Relatório Aprendizado Profundo

Resumo

Este trabalho apresenta o desenvolvimento de um sistema de classificação de imagens voltado para a avaliação do grau de estenose nasal em cães braquicefálicos, com foco em análise automatizada por aprendizagem profunda. O sistema foi implementado em Python utilizando bibliotecas como PyTorch, Torchvision, PIL e Matplotlib, permitindo o treinamento e a avaliação de múltiplas arquiteturas de redes neurais profundas, incluindo ResNet18, MobileNetV2 e EfficientNet-B0, adaptadas para o número de classes do conjunto de dados.

O conjunto de imagens utilizado foi obtido e anotado a partir de um projeto que participei, contendo imagens anotadas quanto ao grau de estenose, divididas em classes “mild” e “severe”. O software permite o ajuste de hiperparâmetros essenciais, como número de épocas, tamanho do lote (batch size), taxa de aprendizado (learning rate), momento do otimizador, e tolerância para parada antecipada, possibilitando a realização de experimentos para avaliar o impacto dessas variações na performance das redes.

O sistema realiza pré-processamento das imagens, incluindo redimensionamento, normalização e aumentos de dados (flip horizontal e pequenas rotações), garantindo maior robustez do modelo. Durante o treinamento, métricas como loss médio, precisão, recall e F1-score são calculadas e salvas, além da geração de matriz de confusão e curvas de aprendizado, permitindo análise detalhada do desempenho de cada arquitetura.

Os experimentos realizados demonstraram que o sistema é capaz de diferenciar com boa precisão os graus de estenose em braquicefálicos, oferecendo uma ferramenta prática e reprodutível para pesquisas veterinárias. A proposta contribui para a área ao fornecer um pipeline completo de pré-processamento, treinamento, avaliação e visualização de resultados de redes neurais profundas, facilitando tanto a análise de desempenho quanto a exploração de diferentes arquiteturas e configurações de hiperparâmetros.

Introdução

A classificação automática de imagens é uma área da inteligência artificial que utiliza redes neurais profundas para identificar e categorizar padrões visuais em dados complexos. Esses modelos são capazes de extrair características relevantes diretamente das imagens, permitindo aplicações em diversas áreas, como medicina, biologia, engenharia e análise de dados. Em particular, o reconhecimento de padrões visuais em imagens médicas tem se mostrado essencial para apoiar diagnósticos e pesquisas clínicas, oferecendo precisão e consistência superiores à análise manual. Apesar do grande potencial das redes neurais, a construção e avaliação de modelos de classificação ainda apresentam desafios, como a escolha da arquitetura adequada, a definição de hiperparâmetros e o tratamento de conjuntos de dados limitados ou desbalanceados. Esses fatores influenciam diretamente o desempenho do modelo e a confiabilidade das previsões.

Dessa forma, este trabalho visa desenvolver um sistema de classificação de imagens para avaliação do grau de estenose nasal em cães braquicefálicos, permitindo o treinamento e teste de diferentes arquiteturas profundas, como ResNet18, MobileNetV2 e EfficientNet-B0. O sistema oferece ferramentas para ajuste de hiperparâmetros, visualização de métricas de desempenho, geração de matrizes de confusão e curvas de aprendizado, proporcionando uma análise detalhada do comportamento do modelo. A proposta tem como objetivo principal facilitar a avaliação automática do grau de estenose, oferecendo uma ferramenta prática, precisa e reprodutível para pesquisas veterinárias.

Estrutura do Software

1. Escolha das Bibliotecas

Para viabilizar o desenvolvimento do sistema de classificação de imagens, foram selecionadas bibliotecas que oferecem suporte eficiente ao processamento de dados, construção de redes neurais profundas, manipulação de imagens e visualização de resultados. A linguagem Python, amplamente utilizada na área acadêmica e científica, fornece um ecossistema robusto para desenvolvimento de aplicações de aprendizado profundo e análise de dados.

A biblioteca PyTorch foi escolhida como framework principal para a construção, treinamento e avaliação das redes neurais profundas. Sua flexibilidade permite adaptar arquiteturas pré-treinadas, como ResNet18, MobileNetV2 e EfficientNet-B0, e realizar ajustes finos nas camadas finais para atender ao número de classes do conjunto de dados. Além disso, PyTorch possibilita a utilização de GPUs para acelerar o treinamento e oferece recursos para implementação de técnicas como congelamento de camadas, cálculo de loss e métricas de desempenho. A biblioteca Torchvision foi utilizada para o acesso a modelos pré-treinados, bem como para o carregamento e pré-processamento de imagens, incluindo transformações como redimensionamento, normalização, flips horizontais e rotações aleatórias, garantindo maior robustez do modelo diante da variabilidade do conjunto de dados.

Para manipulação e carregamento de imagens, a biblioteca PIL (Python Imaging Library) foi utilizada, permitindo abrir, converter e transformar imagens em tensores compatíveis com as redes neurais. Já a biblioteca Matplotlib foi empregada na visualização de métricas de treinamento, geração de curvas de loss e acurácia e plotagem da matriz de confusão, facilitando a análise do desempenho dos modelos treinados.

2. Interação com o Usuário e Visualização de Resultados

A interação do usuário com o sistema é realizada principalmente por meio de scripts Python e diretórios de entrada/saída, o que permite controle completo sobre o treinamento, validação e teste dos modelos. O usuário pode definir parâmetros do experimento, como a arquitetura da rede, número de épocas, tamanho do lote, taxa de aprendizado e diretórios de dados, editando o arquivo de configuração de hiperparâmetros (hyperparameters.py). Embora o sistema não possua interface gráfica, foram implementadas funcionalidades para visualização e análise de resultados, facilitando a compreensão do desempenho dos modelos. Entre essas funcionalidades estão:

- Métricas de desempenho: cálculo e salvamento de loss médio, precisão, recall e F1-score, tanto globalmente quanto por classe, permitindo avaliação detalhada do modelo.

- Matriz de confusão: geração de heatmaps para análise das classificações corretas e incorretas, facilitando a identificação de padrões de erro.

- Curvas de treinamento e validação: plotagem de gráficos de loss e acurácia ao longo das épocas, auxiliando na análise da convergência do modelo e no ajuste de hiperparâmetros.

- Checkpoint do modelo: salvamento automático do melhor modelo durante o treinamento, permitindo que o usuário retome experimentos ou utilize o modelo treinado para classificar novas imagens.

3. Organização de Diretórios e Arquivos

O software foi organizado de forma modular, facilitando a manutenção, reutilização e execução dos experimentos. A estrutura de diretórios e arquivos é a seguinte:

- **data/**: contém o conjunto de imagens utilizado para treinamento, validação e teste. As imagens estão organizadas em subpastas separadas por conjunto (train/, val/, test/) e por classe, garantindo compatibilidade com o ImageFolder do PyTorch.

- **results/**: pasta onde são salvos os resultados de cada rede treinada. Cada execução gera métricas de classificação geral e por classe, matriz de confusão e gráficos das curvas de treinamento (loss e acurácia), permitindo análise detalhada do desempenho do modelo.

- **runs/**: armazena as execuções realizadas, possibilitando rastrear diferentes experimentos e comparar resultados entre arquiteturas e combinações de hiperparâmetros.

- **src/**: contém os módulos principais e auxiliares do software, organizados da seguinte forma:
    - **checkpoints/**: armazena os pesos salvos durante o treinamento das redes, garantindo que os melhores modelos possam ser reutilizados ou avaliados posteriormente.

    - **utils/**: inclui scripts auxiliares para pré-processamento de dados, plotagem de métricas e salvamento de métricas, como:

        - plotMetricas.py: responsável pela geração dos gráficos de loss e acurácia.

        - preparaConjuntoDados.py: script de preparação e organização do conjunto de imagens, caso necessário.

        - salvaMetricas.py: calcula e salva métricas de desempenho (loss, precisão, recall, F1-score) e gera a matriz de confusão.

    - **architectures.py**: define funções para carregar e adaptar arquiteturas pré-treinadas (ResNet18, MobileNetV2, EfficientNet-B0, etc.) para o número de classes do dataset.

    - **hyperparameters.py**: arquivo de configuração centralizada de hiperparâmetros de treinamento, validação e processamento de dados.

    - **main.py**: script principal que organiza o fluxo completo do sistema, incluindo carregamento de dados, seleção de arquitetura, treinamento, validação, salvamento de checkpoints e geração de resultados.

Essa organização permite que o sistema seja executado de forma modular e reprodutível, facilitando a execução de diferentes experimentos e a análise comparativa entre arquiteturas e configurações de hiperparâmetros.

4. Pipeline de Execução / Fluxo do Sistema

O sistema de classificação de imagens segue um fluxo estruturado, permitindo que os dados sejam processados e avaliados de forma organizada e reprodutível. O pipeline de execução pode ser descrito em etapas:

1. **Pré-processamento de imagens**: todas as imagens são redimensionadas para o tamanho definido nos hiperparâmetros, normalizadas e, no caso do conjunto de treino, submetidas a transformações adicionais, como flips horizontais e pequenas rotações, aumentando a robustez do modelo.

2. **Carregamento de datasets**: as imagens de treino, validação e teste são carregadas utilizando o DataLoader do PyTorch, que organiza os dados em batches e permite shuffle automático, garantindo eficiência no processamento durante o treinamento.

3. **Seleção da arquitetura da rede**: o usuário define a arquitetura desejada (ResNet18, MobileNetV2, EfficientNet-B0 ou SqueezeNet), que é adaptada para o número de classes do dataset através da modificação das camadas finais.

4. **Treinamento do modelo**: o modelo é treinado com cálculo do **loss** por meio da função de perda cross-entropy e atualização dos pesos via otimização (SGD ou outro otimizador configurado nos hiperparâmetros). O treinamento inclui técnicas de **congelamento de camadas** e **early stopping**, garantindo que o modelo não seja sobreajustado.

5. **Avaliação e validação**: durante o treinamento, o desempenho é monitorado no conjunto de validação, com cálculo de métricas como loss, acurácia, precisão, recall e F1-score. O modelo com melhor desempenho é salvo como checkpoint, permitindo retomada de experimentos.

6. **Geração de resultados**: ao final do treinamento, são geradas métricas globais e por classe, matriz de confusão e gráficos de curvas de aprendizado, que são armazenados na pasta results/ para análise comparativa entre diferentes arquiteturas e configurações.

Esse pipeline garante que todas as etapas do experimento, desde o pré-processamento até a análise de resultados, sejam conduzidas de forma clara, estruturada e facilmente reprodutível.

5. Modularidade e Reusabilidade

O software foi projetado de forma modular, permitindo que diferentes componentes sejam reutilizados ou substituídos sem impactar todo o sistema. Entre os principais aspectos de modularidade destacam-se:

- **Arquiteturas intercambiáveis**: é possível alternar facilmente entre ResNet18, MobileNetV2, EfficientNet-B0 e SqueezeNet, adaptando apenas a camada final para o número de classes do dataset.

- **Configuração centralizada de hiperparâmetros**: todos os parâmetros de treinamento e processamento de dados estão concentrados no arquivo hyperparameters.py, facilitando ajustes e experimentos sem necessidade de alterar o código principal.

- **Módulos auxiliares reutilizáveis**: scripts como salvaMetricas.py e plotMetricas.py podem ser usados em outros projetos de classificação, permitindo cálculos de métricas e geração de gráficos de forma independente.

- **Execução de múltiplos experimentos**: a pasta runs/ permite registrar diferentes execuções, mantendo a rastreabilidade e possibilitando comparações entre redes, hiperparâmetros e transformações aplicadas.

Essa modularidade e reusabilidade tornam o sistema flexível, escalável e adequado para experimentos comparativos em diferentes cenários de classificação de imagens.

Resultados

Foram utilizadas três arquiteturas profundas diferentes:

- ResnetV18
- MobileNetV2
- EfficientNet-B0

Como parte do experimento, foram utilizadas 3 arquiteturas diferentes com ao menos 3 diferentes hiperparâmetros:

Experimento 1 – ResNetV18:

Hiperparâmetros utilizados:

- Épocas: 50
- Paciência (early stopping):30
- Learning rating: 0.0001

Resultados:

Métricas de classificação

- Loss medio: 0.5506
- Precision: 0.7647
- Recall: 0.7647
- F1-score: 0.7647

Métricas por classe:

CLASSE	PRECISAO	REVOCAÇAO	F1	SUPORTE
mild	0.7500	0.7500	0.7500	8
severe	0.7778	0.7778	0.7778	9

Análise dos resultados:

O modelo ResNet18 apresentou o melhor desempenho geral entre as três arquiteturas avaliadas. O loss médio de 0.5506 indica boa convergência, mostrando que o modelo conseguiu ajustar adequadamente os pesos durante o treinamento. As métricas globais (Precision, Recall e F1 = 0.7647) revelam ótimo equilíbrio entre as classes, o que é reforçado pelos resultados por classe: tanto *mild* quanto *severe* apresentam valores muito próximos (diferenças inferiores a 0.03). Isso demonstra que a ResNet18 não favorece nenhuma categoria e consegue capturar bem os padrões presentes em ambas. O uso de early stopping com paciência de 30 épocas foi adequado, permitindo que o modelo treinasse o suficiente sem sobreajustar. Esses resultados indicam que a ResNet18 é robusta e se adapta muito bem ao dataset utilizado.

Experimento 2- MobilleNetV2

Hiperparâmetros utilizados:

- Épocas: 30
- Paciência (early stopping):10
- Learning rating: 0.01

Resultados:

Métricas de classificação

- Loss medio: 0.7870
- Precision: 0.7684
- Recall: 0.5882
- F1-score: 0.4858

Métricas por classe:

CLASSE	PRECISAO	REVOCAÇAO	F1	SUPORTE
mild	1.0000	0.1250	0.2222	8
severe	0.5625	1.0000	0.7200	9

Análise dos resultados:

A MobileNetV2 apresentou dificuldades significativas de generalização neste experimento. Embora a precisão global (0.7684) pareça alta à primeira vista, isso não reflete um bom desempenho real, já que o recall global foi extremamente baixo (0.5882), resultando em um F1-score muito reduzido (0.4858).

A análise por classe revela um problema claro de desequilíbrio na sensibilidade:

- Para a classe *mild*, o modelo alcançou 100% de precisão, mas apenas 12,5% de recall, indicando que a rede praticamente nunca reconhece as imagens dessa classe — apesar de não errar quando o faz.

- Para a classe *severe*, o comportamento é o oposto: recall 100%, mas precisão apenas 56,25%, indicando forte tendência de classificar muitas amostras como *severe*, mesmo quando incorreto.

Esse padrão indica um viés forte na decisão do modelo, possivelmente causado por:

1. Learning rate muito alto (0.01), que dificulta a convergência para soluções estáveis.
2. Paciência reduzida (10), fazendo o early stopping interromper o treinamento antes que a rede aprendesse adequadamente.
3. A natureza mais leve da MobileNetV2, que costuma exigir hiperparâmetros mais sensíveis.

Em resumo, o modelo não conseguiu aprender fronteiras de decisão adequadas para o dataset, apresentando comportamento desequilibrado e baixa capacidade de generalização.

Experimento 3 – EfficientNetB0

Hiperparâmetros utilizados:

- Épocas: 100
- Paciência (early stopping):40
- Learning rating: 0.0001

Resultados:

Métricas de classificação

- Loss medio: 0.6358
- Precision: 0.6787
- Recall: 0.6471
- F1-score: 0.6203

Métricas por classe:

CLASSE	PRECISAO	REVOCAÇAO	F1	SUPORTE
mild	0.7500	0.3750	0.5000	8
severe	0.6154	0.88889	0.7273	9

Análise dos resultados:

A EfficientNet-B0 apresentou desempenho intermediário entre as três arquiteturas. O loss médio de 0.6358 indica uma convergência razoável, mas não tão consistente quanto a ResNet18. As métricas globais sugerem um modelo equilibrado (F1 = 0.6203), mas a análise por classe revela comportamentos distintos:

- Para a classe *mild*, o recall foi baixo (0.3750), indicando dificuldade em identificar corretamente casos dessa categoria.

- Para a classe *severe*, o recall foi bastante alto (0.8889), o que indica tendência do modelo em reconhecer melhor esta classe.

Apesar disso, a precisão da classe *mild* foi boa (0.75), o que mostra que quando o modelo identifica corretamente, a chance de acerto é alta — porém ele erra por não detectar muitas instâncias desta classe. Esse comportamento sugere que a EfficientNet aprendeu padrões relevantes, mas ainda apresenta viés em direção à classe *severe*. A arquitetura pode estar exigindo:

- mais dados
- ajustes de data augmentation
- ou fine-tuning mais agressivo

Mesmo assim, o desempenho foi consistentemente superior ao da MobileNetV2 e demonstrou boa capacidade de generalização.

Discussão

A comparação entre as três arquiteturas profundas — ResNet18, MobileNetV2 e EfficientNet-B0 — evidencia diferenças importantes na capacidade de generalização, estabilidade do treinamento e sensibilidade às escolhas de hiperparâmetros. Entre os modelos testados, a ResNet18 apresentou o melhor desempenho global, demonstrando maior robustez e equilíbrio na classificação das duas categorias do dataset. O loss relativamente baixo (0.5506) e o F1-score de 0.7647 indicam que o modelo conseguiu aprender representações discriminativas de forma consistente. Além disso, o equilíbrio entre as métricas das classes mild e severe mostra que o modelo não sofreu com viés de classificação, algo essencial em tarefas onde qualquer classe tem igual importância.

Por outro lado, a MobileNetV2 apresentou o pior desempenho entre as arquiteturas avaliadas. Apesar da precisão global relativamente alta, o recall muito baixo e o F1-score reduzido (0.4858) evidenciam problemas sérios de generalização. A análise por classe mostra um padrão de comportamento enviesado: a rede praticamente ignorou a classe mild, classificando quase todas as instâncias como severe. Essa tendência pode estar relacionada ao learning rate excessivamente alto (0.01), que provavelmente impediu a convergência adequada, além da paciência reduzida no early stopping, que interrompeu o treinamento antes que o modelo pudesse estabilizar os pesos. Como uma arquitetura leve, a MobileNetV2 costuma exigir maior cuidado com hiperparâmetros e fine-tuning, o que não ocorreu neste caso.

A EfficientNet-B0 apresentou desempenho intermediário, superando a MobileNetV2, mas ainda abaixo da ResNet18. O loss (0.6358) e o F1-score (0.6203) revelam que o modelo conseguiu aprender padrões relevantes, mas ainda mostra dificuldade em reconhecer amostras da classe mild, como indicado pelo recall reduzido (0.375). Apesar disso, a arquitetura demonstrou certa estabilidade ao apresentar boa precisão na mesma classe e um recall elevado para severe. Isso sugere que a EfficientNet-B0 está capturando características discriminativas, mas ainda depende de ajustes adicionais, como aumento de dados, técnicas de data augmentation mais agressivas ou um fine-tuning mais profundo das camadas iniciais.

De modo geral, os resultados indicam que arquiteturas mais profundas e robustas como a ResNet18 se adaptaram melhor ao dataset utilizado, oferecendo melhor equilíbrio entre precisão e sensibilidade. Já modelos mais leves, como a MobileNetV2, foram mais sensíveis a configurações inadequadas de hiperparâmetros e apresentaram forte viés na classificação. A EfficientNet-B0 demonstrou bom potencial, mas ainda não superou a consistência da ResNet18. Esses achados reforçam a importância da escolha adequada de hiperparâmetros e da necessidade de validação cuidadosa ao trabalhar com modelos de diferentes complexidades, especialmente em datasets com classes próximas ou pouco representadas.

Conclusão

O desenvolvimento do sistema de classificação do grau de estenose das narinas atingiu com sucesso os objetivos propostos, permitindo avaliar diferentes arquiteturas profundas e compreender seu desempenho diante de um dataset desafiador. A ResNet18 mostrou-se a solução mais estável e equilibrada, evidenciando boa capacidade de generalização e classificações consistentes entre as classes. A EfficientNet-B0 apresentou desempenho intermediário, revelando potencial para melhorias com ajustes adicionais de treinamento. Já a MobileNetV2 demonstrou limitações importantes, reforçando a necessidade de uma configuração mais sensível de hiperparâmetros. De forma geral, os resultados obtidos demonstram a viabilidade do uso de redes neurais profundas na classificação do grau de estenose nasal, oferecendo uma base sólida para aprimoramentos futuros, como expansão do conjunto de dados, novas estratégias de pré-processamento e exploração de arquiteturas ainda mais robustas.

Tempo gasto

Estima-se que o desenvolvimento do projeto foi realizado em cerca de 1 semanas de trabalho, divididas entre pesquisa, desenvolvimento, testes e documentação.
Distribuição do Tempo:

- Pesquisa e Aprendizado: ~1 horas
- Construção do Código: ~6 horas
- Testes e Debugging: ~30 minutos
- Documentação: ~20minutos

Demonstração do software

- Assista ao vídeo para ver o funcionamento do software:

https://youtu.be/Hcv2ihhjPck

Referências

- **PYTHON SOFTWARE FOUNDATION. Tkinter — Python interface to Tcl/Tk.**
Disponível em: https://docs.python.org/3/library/tkinter.html

- **Torchvision — Datasets, ImageFolder e Transforms.**
Disponível em: https://pytorch.org/vision/stable/transforms.html

- **Matplotlib — Documentação Oficial.**
Disponível em: https://matplotlib.org/stable/contents.html

- **Redes Convolucionais Profundas**
Disponível em: https://www.youtube.com/watch?v=5zQgE94ky8w&authuser=5