Instalação das Bibliotecas

Para começar, instalamos duas bibliotecas essenciais: **OpenCV** e **TensorFlow Hub**, que serão usadas para processar os vídeos e carregar o modelo pré-treinado de reconhecimento de ações.

```
!pip install -q opencv-python tensorflow_hub
```

Importação das Bibliotecas

Neste trecho, importamos todas as bibliotecas necessárias para processamento de vídeo, carregamento do modelo, manipulação de arquivos temporários e exibição de resultados no notebook. O print exibi a versão atual do TensorFlow instalada no ambiente, garantindo que é compatível com o modelo a ser carregado.

```
import tensorflow as tf
import tensorflow_hub as hub
from google.colab.patches import cv2_imshow
import re
import os
import tempfile
import ssl
import cv2
import numpy as np
from urllib import request
from IPython.display import HTML
from base64 import b64encode

print("TensorFlow version: ", tf.__version__)
```

```
TensorFlow version:  2.19.0
```

Carregando os Rótulos do Kinetics-400

O modelo I3D utiliza a lista de ações do dataset **Kinetics-400**, que contém 400 classes diferentes.
Neste trecho, fazemos o download dessa lista diretamente do repositório oficial e a carregamos em uma lista Python.

```
url_classes = 'https://raw.githubusercontent.com/deepmind/kinetics-i3d/master/data/label_map.txt'
with request.urlopen(url_classes) as file:
  label = [row.decode('utf-8').strip() for row in file.readlines()]

print("Total de classes: ", len(label))
print("Algumas classes: ", label[:10])
```

```
Total de classes:  400
Algumas classes:  ['abseiling', 'air drumming', 'answering questions', 'applauding', 'applying cream', 'archery', 'arm wrestling', 'arra
```

Configuração Inicial para Download dos Vídeos

Antes de processar qualquer vídeo, precisamos definir onde eles serão buscados e onde serão armazenados temporariamente no ambiente.
Neste trecho, criamos essas configurações e definimos uma função para listar vídeos conhecidos.

```
root_folder = 'https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/'
context = ssl._create_unverified_context()
cache_videos = tempfile.mkdtemp()
print("Pasta temporária para vídeos:", cache_videos)

def list_videos():
  known_videos = [
    'v_PizzaTossing_g23_c03.avi',
    'v_ApplyEyeMakeup_g01_c01.avi',
    'v_Archery_g01_c01.avi'
]
  return known_videos

print("Lista de vídeos conhecidos:", list_videos())
```

```
Pasta temporária para vídeos: /tmp/tmpy8j137dx
Lista de vídeos conhecidos: ['v_PizzaTossing_g23_c03.avi', 'v_ApplyEyeMakeup_g01_c01.avi', 'v_Archery_g01_c01.avi']
```

Função para Baixar e Armazenar o Vídeo Localmente

Esta função é responsável por fazer o download de um vídeo da internet e salvar em uma pasta temporária.
Caso o vídeo já tenha sido baixado antes, ela evita o download repetido.

```
def save_video(video_name):
  cache_path = os.path.join(cache_videos, video_name)
  if not os.path.exists(cache_path):
    url_path = request.urljoin(root_folder, video_name)
    print("Baixando de:", url_path)
    data = request.urlopen(url_path, context=context).read()
    with open(cache_path, 'wb') as f:
      f.write(data)
    print("Video salvo em:", cache_path)
  else:
    print("Vídeo já existe em cache:", cache_path)
  return cache_path

video_name = 'v_PizzaTossing_g23_c03.avi'
video_path = save_video(video_name)

print("Caminho final do vídeo:", video_path)
print("Arquivo existe?", os.path.exists(video_path))
```

```
Baixando de: https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_PizzaTossing_g23_c03.avi
Video salvo em: /tmp/tmpy8j137dx/v_PizzaTossing_g23_c03.avi
Caminho final do vídeo: /tmp/tmpy8j137dx/v_PizzaTossing_g23_c03.avi
Arquivo existe? True
```

Função para Carregar o Vídeo com OpenCV

Esta função lê um vídeo salvo em disco, extrai os frames, redimensiona para o tamanho esperado pelo modelo e converte tudo em um array NumPy normalizado entre 0 e 1.

```
def load_video(path, visualize=False):
  cap = cv2.VideoCapture(path)
  frames = []

  if not cap.isOpened():
    print("Não foi possível abrir o vídeo:", path)
    return np.array([])

  while True:
    connected, frame = cap.read()
    if not connected:
      break

    frame = cv2.resize(frame, (224, 224))
    frames.append(frame)

  cap.release()

  if not frames:
    print("Nenhum frame foi lido do vídeo.")
    return np.array([])

  frames = np.array(frames, dtype=np.float32)

  if visualize and frames.size > 0:
    cv2_imshow(frames[0].astype(np.uint8))

  return frames / 255.0

video_frames = load_video(video_path, visualize=False)
print("Formato do array de frames:", video_frames.shape)
```

```
Formato do array de frames: (148, 224, 224, 3)
```

Conversão e Visualização do Vídeo no Notebook

Este trecho verifica se o vídeo foi carregado corretamente e, caso positivo, converte o arquivo `.avi` para `.mp4` e exibe o vídeo dentro do notebook.

```
if video_frames.size > 0:
  output_mp4_path = "/content/output.mp4"
  !ffmpeg -y -i "$video_path" "$output_mp4_path"

  if os.path.exists(output_mp4_path):
    mp4 = open(output_mp4_path, 'rb').read()
    data_url = 'data:video/mp4;base64,' +b64encode(mp4).decode()

    display(HTML(f"""
      <video width="400" controls>
```

```
        <source src="{data_url}" type="video/mp4">
      </video>
      """))
  else:
    print("Falha ao criar o arquivo mp4")
```

Carregando o Modelo I3D do TensorFlow Hub

Nesta etapa, carregamos o modelo **I3D (Inflated 3D ConvNet)** pré-treinado no dataset **Kinetics-400** diretamente do **TensorFlow Hub**. Esse modelo é o responsável por fazer o reconhecimento de ações a partir dos frames do vídeo.

```
print("Carregando modelo do TensorFlow Hub (pode demorar um pouco)...")
model = hub.load('https://tfhub.dev/deepmind/i3d-kinetics-400/1').signatures['default']
print("Modelo carregado com sucesso!")
```

```
Carregando modelo do TensorFlow Hub (pode demorar um pouco)...
Modelo carregado com sucesso!
```

Preparando o Vídeo para o Modelo e Realizando a Inferência

Depois de carregar o vídeo e o modelo, precisamos ajustar o formato dos frames para o padrão esperado pelo I3D e então realizar a previsão da ação presente no vídeo.

```
if video_frames.size == 0:
  raise ValueError("O vídeo não possui frames. Verifique o caminho ou o download.")

test_video = tf.constant(video_frames, dtype=tf.float32)[tf.newaxis, ...]
print("Shape do video para o modelo:", test_video.shape)

outputs = model(test_video)
logits = outputs['default'][0]
probabilities = tf.nn.softmax(logits).numpy()
```

```
Shape do video para o modelo: (1, 148, 224, 224, 3)
```

Analisando os Resultados da Inferência

Agora que temos as probabilidades previstas para cada uma das 400 classes do modelo, precisamos identificar qual ação o modelo acredita estar sendo realizada no vídeo.

```
top_class = np.argmax(probabilities)
print("Classe mais provável (indice):", top_class)
print("Ação mais provável::", label[top_class])
print("Confiança: {:.2f}%".format(probabilities[top_class]*100))
print("\nTop 5 ações principais:")
top_idx = np.argsort(probabilities)[::-1][:5]
for i in top_idx:
  print(f"{label[i]:35}: {probabilities[i]*100:5.2f} %")
```

```
Classe mais provável (indice): 188
Ação mais provável:: making pizza
Confiança: 36.47%

Top 5 ações principais:
making pizza                        : 36.47 %
punching bag                        : 28.43 %
catching or throwing frisbee        :  6.93 %
pumping fist                        :  5.17 %
washing dishes                      :  4.70 %
```

Clique duas vezes (ou pressione "Enter") para editar