

Trabalho Visão Computacional

Gabriel Tavares
Luana Lauschner

2026

1 Estratégias de Implementação

A implementação do trabalho foi realizada em Python, utilizando a biblioteca PyTorch, escolhida por sua flexibilidade na construção de redes neurais convolucionais e pelo suporte a modelos pré-treinados amplamente utilizados na literatura. O código foi organizado de forma modular, permitindo a realização de múltiplos experimentos com diferentes arquiteturas e configurações de hiperparâmetros. O trabalho foi desenvolvido e versionado em um repositório GitHub¹.

1.1 Implementação da VGG16

Na primeira etapa, foi implementada do zero a arquitetura VGG16, originalmente proposta por Simonyan e Zisserman (**SIMONYAN; ZISSEMAN, 2014**), por meio da composição explícita de blocos convolucionais contendo camadas de convolução 3×3 , funções de ativação ReLU e operações de *max pooling*. Opcionalmente, foram incorporadas camadas de Batch Normalization e dropout, permitindo avaliar o impacto dessas técnicas na regularização e estabilidade do treinamento.

A parte classificadora da rede foi composta por camadas totalmente conectadas, com funções de ativação ReLU e taxas de dropout configuráveis. Todas as variações arquiteturais foram controladas por parâmetros globais, facilitando a comparação entre diferentes configurações.

1.2 Transfer Learning

Na segunda etapa, foi aplicada a técnica de *transfer learning* utilizando os modelos VGG16, ResNet50 e DenseNet121 pré-treinados no dataset ImageNet (**DENG et al., 2009**). A arquitetura ResNet50 emprega aprendizado residual profundo (**HE et al., 2016**), enquanto a DenseNet121 utiliza conexões densas entre camadas para favorecer reutilização de características e melhor propagação de gradientes (**HUANG et al., 2017**). Para cada modelo, a camada de classificação original foi substituída por uma nova camada compatível com as 10 classes do CIFAR-10 (**KRIZHEVSKY; HINTON, 2009**).

¹ <<https://github.com/gabrielctavares/DCC197-VisaoComputacional>>

Foram realizados experimentos com congelamento total do *backbone* e com o descongelamento progressivo das últimas camadas convolucionais, visando analisar o efeito do *fine-tuning* no desempenho final dos modelos.

1.3 Pré-processamento e Treinamento

As imagens foram normalizadas utilizando estatísticas compatíveis com as do conjunto de dados. No conjunto de treinamento, técnicas de data augmentation foram aplicadas em alguns experimentos, incluindo rotações aleatórias e espelhamento horizontal, com o objetivo de aumentar a capacidade de generalização das redes neurais.

O treinamento foi realizado utilizando o otimizador Adam e a função de perda *Cross-Entropy*. O desempenho dos modelos foi avaliado no conjunto de teste por meio da acurácia global e da análise por classe, incluindo a geração de matrizes de confusão, conforme solicitado no enunciado do trabalho.

2 Descrição dos Experimentos

Esta seção descreve a configuração computacional utilizada nos experimentos, os parâmetros avaliados, os resultados obtidos para as diferentes arquiteturas e configurações testadas, bem como a análise das tabelas, matrizes de confusão e demais evidências experimentais.

2.1 Ambiente de Execução

Todos os experimentos foram executados em ambiente local, com o código versificado e disponibilizado em um repositório GitHub. O desenvolvimento foi realizado em Python, utilizando a biblioteca PyTorch para a implementação, treinamento e avaliação das redes neurais convolucionais. A execução dos experimentos foi conduzida em uma máquina dedicada, cujas especificações de hardware e software são apresentadas na Tabela 1. Esse ambiente possibilitou a execução controlada e reproduzível de arquiteturas profundas, como VGG16, ResNet50 e DenseNet121, dentro das limitações computacionais do contexto acadêmico.

Componente	Especificação
Sistema Operacional	Ubuntu 20.04 LTS
GPU	NVIDIA GTX 1060 (6 GB)
CPU	Intel Core i7-870
Memória RAM	16 GB
Framework	PyTorch 1.12 + CUDA 11.3

Tabela 1 – Especificações do ambiente utilizado nos experimentos.

2.2 Configuração Geral dos Experimentos

Os experimentos foram conduzidos utilizando o dataset CIFAR-10, composto por 10 classes balanceadas. Como métrica principal de avaliação foi utilizada a acurácia no conjunto de teste, complementada pela análise de acurácia por classe e por meio de matrizes de confusão normalizadas.

Ao longo dos experimentos, foram sistematicamente variados os seguintes parâmetros:

- taxa de aprendizado;
- tamanho do *batch*;
- número de épocas;
- uso de *weight decay*;
- uso ou não de Batch Normalization;
- uso ou não de dropout;
- uso de *data augmentation*;
- estratégia de congelamento ou descongelamento de camadas nos modelos pré-treinados.

2.3 Resultados

A análise dos resultados segue práticas consolidadas na literatura de aprendizado profundo, considerando tanto métricas quantitativas quanto a avaliação qualitativa por meio de matrizes de confusão, conforme recomendado em estudos clássicos de classificação visual em larga escala. Nesta subseção são apresentados os resultados obtidos nos experimentos, contemplando a acurácia global, o desempenho por classe e os padrões de erro observados, de modo a facilitar a comparação entre as abordagens avaliadas por meio de tabelas comparativas entre arquiteturas e figuras com matrizes de confusão selecionadas.

A Tabela 2 apresenta uma visão geral das melhores configurações obtidas para cada arquitetura avaliada, permitindo uma comparação direta entre a VGG16 implementada do zero e os modelos que utilizam *transfer learning*.

Modelo	Configuração (ID)	Acc (%)	Img	LR	WD
VGG16 (do zero)	vgg16_full	90.12	224	0.0001	0.0005
VGG16 (pré-treino)	vgg16_no_freeze	91.60	32	0.0001	0.0005
ResNet50 (pré-treino)	resnet50_wd0	87.96	32	0.0001	0
DenseNet121 (pré-treino)	densenet121_no_freeze	88.11	32	0.0001	0.0005

Tabela 2 – Comparação entre as melhores configurações por arquitetura. (Img: tamanho da imagem; LR: *learning rate*; WD: *weight decay*).

2.3.1 VGG16

Os resultados obtidos com a VGG16 implementada do zero evidenciam o impacto direto das escolhas arquiteturais e de hiperparâmetros. O modelo *baseline* apresentou acurácia inferior a 50%, com forte concentração de erros em classes visualmente semelhantes, como *cat* e *dog*, conforme evidenciado pela matriz de confusão da Figura 1a.

A remoção da Batch Normalization levou ao colapso completo do treinamento, com previsões degeneradas, nas quais todas as amostras passaram a ser classificadas em uma única classe. Esse comportamento pode ser observado na Figura 1b, evidenciando a falha do processo de aprendizado na ausência desse mecanismo de normalização.

A inclusão de regularização por *weight decay* promoveu uma melhora substancial no desempenho, elevando a acurácia para valores próximos a 85%. Esse efeito é refletido na matriz de confusão apresentada na Figura 1c, que mostra uma diagonal mais pronunciada e redução das confusões entre classes. Ajustes adicionais, como o aumento do número de épocas e variações na taxa de aprendizado, apresentaram ganhos marginais, enquanto a redução do tamanho do *batch* impactou negativamente o desempenho. Além disso, a desativação do dropout (Figura 1d) não comprometeu significativamente o desempenho em relação às configurações regularizadas, mantendo uma diagonal bem definida na matriz de confusão, embora com confusões residuais entre classes semanticamente próximas.

O uso de *data augmentation* contribuiu para uma leve melhora na generalização, como mostrado na Figura 1e. A melhor configuração da Parte 1 foi o modelo *vgg16_full*, que combinou regularização completa, *data augmentation* e aumento da dimensão de entrada para 224×224 , alcançando acurácia superior a 90% e apresentando uma matriz de confusão com diagonal dominante (Figura 1f).

2.3.2 Transfer Learning

Nos experimentos de *transfer learning*, foram avaliadas as arquiteturas VGG16, ResNet50 e DenseNet121 pré-treinadas no ImageNet. Conforme apresentado na Tabela 3, observa-se que o congelamento total do *backbone* limita significativamente o desempenho, enquanto estratégias com ajuste fino parcial ou completo apresentam resultados superiores.

Para a VGG16 pré-treinada, a configuração *no_freeze* foi a mais eficaz, superando 91% de acurácia. Esse ganho pode ser visualmente confirmado pela matriz de confusão da Figura 2b, que apresenta redução expressiva das confusões entre classes de animais quando comparada à versão com *freeze* total (Figura 2a).

Resultados semelhantes foram observados para ResNet50 e DenseNet121. Como mostrado nas Figuras 2c e 2e, as versões com *freeze* total apresentam dispersão significativa fora da diagonal principal. Em contraste, as versões com *fine-tuning* completo (Figuras 2d e 2f) exibem diagonais mais definidas e melhor separação entre classes semanticamente próximas.

Arquitetura	Configuração	Freeze backbone	Acc (%)
VGG16	vgg16_freeze	Sim	67.93
VGG16	vgg16_unfreeze_20	Sim (parcial)	90.98
VGG16	vgg16_no_freeze	Não	91.60
ResNet50	resnet50_freeze	Sim	50.52
ResNet50	resnet50_unfreeze_50	Sim (parcial)	80.52
ResNet50	resnet50_no_freeze	Não	87.94
DenseNet121	densenet121_freeze	Sim	52.42
DenseNet121	densenet121_unfreeze_30	Sim (parcial)	69.80
DenseNet121	densenet121_no_freeze	Não	88.11

Tabela 3 – Comparaçāo de estratégias de congelamento e ajuste fino (*fine-tuning*) em modelos pré-treinados.

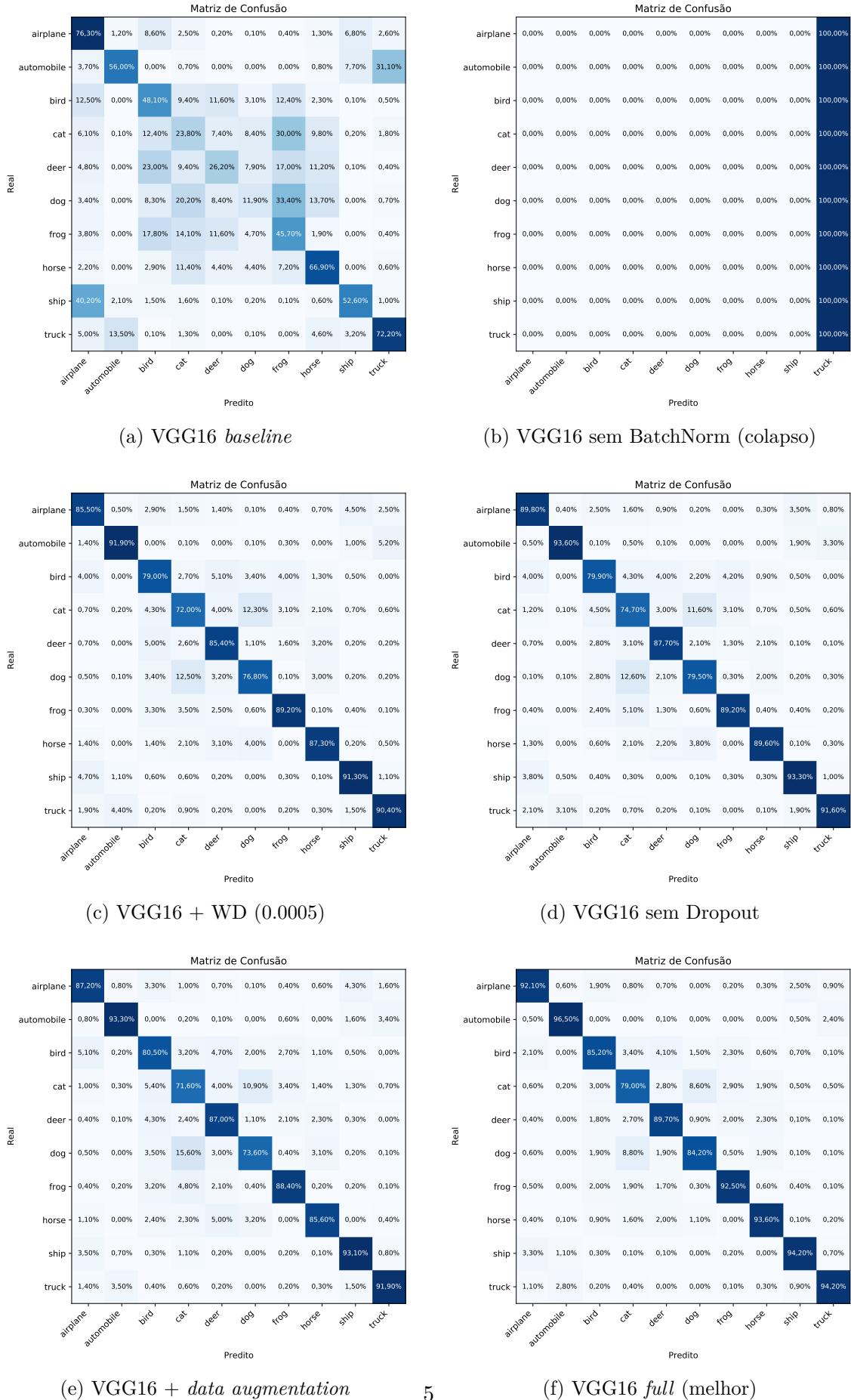


Figura 1 – Matrizes de confusão (normalizadas) para experimentos selecionados da VGG16 implementada do zero.

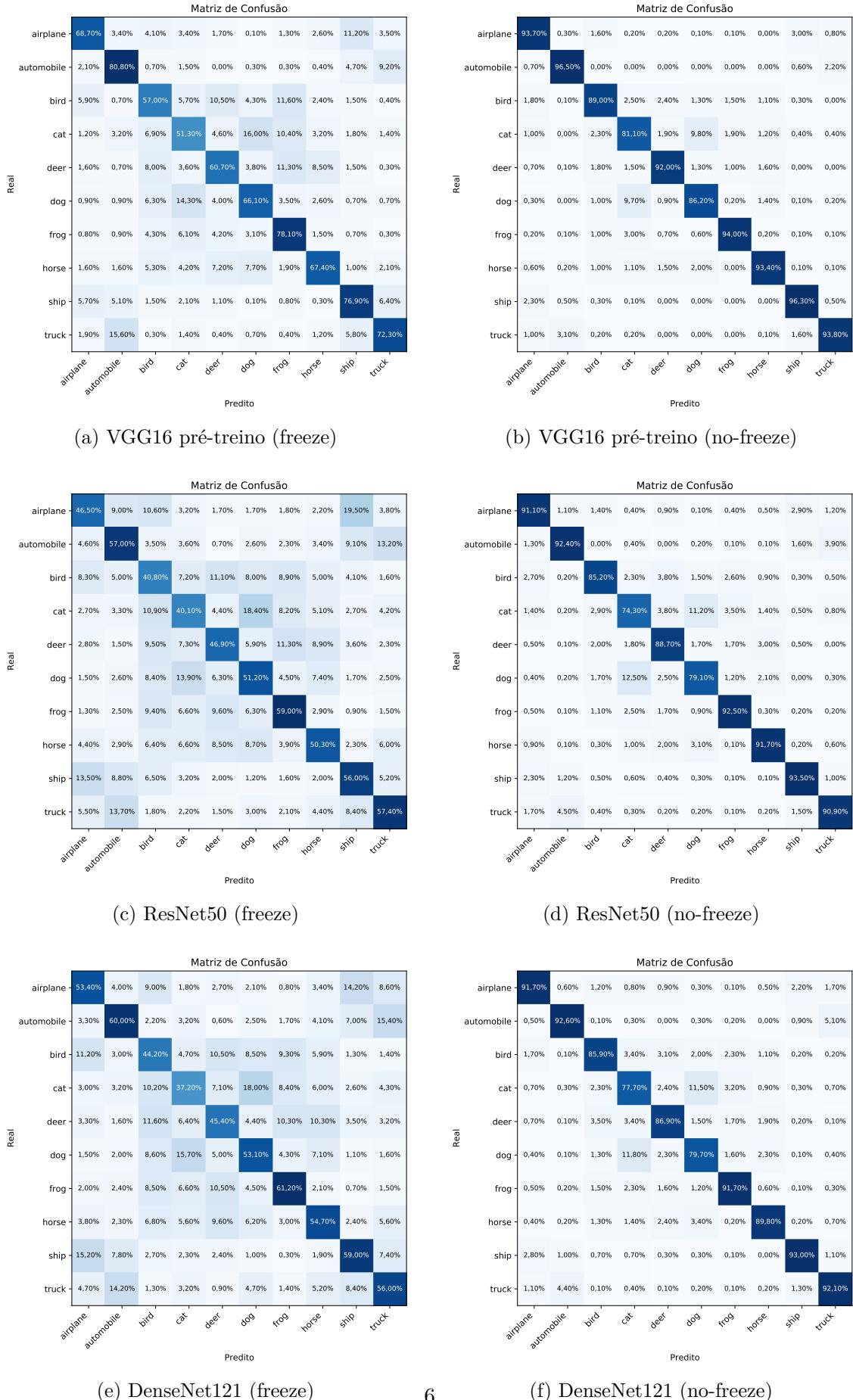


Figura 2 – Matrizes de confusão (normalizadas) para *transfer learning*, comparando *freeze* vs. *no-freeze*.

3 Conclusão

Neste trabalho foi realizada uma avaliação experimental de redes neurais convolucionais aplicadas ao problema de classificação de imagens, considerando tanto a implementação de arquiteturas do zero quanto o uso de técnicas de *transfer learning*. Foram conduzidos experimentos com a arquitetura VGG16 implementada do zero e com modelos pré-treinados amplamente utilizados na literatura, incluindo VGG16, ResNet50 e DenseNet121, avaliando diferentes estratégias de treinamento, regularização e ajuste fino.

Os resultados obtidos demonstraram de forma clara a influência das escolhas arquiteturais e dos hiperparâmetros no desempenho dos modelos. No caso da VGG16 implementada do zero, observou-se que a ausência de mecanismos fundamentais, como a Batch Normalization, inviabiliza o aprendizado, levando ao colapso do treinamento. De forma semelhante, a remoção do termo de regularização por *weight decay* resultou em comportamento instável do modelo, com forte degradação da capacidade de generalização. Nesses cenários, apesar de eventuais melhorias no ajuste aos dados de treinamento, o desempenho no conjunto de teste foi significativamente comprometido, evidenciando sobreajuste. A reintrodução do *weight decay* mostrou-se essencial para estabilizar o processo de otimização e permitir a obtenção de modelos com desempenho consistente.

Por outro lado, a combinação de regularização adequada, *data augmentation* e aumento da resolução de entrada resultou em ganhos expressivos de desempenho, culminando em acurácia superior a 90% na melhor configuração avaliada. As matrizes de confusão evidenciaram redução significativa das confusões entre classes semanticamente próximas, especialmente nas configurações mais completas, indicando melhor capacidade de generalização do modelo.

Nos experimentos de *transfer learning*, os modelos pré-treinados apresentaram desempenho consistentemente superior quando comparados às versões treinadas do zero, principalmente quando foi adotado o ajuste fino completo das camadas do *backbone*. Estratégias de congelamento total mostraram-se limitantes, enquanto o descongelamento parcial ou completo permitiu melhor adaptação dos modelos ao conjunto de dados utilizado. Entre as arquiteturas avaliadas, a VGG16 pré-treinada com ajuste fino completo apresentou o melhor desempenho global, seguida de perto pela DenseNet121, enquanto a ResNet50 apresentou desempenho competitivo, porém inferior nas configurações analisadas. Esses resultados estão alinhados com observações reportadas na literatura, nas quais arquiteturas profundas com regularização adequada e ajuste fino completo de modelos pré-treinados tendem a apresentar melhor capacidade de generalização em tarefas de classificação visual.

Como feedback pessoal, o desenvolvimento deste trabalho foi extremamente enriquecedor, pois possibilitou a consolidação prática de conceitos teóricos abordados ao longo da disciplina, especialmente no que diz respeito ao papel da regularização, do *transfer learning* e da análise qualitativa de erros. Como sugestão para trabalhos futuros, sugere-se o aprofundamento no estudo de arquiteturas baseadas em *Vision Transformers*, explorando a aplicação de mecanismos de atenção no contexto de classificação de imagens, o que pode contribuir para uma melhor compreensão das diferenças de representação e generalização entre essas abordagens.

Referências

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. *2009 IEEE conference on computer vision and pattern recognition*. Miami, FL, USA, 2009. p. 248–255. Citado na página [1](#).

HE, K. et al. Deep residual learning for image recognition. In: IEEE. *Proceedings of the IEEE conference on computer vision and pattern recognition*. Las Vegas, NV, USA, 2016. p. 770–778. Citado na página [1](#).

HUANG, G. et al. Densely connected convolutional networks. In: IEEE. *Proceedings of the IEEE conference on computer vision and pattern recognition*. Honolulu, HI, USA, 2017. p. 4700–4708. Citado na página [1](#).

KRIZHEVSKY, A.; HINTON, G. *Learning Multiple Layers of Features from Tiny Images*. Toronto, Canada, 2009. Citado na página [1](#).

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Citado na página [1](#).