Relatório da Classificação de Imagens com Redes Neurais Convolucionais - Lista 12

Samuel Horta de Faria
801528
https://github.com/SamuelHortadeFaria/IA_Lista12

Junho de 2025

Sumário

1 Introdução

A classificação de imagens é um dos problemas fundamentais no campo da visão computacional e da inteligência artificial. O desafio consiste em treinar um modelo computacional para categorizar uma imagem com base em seu conteúdo visual. O problema "Dogs vs. Cats", popularizado pela plataforma Kaggle, serve como um excelente caso de estudo para a aplicação de técnicas de aprendizado profundo (*Deep Learning*).

O objetivo deste trabalho foi construir e treinar uma Rede Neural Convolucional (CNN), um tipo de arquitetura de rede neural especialmente projetada para processar dados com uma topologia de grade, como as imagens. As CNNs são o estado da arte para muitas tarefas de visão computacional, pois são capazes de aprender hierarquias de características espaciais de forma automática.

Este relatório documenta a metodologia empregada, dividida nas seguintes etapas:

1. **Preparação dos Dados:** Organização dos arquivos de imagem e extração de rótulos.
2. **Pré-processamento:** Divisão dos dados, normalização e aplicação de *data augmentation*.
3. **Construção do Modelo:** Definição da arquitetura da CNN, camada por camada.
4. **Treinamento e Avaliação:** Compilação do modelo, treinamento e análise das métricas de desempenho.
5. **Predição:** Utilização do modelo treinado para classificar novas imagens.

2 Metodologia

A solução foi desenvolvida em um ambiente Jupyter Notebook, utilizando Python como linguagem de programação e o framework TensorFlow com sua API de alto nível, Keras.

2.1 Preparação dos Dados

O conjunto de dados inicial continha uma pasta de treino com 25.000 imagens, divididas igualmente entre 12.500 gatos e 12.500 cachorros, e uma pasta de teste com 12.500 imagens.

O primeiro passo foi ler os nomes dos arquivos do diretório de treino e extrair a classe ("cat"ou "dog") para cada um. Essas informações foram organizadas em um DataFrame da biblioteca `pandas`, facilitando a manipulação. Os rótulos textuais foram então convertidos para um formato numérico binário para o treinamento: **0 para gato** e **1 para cachorro**.

2.2 Pré-processamento e Geração de Dados

Para avaliar o modelo de forma robusta e evitar o sobreajuste (*overfitting*), o conjunto de treino foi dividido em dois subconjuntos:

- **Conjunto de Treinamento (90%):** 22.500 imagens, usadas para ajustar os pesos do modelo.
- **Conjunto de Validação (10%):** 2.500 imagens, usadas para avaliar o desempenho do modelo em dados não vistos ao final de cada época.

A ferramenta `ImageDataGenerator` do Keras foi fundamental nesta etapa, configurada para realizar duas tarefas principais:

1. Normalização dos Pixels: Os valores dos pixels de cada imagem, que variam de 0 a 255, foram reescalados para o intervalo [0, 1]. Isso ajuda a estabilizar e acelerar o processo de treinamento.

2. Aumento de Dados (*Data Augmentation*): Para o conjunto de treino, foram aplicadas transformações aleatórias em tempo real nas imagens (rotações, zooms, cisalhamentos, etc.). Essa técnica aumenta artificialmente a diversidade do conjunto de dados, forçando o modelo a aprender características mais gerais e robustas.

2.3 Arquitetura do Modelo

Foi projetada uma arquitetura de CNN sequencial, composta por blocos de convolução e *pooling*, seguidos por camadas densas para classificação. A estrutura é a seguinte:

- **Bloco Convolucional 1:** Uma camada `Conv2D` com 32 filtros (kernel 3x3) e ativação ReLU, seguida por uma camada `MaxPooling2D`.
- **Bloco Convolucional 2:** Uma camada `Conv2D` com 64 filtros e ativação ReLU, seguida por `MaxPooling2D`.
- **Bloco Convolucional 3:** Uma camada `Conv2D` com 128 filtros e ativação ReLU, seguida por `MaxPooling2D`.
- **Camada de Achatamento (Flatten):** Transforma os mapas de características 2D resultantes em um vetor 1D.
- **Camadas de Classificação:** Uma camada `Dense` com 512 neurônios (ReLU), uma camada `Dropout` com taxa de 0.5 para regularização, e a camada de saída `Dense` com 1 neurônio e ativação `sigmoid`, ideal para classificação binária.

O resumo da arquitetura, gerado pelo Keras, é apresentado abaixo.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896
```

```
 max_pooling2d (MaxPooli   (None, 63, 63, 32)        0
 ng2D)
 conv2d_1 (Conv2D)         (None, 61, 61, 64)        18496
 max_pooling2d_1 (MaxPool  (None, 30, 30, 64)        0
 ing2D)
 conv2d_2 (Conv2D)         (None, 28, 28, 128)       73856
 max_pooling2d_2 (MaxPool  (None, 14, 14, 128)       0
 ing2D)
 flatten (Flatten)         (None, 25088)             0
 dense (Dense)             (None, 512)               12845568
 dropout (Dropout)         (None, 512)               0
 dense_1 (Dense)           (None, 1)                 513
=================================================================
Total params: 12,939,329
Trainable params: 12,939,329
Non-trainable params: 0
```

Listing 1: Resumo da Arquitetura da CNN

3 Treinamento e Compilação

Antes do treinamento, o modelo foi compilado com os seguintes parâmetros:

- **Otimizador:** `adam`, um algoritmo de otimização eficiente.
- **Função de Perda:** `binary_crossentropy`, a escolha padrão para problemas de classificação binária.
- **Métricas:** `accuracy`, para monitorar a acurácia do modelo.

O modelo foi treinado por **3 épocas**, utilizando os geradores de dados criados na etapa de pré-processamento.

4 Resultados e Análise

O desempenho do modelo foi monitorado ao longo das 3 épocas. Ao final, o modelo alcançou uma **acurácia de validação de 77.12%** com uma **perda de 0.4868**.

A evolução da acurácia e da perda é visualizada nos gráficos das Figuras 1 e 2. Observa-se que tanto a acurácia de treino quanto a de validação aumentaram, enquanto a perda diminuiu em ambos os conjuntos, indicando que o modelo estava aprendendo efetivamente.

No entanto, a análise dos gráficos também revela um distanciamento entre as curvas de treinamento e validação. A acurácia de treino (linha azul) cresce mais rapidamente que a de validação (linha vermelha), e a perda de treino diminui mais acentuadamente. Isso sugere o início de um leve sobreajuste (*overfitting*), onde o modelo começa a memorizar os dados de treino em vez de generalizar para dados novos.

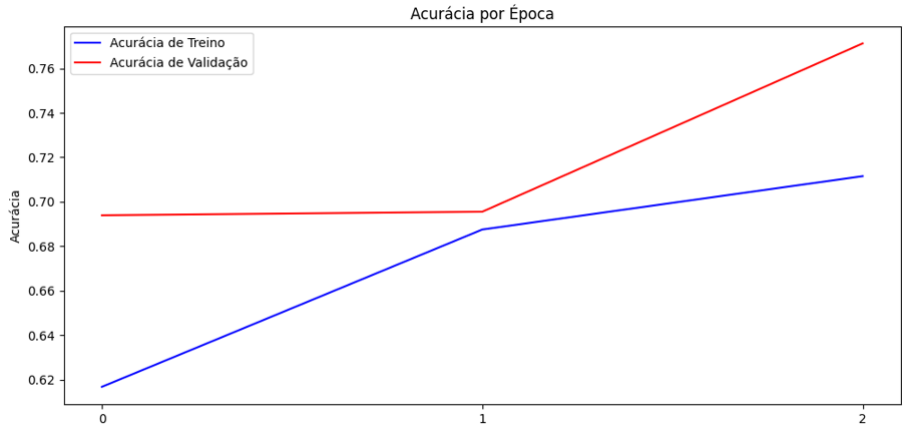

Figura 1: Gráfico da Acurácia por Época.

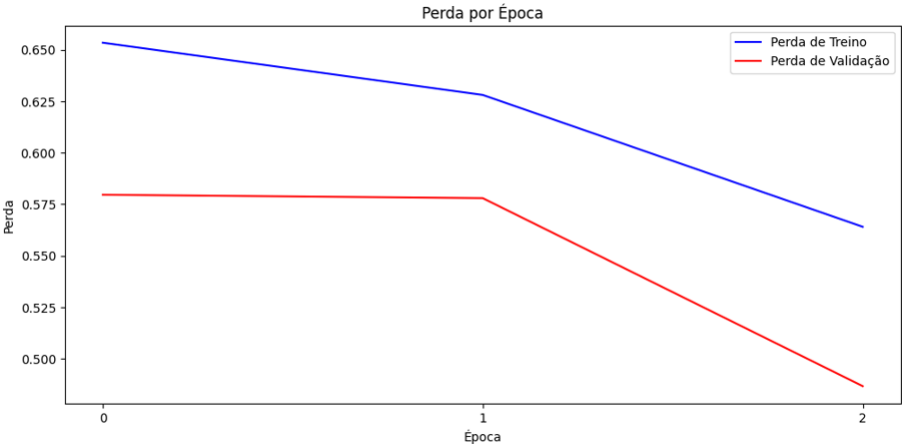

Figura 2: Gráfico da Perda (Loss) por Época.

5 Predição e Geração de Submissão

Após o treinamento, o modelo foi utilizado para prever as classes das 12.500 imagens do conjunto de teste. Um gerador de dados foi criado para este conjunto, aplicando apenas a normalização de pixels. As predições geradas pela camada `sigmoid` (valores de probabilidade) foram convertidas para classes binárias (0 para gato, 1 para cachorro) usando um limiar de 0.5. Finalmente, os resultados foram formatados em um arquivo para submissão.

6 Conclusão

O projeto demonstrou com sucesso a aplicação de uma Rede Neural Convolucional para a classificação de imagens. O pipeline implementado resultou em um classificador que atingiu uma **acurácia de validação de aproximadamente 77.12%** após 3 épocas de treinamento.

O modelo apresentou uma clara capacidade de aprendizado, conforme visto pelo aumento da acurácia e diminuição da perda em ambos os conjuntos de treino e

validação. Contudo, a análise do desempenho revelou que o modelo começa a exibir sinais de um **leve sobreajuste**, indicando que, embora o aprendizado tenha ocorrido, há espaço para melhorias na generalização.

Como trabalhos futuros para aprimorar o desempenho, sugere-se:

- **Aumento do Treinamento:** Treinar o modelo por mais épocas pode permitir uma convergência mais fina dos pesos e, consequentemente, alcançar uma acurácia maior.
- **Ajuste de Hiperparâmetros:** Otimizar parâmetros como a taxa de aprendizado, o tamanho dos lotes e a própria arquitetura da rede.
- **Transfer Learning:** Utilizar uma arquitetura de CNN pré-treinada em um conjunto de dados maior (como ImageNet) e ajustá-la para este problema específico, o que pode acelerar o treinamento e melhorar o desempenho final.