que cada variável tenha média 0 e desvio padrão 1. Isso assegura que todas as variáveis tenham o mesmo peso na análise de componentes principais. isto é, todas as informações terão a mesma importância na análise do PCA.



```
1 # Normalização
2 scaler = StandardScaler()
3 X_treino_norm = scaler.fit_transform(X_treino)
4 X_teste_norm = scaler.transform(X_teste)
```

Figura 4: Código de normalização

As funções `scaler.fit_transform(X_treino)` e `scaler.transform(X_teste)` são utilizadas, respectivamente, para normalizar os dados de treino e aplicar a mesma transformação nos dados de teste, garantindo consistência entre as duas bases.

6 PCA

A Análise de Componentes Principais (PCA) é uma técnica de redução de dimensionalidade. Isso é, o PCA nos ajuda a reduzir o número de características de um conjunto de dados enquanto mantém o maior número de informação possível. Em termos simples, ele permite transformar um conjunto grande de variáveis (ou "informações") em um conjunto menor, mantendo o que é mais importante. Isso é especialmente útil quando trabalhamos com imagens, que possuem milhares de pixels, ou seja, muitas dimensões.

Ao aplicar o PCA no conjunto de dados de imagens, conseguimos reduzir drasticamente a quantidade de informação a ser processada pelo modelo, tornando o treinamento mais rápido e menos sujeito a erros causados por "ruídos" (informações irrelevantes). O parâmetro mais importante no PCA é o percentual de variância que queremos manter, ou seja, o quanto da informação dos dados originais queremos preservar.

6.1 Análise de Componentes Principais

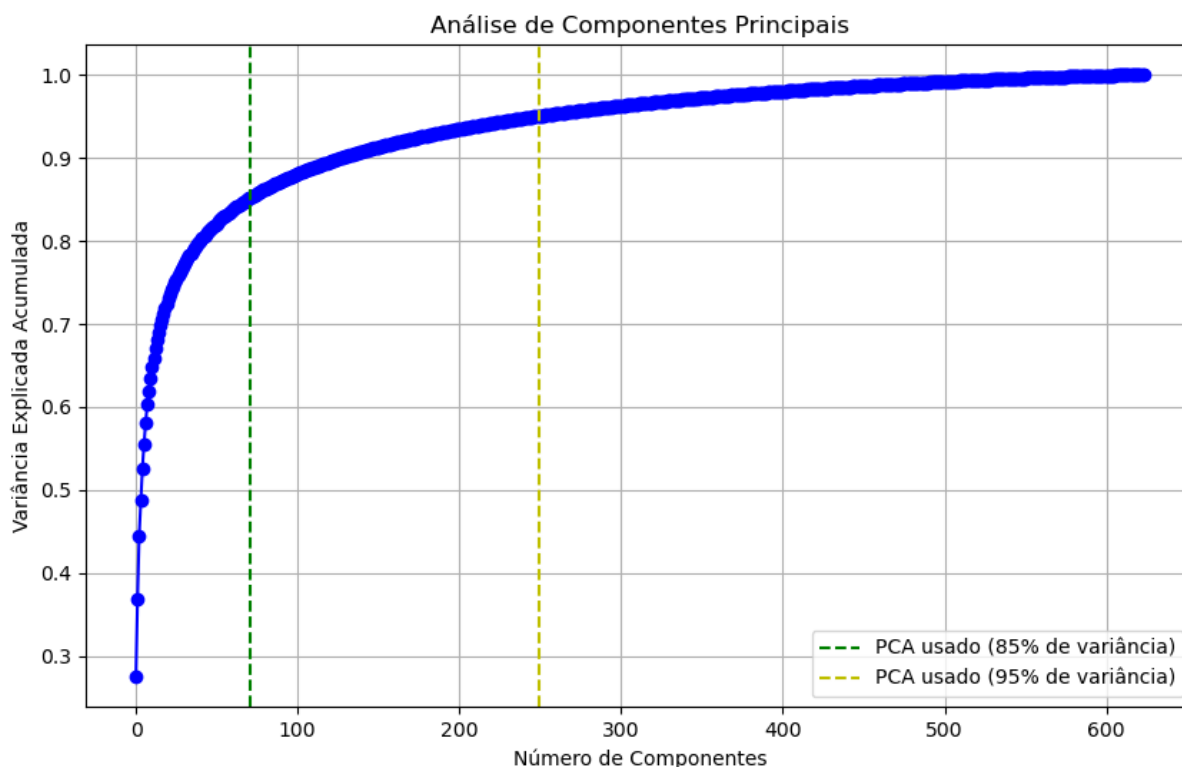


Figura 5: Gráfico da variância acumulada

A partir do gráfico de variância acumulada, observa-se que os componentes acima do número 249 (equivalente a 95% da variância explicada) acrescentam pouca informação nova. O acréscimo de mais componentes, inclusive até acima de 600, não melhora o modelo, pelo contrário, acaba introduzindo ruído e prejudicando a generalização nos dados de validação. Isso evidencia que incluir muitos componentes não é sempre vantajoso.

Com 95% de variância, por exemplo, a avaliação nos dados de teste foi boa, mas o desempenho nos dados de validação caiu, indicando que o modelo estava muito adaptado aos dados de treino (overfitting). Neste projeto, foram testados diferentes valores, e o melhor resultado foi alcançado com 85% da variância, o que corresponde a manter **71 componentes principais**. Essa escolha gerou um bom balanço entre desempenho e simplicidade.

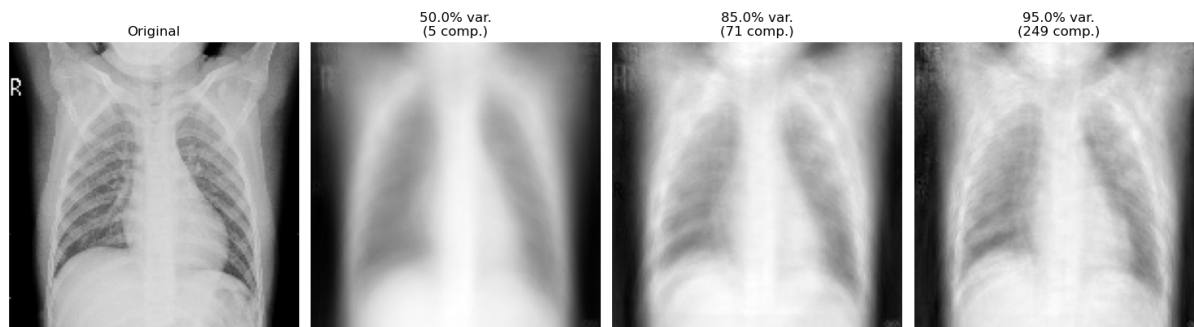


Figura 6: Reconstrução de imagens com diferentes componentes do PCA

6.2 Implementação

```

1 # PCA
2 pca = PCA(n_components=0.85) # 85% de variância
3 X_treino_pca = pca.fit_transform(X_treino_norm)
4 X_teste_pca = pca.transform(X_teste_norm)
5
6 print(f"Número de componentes selecionados: {pca.n_components_}")

```

Figura 7: Código para aplicação do PCA

No código, utilizamos a função `PCA()` da biblioteca `sklearn`, com o parâmetro `0.85` indicando que desejamos manter 85% da variância original. Internamente, o PCA calcula quais combinações de variáveis (ou pixels) carregam mais informação e projeta os dados em um novo espaço com menos dimensões, preservando essa estrutura. A função `fit_transform()` é aplicada aos dados de treino e serve para calcular essas combinações principais (os chamados componentes principais) e ao mesmo tempo transformar os dados, enquanto `transform()` aplica a mesma projeção aos dados de teste, garantindo que os dados de teste sejam projetados no mesmo espaço que os de treino.

7 Regressão Logística

A Regressão Logística é um modelo matemático utilizado para problemas de classificação. Dado um conjunto de entradas e categorias, ela estima a probabilidade de uma nova entrada pertencer a uma das classes, ou seja, a qual classe ela possui mais similaridade.

Neste projeto, as imagens de raio-X são classificadas entre "normal" e "pneumonia". O modelo calcula a probabilidade de uma imagem indicar pneumonia. Caso essa probabilidade seja superior a 50%, a imagem é classificada como "pneumonia".

Esse modelo é adequado por se tratar de um problema binário e por sua eficiência com dados previamente processados, como os resultantes do PCA. O modelo foi treinado usando até 5000 iterações para garantir que o modelo convergisse corretamente.

7.1 Implementação

```
1 # Regressão logística
2 model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=5000)
3 model.fit(X_treino_pca, y_treino)
```

Figura 8: Código da Regressão Logística

Utilizou-se a função `LogisticRegression()` com o solucionador `lbfgs`, eficiente para grandes volumes de dados. O parâmetro `max_iter=5000` define o número máximo de iterações que o modelo pode fazer para ajustar os pesos internos, quanto maior esse número, maior a chance do modelo encontrar uma solução estável

Depois de criado, o modelo é treinado com `model.fit(X_treino_pca, y_treino)`, o que significa que ele aprenderá padrões nos dados reduzidos pelo PCA para tentar prever corretamente se uma imagem indica pneumonia ou não.

8 Avaliação

8.1 Implementação

Após o treinamento, o modelo realiza previsões sobre as imagens do conjunto de teste usando `y_pred = model.predict(X_teste_pca)`. Isso significa que o modelo tenta adivinhar, com base no que aprendeu, se cada imagem representa um caso "normal" ou de "pneumonia".

```
1 # Avaliação
2 y_pred = model.predict(X_teste_pca)
3 print(classification_report(y_teste, y_pred))
```

Figura 9: Código de avaliação do modelo

Para avaliação, utilizamos `classification_report()`, que compara as previsões com os rótulos reais e apresenta métricas como precisão, recall e F1-score.

8.2 Resultado

	precision	recall	f1-score	support
0	0.70	0.85	0.77	1341
1	0.95	0.87	0.91	3875
accuracy			0.87	5216
macro avg	0.82	0.86	0.84	5216
weighted avg	0.88	0.87	0.87	5216

Figura 10: Resultados da Regressão Logística

Os resultados mostram alta precisão para pneumonia (95%), mas menor precisão para casos normais (70%). Isso significa que em 30% dos casos normais, o modelo classificou erroneamente como pneumonia, isso em um cenário médico real seria um erro crítico.

O **recall** indica a proporção de casos positivos corretamente identificados, ele calcula esse resultado pegando os casos de Verdadeiros Positivos e dividindo pelos casos de Falsos Positivos + Verdadeiros Positivos. O **F1-score** é a média harmônica entre precisão e recall. O **support** é a quantidade de dados que cada uma das categorias possuía, nele esta evidente o desbalanceamento dos dados, os quais possuem muitas mais imagens de casos de pneumonia, prejudicando nosso modelo. Essa coluna evidencia esse problema que ocasionou na grande porcentagem de erro nos casos em que a pessoa possuía a doença mas não foi detectada.

Apesar da acurácia geral de 87% ser muito boa, o modelo ainda apresenta limitações. Em contextos médicos, precisão e recall são métricas mais importantes que a acurácia.

Além dessas métricas, o resultado possui as médias macro avg, média simples das métricas, que ignora o desbalanceamento, e a weighted avg, que é a média ponderada pelo support. Ambas saíram com bons resultados.

9 Validação

Após o treinamento do modelo com o conjunto de treino e avaliar seu desempenho com os dados de teste, é necessário verificar se ele realmente aprendeu e não apenas memorizou os dados que já viu. O dataset inclui um conjunto de validação independente, composto por imagens que o modelo nunca viu antes. Isso permite verificar se o modelo é capaz de generalizar, ou seja, de tomar boas decisões em situações reais e com dados novos.

É comum que modelos que vão muito bem nos dados de treino e teste apresentem desempenho inferior na validação, principalmente se estiverem ajustados demais aos dados anteriores (o que chamamos de overfitting). Foi exatamente isso que observamos quando usamos 95% da variância no PCA: o modelo teve ótimo desempenho nos testes, mas caiu na validação. Ao reduzir para 85%, conseguimos um modelo mais equilibrado, que teve bom desempenho tanto nos dados de treino quanto nos de validação.

9.1 Implementação

A estrutura do código é semelhante às etapas anteriores, com inclusão da matriz de confusão para visualização dos acertos e erros.

```
1 # Validação
2 X_val, y_val = carregar_imagens(valida_dataset)
3 X_val_scaled = scaler.transform(X_val)
4 X_val_pca = pca.transform(X_val_scaled)
5
6 y_val_pred = model.predict(X_val_pca)
7
8 print(classification_report(y_val, y_val_pred, target_names=pastas))
9
10 matriz_val = ConfusionMatrixDisplay.from_predictions(y_val, y_val_pred, display_labels=pastas, cmap='Purples')
11 matriz_val.ax_.set_title('Matriz de Confusão - Validação')
12 plt.show()
```

Figura 11: Código da validação

9.2 Saídas

	precision	recall	f1-score	support
NORMAL	0.88	0.88	0.88	8
PNEUMONIA	0.88	0.88	0.88	8
accuracy			0.88	16
macro avg	0.88	0.88	0.88	16
weighted avg	0.88	0.88	0.88	16

Figura 12: Resultados da validação

Na saída do programa, temos o resultados das métricas já vistas anteriormente. Com taxa de acerto de 88% na validação, o modelo apresenta desempenho mais robusto e equilibrado. A acurácia e os F1-scores também melhoraram, mostrando um alto nível de acerto do modelo na validação.

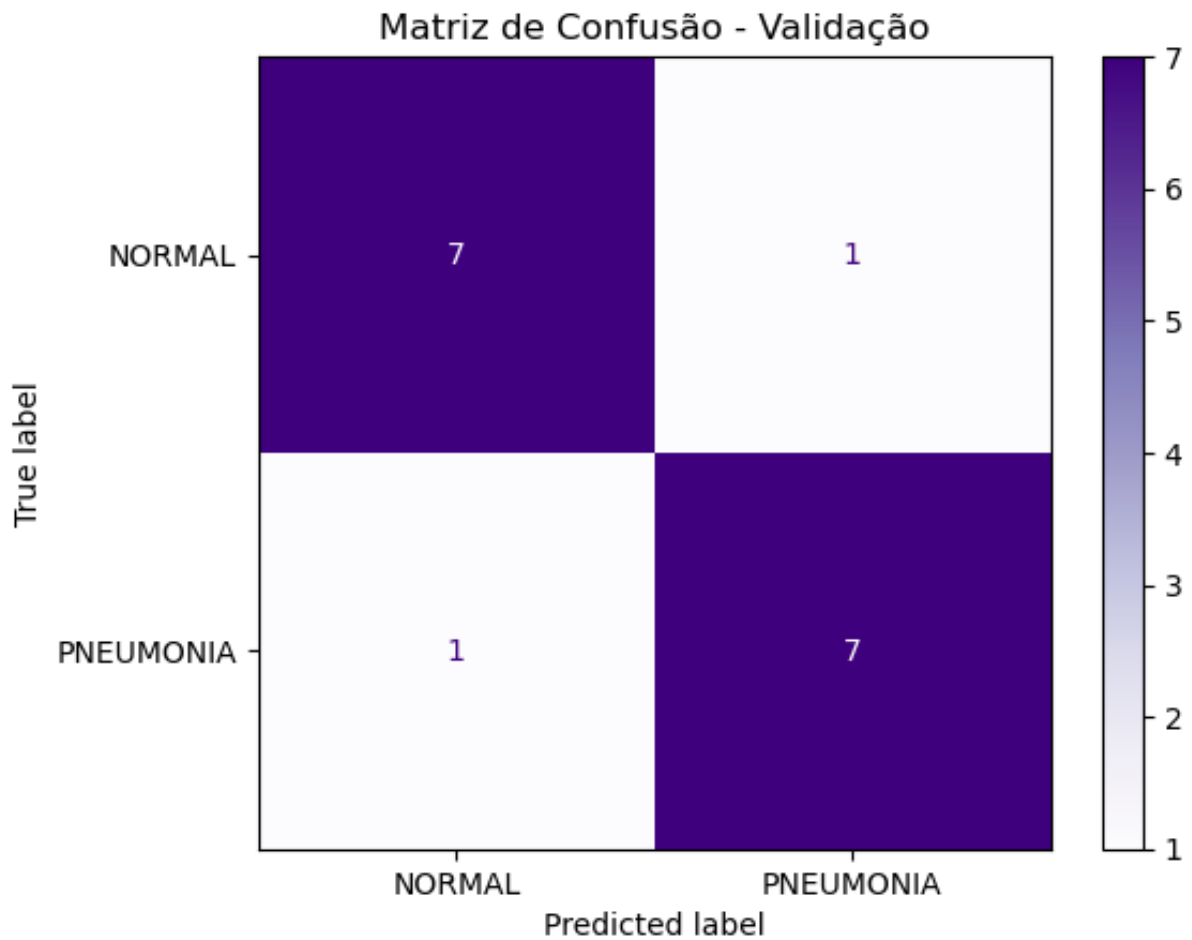


Figura 13: Matriz de confusão da validação

A matriz de confusão nos mostra a quantidade de erros do modelo para cada situação. O true label mostra as situações reais, e o predicted label os chutes do modelo. A matriz de confusão mostra que o modelo cometeu apenas dois erros: um caso normal classificado como pneumonia e vice-versa.

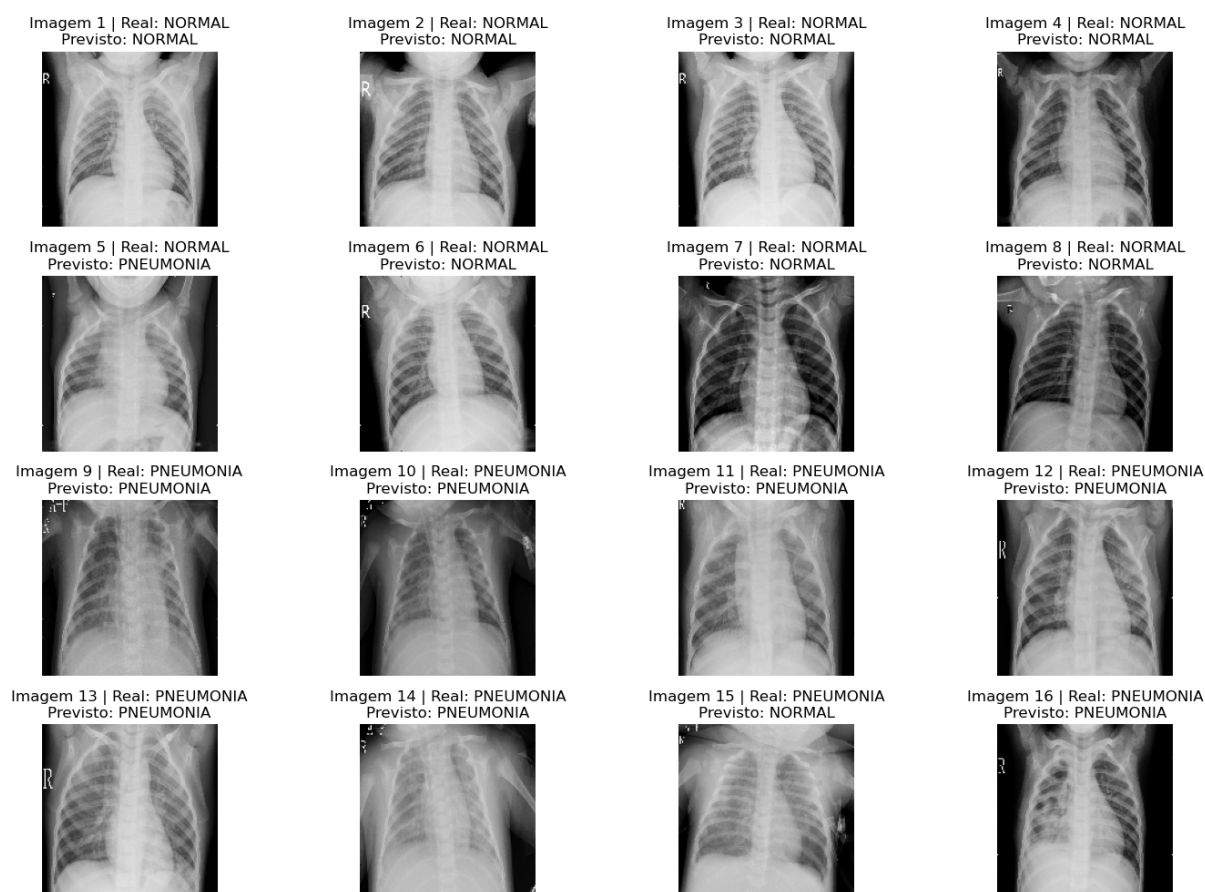


Figura 14: Exemplos de imagens da validação com seus rótulos reais e previstos

10 Referências

- KERMANY, Daniel S. et al. *Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning*. Cell, Volume 172, Issue 5, 11221131.e9.
- KERMANY, Daniel; ZHANG, Kang; GOLDBAUM, Michael. *Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification*. Mendeley Data, V2, 2018. DOI: <https://doi.org/10.17632/rscbjbr9sj.2>
- MOONEY, Paul Timothy. *Chest X-Ray Images (Pneumonia)*. Kaggle. Disponível em: <https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia/data>

11 Links

- Código completo disponível em: <https://github.com/ArthMelo/Deteccao-de-Pneumonia.git>