Relatório de Treinamento de Rede Neural Convolucional (CNN) para identificar Algarismos

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)

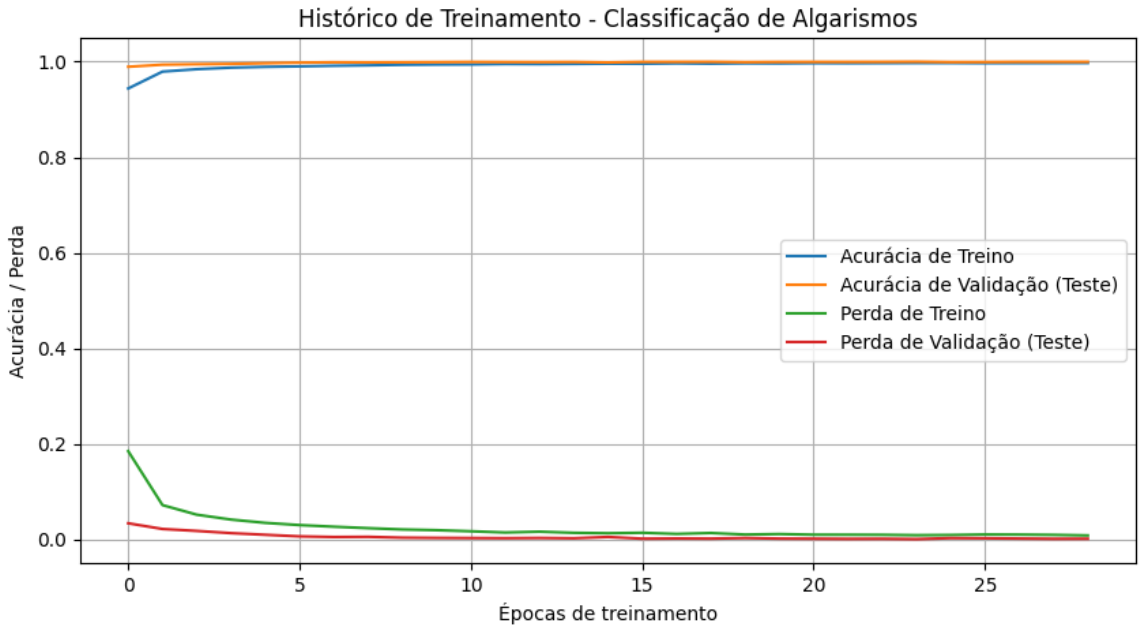
```
TIPO DE TREINAMENTO:
   - Tipo: Aprendizado Supervisionado
   - Algoritmo: Backpropagation
   - Otimizador: Adam (configuração padrão)
   - Função de Perda (Loss): Sparse Categorical Crossentropy

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

--- Camada 1: conv2d (Conv2D) ---
   - Neurônios/Filtros: 32
   - Formato da Matriz de Pesos (Wij): (3, 3, 1, 32)
   - Formato do Vetor de Bias (bi): (32,)

--- Camada 2: max_pooling2d (MaxPooling2D) ---

--- Camada 3: conv2d_1 (Conv2D) ---
   - Neurônios/Filtros: 64
```

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

--- Camada 4: max_pooling2d_1 (MaxPooling2D) ---

--- Camada 5: flatten (Flatten) ---

--- Camada 6: dense (Dense) ---
   - Neurônios/Filtros: 128
   - Formato da Matriz de Pesos (Wij): (1600, 128)
   - Formato do Vetor de Bias (bi): (128,)

--- Camada 7: dropout (Dropout) ---

--- Camada 8: dense_1 (Dense) ---
   - Neurônios/Filtros: 10
   - 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: 100.00%

4. Gráfico do Histórico de Treinamento



5. Resultados da Predição na Base de Testes

Legenda das Classes:

```
Legenda: Índice -> Algarismo
0 -> 0
1 -> 1
2 -> 2
3 -> 3
4 -> 4
5 -> 5
6 -> 6
7 -> 7
8 -> 8
9 -> 9
```

Previsões Finais (Amostra):

	Previsao_de_Algarismo
0	7
1	2
2	1
3	0
4	4
5	1
6	4
7	9
8	5
9	9
10	0
11	6
12	9
13	0
14	1
15	5
16	9
17	7
18	3
19	4
20	9
21	6
22	6
23	5
24	4
25	0
26	7
27	4
28	0
29	1

6. Código Fonte (treinamento.py)

```
import tensorflow as tf
from tensorflow import keras
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 de ALGARISMOS...")
try:
    # [AJUSTE] Alterado o nome dos arquivos para o dataset de algarismos.
    # O 'sep=',' é adicionado para garantir que o pandas use a vírgula como separador.
    dados_treino = pd.read_csv('./content/AlgarismosTreinamento.txt', header=None, sep=',')
    dados_teste = pd.read_csv('./content/AlgarismosTeste.txt', header=None, sep=',')
except FileNotFoundError:
    print("\nERRO: Verifique se os arquivos 'AlgarismosTreinamento.txt' e 'AlgarismosTeste.txt' estão na pasta
'./content/'.")
    exit()

# O restante da lógica de separação de dados permanece idêntico
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

# [AJUSTE] Alterada a lista de nomes das classes para representar os algarismos de 0 a 9.
nomes_classes = [
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
]
num_classes = len(nomes_classes)

# Normalização dos dados (idêntico ao script anterior)
X_treino = X_treino / 255.0
X_teste = X_teste / 255.0

# Redimensionamento para o formato da imagem para a CNN (idêntico ao script anterior)
# Formato: (numero_de_imagens, altura, largura, canais_de_cor)
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) ---
# A mesma arquitetura CNN é muito eficaz para este problema também.
# Topologia: Conv(32)->Pool->Conv(64)->Pool->Flatten->Dense(128)->Dropout->Dense(10)
```

```
modelo = keras.Sequential([
    # Camada 1: Convolucional + Pooling
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),

    # Camada 2: Convolucional + Pooling
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Camada 3: Achatamento para preparar para as camadas densas
    Flatten(),

    # Camada 4: Densa (totalmente conectada) com Dropout para evitar overfitting
    Dense(128, activation='relu'),
    Dropout(0.5),

    # Camada 5: Saída, com um neurônio para cada classe (0-9)
    Dense(num_classes, activation='softmax')
])

# Compilação do modelo (idêntico ao script anterior)
modelo.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])


# --- 3. TREINAMENTO DO MODELO ---
# Tipo de treinamento: Aprendizado Supervisionado com Backpropagation e otimizador Adam.

callbacks = [
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True),
    # [AJUSTE] Alterado o nome do arquivo do melhor modelo para evitar conflitos.
                    keras.callbacks.ModelCheckpoint('melhor_modelo_algarismos.keras',     save_best_only=True,
monitor='val_accuracy', mode='max')
]

print("\nIniciando o treinamento do modelo CNN para classificar algarismos...")
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...")
# [AJUSTE] Carregando o modelo específico de algarismos.
modelo = keras.models.load_model('melhor_modelo_algarismos.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%}')

# A geração dos relatórios textuais permanece idêntica
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("TOPOLOGIA, PESOS (Wij) E BIAS (bi) POR CAMADA:\n")
for i, camada in enumerate(modelo.layers):
    string_io_details.write(f"\n--- Camada {i+1}: {camada.name} ({camada.__class__.__name__}) ---\n")
    if camada.count_params() > 0:
            string_io_details.write(f"    -  Neurônios/Filtros: {camada.units  if  hasattr(camada,  'units')  else
camada.filters}\n")
    if len(camada.get_weights()) > 0:
        pesos, biases = camada.get_weights()
        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=['Previsao_de_Algarismo'])


# --- 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 - Classificação de Algarismos')
plt.ylabel('Acurácia / Perda')
plt.xlabel('Épocas de treinamento')
plt.legend()
plt.grid(True)

# Cria o diretório de resultados se ele não existir
os.makedirs('./result', exist_ok=True)
# [AJUSTE] Salvando os resultados em uma nova pasta para não misturar com o exercício anterior.
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('Legenda: Índice -> Algarismo\n')
    for i, nome in enumerate(nomes_classes):
        f.write(f'{i} -> {nome}\n')
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'.")
```