

Documentação do Projeto: Autoencoder Convolucional com CIFAR-10

Autor: Gabriel Castelo **Tipo de Modelo:** Convolutional Autoencoder (CAE) **Dataset:** CIFAR-10 **Framework:** TensorFlow/Keras

1. Definição do Problema e Conceito

Este projeto implementa uma rede neural de aprendizado profundo não supervisionado chamada **Autoencoder**. O objetivo principal é aprender a **comprimir** e **reconstruir** essas imagens com a menor perda de informação possível.

O que é um Autoencoder?

Um Autoencoder é composto por duas partes principais que trabalham em conjunto:

1. **Encoder (Codificador):** Recebe a imagem original e a comprime em uma representação menor e densa, chamada de *espaço latente* (latent space). Ele aprende a extrair apenas as características mais essenciais (bordas, texturas, formas).
2. **Decoder (Decodificador):** Pega essa representação comprimida e tenta recriar a imagem original a partir dela.

Neste projeto, utilizamos camadas convolucionais (Conv2D), que são ideais para processamento de imagens, pois preservam a estrutura espacial dos dados melhor que as camadas densas tradicionais.

2. O Dataset: CIFAR-10

O modelo é treinado no conjunto de dados **CIFAR-10**, um benchmark padrão em visão computacional.

- **Conteúdo:** 60.000 imagens coloridas.
- **Dimensões:** 32x32 pixels (Baixa resolução).
- **Canais:** 3 (RGB).
- **Classes:** 10 (Avião, Automóvel, Pássaro, Gato, Veado, Cão, Sapo, Cavalo, Navio, Caminhão).

3. Arquitetura da Rede Neural

O código define uma arquitetura simétrica de compressão e descompressão.

Pré-processamento

As imagens são carregadas e normalizadas. Os valores dos pixels, originalmente entre 0 e 255, são divididos por 255.0 para ficarem no intervalo **[0, 1]**. Isso acelera a convergência do treinamento.

Estrutura do Modelo

1. Encoder (Compressão):

- A imagem entra com tamanho (32, 32, 3).
- Passa por camadas de Convolução seguidas de MaxPooling2D.
- O MaxPooling reduz a dimensão da imagem pela metade a cada etapa (32 -> 16 -> 8 -> 4).
- Ao final do encoder, a imagem é representada por um volume de (4, 4, 128).

2. Decoder (Reconstrução):

- Recebe o volume (4, 4, 128).
- Utiliza camadas de Conv2D e UpSampling2D.
- O UpSampling duplica a dimensão da imagem a cada etapa (4 -> 8 -> 16 -> 32).
- A última camada usa ativação **Sigmoid** para garantir que os pixels de saída estejam entre 0 e 1, correspondendo à normalização da entrada.

4. Exemplo de Uso e Execução

O script executa o pipeline completo automaticamente. Abaixo, o fluxo de execução:

Treinamento

O modelo é compilado usando o otimizador **Adam** e a função de perda **MSE (Mean Squared Error)**. O MSE é escolhido porque queremos minimizar a diferença pixel a pixel entre a entrada e a saída.

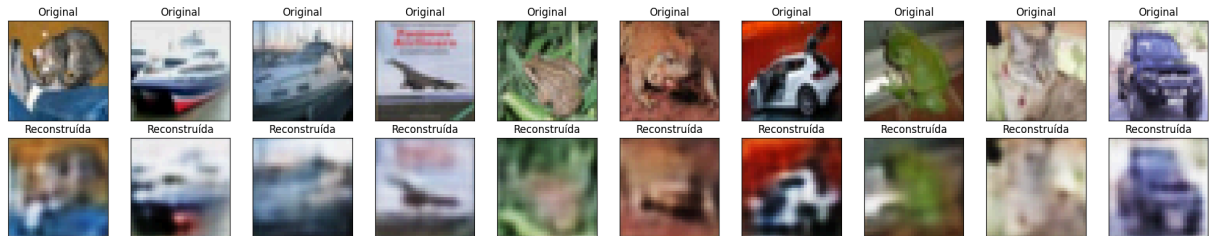
```
# O alvo (y) é a própria imagem de entrada (x)
autoencoder.fit(
    train_images, train_images,
    epochs=10,
    batch_size=128,
    ...
)
```

Visualização dos Resultados

Após o treinamento, o script seleciona imagens de teste, passa pelo autoencoder e exibe o "Antes e Depois".

Saída Visual Esperada: O código gera um gráfico onde a linha superior mostra as imagens originais nítidas e a linha inferior mostra as reconstruções feitas pelo modelo.

- **Originais:** Imagens pixeladas 32x32 nítidas.
- **Reconstruídas:** Versões levemente "embaçadas" das originais. Esse desfoque é normal e indica que o modelo aprendeu os padrões gerais, mas perdeu detalhes de alta frequência devido à compressão no espaço latente.



Curva de Aprendizado

O código também plota um gráfico de Loss (Perda) ao longo das épocas. A curva deve ser descendente, indicando que o modelo está aprendendo a reconstruir as imagens com cada vez menos erro.

5. Requisitos para Execução

Para rodar este script o ambiente deve ter as seguintes bibliotecas instaladas

- Python 3.x
- TensorFlow 2.x (`import tensorflow`)
- NumPy (`import numpy`)
- Matplotlib (`import matplotlib`)