CLASSIFICAÇÃO
DE
DESENHOS ANIMADOS

Equipe

Camila Vanessa de Matos Sousa

Pedro Rafael Pereira de Oliveira

Contextualização

- Para que serve?
- É importante?
- Não perca o tempo que você não tem.

←——(Vocês aí tudo triste porque não sabem o nome do desenho)

Objetivo Principal

- Objetivo
- Aplicação Prática
- Desafios

Base de Dados (Construção)

- 1ª Tentativa: Salvar da Internet
- 2ª Tentativa: Banco de Dados pronto
- 3ª Tentativa: Salvar frames do episódio (com intervalo)

Base de Dados (Construção)

Salvar os frames com intervalo x

```python
import cv2
import os

# Abra o vídeo
video_path = 'Bob Esponja _ O Estrangulador contra o Bob Esponja! _ Bob Esponja em Português.mp4'
cap = cv2.VideoCapture(video_path)

# Verifique se o vídeo foi aberto com sucesso
if not cap.isOpened():
    print("Erro ao abrir o vídeo.")
    exit()

# Crie uma pasta para armazenar os frames
output_folder = 'Bob'
os.makedirs(output_folder, exist_ok=True)

# Inicialize um contador para os frames
frame_count = 1

# Intervalo de frames a serem salvos
frame_interval = 300

# Loop para capturar e salvar frames a cada intervalo
while True:
    ret, frame = cap.read()
    if not ret:
        break  # Sai do loop quando todos os frames foram lidos

    # Verifique se o frame atual deve ser salvo com base no intervalo
    if frame_count % frame_interval == 0:
        # Salve o frame como uma imagem na pasta de saída
        frame_filename = os.path.join(output_folder, f'Bob_{frame_count:04d}.jpg')
        cv2.imwrite(frame_filename, frame)

    frame_count += 1

# Libere o objeto de captura e finalize
cap.release()
cv2.destroyAllWindows()

print(f"{frame_count} frames salvos em '{output_folder}' com intervalo de {frame_interval} frames.")
```

Vê se há imagens semelhantes

```python
!pip install imagehash
from PIL import Image
import imagehash
import os

pasta_imagens = 'Bob'

# Lista para armazenar os hashes das imagens
hashes = []

# Dicionário para armazenar imagens semelhantes
imagens_semelhantes = {}

# Função para calcular o hash de uma imagem
def calcular_hash(imagem):
    return str(imagehash.dhash(imagem))

# Iterar sobre as imagens na pasta
for arquivo in os.listdir(pasta_imagens):
    if arquivo.endswith('.jpg'):  # Verifique se o arquivo é uma imagem
        caminho_imagem = os.path.join(pasta_imagens, arquivo)
        imagem = Image.open(caminho_imagem)
        hash_imagem = calcular_hash(imagem)

        # Verifique se o hash da imagem já existe na lista
        if hash_imagem in hashes:
            # Adicione o arquivo ao dicionário de imagens semelhantes
            if hash_imagem not in imagens_semelhantes:
                imagens_semelhantes[hash_imagem] = []
            imagens_semelhantes[hash_imagem].append(caminho_imagem)
        else:
            # Adicione o hash à lista de hashes
            hashes.append(hash_imagem)

# Agora, imagens_semelhantes contém grupos de imagens semelhantes
# Mantenha apenas uma imagem de cada grupo e exclua as outras
for hash, imagens in imagens_semelhantes.items():
    if len(imagens) > 1:
        # Mantenha apenas a primeira imagem no grupo (ou escolha uma específica)
        imagem_a_manter = imagens[0]
        for imagem_a_excluir in imagens[1:]:
            os.remove(imagem_a_excluir)

print("Imagens semelhantes foram removidas, deixando apenas uma de cada grupo.")
```

Metodologia

- Algoritmo escolhido: CNN (Convolutional Neural Network)
- Pré-processamento
- Divisão dos dados: k-fold

Arquitetura do modelo

- Convolutional Neural Network (CNN)
  - Camadas usadas
    - Padrão
    - Aprimorado/modificado

Treinamento do Modelo

- Treinamento do modelo
    - Tempo: Padrão X Aprimorado
- Métricas

Treinamento do Modelo

- Camadas utilizadas

(Algoritmo padrão)

Camadas usadas no algoritmo padrão

- Camada convolucional e de pooling
  model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
  model.add(tf.keras.layers.MaxPooling2D((2, 2)))
  model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(tf.keras.layersMaxPooling2D((2, 2)))
  model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
- Flatten para camadas densas
  model.add(tf.keras.layers.Flatten())
- Camada densa
  model.add(tf.keras.layers.Dense(64, activation='relu'))
- Camada de saída
  model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

(Algoritmo aprimorado)

Camadas usadas no algoritmo aprimorado

- Camadas convolucionais e de pooling
  model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(224, 224, 3)))
  model.add(tf.keras.layers.MaxPooling2D((2, 2)))
  model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D((2, 2)))
  model.add(tf.keras.layers.Conv2D(256, (3, 3), activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D((2, 2)))
- Flatten para camadas densas
  model.add(tf.keras.layers.Flatten())
- Camadas densas
  model.add(tf.keras.layers.Dense(512, activation='relu'))
  model.add(tf.keras.layers.Dropout(0.5)) # Dropout para reduzir o overfitting
  model.add(tf.keras.layers.Dense(512, activation='relu'))
  model.add(tf.keras.layers.Dropout(0.5)) # Dropout para reduzir o overfitting
- Camada de saída
  model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

Resultados

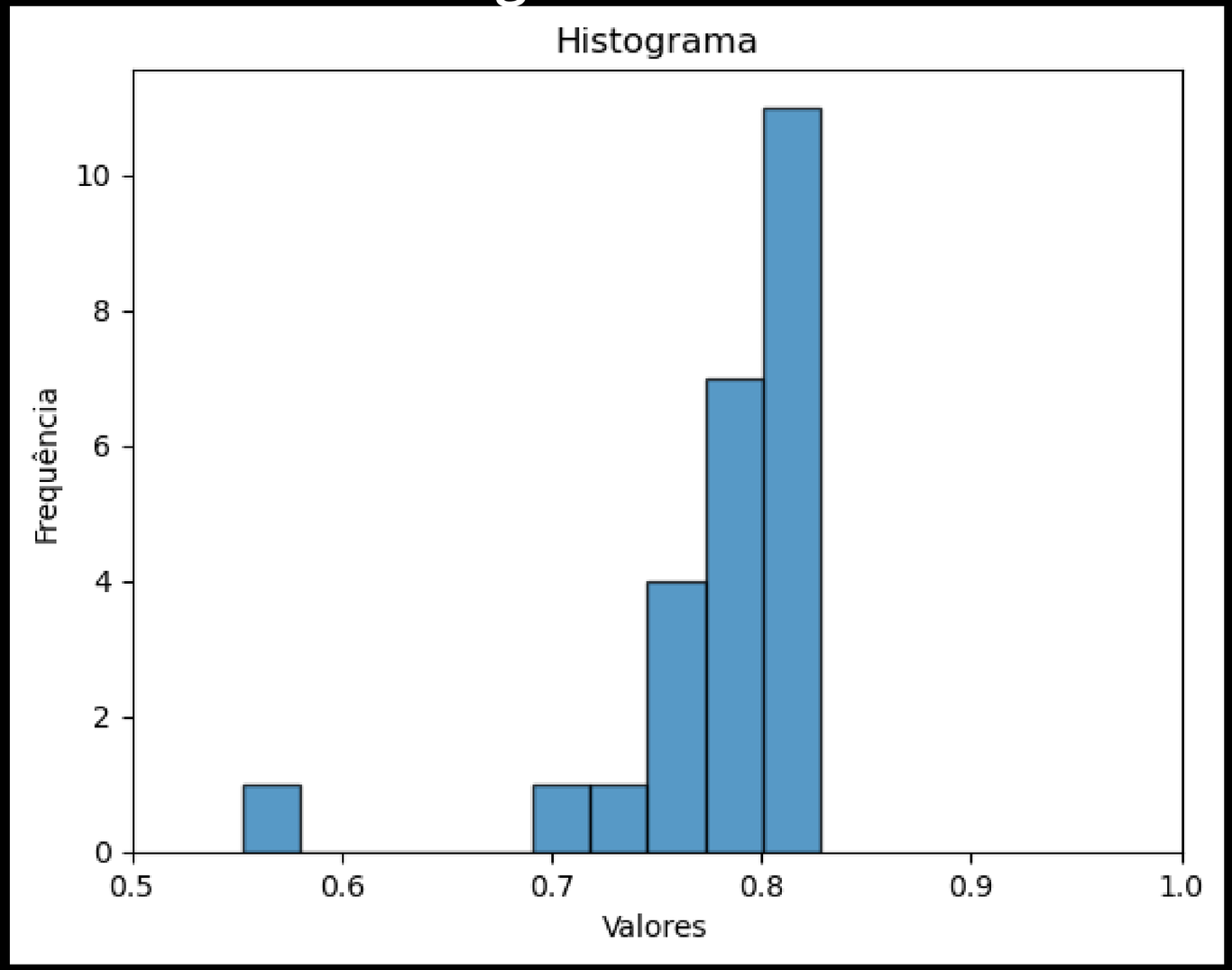
- Estatísticas e métricas obtidas
  - Acurácia
  - Matriz de confusão
  - Precisão por classe
  - Tempo

Tempo

- Padrão: 110,6 minutos
- Aprimorado: 325,4 minutos

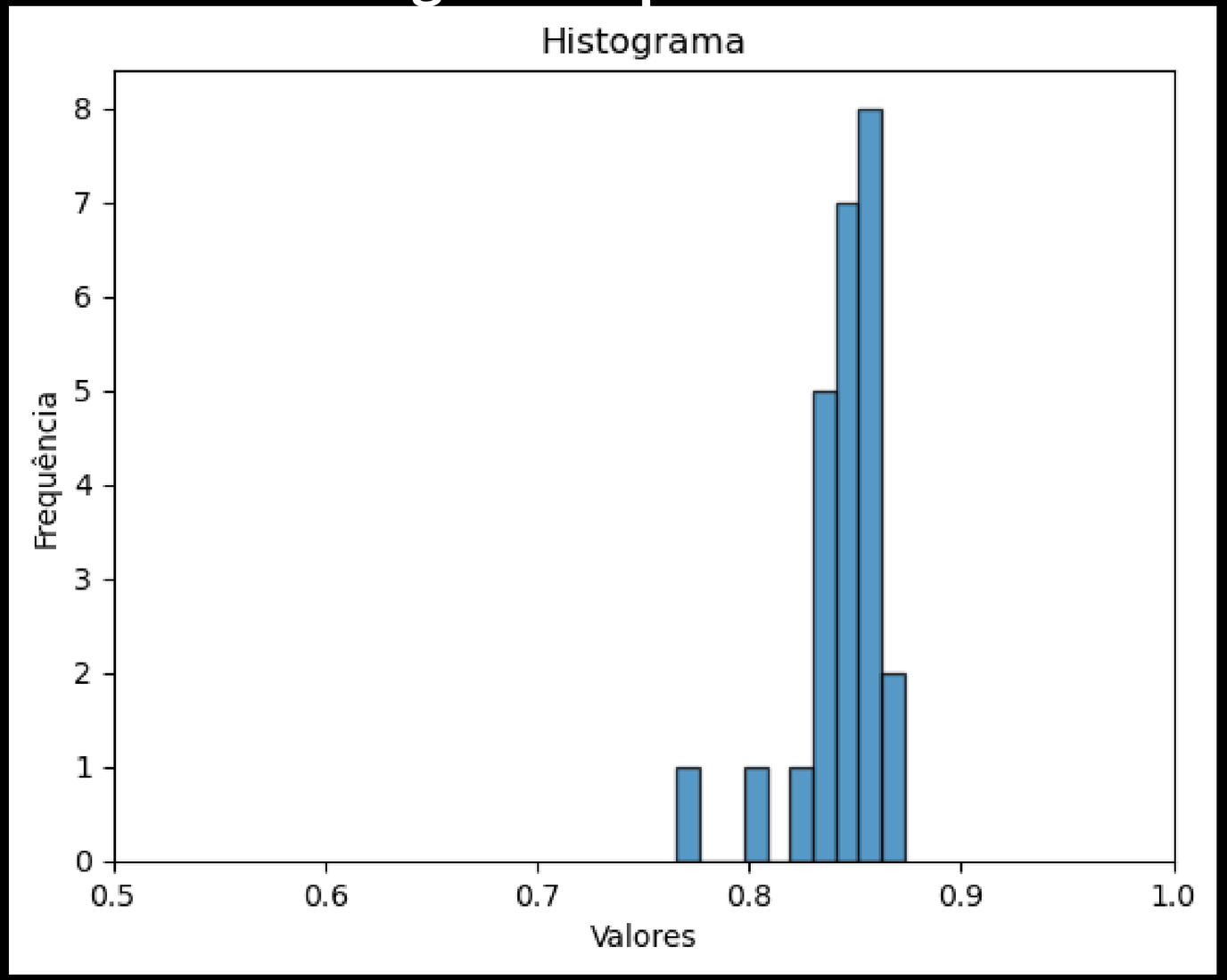
Acurácia de Validação

Histograma Padrão



Acurácia de Validação

Histograma Aprimorado



Teste Estatístico

- Teste de normalidade:
  - Hipótese Nula (H0): Os dados seguem uma distribuição normal
  - Hipótese Alternativa (H1): Os dados não seguem uma distribuição normal

```python
def teste_normalidade(sample):
    # Aplicando o teste de Shapiro-Wilk
    statistic, p_value = stats.shapiro(sample)

    # Verificando o resultado
    if p_value > 0.05:
        print("A amostra parece ser normal.")
    else:
        print("A amostra não parece ser normal.")

    plot_histogram(sample, xlim=(0.5, 1.0))
```

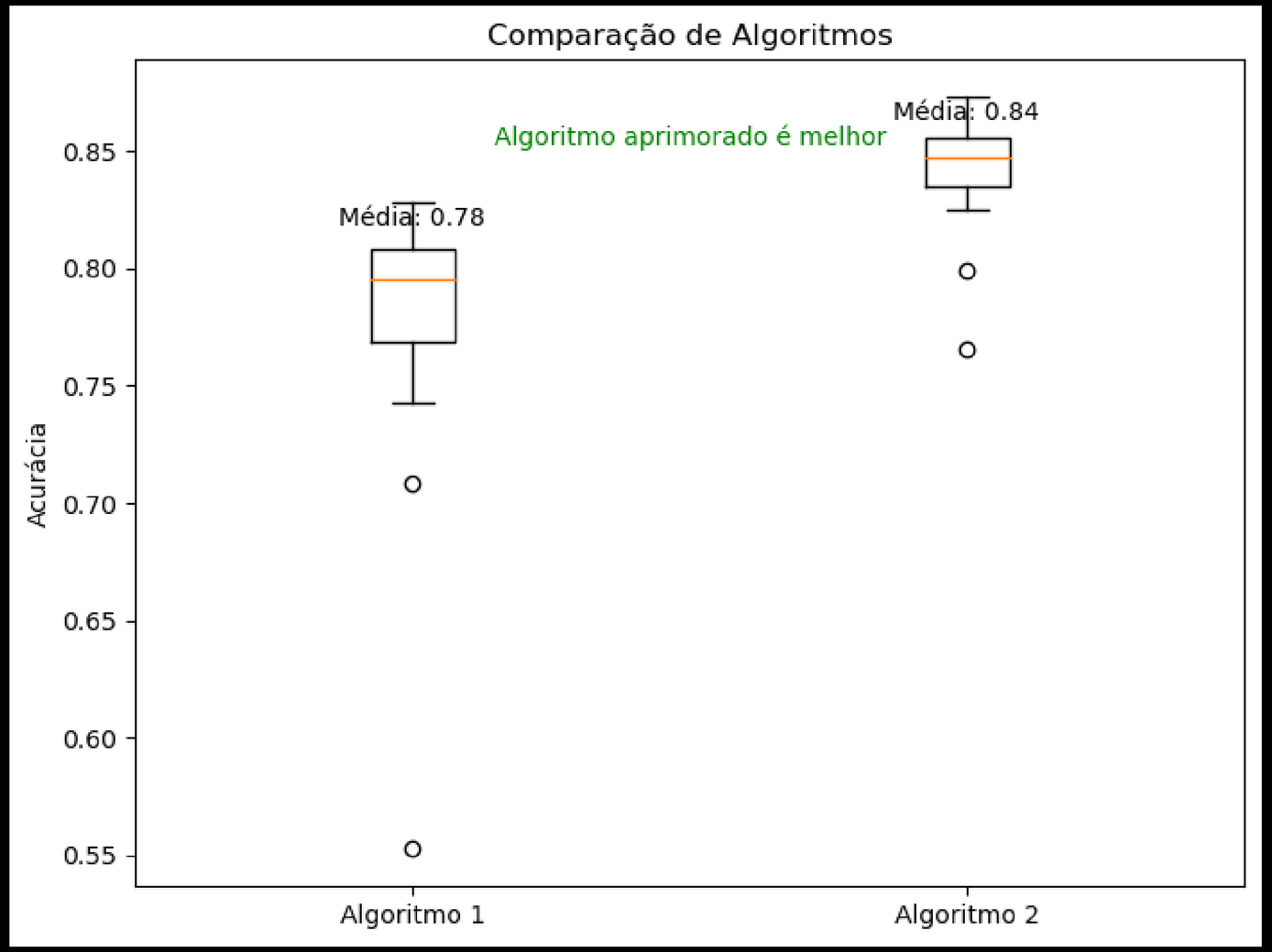
Teste Estatístico

- Qual o melhor?
- Hipótese Nula (H0): Não há diferença significativa entre as amostras
- Hipótese Alternativa (H1): Há diferença significativa

```python
def comparar_algoritmos(amostra1, amostra2, alpha=0.05):
    # Realize o teste de Mann-Whitney U
    stat, p = stats.mannwhitneyu(amostra1, amostra2, alternative='two-sided')

    # Plote um gráfico de caixa para cada amostra
    plt.figure(figsize=(8, 6))
    plt.boxplot([amostra1, amostra2], labels=['Algoritmo 1', 'Algoritmo 2'])
    plt.title('Comparação de Algoritmos')
    plt.ylabel('Acurácia')

    # Calcule medidas resumo
    media_amostra1 = round(sum(amostra1) / len(amostra1), 2)
    media_amostra2 = round(sum(amostra2) / len(amostra2), 2)

    # Adicione informações sobre a média no gráfico
    plt.text(1, max(amostra1) - 0.01, f'Média: {media_amostra1}', ha='center')
    plt.text(2, max(amostra2) - 0.01, f'Média: {media_amostra2}', ha='center')

    # Adicione resultado do teste ao gráfico
    if p < alpha:
        if media_amostra1 > media_amostra2:
            plt.text(1.5, max(max(amostra1), max(amostra2)) - 0.02, 'Algoritmo 1 é melhor', ha='center', color='green')
        else:
            plt.text(1.5, max(max(amostra1), max(amostra2)) - 0.02, 'Algoritmo 2 é melhor', ha='center', color='green')
    else:
        plt.text(1.5, max(max(amostra1), max(amostra2)) - 0.02, 'Não há diferença significativa', ha='center', color='red')

    plt.show()

    # Retorne o resultado do teste e qual algoritmo é considerado melhor ou se não há diferença significativa
    if p < alpha:
        if media_amostra1 > media_amostra2:
            return "Algoritmo padrão é melhor"
        else:
            return "Algoritmo aprimorado é melhor"
    else:
        return "Não há diferença significativa"
```

Teste Estatístico

- Qual o melhor?



TESTANDO AO VIVO

ACABOU,
ESTAMOS ABERTOS A PERGUNTAS,
SÓ NÃO QUEREMOS RESPONDER.