Taller Nº 3 - MediaPipe			
Asignatura	Tópicos Especiales - Visión Artificial	**Código**	0756
Profesor	José Carlos Rangel Ortiz		

1. Introducción

Este taller consiste en la presentación y uso de las librerías CustomTKinter[1] para generar interfaces de usuario y MediaPipe[2] la cual provee una variedad de modelos basados en ML para diferentes tareas en Visión Artificial.

1.1. MediaPipe

Librerías como MediaPipe permiten a los usuarios desarrollar tareas como segmentación de imágenes, detección facial, reconocimiento de objetos, reconocimiento de gestos entre otros.

MediaPipe es una biblioteca de código abierto de Google que proporciona soluciones de visión por computadora para aplicaciones en tiempo real. La biblioteca está diseñada para ser fácil de usar y escalable, y se puede utilizar para una variedad de tareas, como la detección de objetos, el seguimiento de manos y el reconocimiento facial.

Puede ser utilizada para una variedad de aplicaciones, entre las que se incluyen:[3].

Realidad aumentada y virtual : MediaPipe se puede utilizar para crear experiencias de realidad aumentada y virtual más inmersivas. Por ejemplo, se puede utilizar para detectar objetos en el mundo real y superponerlos con imágenes o videos virtuales.

Inteligencia artificial : MediaPipe se puede utilizar para desarrollar aplicaciones de inteligencia artificial que requieren procesamiento de imágenes en tiempo real. Por ejemplo, se puede utilizar para detectar emociones en las caras o para controlar dispositivos con gestos.

Robótica : MediaPipe se puede utilizar para desarrollar robots que puedan navegar por el mundo y realizar tareas de forma autónoma. Por ejemplo, se puede utilizar para detectar obstáculos o para identificar objetos.

MediaPipe ofrece una variedad de características que la hacen una biblioteca de visión por computadora poderosa y versátil. Estas características incluyen:

[1] CustomTKinter
[2] MediaPipe
[3] Generado por Gemini

Eficiencia : MediaPipe está diseñado para ser eficiente en términos de recursos, lo que lo hace ideal para aplicaciones en tiempo real.

Facilidad de uso : MediaPipe es fácil de usar, incluso para desarrolladores sin experiencia en visión por computadora.

Escalabilidad : MediaPipe se puede escalar para adaptarse a una variedad de necesidades.

Esta es una librería en constante desarrollo y que puede ser utilizada en tanto en aplicaciones de escritorio con Python, sistemas Android, iOS, Raspberry Pi y Aplicaciones Web. En la página web del proyecto se encuentran ejemplos, demos y notebooks para probar las diferentes aplicaciones de la librería.

1.2. CustomTKinter

CustomTKinter es una biblioteca de Python que proporciona una interfaz gráfica de usuario (GUI) basada en Tkinter[4]. La biblioteca está diseñada para ser fácil de usar y personalizable, y se puede utilizar para crear una variedad de aplicaciones de escritorio. Para comprender el funcionamiento de las interfaces en Python se puede consultar el siguiente enlace, el cual ofrece un introducción rápida a esta metodología de trabajo en Python.

CustomTKinter ofrece una serie de ventajas sobre Tkinter, entre las que se incluyen:

Facilidad de uso : CustomTKinter es más fácil de usar que Tkinter, ya que proporciona widgets y funciones predefinidos que facilitan la creación de interfaces gráficas de usuario complejas.

Personalización : CustomTKinter ofrece una amplia gama de opciones de personalización, lo que permite a los desarrolladores crear interfaces gráficas de usuario únicas y atractivas.

Compatibilidad : CustomTKinter es compatible con Tkinter, lo que significa que los desarrolladores pueden reutilizar código existente al migrar de Tkinter a CustomTKinter.

CustomTKinter se puede utilizar para crear una variedad de aplicaciones de escritorio, entre las que se incluyen:

Aplicaciones de productividad : CustomTKinter se puede utilizar para crear aplicaciones de productividad, como hojas de cálculo, procesadores de texto y editores de código.

Aplicaciones de entretenimiento : CustomTKinter se puede utilizar para crear aplicaciones de entretenimiento, como juegos, reproductores multimedia y herramientas de edición de fotos.

Aplicaciones educativas : CustomTKinter se puede utilizar para crear aplicaciones educativas, como cursos en línea, simulaciones y juegos educativos.

[4] TKinter

2. Detección Facial en Tiempo Real

Esta sección se enfocará en explicar las porciones relevantes del código suministrado para el taller. Este código crea una ventana con 3 botones para activar la entrada de vídeo al programa y para activar o desactivar la función de detección de los modelos de MediaPipe. Se estará trabajando con el archivo **guiMediaPipeFace.py**, disponible en los archivos del taller.

Si al ejecutar su programa surge un error de la librería *imutils* no encontrada, puede instalarla usando el siguiente código en un terminal.

```
pip install imutils
```

1. Iniciamos importando las librerías que se requieren para ejecutar la aplicación.

```
from PIL import Image
from PIL import ImageTk
import cv2
import imutils
import customtkinter
import numpy as np
from typing import Tuple, Union
import math
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
from mediapipe import solutions
```

2. Dentro del código se encuentran las funciones siguientes

   - `iniciar()`: Se encarga de inicializar la instancia global de la cámara o entrada de datos de la aplicación.
   - `visualizar()`: Luego de ejecutada la función iniciar, se invoca este método para tomar la entrada los frames de entrada de la cámara o vídeo y los configura para que sean visualizados en la interfaz. Esta función aplica las modificaciones a imágenes cuando el usuario active la función de detección.
   - `detectar()`: Esta función se encarga de activar o desactivar los procedimientos de detección en los frames enviados por la cámara.
   - `finalizar()`: Esta función detiene el flujo de imágenes que son enviados a la interfaz gráfica.
   - `modificar()`: Esta función se invoca cuando se activan las detecciones en las imagen enviada por la cámara. La llamada a esta función esta controlada por la variable Global Booleana `det`.

3. Los métodos `_normalized_to_pixel_coordinates` y `visualize` son definidas en el demo para Python de MediaPipe[5]. Estos métodos se encargan de dibujar el boundingBox sobre las caras detectadas en las imágenes de entrada.

4. EL método `detectorFace` utiliza el detector definido para buscar caras en las imágenes de entrada. Para permitir el procesamiento de la imagen de entrada se debe convertir la imagen a un formato `mp.Image` el cual es el utilizado por MediaPipe.

```
def detectorFace(img):
  global detector

  # Load the input image.
  image = mp.Image(image_format=mp.ImageFormat.SRGB, data=img)

  # Detect face from the input image.
  detection_result = detector.detect(image)

  # Process the classification result and visualize it.
  image_copy = np.copy(image.numpy_view())
  annotated_image = visualize(image_copy, detection_result)
  #rgb_annotated_image = cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB)
  return annotated_image
```

5. Los detectores de MediaPipe son modelos entrados y listos para su uso. Estos modelos son un archivo que se descarga de su respectiva página web y se debe cargar en el programa. Por eficiencia en la ejecución este archivo se debe cargar solo una vez al iniciar la aplicación ya que cargar el archivo en cada momento de uso aumenta la latencia en la respuesta en la aplicación. En este caso al ser un modelo de detección facial se utiliza el archivo **blaze_face_short_range.tflite**.

```
model_file = open('blaze_face_short_range.tflite', "rb")
model_data = model_file.read()
model_file.close()
```

6. Una vez cargado el modelo se debe proceder con la creación de la instancia de detector, para lo cual se requieren la información del modelo a utilizar. En este caso nuestro detector es una variable global que es utilizad por diferentes métodos en la aplicación. Lo cual se realiza con el siguiente código.

```
base_options = python.BaseOptions(model_asset_buffer=model_data)
options = vision.FaceDetectorOptions(base_options=base_options)
detector = vision.FaceDetector.create_from_options(options)
```

7. El siguiente código esta encargado de crear la GUI para la utilización del programa. Este fragmento crea una ventana con 3 botones y un label en el cual se ubica el resultado de

[5] Demo Face Detection

la detección. Para más información del diseño de esta interfaz puede consultar este enlace.

```
root = customtkinter.CTk()

btnIniciar = customtkinter.CTkButton(root, text="Iniciar", width=45, command=iniciar)
btnIniciar.grid(column=0, row=0, padx=5, pady=5)

btnFinalizar = customtkinter.CTkButton(root, text="Finalizar", width=45, command=finalizar)
btnFinalizar.grid(column=1, row=0, padx=5, pady=5)

btnMediaPipe = customtkinter.CTkButton(root, text="Detectar", width=45, command=detectar)
btnMediaPipe.grid(column=2, row=0, padx=5, pady=5)

lblVideo = customtkinter.CTkLabel(root, text="")
lblVideo.grid(column=0, row=1, columnspan=3)

root.resizable(width= False, height =False)

root.mainloop()
```

8. Luego de la ejecución nos aparecerá una ventana como se aprecia en la Figura 1.

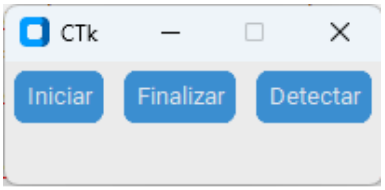

Figura 1: Ventana Inicial del programa

9. Cuando se ha presionado el botón de "Iniciar" nuestra interfaz mostrará los frames entrantes de la cámara o de un vídeo. Por lo cual, nuestra interfaz se vería como se muestra en la Figura 2.

10. Cuando se selecciona el botón "Detectar" el sistema inicia a procesar cada imagen de entrada y mostrar el resultado. Ver Figura 3

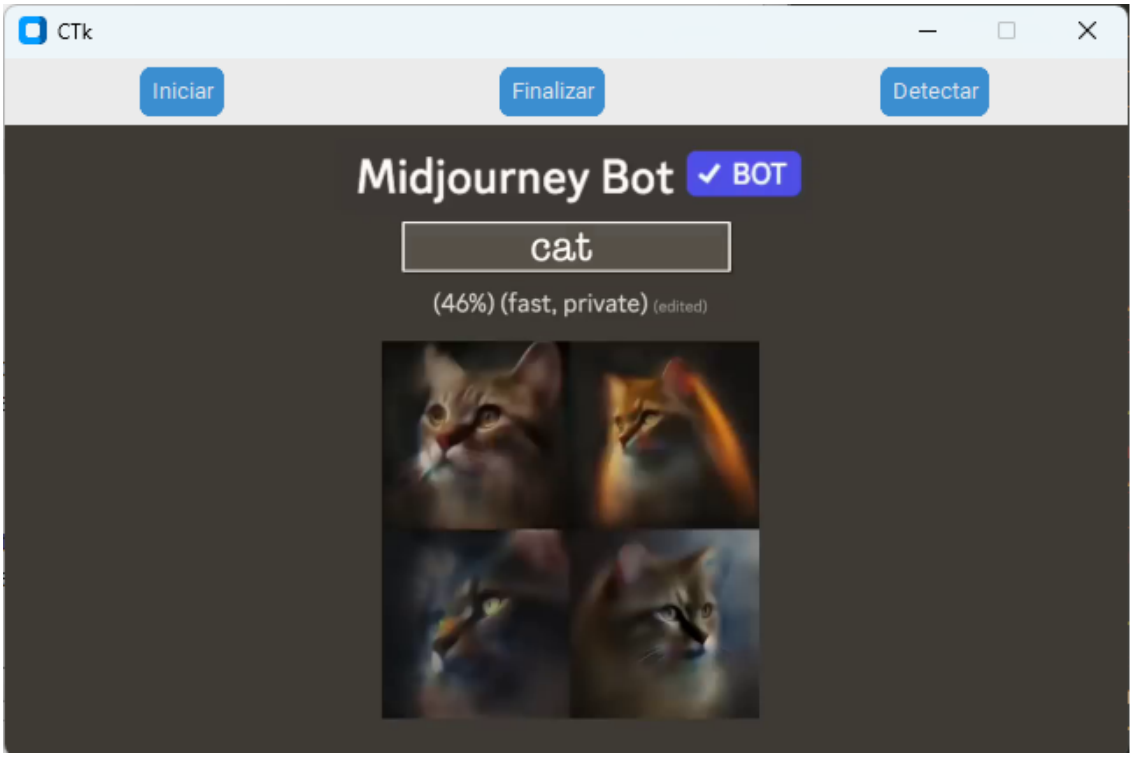

Figura 2: Interfaz mostrando los frames.

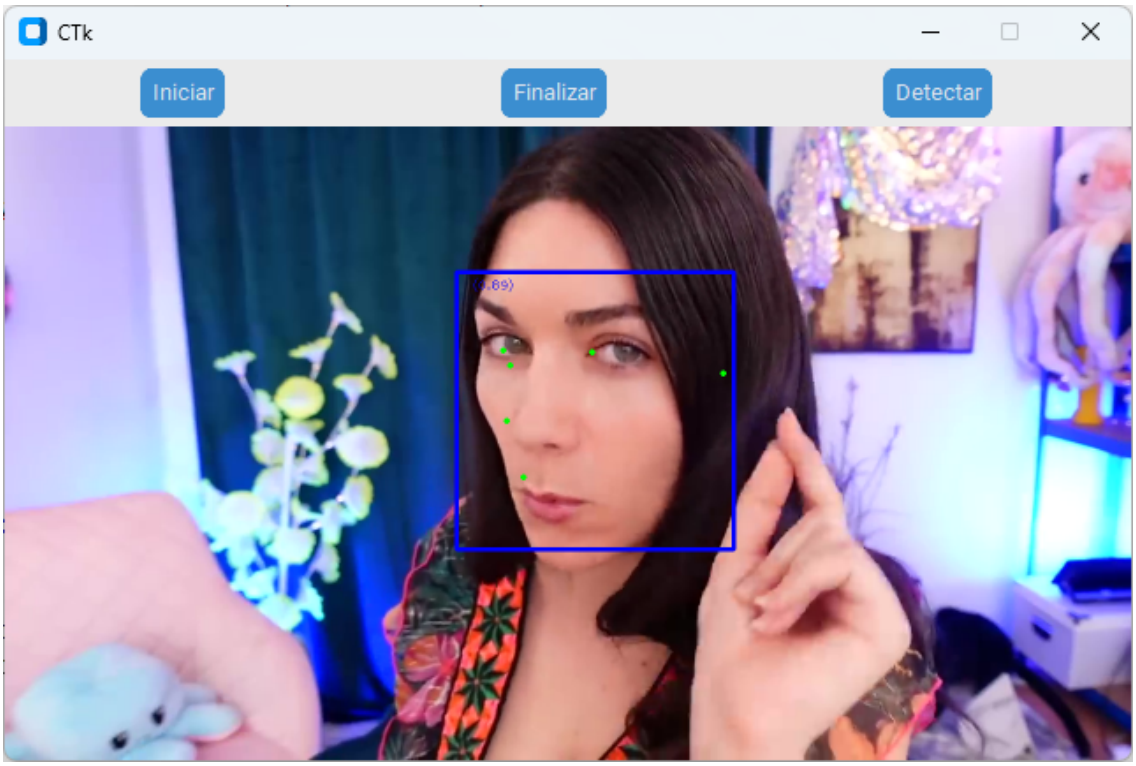

Figura 3: Interfaz mostrando los frames.

Entregables del taller

1. Dentro de los archivos del taller encontrará 2 notebooks tomados de los demos de MediaPipe para visión artificial.
2. Seleccione uno de estos notebooks y construya su aplicación de detección con interfaz tomando como base el código suministrado.
3. El estudiante es libre de utilizar un ejemplo diferente basados en los demos ofrecidos por MediaPipe en su web. De igual manera puede realizar las modificaciones deseadas en la interfaz.
4. Presente 3 evidencias (imágenes) de la ejecución del programa suministrado y del propuesto.
5. Conteste las siguientes preguntas relativas al modelo seleccionado:
   a) Indique la función que cumple el modelo seleccionado.
   b) Indique el nombre del modelo utilizado.
   c) Indique cual es el nombre de la función en el código que se encarga de dibujar la detección en la imagen de entrada.