Reconhecimento de células em exames de Papanicolau

Davi Dias, Paulo Henrique Sendas, Reynaldo Villar Garavini

Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte – MG – Brasil

Department of Computer Science – Pontifícia Universidade Católica de Minas Gerais

davi.magalhaes@sga.pucminas.br, paulo.resende@sga.pucminas.br,

reynaldo.garavini@sga.pucminas.br

Abstract. *The Pap test, or Pap smear, is a histological examination conducted to detect abnormalities in the cells of the cervix, serving as a primary method for early diagnosis of cervical cancer. This study aims to develop an application that reads Pap smear images and enables the automatic recognition of cancerous cells. The application preprocesses the dataset by extracting cell nuclei coordinates from provided images, cropping them into 100x100 sub-images, and categorizing them based on their class. We implemented a fully graphical interface to perform image manipulations, including visualization, grayscale conversion, histogram generation, and feature extraction using Haralick descriptors and Hu moments. The dataset was divided into training and test sets in a 4:1 ratio to ensure balance.*
In the second phase, features used include the 28 Hu moments (4 sets of 7 characteristics), with the shallow classifier being Support Vector Machine (SVM) and the deep classifier being ResNet50. We implemented both binary (negative class vs. others) and multiclass classification, adjusting pre-trained weights from ImageNet for fine-tuning. Performance was evaluated using accuracy metrics and confusion matrices, with training and validation accuracy tracked across epochs.
Our results demonstrate the potential for automated cervical cell classification, providing a foundation for further improvements in diagnostic tools. This document details the methodologies, classifiers, feature extraction techniques, implementation details, and performance evaluations. The code and accompanying documentation are structured in accordance with SBC standards, and we include analysis of execution times and classification outcomes, highlighting both successes and challenges encountered.

Resumo. *O exame de Papanicolau é um exame histológico realizado para detectar anomalias nas células do colo do útero, servindo como método primário para o diagnóstico precoce do câncer cervical. Este estudo tem como objetivo desenvolver um aplicativo que leia imagens de exames de Papanicolau e permita o reconhecimento automático de células cancerosas. O aplicativo pré-processa o conjunto de dados extraindo coordenadas dos núcleos das células a partir das imagens fornecidas, recortando-as em subimagens de 100x100 pixels e categorizando-as com base em suas classes. Implementamos uma interface totalmente gráfica para realizar manipulações de imagem, incluindo visualização,*

conversão para tons de cinza, geração de histogramas e extração de características usando descritores de Haralick e momentos de Hu. O conjunto de dados foi dividido em conjuntos de treinamento e teste na proporção de 4:1 para garantir o balanceamento.
Na segunda fase, as características utilizadas incluem os 28 momentos de Hu (7 características para cada canal), sendo o classificador raso o Máquina de Vetores de Suporte (SVM) e o classificador profundo o ResNet50. Implementamos tanto a classificação binária (classe negativa vs. outras) quanto a classificação multiclasse, ajustando pesos pré-treinados do ImageNet para ajuste fino. O desempenho foi avaliado usando métricas de acurácia e matrizes de confusão, com a acurácia de treinamento e validação acompanhadas ao longo das épocas. Nossos resultados demonstram o potencial para a classificação automatizada de células cervicais, proporcionando uma base para futuras melhorias em ferramentas de diagnóstico. Este documento detalha as metodologias, classificadores, técnicas de extração de características, detalhes da implementação e avaliações de desempenho. O código e a documentação acompanhante estão estruturados de acordo com os padrões da SBC, e incluímos análise de tempos de execução e resultados de classificação, destacando tanto os sucessos quanto os desafios encontrados.

1. Introdução

O exame de Papanicolau, também conhecido como citologia cervical, é um procedimento amplamente utilizado na detecção precoce do câncer cervical. Este exame envolve a coleta de células do colo do útero, que são então analisadas microscopicamente para identificar possíveis anomalias celulares. O diagnóstico precoce é crucial, pois o câncer cervical é um dos tipos de câncer mais tratáveis quando detectado em estágios iniciais. No entanto, a análise manual das lâminas de Papanicolau pode ser demorada e sujeita a erros humanos, o que destaca a necessidade de soluções automatizadas que possam auxiliar patologistas na detecção de células anormais.

Este estudo propõe o desenvolvimento de um aplicativo que utiliza técnicas avançadas de processamento de imagem e aprendizado de máquina para a classificação automática de células cervicais. O objetivo principal é criar um sistema que possa identificar e classificar células potencialmente cancerosas em imagens de exames de Papanicolau, contribuindo para um diagnóstico mais rápido e preciso.

1.1. Entendimento do problema

A classificação automática de células cervicais em exames de Papanicolau apresenta diversos desafios técnicos e clínicos. Primeiramente, as imagens obtidas a partir desses exames podem variar significativamente em termos de qualidade e características visuais, devido a diferenças na coloração, iluminação e resolução. Além disso, a variabilidade intrínseca nas aparências das células normais e anormais requer técnicas robustas de extração de características e classificação para garantir a precisão do diagnóstico.

Objetivos Específicos:

- Pré-processamento de Imagens: Desenvolver técnicas para segmentação das imagens de exames de Papanicolau, extraindo coordenadas dos núcleos celulares e gerando subimagens de 100x100 pixels para análise subsequente.

- Extração de Características: Implementar a extração de descritores de Haralick e momentos de Hu, que são usados para capturar texturas e formas nas imagens das células.

- Classificação:
  Utilizar dois tipos de classificadores:

  - Classificador Raso: Máquina de Vetores de Suporte (SVM) para a classificação inicial com base nas características extraídas.
  - Classificador Profundo: Rede Neural Convolucional ResNet50, ajustada com pesos pré-treinados do ImageNet, para uma análise mais aprofundada e precisa das imagens.

.

- Avaliação de Desempenho: Avaliar o desempenho dos classificadores usando métricas de acurácia e matrizes de confusão, acompanhando a evolução da acurácia de treinamento e validação ao longo das épocas.

2. Modelo classificador raso binário

Este modelo tem como objetivo avaliar se a imagem a ser classificada é de um exame de uma pessoa doente ou não. O treinamento do modelo é feito a partir de dados extraídos da imagem por meio dos momentos invariantes de Hu de um corte de 100x100 pixels da imagem original, realizado nas coordenadas de cada núcleo de célula fornecidas no arquivo classifications.csv. O corte descrito acima foi separado em uma imagem em tons de cinza e nos canais Hue, Saturation e Value, e de cada um deles foram extraídos os momentos de Hu, que foram utilizados para treinar o modelo.

2.1. Disposição dos dados

Para o treinamento, os dados foram organizados da seguinte maneira: os 7 momentos invariantes de Hu de cada um dos 4 canais da imagem foram dispostos em uma tabela, sendo uma coluna para cada momento de Hu e uma coluna ”id”para o número de cada imagem.

2.2. Filtros aplicados

Antes da extração dos momentos invariantes de Hu, foram aplicados filtros para aprimorar a qualidade da imagem e a identificação dos núcleos das células na binarização. Para cada canal, foram aplicados diferentes filtros com o objetivo de melhorar a acurácia do modelo.

2.2.1. Tons de Cinza

Para a imagem em tons de cinza, foi aplicado o filtro de mediana para suavizar a imagem. Em seguida, foi realizada uma limiarização utilizando o valor de corte 75 para binarizar a imagem.

2.2.2. Canal Hue

Para o canal Hue, foi utilizada a detecção de bordas de Canny para identificar melhor as bordas dos objetos da imagem, já que nesse canal o formato da célula pode ser confundido na binarização com manchas muito claras ou muito escuras. Antes de aplicar o filtro Canny Edges, foi utilizado o filtro Gaussiano para redução de ruído. Como a imagem retornada pelo filtro Canny Edges já é composta apenas por preto e branco, não foi necessário aplicar limiarização. O filtro Canny Edges foi aplicado utilizando a janela de Sobel de tamanho 7 para suavização maior da imagem antes de calcular os gradientes.

2.2.3. Canal Saturation

Para o canal Saturation, apenas a aplicação da limiarização da imagem utilizando o limiar 62 foi suficiente para extrair os dados necessários da imagem binarizada.

2.2.4. Canal Value

Para o canal Value, apenas a aplicação da limiarização da imagem utilizando o limiar 62 foi suficiente para extrair os dados necessários da imagem binarizada.

2.3. Aprofundamento sobre técnicas usadas no treinamento

No treinamento do modelo, foi utilizado o Grid Search para encontrar os melhores hiper parâmetros, dentre eles C, kernel e gamma. O balanceamento dos pesos das classes foi feito utilizando o hiper parâmetro class_weight='balanced', que é responsável por compensar a diferença no tamanho da amostra de classe para classe. Após o uso desse parâmetro.

2.4. Resultados

O modelo atingiu a acurácia de 76% com a matriz de confusão disponível na figura 1

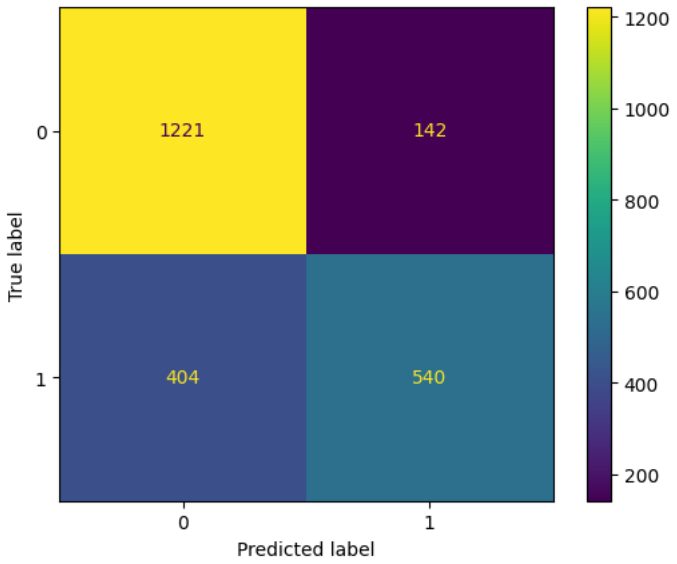

Figura 1. Matriz de confusão Binário

3. Modelo classificador raso multiclasse

Este modelo tem como objetivo avaliar se a imagem a ser classificada é da classe ”Negative for intraepithelial lesion”, ”HSIL”, ”LSIL”, ”ASC-H”, ”ASC-US”ou ”SCC”. O treinamento do modelo é feito a partir de dados extraídos da imagem por meio dos momentos invariantes de Hu de um corte de 100x100 pixels da imagem original, realizado nas coordenadas de cada núcleo de célula fornecidas no arquivo classifications.csv. O corte descrito acima foi separado em uma imagem em tons de cinza e nos canais Hue, Saturation e Value, e de cada um deles foram extraídos os momentos de Hu, que foram utilizados para treinar o modelo.

3.1. Disposição dos dados

Para o treinamento, os dados foram organizados da seguinte maneira: os 7 momentos invariantes de Hu de cada um dos 4 canais da imagem foram dispostos em uma tabela, sendo uma coluna para cada momento de Hu e uma coluna ”id”para o número de cada imagem.

3.2. Filtros aplicados

Antes da extração dos momentos invariantes de Hu, foram aplicados filtros para aprimorar a qualidade da imagem e a identificação dos núcleos das células na binarização. Para cada canal, foram aplicados diferentes filtros com o objetivo de melhorar a acurácia do modelo.

3.2.1. Tons de Cinza

Para a imagem em tons de cinza, foi aplicado o filtro de Gaussiano para suavizar a imagem e, em seguida, o filtro mediana, que também suaviza a imagem. Posteriormente, foi realizada uma limiarização utilizando o valor de corte 75 para binarizar a imagem.

3.2.2. Canal Hue

Para o canal Hue, foi utilizada a detecção de bordas de Canny para identificar melhor as bordas dos objetos da imagem, já que nesse canal o formato da célula pode ser confundido na binarização com manchas muito claras ou muito escuras. Antes de aplicar o filtro Canny Edges, foi utilizado o filtro Gaussiano para redução de ruído. Como a imagem retornada pelo filtro Canny Edges já é composta apenas por preto e branco, não foi necessário aplicar limiarização. O filtro Canny Edges foi aplicado utilizando a janela de Sobel de tamanho 7 para suavização maior da imagem antes de calcular os gradientes.

3.2.3. Canal Saturation

Para o canal Saturation, foi utilizada a detecção de bordas de Canny para identificar melhor as bordas dos objetos da imagem. Antes de aplicar o filtro Canny Edges, foi utilizado o filtro Gaussiano para redução de ruído. Como a imagem retornada pelo filtro Canny Edges já é composta apenas por preto e branco, não foi necessário aplicar limiarização.

3.2.4. Canal Value

Para o canal Value, foi aplicado o filtro de Gaussiano para suavizar a imagem e, em seguida, o filtro mediana, que também suaviza a imagem. Posteriormente, foi realizada uma limiarização utilizando o valor de corte 62 para binarizar a imagem.

3.3. Aprofundamento sobre técnicas usadas no treinamento

No treinamento do modelo, foi utilizado o Grid Search para encontrar os melhores hiperparâmetros, dentre eles C, kernel e gamma. O balanceamento dos pesos das classes foi feito utilizando o hiperparâmetro class_weight='balanced', que é responsável por compensar a diferença no tamanho da amostra de classe para classe. Após o uso desse parâmetro, os pesos gerados foram extraídos e ajustados manualmente a partir da acurácia do modelo.

3.4. Resultados

O modelo atingiu a acurácia de 62% com a matriz de confusão disponível na figura 2

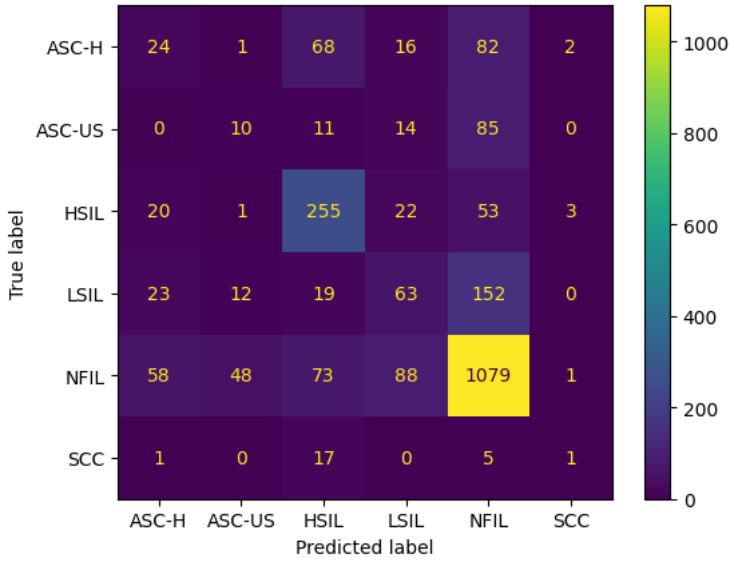

Figura 2. Matriz de confusão Multiclasse

4. Modelo classificador multiclasse profundo

Tem como objetivo treinar e avaliar o modelo de classificação de imagens multiclasse usando a rede neural profunda ResNet50.

4.1. Codificar rótulos multiclasse

Para o codificação de rótulos a codificação de rótulos utiliza 'LabelEncoder' para converter rótulos categóricos em rótulos numéricos e 'to_categorical' converte esses rótulos numéricos em vetores one-hot para a classificação multiclasse.

Os dados são divididos em conjuntos de treinamento e teste na proporção de 80/20 usando 'train_test_split'.

4.2. Calcular pesos das classes

Utiliza 'compute_class_weight' para calcular pesos balanceados para as classes, o que ajuda a lidar com desbalanceamento no conjunto de dados.

4.3. Carregar e ajustar o modelo ResNet50

O modelo base carrega o modelo ResNet50 pré-treinado sem a camada de topo. Com isso, são adicionadas camadas personalizadas específicas para a tarefa, isso inclui, pooling global, camadas densas(512, activation='relu') e dropout(0.5) para regularização. É adicionada uma camada 'Dense' como ativação 'softmax' para a classificação multiclasse.

4.4. Congelar as camadas convolucionais da ResNet50

As camadas convolucionais do modelo base são congeladas para preservar os pesos pré-treinados durante o treinamento inicial.

4.5. Compilação do modelo

O modelo é compilado usando o otimizador Adam, a função de perda 'categorical_crossentropy' e a métrica de acurácia.

4.6. Early stopping

Um callback de Early Stopping é adicionado para interromper o treinamento se o desempenho no conjunto de validação não melhorar após 10 épocas.

4.7. Data augmentation

O 'ImageDataGenerator' é configurado para aplicar diversas transformações nas imagens de treinamento, aumentando a diversidade do conjunto de dados e ajudando na generalização do modelo.

rotation range=40, width shift_range=0.2, height shift range=0.2 ,shear range=0.2, zoom range=0.2, rotation range=40, horizontal flip=True, fill mode='nearest'

Após essa atribuição, o gerador de dados no conjunto de treinamentos é ajustado, em seguida o modelo é treinado com early stopping, data augmentation e class weights.

4.8. Treinamento do modelo

A função 'fit' é usada para treinar o modelo com os dados de treinamento.

'datagen' É o gerador de dados que aplica data augmentation, que é a criação de novas amostras de treinamento a partir das amostras existentes, aplicando transformações como rotações, translações, cisalhamentos, zoom e flips. Isso ajuda a aumentar a diversidade do conjunto de treinamento e melhora a capacidade de generalização do modelo.

'flow(X_train, y_train, batch_size=32)' Esta função gera lotes de dados aumentados a partir dos dados de treinamento 'X_train' e seus rótulos 'y_train'. O 'batch_size=32' indica que cada lote conterá 32 amostras.

'validation_data=(X_test, y_test)' Especifica que os dados de teste ('X_test' e 'y_test') serão usados para validar o desempenho do modelo durante o treinamento. Isso ajuda a monitorar a performance do modelo em dados não vistos, evitando overfitting.

'epochs=70' define que o modelo será treinado por 70 épocas. Uma época é um ciclo completo pelo conjunto de treinamento.

'callbacks=[early_stopping]' é um callback que interrompe o treinamento se o desempenho no conjunto de validação não melhorar após um número especificado de épocas (neste caso, 10 épocas). Ele restaura os pesos do modelo para o estado da melhor época registrada.

'class_weight=class_weights_dict' O class_weight (pesos das classes) é um dicionário que fornece pesos para cada classe, usado para lidar com desbalanceamento de classes. Classes menos frequentes recebem maior peso para que o modelo não seja tendencioso em favor das classes mais frequentes.

4.9. Resultados

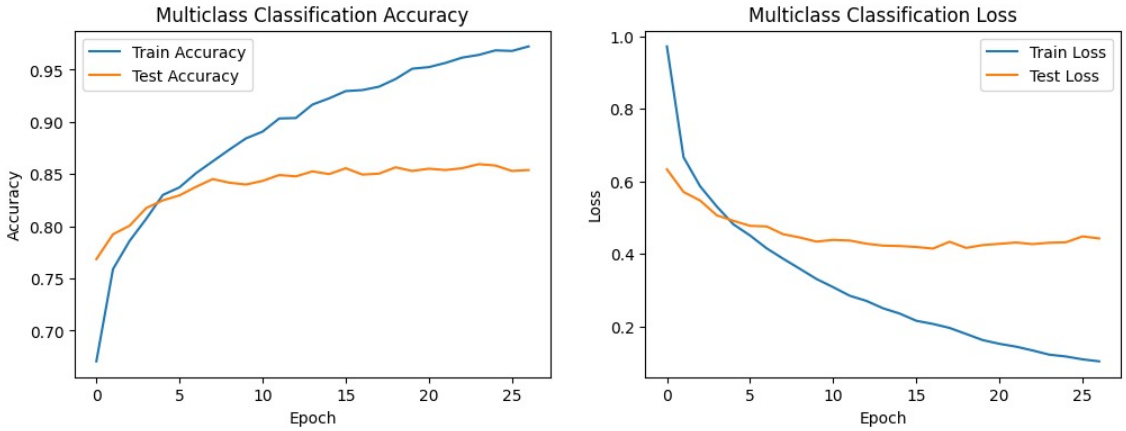

Figura 3. Modelo sem ajustes - Multiclasse Profundo

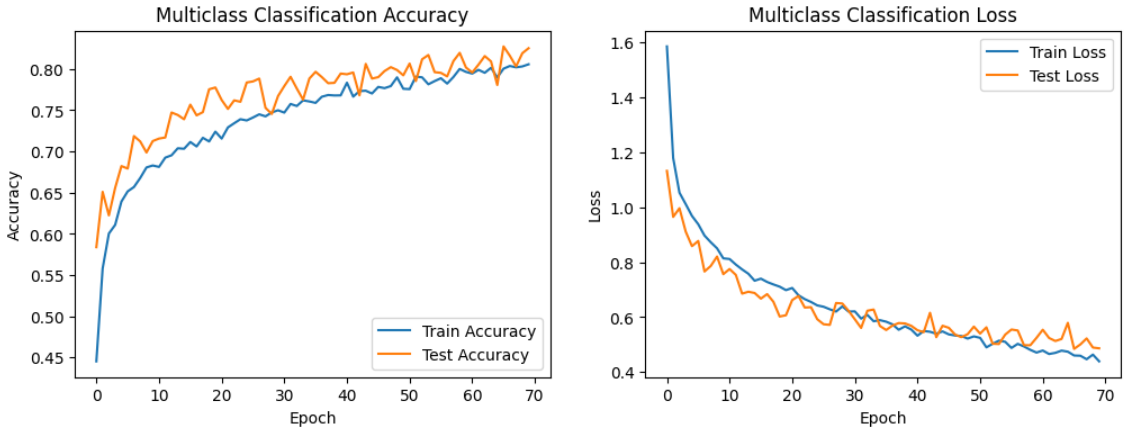

Figura 4. Modelo final - Multiclasse Profundo

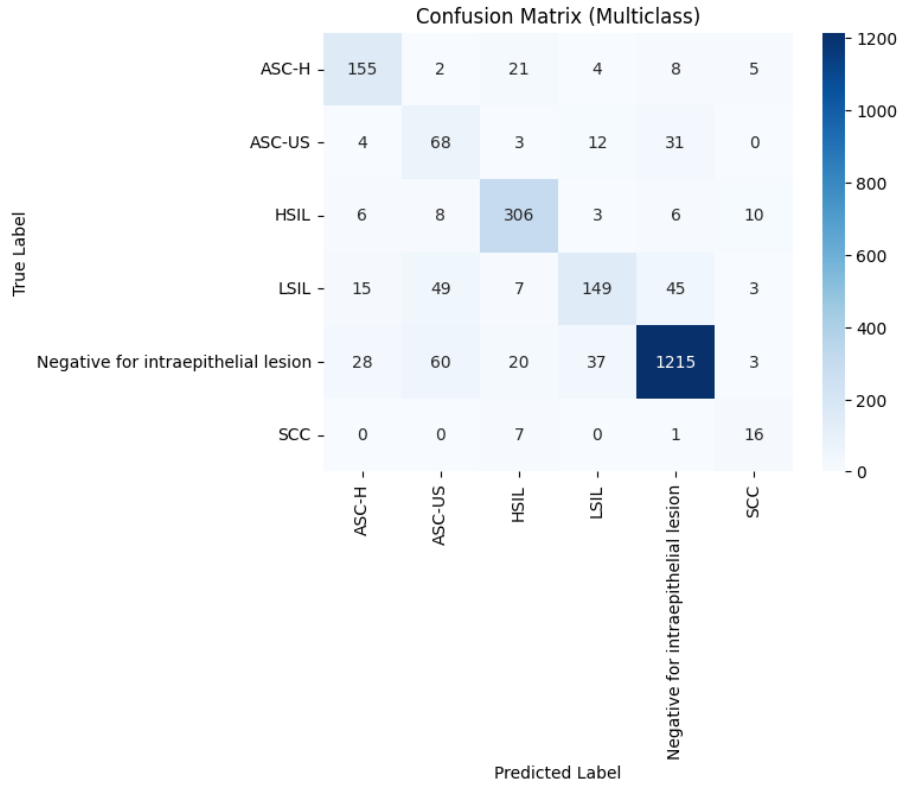

Figura 5. Matriz - Modelo final - Multiclasse Profundo

Após os primeiros testes, que não continha regularização dos dados, nas primeiras execuções, foi notado que a acurácia de treinamento começa em cerca de 0.70 (Figura 3) e aumenta rapidamente para aproximadamente 0.95 ao final de 25 épocas. Já a acurácia de validação começa em cerca de 0.80 e se estabiliza em torno de 0.85. A acurácia de treinamento é significativamente maior que a acurácia de validação, indicando um possível overfitting, onde o modelo está performando muito bem nos dados de treinamento, mas não tão bem nos dados de validação. No grafico de perda, a perda de treinamento começa em aproximadamente 1.0 e diminui para cerca de 0.1 e a perda de validação começa em cerca de 0.6 e se estabiliza em torno de 0.4. A perda de treinamento é muito menor que a perda de validação, o que também sugere overfitting. O modelo pode estar se ajustando muito bem aos dados de treinamento, mas não generalizando bem para os dados de validação.

Já no mmodelo final (Figura 4), a acurácia de treinamento começa em aproximadamente 0.45 e aumenta consistentemente até cerca de 0.80 ao final de 70 épocas e a acurácia de validação segue uma trajetória semelhante, começando em cerca de 0.50 e atingindo aproximadamente 0.80 ao final de 70 épocas. A acurácia de treinamento e validação estão próximas, indicando que o modelo está aprendendo bem e não está sofrendo de overfitting significativo. A perda de treinamento começa em aproximadamente 1.6 e diminui consistentemente para cerca de 0.4. A perda de validação começa em cerca de 1.2 e também diminui para aproximadamente 0.4. Tanto a perda de treinamento quanto a de validação estão diminuindo de forma consistente, indicando que o modelo está aprendendo eficientemente.

5. Modelo Classificador Binário Profundo

Este código Python começa com funções para carregar, pré-processar e organizar imagens de diferentes classes a partir de um diretório base, preparando os dados para serem utilizados em um modelo de aprendizado profundo. A função load_and_preprocess_image() tem como objetivo carregar uma imagem a partir de um caminho especificado, redimensioná-la para um tamanho padrão (224x224 pixels), convertê-la para um array NumPy, expandir suas dimensões para incluir um lote e, finalmente, aplicar um pré-processamento adequado para ser utilizada como entrada em um modelo de rede neural. Este pré-processamento é crucial para que o modelo possa ser treinado de forma eficaz e eficiente, garantindo que os dados de entrada estejam na forma esperada pelo modelo ResNet-50. Após o pré-processamento, o codígo utiliza destas imagens já tratadas como um conjuntos de dados para o treinamento de um modelo de aprendizado profundo para um problema de classificação binária. Ele inclui a codificação dos rótulos, divisão dos dados, ajuste de um modelo pré-treinado, congelamento de camadas, compilação do modelo, treinamento avaliação do modelo. Cada etapa é essencial para garantir que o modelo seja treinado de maneira eficaz e eficiente, maximizando sua performance na tarefa de classificação.

5.1. Aprofundamento sobre técnicas usadas no treinamento

5.1.1. Divisão dos Dados

Usa train_test_split para dividir os dados e os rótulos binários em conjuntos de treino e teste. test_size=0.2 significa que 20% dos dados serão usados para teste. random_state é definido para garantir a reprodutibilidade dos resultados.

5.1.2. Cálculo dos Pesos

Usa compute_class_weight para calcular os pesos das classes. class_weight='balanced' ajusta os pesos para serem inversamente proporcionais à frequência das classes no conjunto de dados. Isso é importante para lidar com conjuntos de dados desbalanceados.

5.1.3. Ajuste do Modelo ResNet-50 Pré-treinado

Carregar Modelo Pré-treinado: Carrega a ResNet-50 pré-treinada sem as camadas de classificação superiores (include_top=False).

5.1.4. Adição de Camada GlobalAveragePooling2D

Esta camada reduz a dimensão espacial (altura e largura) da saída convolucional da ResNet-50 ao calcular a média de todas as unidades de ativação em cada mapa de características. É frequentemente usada para substituir as camadas totalmente conectadas finais, reduzindo a quantidade de parâmetros e, assim, ajudando a evitar o overfitting.

5.1.5. Adição da camada Relu

A camada ReLU ajuda a introduzir não-linearidades na rede, permitindo que ela aprenda uma representação mais rica e complexa dos dados.

5.1.6. Adição da camada Dropout

Durante o treinamento, a camada dropout desativa aleatoriamente metade dos neurônios da camada anterior, ajudando a rede a generalizar melhor. Dropout é uma técnica regularizadora que ajuda a prevenir o overfitting, forçando a rede a não depender de neurônios específicos.

5.2. Criação do modelo

Treina o modelo model_bin com os dados de treino X_train_bin e y_train_bin, utilizando os seguintes paramêtros:

5.2.1. Epochs

O valor de 17 foi escolhido com base em experimentações anteriores, onde foi observado que o modelo convergia bem e atingia um bom desempenho após 17 epochs.

5.2.2. Steps per Epoch

O valor 60 foi escolhido com base em experimentações anteriores que eram maiores e geravam overfitting.

5.2.3. Peso das Classes

Através do 'class_weight=class_weights', passamos os pesos de classe ajustam o impacto de cada classe na função de perda durante o treinamento.

5.3. Resultados

Após os primeiros testes, que não continha regulzarização dos dados, menor o número de epochs e maior número de steps per epoch a acurácia de treino subia quase que linearmente tendendo a 100, enquanto a acurácia de teste estava sem crescimento. Analogamente interpretamos o gráfico de perda, no qual, quando os gráficos de perda de teste e treinamento mostram que a perda de teste está subindo enquanto a de treinamento está diminuindo, isso é um forte indicador de overfitting. Onde o modelo estava se tornando muito bom em prever os dados de treinamento, mas está perdendo a capacidade de generalizar para novos dados. Estes resultados podem ser vizualizados na Figura 6 e Figura 7.

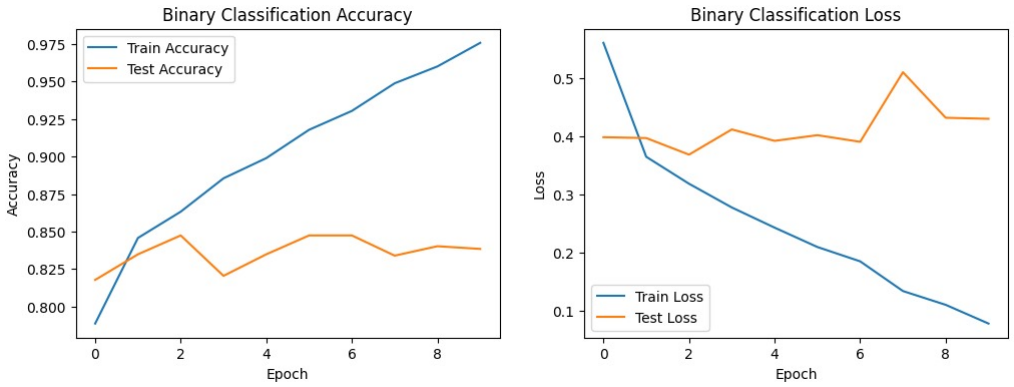

Figura 6. Comportamento do modelo sem ajustes, com os padrões da classe

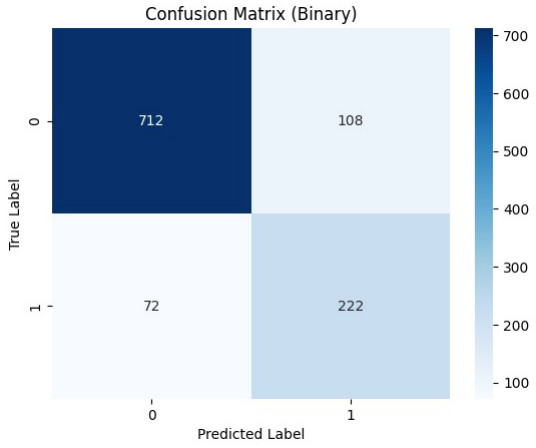

Figura 7. Matriz de confusão do modelo sem ajustes, com os padrões da classe

Ao analisar os gráficos e detectar o overfitting, foram implementados técnicas para melhorar o desempenho, como dropout , aumento de epochs, e dimuição de steps por epoch.

Após as alterações os resultados são bons pois mostram que o modelo está aprendendo eficientemente a partir dos dados de treinamento e está generalizando bem para novos dados. O aumento na acurácia e a diminuição na perda em ambos os conjuntos de dados ao longo das épocas indicam que o treinamento está progredindo de forma eficaz e que o modelo é robusto para prever corretamente a classificação binária em dados não vistos. O que pode ser visualmente compreendido através da na Figura 8 e Figura 9.

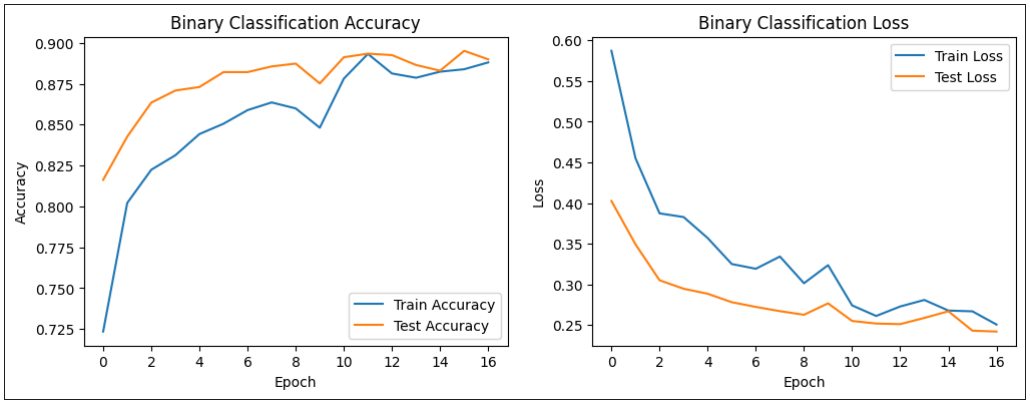

Figura 8. Comportamento final do modelo binário profundo

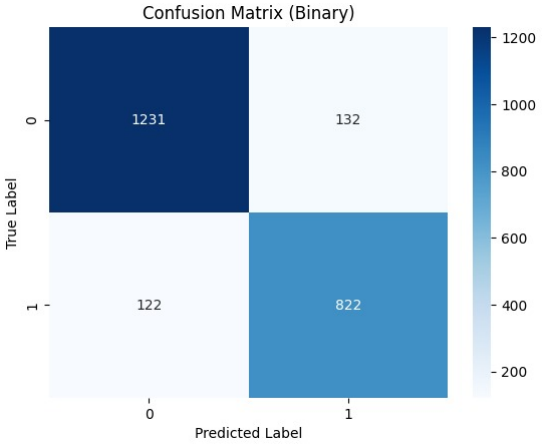

Figura 9. Matriz de confusão do modelo binário profundo

6. Métricas

Comparativo		
Algoritmo	Tempo	Acurácia
Resnet Binario	48 minutos	89%
Resnet Multi	125 minutos	81%
SVM Binario	4,3 segundos	76%
SVM Multiclasse	8 segundos	62%

7. Conclusão

A avaliação comparativa entre um modelo de classificação raso (SVM) e um profundo (ResNet) revelou diferenças significativas em termos de desempenho. O SVM, aplicado aos momentos de HU das imagens nem sempre claras, alcançou uma acurácia de 62% e 76%, enquanto a ResNet, com sua capacidade de aprendizado profundo, obteve uma acurácia notável de 81% e 89%. Essa disparidade demonstra a vantagem do modelo profundo quando se trata de capturar e aprender características complexas e abstratas dos dados, especialmente em cenários onde a clareza das imagens pode variar.

Em conclusão, os resultados destacam que, apesar da abordagem mais simples o SVM pode ser aplicável e interpretável em condições específicas ele possui execução muito rápida. Porém quando se dispõe de recursos de processamento suficientes, investir em modelos profundos como a ResNet resultou em melhorias significativas no desempenho de classificação. Portanto, a escolha do modelo ideal deve considerar não apenas a complexidade dos dados e a clareza das imagens, mas também a disponibilidade de recursos computacionais para processamento mais intensivo.

8. Referências das bibliotecas

8.1. tensorflow.keras

Utilizada no treinamento dos modelos profundos.

8.2. Matplotlib e Seaborn

Utilizada para plotar gráficos.

8.3. sklearn

Utilizada para treinamento de modelos rasos.

8.4. tkinter

Utilizada para criação da interface

8.5. ttkthemes

Utilizada para alterar o visual da interface tkinter

8.6. Numpy e Pandas

Utilizada para manipulação de dados

8.7. cv2 e PIL

Utilizada para aplicar filtros e manipulação de imagens

8.8. joblib

Utilizada para salvar o modelo raso em arquivo

8.9. os

Utilizada para manipulação de arquivos e diretórios

Referências

[1] Marcone J. Freitas. Pap Test Image Classification. Journal of Imaging, 2021. http://www.decom.ufop.br/prof/marcone/projects/ppm676-17/Jimaging-2021-PapTestImageClassification.pdf.

[2] International Journal of Data Science and Artificial Intelligence (IJODAS). International Journal of Data Science and Artificial Intelligence. 2023. https://jurnal.yoctobrain.org/index.php/ijodas.