ório de Treinamento de Rede Neural Convolucional (CNN) para identificar Roupas em Ima

Autor: Pedro Henrique Vilaça Valverde

1. Resumo do Modelo (Arquitetura)

```
Model: "sequential"

 Layer (type)                        Output Shape                      Param #

| conv2d (Conv2D)                   | (None, 26, 26, 32)          |             320 |
+-----------------------------------------------------------------------------+
| max_pooling2d (MaxPooling2D)      | (None, 13, 13, 32)          |               0 |
+-----------------------------------------------------------------------------+
| conv2d_1 (Conv2D)                 | (None, 11, 11, 64)          |          18,496 |
+-----------------------------------------------------------------------------+
| max_pooling2d_1 (MaxPooling2D)    | (None, 5, 5, 64)            |               0 |
+-----------------------------------------------------------------------------+
| flatten (Flatten)                 | (None, 1600)                |               0 |
+-----------------------------------------------------------------------------+
| dense (Dense)                     | (None, 128)                 |         204,928 |
+-----------------------------------------------------------------------------+
| dropout (Dropout)                 | (None, 128)                 |               0 |
+-----------------------------------------------------------------------------+
| dense_1 (Dense)                   | (None, 10)                  |           1,290 |
+-----------------------------------+-----------------------------+-----------------+
 Total params: 675,104 (2.58 MB)
 Trainable params: 225,034 (879.04 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 450,070 (1.72 MB)
```

2. Detalhes da Rede (Pesos e Bias)

```
TIPO DE TREINAMENTO:
   - Tipo: Aprendizado Supervisionado
   - Algoritmo: Backpropagation
   - Otimizador: Adam (configuração padrão)
   - Função de Perda (Loss): Sparse Categorical Crossentropy

PESOS (Wij) E BIAS (bi) POR CAMADA:

--- Camada 0: conv2d ---
   - Formato da Matriz de Pesos (Wij): (3, 3, 1, 32)
   - Formato do Vetor de Bias (bi): (32,)

--- Camada 2: conv2d_1 ---
   - Formato da Matriz de Pesos (Wij): (3, 3, 32, 64)
   - Formato do Vetor de Bias (bi): (64,)

--- Camada 5: dense ---
```

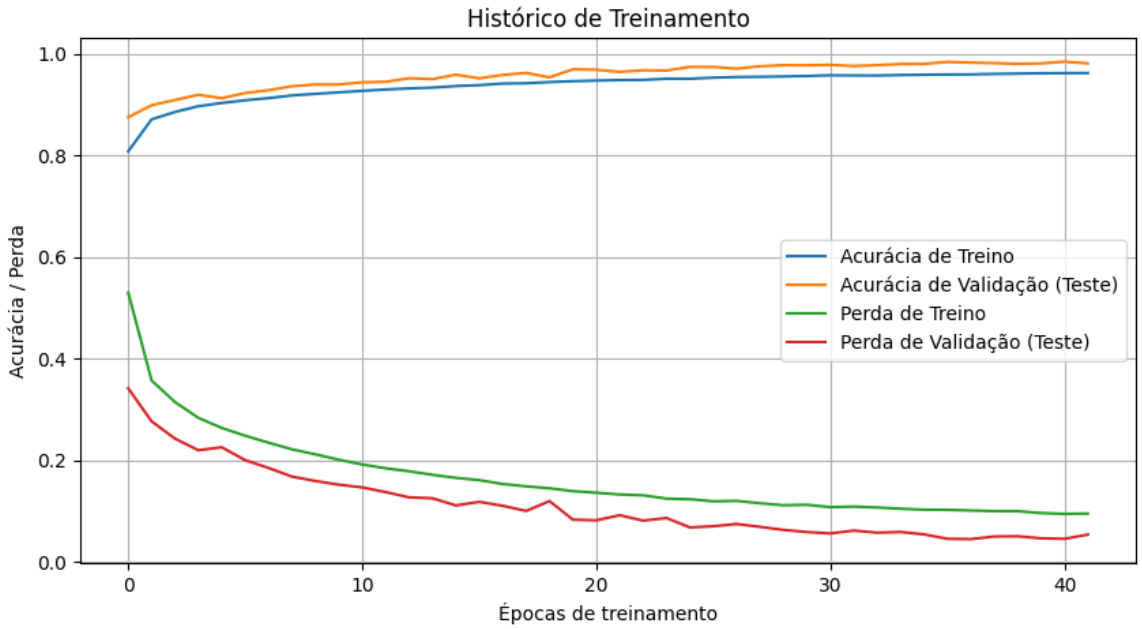
```
   - Formato da Matriz de Pesos (Wij): (1600, 128)
   - Formato do Vetor de Bias (bi): (128,)

--- Camada 7: dense_1 ---
   - Formato da Matriz de Pesos (Wij): (128, 10)
   - Formato do Vetor de Bias (bi): (10,)
```

3. Desempenho do Treinamento

Acurácia final na base de testes: 98.44%

4. Gráfico do Histórico de Treinamento



5. Resultados da Predição na Base de Testes

Legenda das Classes:

Camisetas/Top
Calça
Suéter
Vestidos
Casaco
Sandálias
Camisas
Tênis
Bolsa
Botas

Previsões Finais (Amostra):

	Previsão de Classe
0	Botas
1	Suéter
2	Calça
3	Calça
4	Camisas
5	Calça
6	Casaco
7	Camisas
8	Sandálias
9	Tênis
10	Casaco
11	Sandálias
12	Tênis
13	Vestidos
14	Casaco
15	Calça
16	Suéter
17	Casaco
18	Bolsa
19	Camisetas/Top
20	Suéter
21	Sandálias
22	Tênis
23	Botas
24	Calça
25	Casaco
26	Camisas
27	Camisetas/Top
28	Botas
29	Vestidos

6. Código Fonte (treinamento.py)

```
import tensorflow as tf
from tensorflow import keras
# [NOVO] Importando as camadas necessárias para a CNN
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import numpy as np
import pandas as pd
import warnings
import io
import os
from contextlib import redirect_stdout
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')

# --- 1. CARREGAMENTO E PRÉ-PROCESSAMENTO DOS DADOS ---

print("Carregando bases de treino e teste...")
try:
    dados_treino = pd.read_csv('./content/Treino Reconhecimento de Roupas.csv', header=None)
    dados_teste = pd.read_csv('./content/Teste Reconhecimento de Roupas.csv', header=None)
except FileNotFoundError:
     print("\nERRO: Verifique se os arquivos 'Treino Reconhecimento de Roupas.csv' e 'Teste Reconhecimento de
Roupas.csv' estão na pasta './content/'.")
    exit()

X_treino_df = dados_treino.iloc[:, 1:]
y_treino_series = dados_treino.iloc[:, 0]
X_teste_df = dados_teste.iloc[:, 1:]
y_teste_series = dados_teste.iloc[:, 0]

X_treino = X_treino_df.values
y_treino = y_treino_series.values
X_teste = X_teste_df.values
y_teste = y_teste_series.values

nomes_classes = [
    'Camisetas/Top', 'Calça', 'Suéter', 'Vestidos', 'Casaco',
    'Sandálias', 'Camisas', 'Tênis', 'Bolsa', 'Botas'
]
num_classes = len(nomes_classes)

# Normalização dos dados (continua igual)
X_treino = X_treino / 255.0
X_teste = X_teste / 255.0

# --- [NOVO] REDIMENSIONAMENTO PARA O FORMATO DA IMAGEM ---
# As CNNs esperam um formato (altura, largura, canais_de_cor)
# Nossas imagens são 28x28 com 1 canal (escala de cinza).
X_treino = X_treino.reshape(-1, 28, 28, 1)
X_teste = X_teste.reshape(-1, 28, 28, 1)

print("Carregamento e pré-processamento concluídos.")


# --- 2. CONSTRUÇÃO DO MODELO DE REDE NEURAL (ARQUITETURA CNN) ---

modelo = keras.Sequential([
    # Camada Convolucional: aplica 32 filtros (3x3) para extrair características
```

```
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    # Camada de Pooling: reduz a dimensão da imagem (para 14x14), mantendo as características mais importantes
    MaxPooling2D((2, 2)),

    # Segundo bloco convolucional para aprender características mais complexas
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Achata a matriz de características para um vetor, para conectar com as camadas densas
    Flatten(),

    # Camadas densas para a classificação final (similar ao modelo anterior)
    Dense(128, activation='relu'),
    Dropout(0.5), # Dropout é ainda mais importante em CNNs
    Dense(num_classes, activation='softmax')
])

modelo.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])


# --- 3. TREINAMENTO DO MODELO ---

callbacks = [
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1),
         keras.callbacks.ModelCheckpoint('melhor_modelo.keras',  save_best_only=True,  monitor='val_accuracy',
mode='max')
]

print("\nIniciando o treinamento do modelo CNN...")
resultado_treinamento = modelo.fit(X_treino, y_treino, batch_size=32, epochs=50,
                                   validation_data=(X_teste, y_teste), verbose=2,
                                   callbacks=callbacks)
print("Treinamento finalizado.")


# --- 4. AVALIAÇÃO E PREPARAÇÃO DOS RESULTADOS ---

print("\nCarregando o melhor modelo salvo e preparando os resultados...")
modelo = keras.models.load_model('melhor_modelo.keras')

perda_teste, acuracia_teste = modelo.evaluate(X_teste, y_teste, verbose=0)
print(f'\nAcurácia final na base de testes (melhor modelo): {acuracia_teste:.2%}')

# O restante do código para salvar os resultados permanece igual...
string_io_summary = io.StringIO()
with redirect_stdout(string_io_summary):
    modelo.summary()
resumo_modelo_texto = string_io_summary.getvalue()

string_io_details = io.StringIO()
string_io_details.write("TIPO DE TREINAMENTO:\n")
string_io_details.write("   - Tipo: Aprendizado Supervisionado\n")
string_io_details.write("   - Algoritmo: Backpropagation\n")
string_io_details.write(f"   - Otimizador: Adam (configuração padrão)\n")
string_io_details.write(f"   - Função de Perda (Loss): Sparse Categorical Crossentropy\n\n")
string_io_details.write("PESOS (Wij) E BIAS (bi) POR CAMADA:\n")
for i, camada in enumerate(modelo.layers):
    if len(camada.get_weights()) > 0:
```

```
        pesos, biases = camada.get_weights()
        string_io_details.write(f"\n--- Camada {i}: {camada.name} ---\n")
        string_io_details.write(f"   - Formato da Matriz de Pesos (Wij): {pesos.shape}\n")
        string_io_details.write(f"   - Formato do Vetor de Bias (bi): {biases.shape}\n")
detalhes_rede_texto = string_io_details.getvalue()

probabilidades = modelo.predict(X_teste)
indices_preditos = np.argmax(probabilidades, axis=1)
classes_preditas = [nomes_classes[i] for i in indices_preditos]
Y_Resposta = pd.DataFrame(data=classes_preditas, columns=['Previsão de Classe'])


# --- 5. GERAÇÃO E SALVAMENTO DO GRÁFICO DE TREINAMENTO ---

print("Gerando gráfico do histórico de treinamento...")
plt.figure(figsize=(10, 5))
plt.plot(resultado_treinamento.history['accuracy'], label='Acurácia de Treino')
plt.plot(resultado_treinamento.history['val_accuracy'], label='Acurácia de Validação (Teste)')
plt.plot(resultado_treinamento.history['loss'], label='Perda de Treino')
plt.plot(resultado_treinamento.history['val_loss'], label='Perda de Validação (Teste)')
plt.title('Histórico de Treinamento')
plt.ylabel('Acurácia / Perda')
plt.xlabel('Épocas de treinamento')
plt.legend()
plt.grid(True)

os.makedirs('./result', exist_ok=True)
plt.savefig('./result/Historico_de_treinamento.png')
print("Gráfico salvo em ./result/Historico_de_treinamento.png")


# --- 6. SALVANDO OS RESULTADOS EM ARQUIVOS EXTERNOS ---

print("\nSalvando todos os resultados em arquivos na pasta './result'...")
with open('./result/resumo_modelo.txt', 'w', encoding='utf-8') as f:
    f.write(resumo_modelo_texto)
with open('./result/detalhes_rede.txt', 'w', encoding='utf-8') as f:
    f.write(detalhes_rede_texto)
with open('./result/legenda_classes.txt', 'w', encoding='utf-8') as f:
    f.write('\n'.join(nomes_classes))
Y_Resposta.to_csv('./result/previsoes_finais.csv', index=False)
with open('./result/acuracia_teste.txt', 'w', encoding='utf-8') as f:
    f.write(f"{acuracia_teste:.2%}")
print("\nTodos os arquivos foram salvos com sucesso na pasta './result'.")
```