Relatório Final - Reconhecimento de Dígitos Numéricos com Keras e MNIST

Este relatório descreve o desenvolvimento de um sistema de reconhecimento de dígitos manuscritos (0 a 9) usando redes neurais com Keras e o dataset MNIST. O foco é apresentar o funcionamento dos códigos de treinamento e predição, e as instruções para sua execução.

1. Objetivo

O projeto visa criar e treinar um modelo de rede neural para reconhecer dígitos numéricos de 0 a 9 a partir de imagens, utilizando o dataset MNIST.

2. Funcionamento Detalhado do Código

2.1 Código de Treinamento da IA

Este script treina uma rede neural para reconhecer dígitos escritos à mão (de 0 a 9) usando o famoso dataset MNIST. O processo é dividido em 5 etapas principais:

Configuração e Importações

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import os

# --- Parâmetros Principais ---
NUM_CLASSES = 10
INPUT_SHAPE = (28, 28)
EPOCHS = 10
BATCH_SIZE = 64
MODEL_PATH = "models/modelo_mnist_custom.h5"
```

Importa as bibliotecas necessárias (TensorFlow, Keras, NumPy) e define os parâmetros essenciais do treinamento, como o número de épocas, o tamanho do lote (batch size) e onde salvar o modelo final. Manter isso no início facilita a experimentação.

Carregamento e Preparação dos Dados

```
# Carrega o dataset MNIST
(x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()

# Normaliza as imagens (escala de 0 a 1)
```

```
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# Converte os rótulos para o formato one-hot encoding
y_train = keras.utils.to_categorical(y_train, NUM_CLASSES)
y_test = keras.utils.to_categorical(y_test, NUM_CLASSES)
```

Carrega os dados de treino e teste do MNIST. As imagens são **normalizadas** (valores de pixel de 0-255 para 0-1) para ajudar o modelo a treinar melhor. Os rótulos (0, 1, 2...) são transformados em vetores (ex: 5 -> [0,0,0,0,0,1,0,0,0,0]), que é o formato esperado pela rede neural.

Construção da Arquitetura da Rede Neural

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=INPUT_SHAPE),
    keras.layers.Dense(256, activation="relu"),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(NUM_CLASSES, activation="softmax")
])
```

Define a estrutura da nossa IA. É um modelo sequencial simples:

1. **Flatten**: Transforma a imagem 28x28 em um vetor linear.
2. **Dense**: Camadas de neurônios totalmente conectadas que aprendem os padrões. A ativação relu é uma escolha padrão e eficiente.
3. **Dropout**: "Desliga" alguns neurônios durante o treino para evitar que o modelo apenas "decore" os dados (overfitting).
4. **Dense (final)**: A camada de saída com 10 neurônios (um para cada dígito) e ativação softmax, que calcula a probabilidade de a imagem pertencer a cada classe.

Compilação do Modelo

```
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

Prepara o modelo para o treinamento, definindo três coisas cruciais:

- **optimizer="adam"**: O algoritmo que atualiza o modelo para melhorar seu desempenho (minimizar o erro). Adam é uma escolha robusta e popular.
- **loss="categorical_crossentropy"**: A função que mede o "erro" do modelo. O objetivo é minimizar esse valor.
- **metrics=["accuracy"]**: A métrica que queremos monitorar. Aqui, focamos na acurácia (percentual de acertos).

Treinamento e Avaliação

```
# Callbacks para salvar o melhor modelo e parar o treino se não houver melhora
callbacks = [
    keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
    keras.callbacks.ModelCheckpoint(filepath=MODEL_PATH, save_best_only=True)
]

# Inicia o treinamento
model.fit(x_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE,
          validation_split=0.1, callbacks=callbacks)

# Avalia o modelo final com dados nunca vistos antes
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Acurácia no teste: {accuracy*100:.2f}%")
```

Esta é a etapa final.

- **model.fit()**: Inicia o processo de treinamento com os dados de treino. 10% desses dados são usados para validação a cada época.
- **callbacks**: São "ajudantes" inteligentes. Um salva a melhor versão do modelo e o outro para o treinamento se a performance parar de melhorar, economizando tempo.
- **model.evaluate()**: Após o treino, testa o modelo no conjunto de teste para verificar sua performance real em dados novos, fornecendo a acurácia final.

2.2 Código de Treinamento da IA

Configuração e Carregamento do Modelo

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

```
import matplotlib.pyplot as plt
import os

MODEL_NAME = "modelo_mnist_custom.h5"

# --- Verifica se o modelo existe e o carrega ---
if not os.path.exists(MODEL_NAME):
    print(f"ERRO: O arquivo do modelo '{MODEL_NAME}' não foi
encontrado.")
    exit()

model = keras.models.load_model(MODEL_NAME)
model.summary()
```

Primeiro, importa as bibliotecas necessárias. Em seguida, ele verifica se o arquivo do modelo treinado (modelo_mnist_custom.h5) existe. Se existir, ele o carrega na memória usando keras.models.load_model() e exibe um resumo de sua arquitetura. Caso contrário, o programa para com um erro.

Preparação dos Dados de Teste

```
# Carrega apenas o conjunto de teste do MNIST
(_, _), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normaliza as imagens (escala de 0 a 1)
x_test = x_test.astype("float32") / 255.0
```

Carrega o conjunto de dados de teste do MNIST, que o modelo nunca viu durante o treinamento. Assim como no script de treino, as imagens são **normalizadas** para a escala de 0 a 1, garantindo que os dados de entrada tenham o mesmo formato que os dados de treinamento.

Seleção da Imagem e Predição

```
# Escolhe uma imagem aleatória do conjunto de teste
index_to_predict = np.random.randint(0, len(x_test))
img_to_predict = x_test[index_to_predict]
true_label = y_test[index_to_predict]

# Prepara a imagem para o modelo e faz a predição
input_for_model = np.expand_dims(img_to_predict, axis=0)
predictions = model.predict(input_for_model)
```

```
# Extrai o resultado com maior probabilidade
predicted_label = np.argmax(predictions[0])
confidence = np.max(predictions[0]) * 100
```

Seleciona uma imagem aleatória do conjunto de teste para ser o nosso desafio.
A imagem é preparada para ter o formato que o modelo espera (adicionando uma dimensão extra).
model.predict() executa a imagem através da rede neural.
A rede retorna um vetor de 10 probabilidades. O np.argmax() encontra qual dígito (índice) teve a maior probabilidade, que se torna a previsão final.

Exibição dos Resultados

```
# Mostra a imagem com o resultado
plt.imshow(img_to_predict, cmap="gray")
plt.title(
    f"Rótulo Real: {true_label}\nPredição: {predicted_label}
(Confiança: {confidence:.2f}%)",
    color="green" if predicted_label == true_label else "red",
)
plt.axis("off")
plt.show()

# Imprime os resultados no terminal
print(f"Rótulo REAL da imagem: {true_label}")
print(f"Dígito PREVISTO pelo modelo: {predicted_label}")
```

Apresenta o resultado da predição de forma clara para o usuário.

- Uma janela do Matplotlib exibe a imagem original, o dígito que ela realmente representa (Rótulo Real) e o que o modelo previu (Predição), junto com a confiança.
- O título fica **verde** se o modelo acertou e **vermelho** se errou, facilitando a interpretação visual.
- As mesmas informações são impressas no console para um registro detalhado.

3. Instruções para Executar o Treinamento

1. Clone o Repositório: git clone https://github.com/HenriqueDC2003/DeepLearning_Digitos_Numericos.git e cd DeepLearning_Digitos_Numericos.
2. Crie e Ative o Ambiente Virtual (.venv):

- python -m venv .venv
- Windows: .\.venv\Scripts\activate
- macOS/Linux: source ./.venv/bin/activate

3. Instale as Dependências: pip install -r requirements.txt (com o .venv ativado).
4. Execute o Script: Vá para cd src e execute python treino_mnist.py. O modelo modelo_mnist_custom.h5 será salvo em models/.

4. Instruções para Realizar o Reconhecimento

1. Pré-requisitos: Modelo models/modelo_mnist_custom.h5 deve existir.
2. Ative o Ambiente Virtual: Se não estiver ativo, ative-o (ver Seção 3).
3. Execute o Script: Vá para cd src e execute python prever_mnist.py. Uma imagem aleatória do MNIST será exibida com sua predição.