Relatório Técnico: Desenvolvimento de uma CNN para Classificação de Imagens de Cães e Gatos

Edson Pimenta de Almeida

Junho de 2025

Sumário

1 Introdução

Este relatório documenta o processo de desenvolvimento de uma Rede Neural Convolucional (CNN) para resolver a tarefa de classificação binária de imagens, especificamente, para diferenciar cães e gatos. O projeto, baseado no popular dataset "Dogs vs. Cats", serve como um estudo de caso prático para a aplicação de técnicas de Deep Learning em visão computacional.

O objetivo principal foi percorrer um ciclo de vida completo de um projeto de Machine Learning, desde a obtenção e preparação dos dados até a construção, treinamento, depuração e avaliação de um modelo preditivo. Ao longo do desenvolvimento, foram encontrados e superados diversos desafios técnicos, como a incompatibilidade de frameworks, o acesso a dados e a estagnação do aprendizado do modelo, tornando a experiência rica em aprendizados práticos.

2 Metodologia

A solução foi desenvolvida utilizando a linguagem Python e a biblioteca PyTorch. A metodologia foi dividida em quatro etapas principais: preparação dos dados, definição da arquitetura do modelo, treinamento e otimização, e avaliação final.

2.1 Preparação do Ambiente e Dados

A fonte de dados inicial seria a competição "Dogs vs. Cats"do Kaggle. No entanto, o acesso ao dataset mostrou-se bloqueado para novos usuários. A solução foi utilizar um dataset espelho, disponível publicamente na mesma plataforma, que continha as mesmas 25.000 imagens de treino.

As imagens foram divididas programaticamente em três conjuntos para garantir uma avaliação robusta do modelo:

- **Conjunto de Treinamento:** 70% das imagens.
- **Conjunto de Validação:** 15% das imagens, usado para monitorar o desempenho durante o treinamento.
- **Conjunto de Teste:** 15% das imagens, usado para a avaliação final do modelo treinado.

Um script utilizando as bibliotecas ‘os‘, ‘shutil‘ e ‘sklearn‘ foi criado para realizar essa divisão e organizar os arquivos na estrutura de diretórios esperada pelo PyTorch (‘/train/-class‘, ‘/validation/class‘, etc.).

2.2 Framework e Pré-processamento

O projeto foi inicialmente planejado para TensorFlow, mas devido a uma incompatibilidade com a versão de Python utilizada (3.13), foi decidido migrar para o framework **PyTorch**. Esta escolha se mostrou vantajosa, oferecendo maior flexibilidade e compatibilidade.

O pré-processamento foi definido usando ‘torchvision.transforms‘. Para o conjunto de treino, foram aplicadas técnicas de *Data Augmentation* para aumentar a robustez do modelo, incluindo:

- Redimensionamento para 150x150 pixels.
- Rotações aleatórias de até 40 graus.
- Inversão horizontal aleatória.
- Normalização dos valores dos pixels para o intervalo [-1, 1].

Para os conjuntos de validação e teste, apenas o redimensionamento e a normalização foram aplicados. Os dados foram então carregados em lotes (batches) de 32 imagens através da classe ‘DataLoader‘.

2.3 Arquitetura do Modelo

A arquitetura da CNN foi definida como uma classe em PyTorch, herdando de ‘torch.nn.Module‘. A rede é composta por:

1. **Três blocos convolucionais:** Cada um contendo uma camada ‘Conv2d‘ (com 32, 64 e 128 filtros, respectivamente), uma função de ativação ‘ReLU‘ e uma camada ‘MaxPool2d‘ para subamostragem.
2. **Um classificador:** Após o achatamento (flatten) da saída dos blocos convolucionais, os dados passam por uma camada densa (‘Linear‘) de 512 neurônios com ativação ‘ReLU‘, uma camada de ‘Dropout‘ com taxa de 0.5 para regularização, e uma camada de saída final com 1 neurônio para a classificação binária.

2.4 Treinamento e Otimização

O modelo foi treinado com o otimizador ‘Adam‘ e uma taxa de aprendizado de 0.001. Durante os testes iniciais, o modelo apresentou estagnação, com acurácia travada em 50%, indicando que não estava aprendendo. A causa foi diagnosticada como uma instabilidade numérica na combinação da função de perda ‘BCELoss‘ com uma ativação ‘Sigmoid‘ aplicada separadamente no modelo.

A solução foi substituir a função de perda pela mais estável **‘BCEWithLogitsLoss‘**, que combina internamente a sigmoide e o cálculo da perda, e remover a ativação sigmoide da camada final do modelo. Após esta correção e o aumento do número de épocas de treinamento para 20, o modelo passou a aprender de forma eficaz.

3 Resultados e Discussão

O processo de treinamento corrigido resultou em um modelo com aprendizado consistente, como ilustrado pelos gráficos de acurácia e perda na Figura 1.

As curvas demonstram que o modelo não sofreu de overfitting significativo, pois as métricas de validação acompanharam de perto as de treinamento. A avaliação final no conjunto de teste, com dados completamente inéditos, confirmou o bom desempenho, atingindo uma **acurácia de 82%**. O relatório de classificação detalhado é apresentado na Tabela 1.

A análise das métricas revela um comportamento interessante: o modelo possui um recall muito alto para gatos (91%), indicando que ele raramente falha em identificar um gato. Por outro lado, sua precisão para cachorros é maior (89%), mostrando que, quando

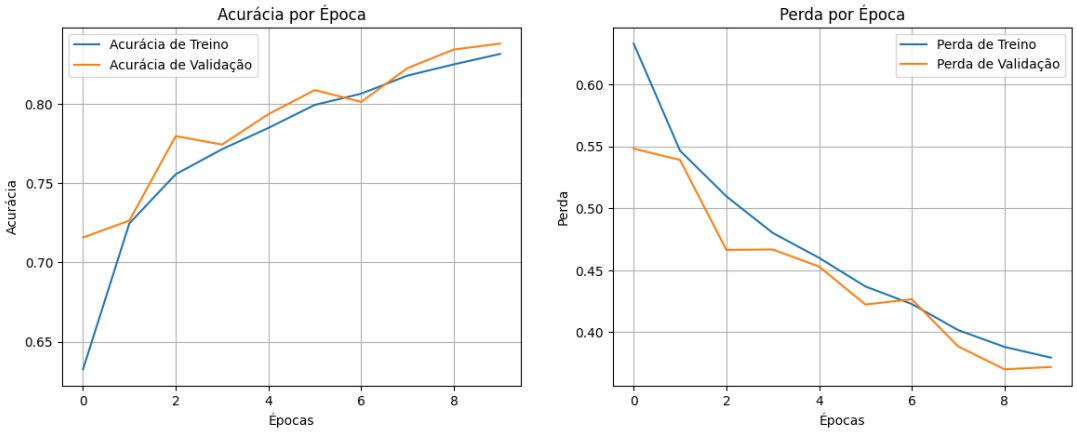

Figura 1: Curvas de Acurácia e Perda por Época nos conjuntos de treino e validação.

Tabela 1: Relatório de Classificação no Conjunto de Teste.

Classe	**Precision**	**Recall**	**F1-Score**	**Support**
cats	0.77	0.91	0.84	1875
dogs	0.89	0.74	0.81	1875
accuracy	-	-	0.82	3750
macro avg	0.83	0.82	0.82	3750
weighted avg	0.83	0.82	0.82	3750

ele prevê "cachorro", sua confiança é alta. O principal tipo de erro foi classificar cachorros como gatos, indicando um leve viés do modelo.

4 Testes com Novas Imagens

Para validar a aplicabilidade prática do modelo, foi criada uma função para classificar imagens novas obtidas da internet. A função carrega a imagem, aplica as mesmas transformações do conjunto de teste e alimenta o modelo treinado. Os testes demonstraram que o modelo é capaz de classificar corretamente novas imagens com alta confiança, como exemplificado na Figura 2 e na Figura 3.

5 Conclusão

O projeto atingiu com sucesso seu objetivo de desenvolver uma Rede Neural Convolucional para a classificação de imagens de cães e gatos. O modelo final demonstrou uma performance robusta, com 82% de acurácia, e seu comportamento foi analisado em detalhe.

Mais significativamente, a jornada de desenvolvimento serviu como um exercício prático valioso, forçando a superação de desafios reais como a incompatibilidade de ferramentas, problemas de acesso a dados e a depuração de redes neurais. Os aprendizados obtidos, especialmente a importância da estabilidade numérica no treinamento e a flexibilidade oferecida pelo PyTorch, são fundamentais para futuros projetos na área. Como próximo passo, sugere-se a aplicação de técnicas de *Transfer Learning* para buscar um desempenho ainda superior.

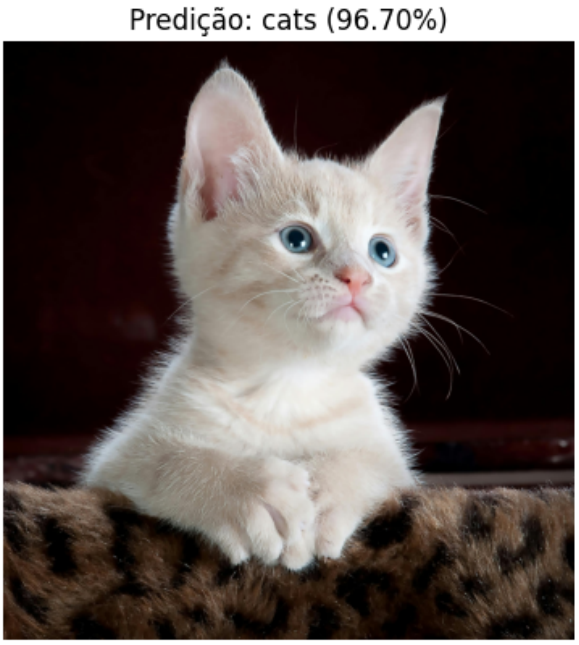

Figura 2: Exemplo de classificação de uma nova imagem de um gato com o modelo treinado.

Figura 3: Exemplo de classificação de uma nova imagem de um cachorro com o modelo treinado.