Classificação de Imagens Usando Redes Neurais Convolucionais

Marcel Kendy Rabelo Matsumoto
Sistemas de informação
Universidade Federal de Viçosa
Rio Paranaíba, Brasil
marcel.matsumoto@ufv.br

***Abstract*—Este artigo apresenta uma análise comparativa da classificação de imagens utilizando Redes Neurais Convolucionais (CNNs) com diversas arquiteturas, incluindo DenseNet, InceptionV3, MobileNetV2, ResNet18 e VGG16. O objetivo é avaliar o desempenho desses modelos em uma tarefa específica de classificação de imagens, focando na precisão e na evolução da perda ao longo das épocas. Após explorar a arquitetura e as técnicas de otimização utilizadas em cada modelo, analisei as matrizes de confusão resultantes para avaliar o desempenho na classificação. O treinamento foi realizado utilizando um conjunto de dados de tumores cerebrais, e os resultados mostram níveis variados de eficácia entre os modelos, com ênfase nas taxas de convergência e nas capacidades de generalização.**

***Index Terms*—classificação, Redes Neurais Convolucionais, modelos, análise**

I. Introdução

Este projeto tem como objetivo a implementação de um sistema para a classificação de imagens utilizando Redes Neurais Convolucionais (CNNs, do inglês Convolutional Neural Networks). O sistema foi desenvolvido em Python, utilizando frameworks como PyTorch, scikit-learn e matplotlib, com foco em eficiência e flexibilidade no treinamento de modelos de aprendizado profundo.

A classificação de imagens é uma tarefa central no campo da inteligência artificial, especialmente em áreas como diagnóstico médico e reconhecimento visual. Neste projeto, o foco está na construção de um sistema capaz de treinar e avaliar diferentes modelos de CNNs, aplicados ao conjunto de dados do Brain Tumor MRI Dataset, disponível no Kaggle. Esse dataset contém imagens de ressonância magnética (RM) do cérebro, classificadas em quatro categorias: glioma, meningioma, sem tumor e pituitária. A escolha desse conjunto de dados se deve à sua relevância na área da medicina, especialmente no diagnóstico precoce de tumores cerebrais, um campo onde a detecção rápida e precisa pode ser vital para a escolha do tratamento adequado e a melhoria da qualidade de vida do paciente.

O sistema desenvolvido é capaz de treinar diferentes modelos de redes neurais, avaliar seu desempenho em conjuntos de validação e ajudar na escolha do modelo com a melhor performance. Durante o treinamento, são calculadas as perdas de treinamento e validação, permitindo o monitoramento contínuo do progresso dos modelos. Para isso, foi utilizada a função de perda *cross-entropy-loss*, indicada para problemas de classificação multiclasse, como o presente. Além disso, o sistema é altamente configurável, permitindo a personalização de parâmetros como o número de épocas de treinamento, a tolerância a épocas não produtivas, o otimizador e outros parâmetros críticos.

O projeto inclui a implementação de cinco modelos de CNN amplamente utilizados no campo de visão computacional, a saber:

ResNet-18: Modelo eficiente baseado em redes residuais, ideal para tarefas de classificação de imagens. VGG-16: Um modelo mais simples e profundo, conhecido pela sua arquitetura de camadas convolucionais sucessivas. InceptionV3: Modelo avançado que utiliza múltiplos caminhos de convoluções para capturar informações em diferentes escalas. AlexNet: Um dos primeiros modelos profundos a alcançar grandes avanços em tarefas de reconhecimento visual. DenseNet-121: Modelo com conexões densas entre camadas, o que facilita a troca de informações entre as camadas e melhora a precisão. Com isso, o projeto permite a comparação do desempenho de diferentes modelos de redes neurais em um problema prático de classificação de imagens, facilitando a escolha do modelo mais adequado para uma aplicação específica.

II. Revisão Bibliográfica

Diversos estudos abordaram a aplicação de redes neurais convolucionais (CNNs) na classificação de imagens, incluindo em áreas da saúde como a detecção de tumores cerebrais. A seguir, são apresentados seis trabalhos relevantes que exploram diferentes abordagens e técnicas dentro desse contexto.

A. Pathak et al. (2019)

No trabalho de Pathak et al. (2019) [1], foi desenvolvida uma abordagem baseada em redes neurais convolucionais (CNNs) para a classificação e segmentação de tumores cerebrais em imagens de ressonância magnética (MRI). A pesquisa utilizou núcleos convolucionais pequenos para permitir redes mais profundas e reduzir o risco de overfitting. O modelo desenvolvido não apenas classificou as imagens quanto à presença de tumor, mas também segmentou a área afetada utilizando o Algoritmo de Watershed e operações morfológicas. Além disso, o estudo incluiu o cálculo da área do tumor. O modelo apresentou uma precisão de 98%, demonstrando alta eficiência com baixa complexidade computacional.

B. Vaishnav e Singh (2024)

Vaishnav e Singh (2024) [2] propuseram um sistema baseado em redes neurais convolucionais (CNNs) para a identificação automática de tumores cerebrais em imagens de ressonância magnética (MRI). O método busca superar limitações de abordagens convencionais, que dependem da segmentação manual e análise realizada por radiologistas, frequentemente propensas a erros e demandando muito tempo. As CNNs foram treinadas em um grande conjunto de dados rotulados, permitindo a identificação de padrões e variações associados a diferentes tipos de tumores. O sistema apresentou precisão superior a 99% em certos casos, demonstrando seu potencial para automatizar a detecção de tumores cerebrais, aumentando a eficácia do diagnóstico e, consequentemente, melhorando os resultados para os pacientes.

C. Ayadi et al. (2021)

Ayadi et al. (2021) [3] destacaram a gravidade dos tumores cerebrais, que estão entre os cânceres mais letais e apresentam diversas formas e características, dependendo de fatores como localização e textura. Eles propuseram um sistema automatizado de diagnóstico assistido por computador (CAD) baseado em redes neurais convolucionais profundas (CNNs) para classificar tumores cerebrais em imagens de ressonância magnética (MRI). A abordagem sugere um novo modelo com múltiplas camadas, projetado para lidar com a variabilidade dos tipos de tumor cerebral e o grande volume de dados, superando as limitações de técnicas manuais que podem ser demoradas e propensas a erros humanos. Resultados experimentais em três conjuntos de dados indicam que o modelo proposto apresenta desempenho convincente em comparação com métodos existentes.

D. Srinivas et al. (2022)

Srinivas et al. (2022) [4] enfatizaram a importância da classificação de tumores cerebrais como etapa crucial para avaliar tecidos anormais que ameaçam a vida e oferecer tratamentos eficientes. Utilizando imagens de ressonância magnética (MRI), amplamente reconhecidas pela qualidade superior e ausência de radiação ionizante, o estudo avaliou a aplicação de redes neurais convolucionais (CNNs) baseadas em aprendizado profundo para a detecção automática de tumores. Eles conduziram uma análise comparativa de desempenho entre os modelos pré-treinados VGG-16, ResNet-50 e Inception-v3, que também foram avaliados no meu trabalho, aplicados a um conjunto de dados com 233 imagens de MRI. O modelo VGG-16 destacou-se pela alta precisão, mostrando resultados superiores tanto no treinamento quanto na validação, demonstrando a eficácia do aprendizado por transferência na detecção de tumores cerebrais.

E. Habibzadeh et al. (2018)

Habibzadeh et al. (2018) [5] propuseram um sistema de diagnóstico assistido por computador (CAD) para a classificação diferencial de leucócitos, conhecidos como glóbulos brancos, que desempenham um papel essencial no sistema imunológico. A análise manual de amostras de sangue é demorada, cara e sujeita a erros, especialmente devido à variabilidade entre os tipos e subtipos de leucócitos. Para abordar esses desafios, os autores desenvolveram um framework baseado em aprendizado profundo que inclui etapas de pré-processamento, como correção de distorção de cor, recorte e espelhamento de imagens.

Utilizando arquiteturas ResNet e Inception, o sistema classificou leucócitos em quatro categorias principais: neutrófilos, eosinófilos, linfócitos e monócitos. Em testes realizados com um conjunto de 11.200 amostras de treinamento e 1.244 amostras de validação, o modelo alcançou uma precisão média de 100% com ResNet V1 50, além de resultados promissores de 99,84% e 99,46% com ResNet V1 152 e ResNet 101, respectivamente. A análise estatística também revelou valores elevados de sensibilidade e taxas quase nulas de falsos negativos. Esse estudo destaca a eficácia do uso de técnicas avançadas de aprendizado profundo na classificação de leucócitos, com potencial para melhorar significativamente a confiabilidade e a eficiência no diagnóstico hematológico.

F. Sharma et al. (2022)

Além das aplicações específicas em classificação de tumores cerebrais, é fundamental compreender os avanços gerais em modelos de aprendizado profundo aplicados à classificação de imagens. Sharma e Guleria (2022) [6] realizaram uma análise detalhada sobre diferentes modelos de aprendizado profundo, destacando os métodos mais utilizados, como as Redes Neurais Convolucionais (CNNs), que permanecem como padrão para tarefas de visão computacional e classificação de imagens. No entanto, o estudo também apontou as limitações das CNNs tradicionais ao lidar com dados de entrada mais complexos. Essa revisão teve como objetivo comparar diversas arquiteturas de aprendizado profundo, avaliando seu desempenho em bases de dados populares, e contribuir para a escolha de modelos eficazes para tarefas de classificação e segmentação de imagens em cenários variados.

III. Material e Métodos

A. Modelo de Redes Neurais Convolucionais

Para a tarefa de classificação de imagens, foi utilizado um modelo de rede neural convolucional pré-treinado, com opções para diferentes arquiteturas de redes como *ResNet18*, *VGG16*, *InceptionV3*, *DenseNet121* e *MobileNetV2*. A escolha do modelo é realizada com base na função `initialize_model`, que seleciona e personaliza o modelo conforme o número de classes da tarefa e a possibilidade de congelar as camadas de extração de características (*feature extraction*). A customização das redes envolve a substituição da camada final de classificação de acordo com o número de classes específicas do problema.

B. Pré-processamento dos Dados

O pré-processamento dos dados foi projetado para preparar as imagens para treinamento, validação e teste, além de aumentar a robustez do modelo por meio de técnicas de *data*

augmentation. Os dados foram carregados usando a classe `ImageFolder` do PyTorch, que organiza as imagens em função das classes, e posteriormente divididos em subconjuntos utilizando a técnica de *hold-out*.

1) Divisão dos Dados: Hold-Out: A técnica de *hold-out* foi empregada para dividir o conjunto de dados original em três subconjuntos: treinamento (70% dos dados), validação (20%) e teste (10%). Inicialmente, 20% do conjunto total foi reservado exclusivamente para teste. O restante dos dados foi subdividido, com 70% destinado ao treinamento e 30% à validação. Essa abordagem garante que os dados de teste permaneçam isolados e não influenciem o treinamento ou a validação.

2) Data Augmentation: Para aumentar a diversidade dos dados de treinamento, foi aplicado um conjunto de transformações nos dados, conhecidas como *data augmentation*. As transformações utilizadas foram:

- `RandomResizedCrop`: Redimensiona e recorta aleatoriamente as imagens para um tamanho fixo de 299x299 pixels.
- `RandomHorizontalFlip`: Aplica espelhamento horizontal aleatório às imagens.
- `ColorJitter`: Ajusta aleatoriamente brilho, contraste, saturação e matiz das imagens.
- `RandomRotation`: Rotaciona as imagens em até 15 graus de forma aleatória.
- `ToTensor` e `Normalize`: Convertem as imagens em tensores e normalizam os valores de pixel para que sigam uma distribuição com média [0.485, 0.456, 0.406] e desvio-padrão [0.229, 0.224, 0.225], compatível com modelos pré-treinados.

Essas transformações são aplicadas apenas ao conjunto de treinamento. Para os conjuntos de validação e teste, apenas transformações básicas de redimensionamento, conversão em tensor e normalização foram realizadas, a fim de evitar alterações indesejadas nesses conjuntos.

3) Preparação dos Conjuntos de Dados: O pré-processamento e a divisão foram implementados conforme segue:

- O conjunto original foi carregado usando `ImageFolder`, e transformações foram aplicadas separadamente para treinamento, validação e teste.
- A divisão entre os conjuntos foi realizada com a função `random_split`, garantindo que as proporções de *hold-out* fossem respeitadas.
- As transformações específicas de cada subconjunto foram aplicadas dinamicamente usando a propriedade `.transform`.

C. Otimização de Hiperparâmetros

A otimização de hiperparâmetros foi realizada utilizando uma abordagem de busca aleatória (*random search*), explorando combinações de diferentes taxas de aprendizado, tamanhos de lote e otimizadores. Esse processo foi projetado para identificar a melhor configuração de hiperparâmetros para cada modelo.

1) Abordagem de Otimização: Para cada modelo de rede neural, foram testadas combinações aleatórias de hiperparâmetros dentro de limites predefinidos:

- **Taxas de aprendizado**: {0.01, 0.001, 0.0001}.
- **Tamanhos de lote**: {8, 16, 32}.
- **Otimizadores**: `Adam`, `SGD`.

A porcentagem de combinações testadas foi controlada pelo parâmetro `combinations_percentage`, enquanto o número máximo de combinações foi limitado por `upper_limit_testings`.

2) Fluxo do Processo de Otimização: 1. Para cada modelo, as combinações de hiperparâmetros foram geradas aleatoriamente. 2. O conjunto de treinamento foi dividido em lotes usando o tamanho de lote da combinação corrente. 3. O modelo foi treinado por `num_epochs` épocas em cada combinação, com a perda de validação calculada ao final de cada época. 4. A combinação que resultou na menor perda de validação foi registrada como a melhor para o modelo correspondente.

3) Resultados da Otimização: Os melhores hiperparâmetros para cada modelo foram armazenados em um dicionário (`hyperparams_dict`), permitindo que fossem utilizados no treinamento final do modelo. O processo de otimização foi projetado para ser eficiente, limitando o número de combinações testadas para evitar consumo excessivo de tempo e recursos computacionais.

D. Estratégia de Treinamento

A função de treinamento do modelo é implementada pela função `train_model`, que gerencia o fluxo de treinamento, validação e salvamento do melhor modelo. O fluxo de treinamento é controlado por diversos parâmetros, como número de épocas (`epochs`), paciência para early stopping (`patience`) e timeout (`timeout`), que determinam, respectivamente, o número máximo de épocas de treinamento, o número de épocas sem melhoria na perda de validação antes de interromper o treinamento, e o tempo máximo permitido para o treinamento.

1) Parâmetros da Função de Treinamento: A função `train_model` recebe os seguintes parâmetros:

- `model`: Instância do modelo de rede neural.
- `train_loader`: `DataLoader` para o conjunto de treinamento.
- `val_loader`: `DataLoader` para o conjunto de validação.
- `criterion`: Função de perda utilizada no treinamento.
- `optimizer`: Otimizador utilizado para atualizar os pesos da rede.
- `device`: Indica se o treinamento será feito em `cpu` ou `cuda` (GPU).
- `model_name`: Nome do modelo, utilizado para salvar gráficos e resultados.
- `epochs`: Número máximo de épocas de treinamento.
- `patience`: Número de épocas sem melhoria na perda de validação antes de interromper o treinamento (early stopping).

- `timeout`: Tempo máximo de treinamento permitido. Se `None`, não há limite de tempo.

2) Fluxo de Treinamento: Durante o treinamento, o modelo é alternadamente colocado no modo de treinamento e validação. Para cada época, o modelo é treinado utilizando os dados do `train_loader`, onde os gradientes são calculados e os parâmetros do modelo são atualizados. Após cada época de treinamento, a perda de validação é calculada utilizando o `val_loader`. Caso a perda de validação melhore, o modelo é salvo como o melhor modelo até o momento. O treinamento pode ser interrompido precocemente caso não haja melhoria significativa na perda de validação por um número determinado de épocas consecutivas (early stopping), ou se o tempo de treinamento atingir o limite estipulado.

Bom salientar que durante o treinamento, a progressão da perca de treino e validação são transferidas para a função `save_loss_graph`, que produz um gráfico de linhas mostrando a evolução desses índices ao longo das épocas, para complementar o substrato de avaliação no final do projeto.

E. Avaliação do Modelo

Após o treinamento, o modelo é avaliado no conjunto de teste utilizando a função `evaluate_model`, que calcula métricas de desempenho como *accuracy*, *precision*, *recall*, *f1-score* e a matriz de confusão. Durante a avaliação, os resultados de cada imagem de teste são registrados em um arquivo de log para posterior análise. Além disso, é gerado um gráfico da matriz de confusão, que ilustra o desempenho do modelo em relação às diferentes classes.

1) Parâmetros da Função de Avaliação: A função `evaluate_model` recebe os seguintes parâmetros:

- `model`: O modelo treinado.
- `test_loader`: `DataLoader` contendo o conjunto de teste.
- `device`: O dispositivo de treinamento (`cpu` ou `cuda`).
- `class_names`: Lista contendo os nomes das classes.
- `model_name`: Nome do modelo, utilizado para salvar gráficos e resultados.

2) Fluxo de Avaliação: Durante a avaliação, o modelo é colocado no modo de avaliação (`model.eval()`) e as previsões são feitas sobre o conjunto de teste. As métricas de avaliação são calculadas utilizando as funções `accuracy_score`, `precision_score`, `recall_score` e `f1_score`. A matriz de confusão é gerada com a função `confusion_matrix`, e uma imagem da matriz é salva.

Todos os resultados e porcentagens de classificação de cada imagem do conjunto de teste são registrados em um arquivo de log de texto.

F. Visualização dos Resultados

1) Gráficos e Log de Classificação: Como dito anteriormente, além das métricas de desempenho, os resultados da validação e avaliação são visualizados em gráficos, como o gráfico de progresso da perda de treinamento e validação e a matriz de confusão. A função `save_loss_graph` gera o gráfico de perdas de treinamento e validação, enquanto a função `plot_confusion_matrix` plota e salva a matriz de confusão para avaliar o desempenho do modelo em relação às diferentes classes. Também é salvo um arquivo de texto para cada modelo demonstrando a classificação feita sob cada imagem em questão, com a porcentagem de decisão entre as classes.

2) Monitoramento de Progresso: Informações úteis como: época atual, número total de épocas, parâmetros importantes e o modelo avaliado no momento são impressas no terminal e também em um arquivo log de texto durante todas as etapas descritas, (inicialização, otimização de hiperparâmetros, treinamento e avaliação) do ínicio ao fim.

O tempo total de treinamento é monitorado e impresso durante o processo, com informações sobre o tempo gasto até o momento e o limite de tempo, se houver. Dessa forma, a análise do funcionamento de um algoritmo tão extenso e muitas vezes despendioso em tempo, fica mais compreensível.

No estudo em questão, foi utilizado um processador i7 de 7ª geração e 8GB de memória RAM em um sistema operacional Windows 11 Home.

IV. Resultados e Discussão

A. Resultados por Modelo

1) ResNet18: O modelo ResNet18 alcançou uma acurácia de 87.82% no conjunto de teste. Os valores de precisão, revocação e F1-Score foram 87.77%, 87.82% e 87.63%, respectivamente. A matriz de confusão e a progressão da perda durante o treinamento estão ilustradas nas Figuras 1 e 2.

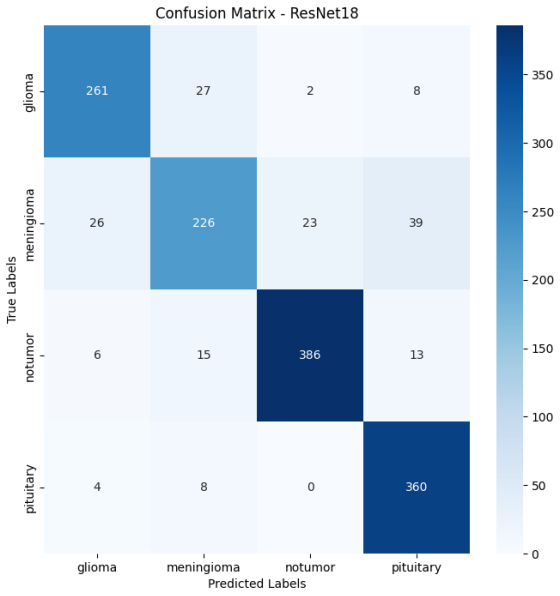

Fig. 1. Matriz de confusão para o modelo ResNet18.

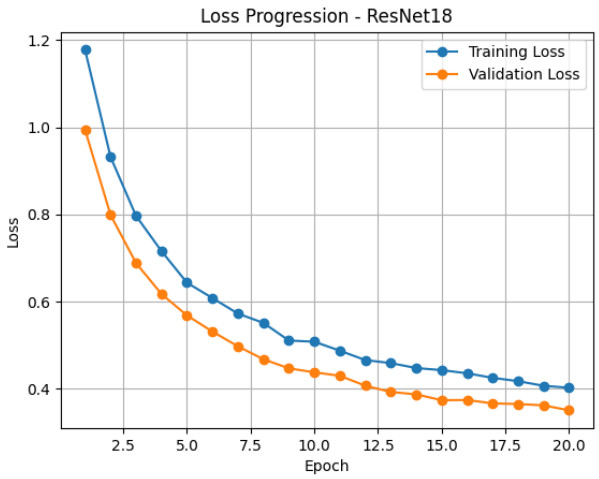

Fig. 2. Progressão da perda durante o treinamento do modelo ResNet18.

2) VGG16: O modelo VGG16 apresentou uma acurácia de 88.96%. Os demais métricos incluem uma precisão de 89.27%, revocação de 88.96% e F1-Score de 89.04%. A Figura 3 mostra a matriz de confusão, enquanto a Figura 4 apresenta a progressão da perda.

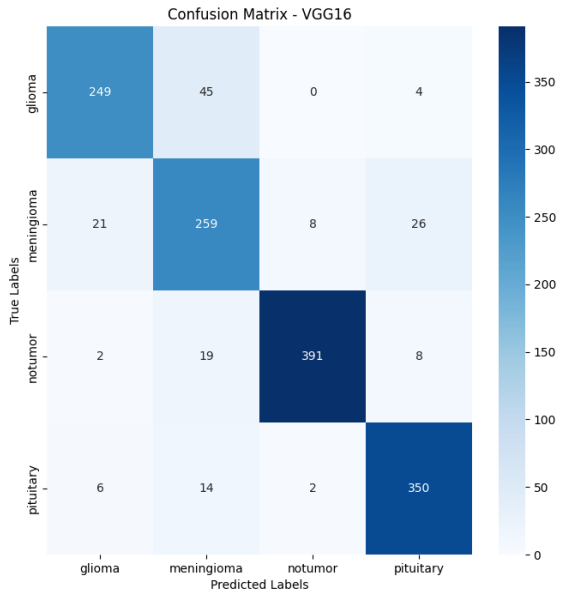

Fig. 3. Matriz de confusão para o modelo VGG16.

3) InceptionV3: Com uma acurácia de 90.10%, o modelo InceptionV3 destacou-se como o mais preciso entre os modelos testados. Os valores de precisão, revocação e F1-Score foram 90.30%, 90.10% e 90.13%, respectivamente. As Figuras 5 e 6 exibem a matriz de confusão e a evolução da perda.

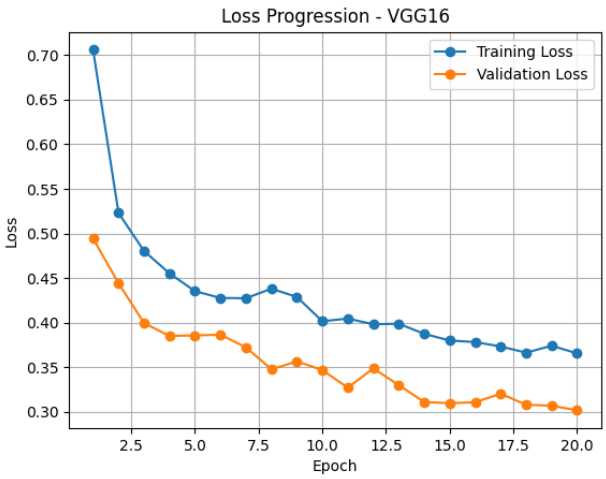

Fig. 4. Progressão da perda durante o treinamento do modelo VGG16.

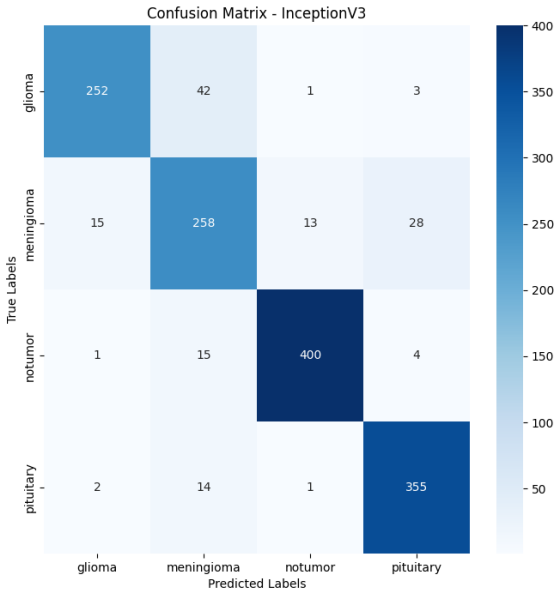

Fig. 5. Matriz de confusão para o modelo InceptionV3.

4) DenseNet: O modelo DenseNet apresentou uma acurácia de 89.53%, com precisão e revocação de 89.43% e F1-Score de 89.43%. Os resultados visuais estão dispostos nas Figuras 11 e 12.

5) MobileNetV2: Finalmente, o modelo MobileNetV2 alcançou uma acurácia de 89.74%, acompanhado de precisão e revocação de 89.74% e F1-Score de 89.73%. As Figuras 9 e 10 mostram os resultados graficamente.

B. Discussão

Os resultados indicam que todos os modelos testados demonstraram desempenho robusto, com acurácias superiores

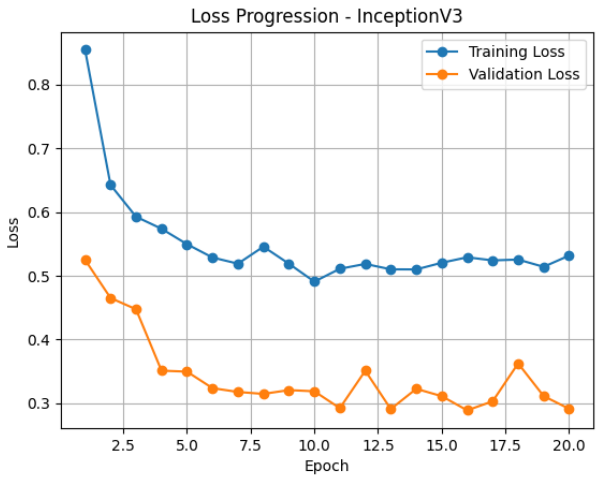

Fig. 6. Progressão da perda durante o treinamento do modelo InceptionV3.

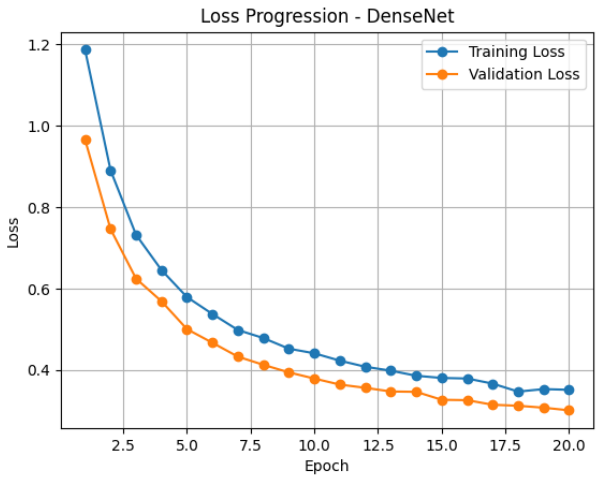

Fig. 8. Progressão da perda durante o treinamento do modelo DenseNet.

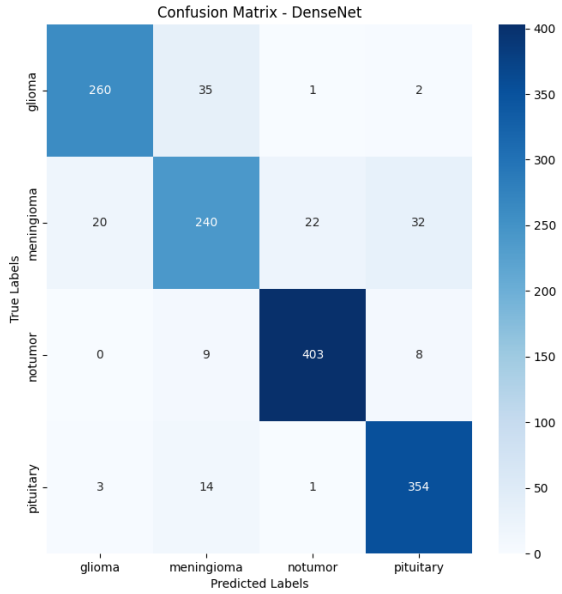

Fig. 7. Matriz de confusão para o modelo DenseNet.

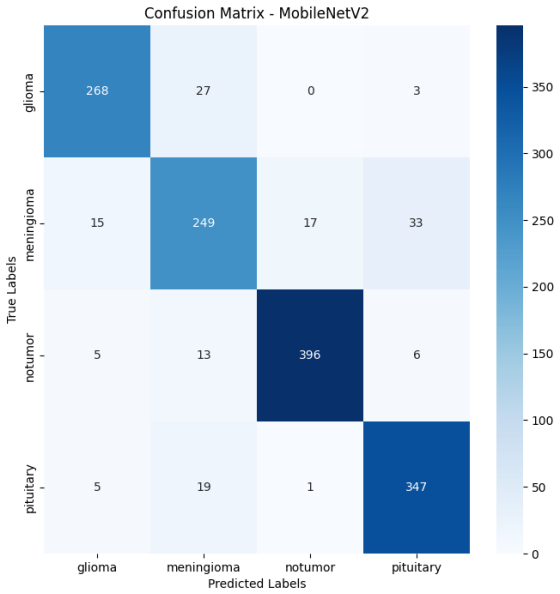

Fig. 9. Matriz de confusão para o modelo MobileNetV2.

a 87%. Entre eles, o modelo InceptionV3 se destacou como o mais eficaz, alcançando uma acurácia de 90.10%. Este resultado pode ser atribuído à sua arquitetura possivelmente mais apropriada para o caso de estudo.

O modelo VGG16 também apresentou um desempenho consistente, com uma acurácia de 88.96%, enquanto o DenseNet e o MobileNetV2 alcançaram resultados ligeiramente superiores a 89%. O ResNet18 manteve um desempenho respeitável de 87.82%.

Os gráficos de progressão da perda mostram que todos os modelos convergiram eficientemente dentro das 20 épocas estipuladas, sem sinais de overfitting ou subutilização de recursos, possivelmente também em influência da otimização de hiperparâmetros realizada.

V. Conclusão

Neste trabalho, foi realizado um estudo comparativo de cinco arquiteturas de redes neurais convolucionais para classificação de imagens. Após a otimização de hiperparâmetros e treinamento, todos os modelos apresentaram desempenho satisfatório, destacando-se o InceptionV3 como o mais eficiente, com acurácia de 90.10%.

Os resultados obtidos demonstram a relevância da escolha adequada de hiperparâmetros e da arquitetura do modelo

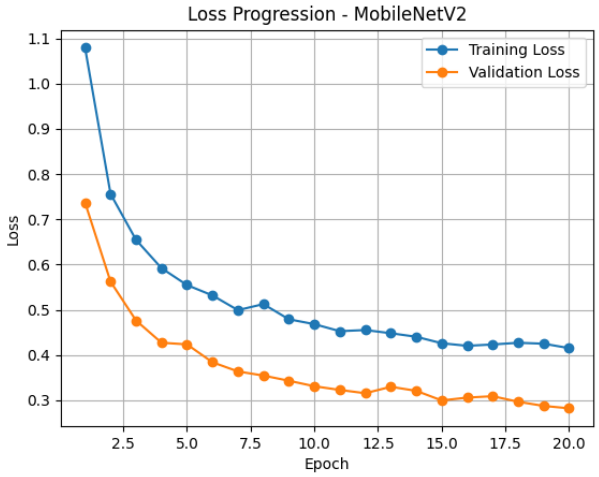

Fig. 10. Progressão da perda durante o treinamento do modelo MobileNetV2.

para aplicações específicas. Como trabalhos futuros, sugere-se explorar estratégias adicionais de data augmentation e regularização, bem como testar as redes em um conjunto de dados maior e mais variado para avaliar sua escalabilidade.

Este estudo reforça a importância de combinações adequadas de arquiteturas e técnicas de treinamento para obter resultados de alta qualidade em tarefas de classificação de imagens.

É importante salientar o fato de que com mais épocas de treinamento e de otimização de hiperparâmetros é muito provável que se obtenha resultados significativamente melhores, que foram limitados pelo tempo e poder de processamento no ambiente em questão.

References

[1] Pathak, Krishna, et al. "Classification of brain tumor using convolutional neural network." 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA). IEEE, 2019.

[2] Vaishnav, P. S. S., Singh, B. (2024, January). Brain Tumor Segmentation and Classification Using Deep Learning. In International Conference on Multi-Strategy Learning Environment (pp. 129-142). Singapore: Springer Nature Singapore.

[3] AYADI, Wadhah et al. Deep CNN for brain tumor classification. Neural processing letters, v. 53, p. 671-700, 2021.

[4] Srinivas, Chetana, et al. "Deep transfer learning approaches in performance analysis of brain tumor classification using MRI images." Journal of Healthcare Engineering 2022.1 (2022): 3264367.

[5] Habibzadeh, Mehdi, et al. "Automatic white blood cell classification using pre-trained deep learning models: Resnet and inception." Tenth international conference on machine vision (ICMV 2017). Vol. 10696. SPIE, 2018.

[6] Sharma, Shagun, and Kalpna Guleria. "Deep learning models for image classification: comparison and applications." 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE). IEEE, 2022.

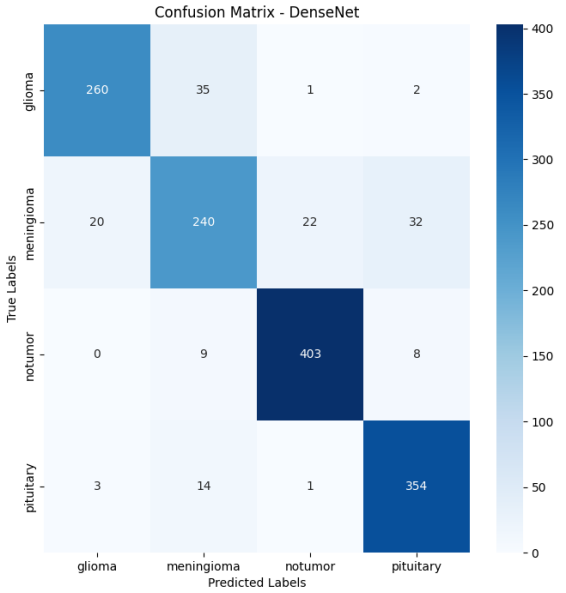

Fig. 11. Matriz de confusão para o modelo DenseNet.

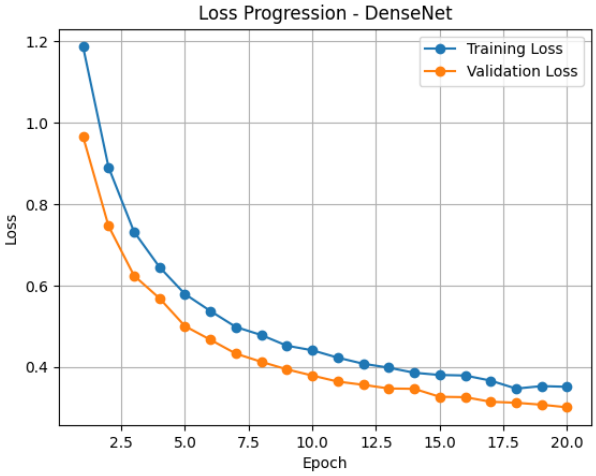

Fig. 12. Progressão da perda durante o treinamento do modelo DenseNet.