Classificador de Animais: Cachorro ou Gato?

Um Projeto Baseado em Redes Neurais

Enzo Paschoalini

Universidade Estadual Paulista 'Júlio de Mesquita Filho' (UNESP)

Departamento de Ciência da Computação - Campus Bauru

Novembro de 2024

Este relatório é apresentado como parte do desenvolvimento do projeto da disciplina de Inteligência Artificial no curso de Ciência da Computação da Universidade Estadual Paulista 'Júlio de Mesquita Filho', Campus de Bauru.

Orientador: Prof. Clayton Pereira

Contents

1 Introdução

O avanço do aprendizado de máquina e das redes neurais trouxe possibilidades antes inimagináveis no campo da classificação de imagens. Redes neurais convolucionais (*CNNs*) têm se destacado como uma das abordagens mais eficazes para tarefas de visão computacional, incluindo reconhecimento facial, diagnóstico médico e, neste caso, a classificação de imagens de animais. Este relatório apresenta o desenvolvimento de um sistema baseado em *CNNs* para classificar imagens de animais em duas categorias: gatos e cachorros. O objetivo principal é demonstrar como técnicas modernas de aprendizado supervisionado podem ser aplicadas para alcançar um modelo eficiente, robusto e capaz de generalizar para novos dados [9].

O projeto utilizou um dataset contendo 4.000 imagens de gatos e cachorros em formato *.png*. Este conjunto de dados é simetricamente distribuído entre as classes, garantindo equilíbrio no treinamento e avaliação. As imagens foram previamente divididas em subconjuntos para treinamento e teste, seguindo a seguinte distribuição:

- 1.500 imagens de gatos e 1.500 de cachorros no conjunto de treinamento.
- 500 imagens de gatos e 500 de cachorros no conjunto de teste.

Essa separação permite avaliar adequadamente a capacidade do modelo de generalizar para exemplos inéditos. Cada imagem possui dimensões ajustadas para 64×64 pixels, o que reduz a complexidade computacional sem comprometer significativamente a qualidade visual necessária para extração de características relevantes.

Para a implementação, utilizou-se o *framework* TensorFlow/Keras, que fornece ferramentas poderosas para construção e treinamento de redes neurais [1]. Além disso, o sistema foi integrado a uma interface interativa desenvolvida com *deploy Streamlit*, tornando possível que usuários não técnicos carreguem imagens e obtenham resultados instantaneamente. Esta interface demonstra como soluções baseadas em inteligência artificial podem ser aplicadas em cenários do mundo real, como aplicativos de classificação de imagens em dispositivos móveis.

A divisão do projeto em duas partes principais — construção do modelo e interface de usuário — reflete a abordagem prática adotada. A primeira etapa concentrou-se na arquitetura do modelo e no treinamento supervisionado com base no conjunto de dados.

A segunda etapa focou na usabilidade, garantindo que os resultados do modelo fossem apresentados de forma acessível e compreensível ao usuário final.

2 Fundamentos de Redes Neurais Convolucionais (CNNs)

2.1 Conceito Geral

Redes neurais convolucionais (*CNNs*) são um tipo especial de rede neural projetada especificamente para processar dados com estrutura de grade, como imagens, vídeos, e até mesmo dados de áudio com representação em espectrograma [10]. Elas são amplamente utilizadas em tarefas de *visão computacional*, como reconhecimento de objetos, classificação de imagens, segmentação semântica, e detecção de anomalias. O diferencial das *CNNs* está em sua capacidade de aprender representações hierárquicas de dados, o que significa que diferentes camadas da rede são especializadas em capturar diferentes tipos de características dos dados de entrada [9].

A principal vantagem das *CNNs* é a capacidade de aprender **características locais** (como bordas e texturas) nas camadas iniciais e **características globais** (como formas e objetos complexos) nas camadas mais profundas [10, 5]. Essa capacidade é possibilitada através de uma estrutura hierárquica, onde as camadas mais profundas combinam as informações extraídas pelas camadas anteriores para aprender representações de alto nível.

O processo de convolução, que é a base da operação das *CNNs*, envolve a aplicação de filtros (ou kernels) sobre as imagens [5]. Esses filtros são matrizes pequenas que se movem (ou convoluem) sobre a imagem de entrada para produzir um mapa de características. [10, 5] são as principais referências que explicam a importância das *CNNs* no campo da visão computacional.

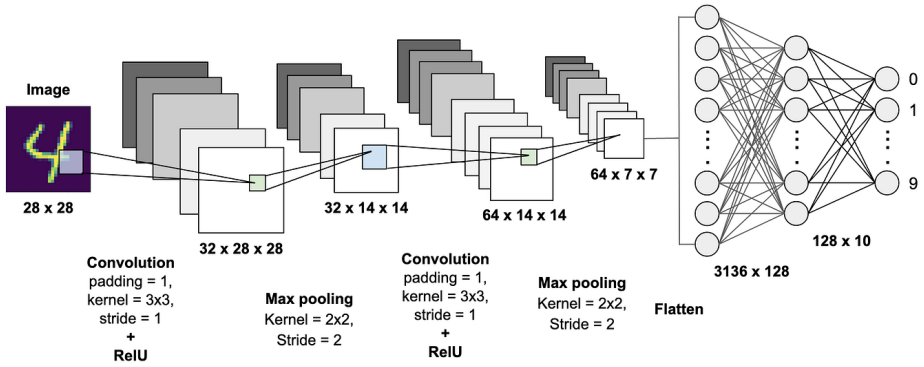

Figure 1: Arquitetura de uma Rede Neural Convolucional (CNN).

2.2 Componentes Fundamentais das *CNNs*

As **camadas convolucionais** são o componente central de uma rede neural convolucional. Elas aplicam filtros, também conhecidos como *kernels*, sobre a imagem de entrada para extrair características locais, como bordas, texturas ou padrões específicos [10, 9]. A principal vantagem da convolução é que ela permite que a rede seja translacionalmente invariante, ou seja, a capacidade de detectar um padrão específico não é afetada pela posição desse padrão na imagem.

As **camadas de pooling** desempenham um papel crucial na redução da dimensionalidade dos dados, mantendo apenas as informações essenciais [5, 10]. Técnicas como o *max pooling* ajudam a preservar as características mais proeminentes, tornando o modelo mais robusto a pequenas transformações.

Após a extração de características, as **camadas densas** realizam a classificação final, combinando as características extraídas nas camadas anteriores [9]. Essas camadas totalmente conectadas ajudam a aprender uma combinação complexa das características, levando à predição final.

Por fim, as técnicas de **regularização**, como o *Dropout* e a *Batch Normalization*, ajudam a evitar o *overfitting* e melhoram a generalização do modelo [14, 7].

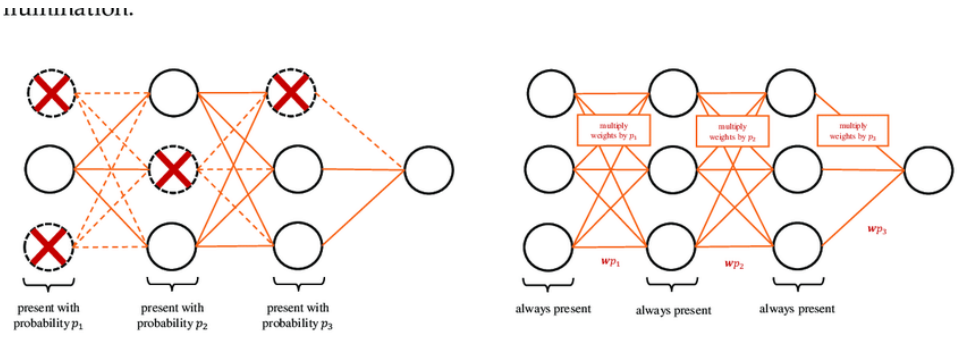

Figure 2: Dropout em redes neurais.

3 Arquitetura do Modelo

A arquitetura implementada foi desenvolvida com foco em equilíbrio entre simplicidade e desempenho. Ela é composta por dois blocos convolucionais, seguidos por camadas densas para integração das características extraídas.

3.1 Estrutura do Modelo

A arquitetura da rede começa com o **Bloco Convolucional 1**, que aplica 32 filtros de tamanho 3×3 na camada `Conv2D`. Cada filtro captura características locais específicas, como bordas e texturas, e a ativação utilizada é ReLU, o que ajuda a evitar o problema de gradientes desaparecendo [4]. Em seguida, a `BatchNormalization` normaliza as ativações de cada camada [7]. A `MaxPooling2D` é usada para reduzir a resolução espacial das saídas da camada convolucional, preservando as características mais relevantes [10].

No **Bloco Convolucional 2**, aplicam-se 64 filtros de tamanho 3×3, com `Batch Normalization` e `MaxPooling2D` novamente, ampliando a capacidade da rede de aprender características mais complexas. Após os blocos convolucionais, a rede passa pelas **Camadas Finais**, com a camada `Flatten` transformando as saídas das camadas anteriores em um vetor unidimensional. As camadas `Dense` de 512 unidades com ReLU e `Dropout` são aplicadas para aprender combinações não lineares [14], enquanto a camada de saída usa `Softmax` para realizar a classificação binária [9].

3.2 Compilação e Treinamento

O treinamento da rede neural foi realizado utilizando o **otimizador Adam**, que ajusta dinamicamente a taxa de aprendizado com base nas médias móveis do gradiente e do

quadrado do gradiente [8]. A taxa de aprendizado inicial foi configurada em 0.001, um valor que balanceia a velocidade e a estabilidade do treinamento.

A função de perda escolhida foi a `categorical_crossentropy`, ideal para problemas de classificação multiclasse, onde as classes são mutuamente exclusivas [11]. Ela mede a discrepância entre as distribuições preditas pela rede e os rótulos verdadeiros, ajudando a minimizar os erros do modelo.

Para avaliar o desempenho, utilizou-se a métrica de **acurácia**, que calcula a proporção de previsões corretas. A escolha da acurácia como métrica principal se justifica pela distribuição simétrica das classes no conjunto de dados [3].

4 Interface com *Streamlit*

O sistema foi integrado a uma aplicação interativa desenvolvida com o *Streamlit*, que permite aos usuários carregar imagens, realizar a classificação e visualizar os resultados em tempo real. A interface foi projetada com foco na acessibilidade, sendo intuitiva e amigável, garantindo que mesmo pessoas sem experiência técnica possam utilizá-la de forma eficiente.

A aplicação utiliza dois arquivos essenciais para carregar o modelo treinado: o arquivo network.json, que contém a arquitetura da rede neural convolucional, e o arquivo weights.hdf5, que armazena os pesos previamente ajustados durante o treinamento. Esses arquivos são carregados no início da execução da aplicação, reconstruindo o modelo com as configurações e parâmetros originais.

Após a inicialização do modelo, a interface disponibiliza um campo para o upload de imagens no formato PNG. Assim que o usuário faz o upload de uma imagem, o sistema processa a entrada, redimensionando-a para 64×64 pixels, de forma a compatibilizá-la com as dimensões esperadas pela rede. Em seguida, a imagem é normalizada e submetida ao modelo, que realiza a predição e retorna a classe correspondente, indicando se a imagem é de um gato ou de um cachorro. O resultado da classificação é exibido diretamente na interface.

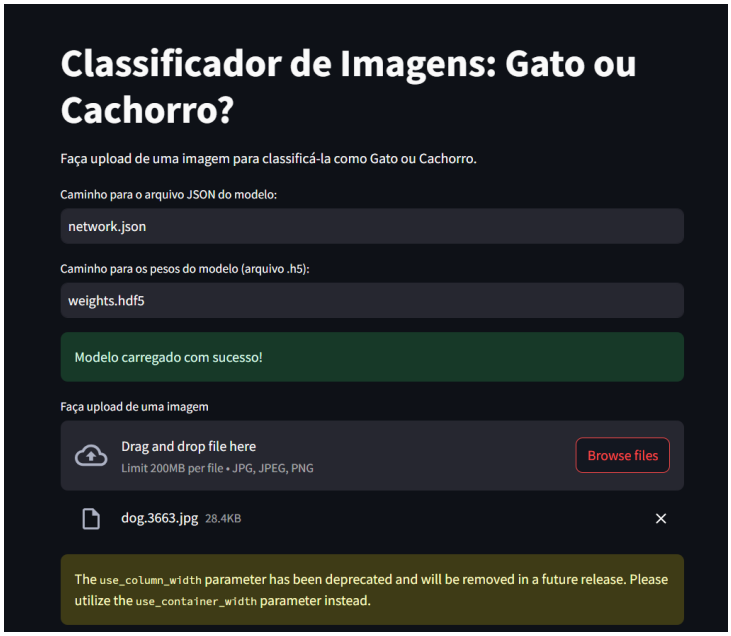

Figure 3: Interface ao carregar modelo e pesos.

Figure 4: Interface ao classificar uma imagem.

5 Resultados e Avaliação

O modelo alcançou uma precisão geral de **79,1%** nos dados de validação, um desempenho consistente com os resultados apresentados pela matriz de confusão e as métricas detalhadas. Este resultado é considerado satisfatório, especialmente considerando o tamanho reduzido das imagens (64×64) e a simplicidade relativa da arquitetura utilizada.

Desempenho por Classe

As métricas por classe demonstram que o modelo tem desempenho equilibrado na classificação de ambas as categorias:

- **Classe 0 (Gato):**
  - **Precision:** 78%, indicando que 78% das predições da classe "gato" estavam corretas.
  - **Recall:** 80%, mostrando que 80% dos gatos presentes no conjunto de validação foram identificados corretamente.
  - **F1-Score:** 79%, representando um equilíbrio entre precisão e sensibilidade.
- **Classe 1 (Cachorro):**
  - **Precision:** 80%, indicando que 80% das predições da classe "cachorro" estavam corretas.
  - **Recall:** 78%, mostrando que 78% dos cachorros foram identificados corretamente.
  - **F1-Score:** 79%, refletindo também o equilíbrio para esta classe.

Desempenho Geral

- **Accuracy:** O modelo apresentou uma precisão geral de 79%, ou seja, 79% das imagens no conjunto de validação foram classificadas corretamente.
- **Macro Average:** As médias macro de precisão, recall e F1-Score são todas 79%, indicando um desempenho uniforme entre as classes.
- **Weighted Average:** O modelo também apresenta médias ponderadas de 79%, o que reflete a distribuição simétrica das classes.

Matriz de Confusão

A matriz de confusão reforça as métricas apresentadas:

- **400** gatos foram corretamente classificados como gatos (verdadeiros positivos para a classe 0).
- **98** gatos foram incorretamente classificados como cachorros (falsos negativos para a classe 0).
- **390** cachorros foram corretamente classificados como cachorros (verdadeiros positivos para a classe 1).
- **110** cachorros foram incorretamente classificados como gatos (falsos negativos para a classe 1).

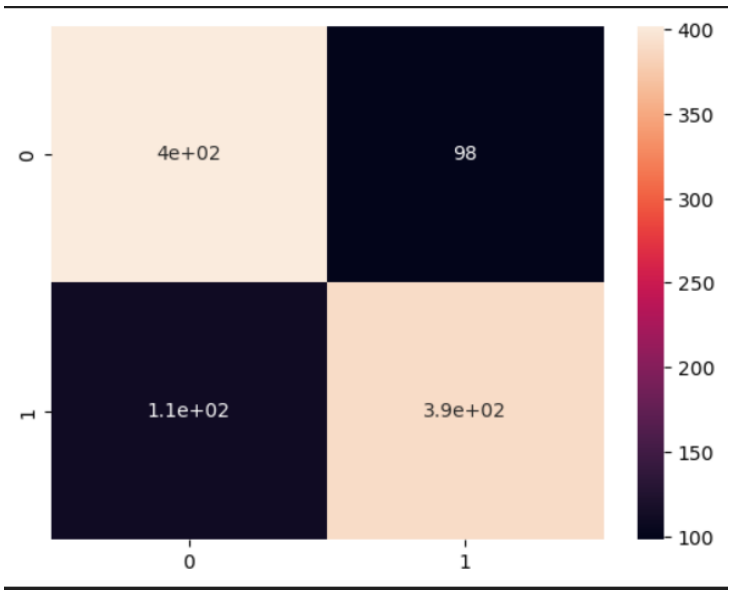

Figure 5: Matriz de confusão usando seaborn.

Esses resultados indicam que o modelo tem desempenho equilibrado entre as classes, com um leve viés em confundir gatos como cachorros e vice-versa. Esse comportamento pode ser atribuído à simplicidade da arquitetura da rede, ao tamanho reduzido das imagens e à possível semelhança visual entre alguns exemplares das classes.

Embora existam margens para melhorias, como a utilização de redes mais profundas ou técnicas de aumento de dados (*data augmentation*), o desempenho atual é promissor. A integração com a aplicação *Streamlit* amplia a utilidade prática do sistema, permitindo que os usuários utilizem o modelo de forma intuitiva e eficiente para classificar imagens de gatos e cachorro

6 Possíveis Melhorias

Uma série de melhorias pode ser implementada para elevar o desempenho do modelo atual. Primeiramente, aumentar a resolução das imagens pode ajudar a capturar mais detalhes visuais, especialmente características sutis que são fundamentais para distinguir entre as classes de gatos e cachorros. Estudos demonstram que resoluções mais altas frequentemente levam a melhores resultados em tarefas de classificação de imagens, pois fornecem informações mais ricas para o treinamento das redes neurais [2].

Outra estratégia promissora é a utilização de *transfer learning*, que consiste em aproveitar modelos pré-treinados em grandes conjuntos de dados, como o ResNet [6] ou o VGG [13]. Esses modelos já aprenderam representações gerais a partir de bases de dados extensas, como o ImageNet [2], e podem ser adaptados ao problema atual por meio de *fine-tuning*, economizando tempo de treinamento e melhorando a precisão.

Adicionalmente, a aplicação de técnicas de aumento de dados (*data augmentation*) pode melhorar significativamente a capacidade de generalização do modelo. Essas técnicas, como rotações, inversões horizontais, ajustes de brilho e recortes aleatórios, aumentam a diversidade do conjunto de treinamento sem a necessidade de adquirir mais dados [12]. Ao aumentar a variabilidade nas imagens de entrada, o modelo é exposto a cenários mais amplos e é menos propenso a *overfitting*, resultando em melhor desempenho em dados não vistos.

Essas melhorias representam caminhos concretos para refinar o modelo, tornando-o mais robusto e eficiente para a tarefa de classificação de imagens.

7 Conclusão

O projeto demonstra como redes neurais convolucionais podem ser aplicadas para resolver problemas reais de classificação de imagens. O uso do *Streamlit* como interface interativa expandiu a acessibilidade da solução, permitindo que não especialistas utilizassem o modelo com facilidade.

References

[1] François Chollet. *Deep Learning with Python*. Manning Publications Co., 2017.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

[3] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[4] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. *Proceedings of the 14th international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B Girshick. Mask r-cnn. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(2):296–307, 2017.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[12] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.