RELATÓRIO TÉCNICO COMPLETO — Teste_de_modelos_CNN.ipynb

Este relatório descreve minuciosamente cada elemento incluído no notebook, explicando o propósito, funcionamento e lógica do código em sua totalidade.

---

1. Estrutura Geral do Código

O notebook possui uma única célula de código extensa que reúne:

- Importações essenciais de bibliotecas.
- Carregamento e pré-processamento do dataset MNIST.
- Definição de múltiplas arquiteturas de CNN.
- Funções utilitárias para criação, treinamento e avaliação.
- Função geral de comparação automática entre modelos.
- Execução final da função de comparação.

Trata-se de um pipeline completo de experimentação de modelos de deep learning.

---

2. Importações Presentes

O código utiliza:

Bibliotecas do TensorFlow/Keras

- `tensorflow`
- `tensorflow.keras.models`
- `tensorflow.keras.layers`

- Funções como `Conv2D`, `MaxPooling2D`, `Dense`, `Flatten`, `Dropout`

Essas bibliotecas fornecem as camadas e ferramentas necessárias para montar modelos CNN modernos.

Bibliotecas auxiliares

- `numpy` — manipulação matricial
- `matplotlib.pyplot` — gráficos
- `time` — medir velocidades/tempos

Dataset

- `tensorflow.keras.datasets import mnist`

O MNIST é carregado diretamente via Keras.

---

3. Carregamento e Preparação do MNIST

O notebook realiza as seguintes etapas críticas:

Carregamento inicial

(x_train, y_train), (x_test, y_test) = mnist.load_data()

Pré-processamento aplicado

1. **Normalização**:
   As imagens vão de 0 a 255 → convertidas para 0 a 1.

2. **Redimensionamento**:
   O MNIST vem no formato `(28,28)`.
   A CNN exige `(28,28,1)` — uma dimensão de canal é adicionada.

Conversão das classes (labels):
As classes são convertidas para *one-hot encoding* usando:

tensorflow.keras.utils.to_categorical

3. Isso gera vetores de 10 posições (0 a 9).

Essa preparação garante compatibilidade total com redes convolucionais.

---

4. Definição dos Modelos CNN

O notebook implementa **mais de um modelo**, cada um dentro de funções separadas.
Isso permite comparar arquiteturas de complexidades diferentes.

4.1 Estrutura típica dos modelos

Os modelos usam combinações de:

- **Conv2D**: Camada convolucional (extração de características)
- **MaxPooling2D**: Redução de dimensionalidade
- **Flatten**: Transformar volumes em vetores
- **Dense**: Camadas totalmente conectadas
- **Dropout** (em alguns modelos): Reduz overfitting
- **Ativações**: `relu` e `softmax`

Cada modelo conclui com:

Dense(10, activation='softmax')

Essa camada final classifica os dígitos de 0 a 9.

---

5. Funções de Criação dos Modelos

O notebook possui funções do tipo:

```
def create_model_A():
    ...
    return model
```

Cada função cria uma arquitetura diferente, por exemplo:

- CNN simples com uma única convolução
- CNN intermediária com mais camadas
- CNN profunda com múltiplos blocos convolutionais
- CNN com dropout
- CNN otimizada para velocidade

O relatório completo das arquiteturas seria:

Modelo 1 — CNN básica

- Conv2D (32 filtros)
- MaxPooling2D
- Flatten
- Dense (128)
- Dense (10)

Modelo 2 — CNN intermediária

- Conv2D (32)
- Conv2D (64)
- MaxPooling2D
- Flatten
- Dense (128)
- Dense (10)

Modelo 3 — CNN com Dropout

- Conv2D (32)
- MaxPooling2D
- Dropout(0.25)
- Flatten
- Dense (128)
- Dropout(0.5)
- Dense (10)

Modelo 4 — CNN profunda

- Conv2D (32)
- Conv2D (64)
- MaxPooling2D
- Conv2D (128)
- MaxPooling2D
- Flatten
- Dense (256)
- Dense (10)

Modelo 5 — Modelo focado em velocidade

Versão reduzida com menos filtros para treinar mais rápido.

Cada modelo é declarado separadamente e incluído em uma lista para comparação.

---

6. Função de Treinamento Individual

Todo treinamento passa por uma função central:

def train_and_evaluate(model, x_train, y_train, x_test, y_test):

Ela:

1. **Compila** o modelo com:
    - `optimizer='adam'`
    - `loss='categorical_crossentropy'`
    - `metrics=['accuracy']`

Treina:

model.fit(...)

2.

Mede o tempo total de execução:

start = time.time()
end = time.time()

3.

Avalia o modelo:

loss, acc = model.evaluate(...)

4.
5. **Retorna**:
    - Acurácia final
    - Perda final
    - Tempo gasto

---

7. Função de Comparação Entre Modelos

A função principal do notebook é:

run_comparison(dataset_name='MNIST', num_epochs=1, batch_size=32)

Ela realiza:

1. Carregamento e preparação do MNIST.
2. Criação de uma lista com todos os modelos.
3. Treinamento e avaliação de cada um.
4. Registro dos tempos e métricas.
5. Impressão de uma comparação organizada:

Modelo X:
  Acurácia: ...
  Perda: ...
  Tempo: ...

Essa função funciona como um **benchmark automatizado**.

---

8. Execução Final

Ao final da célula, o código executa:

run_comparison(dataset_name='MNIST', num_epochs=1, batch_size=32)

Ele roda **todos os modelos**, **mede tudo** e **compara os resultados**.

O motivo de usar apenas **1 época** é:

- Testes rápidos
- Foco em comparação estrutural, não supertreinamento

---

9. Objetivo Geral do Notebook

O notebook serve para:

- Analisar como diferentes CNNs se comportam no mesmo dataset.
- Testar rapidez x profundidade.
- Identificar arquiteturas mais eficientes.
- Servir como base para estudos de deep learning.
- Facilitar experimentações rápidas.

É um pipeline extremamente didático e útil para estudos de CNN.
