

Trabajo Práctico 3: HPC

Acosta Bernascone, Ramiro¹; Funes, Norman Federico¹; Giasone, Santiago Claudio¹
41.129.076, 41.204.592, 41.894.276
Comisión: Miércoles, Grupo: 17

¹Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina

Resumen: El Trabajo Práctico 3 corresponde a la temática de Computación de Altas Prestaciones. En él, se abordarán dos Ejercicios. En el primer ejercicio, se comparará y analizará los tiempos de ejecución del Algoritmo de Inversion de Color en forma secuencial (usando CPU) y su version paralela (con GPU). En dicho ejercicio, se enviará como input una imagen y el Algoritmo creará una imagen negativa a partir de la misma (es decir, una imagen con los colores diametralmente opuestos en el círculo cromático). Mientras que en el segundo ejercicio se desarrollará un Bot de Discord al cual se le dará como input una imagen de una persona y mediante Algoritmo de Deepfake creará un video falso de esa persona. Como experiencias aprendidas podemos destacar los varios usos que descubrimos que se le pueden dar a las HPC a medida que leíamos los Papers para obtener información sobre los Algoritmos que desarrollamos en este Trabajo.

Palabras Clave: HPC, Inversion de Color, Deepfake, Google Colab, GPGPU, CUDA, Algoritmos.

1 Introducción

- Google Colab es una plataforma que permite a cualquier usuario escribir y ejecutar código de Python en el navegador. Es adecuado para tareas de aprendizaje automático y análisis de datos, por lo que resulta una plataforma extremadamente útil para el desarrollo del trabajo practico de Computacion de Altas Prestaciones, que como su nombre lo indica, requiere de un Hardware potente para poder sacar provecho de los cálculos exhaustivos que se podrían llegar a realizar en esta temática. Dicha plataforma tiene como característica el uso del procesador gráfico (GPU) CUDA para el procesamiento de datos, el cual permite ejecutar tareas en forma paralela, lo que hace que el procesamiento de grandes volúmenes de información sea muchísimo más rápido que si se realizará con una CPU, debido a su alta potencia de procesamiento.

En la plataforma, se desarrollaron dos Ejercicios. El primer ejercicio hace referencia al Algoritmo de Inversion de Color. Sobre esto, el usuario debe buscar cualquier imagen que exista en la web y debe obtener su URL. Una vez que el usuario obtenga la URL de la imagen, el Algoritmo de Inversion de Color calculará el complemento de los 3 canales RGB (que representan los colores Rojo-Verde-Azul) que tiene cada pixel que componen la imagen. De esta manera, se obtiene como resultado una imagen con los colores diametralmente opuestos en el círculo cromático al de la imagen original.

Para el segundo ejercicio escogimos el Algoritmo de Deepfake, y lo implementamos a través de un bot de Discord. Dicho bot se deberá alojar en un Servidor de Discord y en el que el usuario deberá ser miembro. El bot recibirá como input una imagen de una persona y creará a partir de esta imagen un video aparentemente real (pero que no lo es, es ficticio) combinando ese input de esa imagen que el usuario envíe y alguno de los dos videos que tiene previamente configurado el bot (que el usuario escogerá por parámetro para generar dicho video).

2 Desarrollo

Para el Ejercicio 1:

- Explicar detalladamente como utilizar el ejercicio.

Para comenzar se debe ingresar una imagen a la cual se desea invertir los colores. Para ello, se debe escoger una imagen que haya en la web e insertar la URL de la misma en el campo “url_imagen”. Se recomienda escoger una imagen que tenga colores fuertes.

Especifique la URL de la imagen:

url_imagen: " <https://columnacero.com/pic/25689/spider-man-un-nuevo-universo.jpg> "

Luego se debe clicar en el boton “play” de la sección “Obtención de imagen”, el cual ejecutará el código que almacenará dicha imagen localmente para posteriormente trabajar con ella.

Obtencion de imagen

```
### Parámetros de ejecución

#@markdown ## Especifique la URL de la imagen:
url_imagen = "https://columnacero.com/pic/25689/spider-man-un-nuevo-universo.jpg" #@param {type:"string"}
#link a imagen de prueba https://columnacero.com/pic/25689/spider-man-un-nuevo-universo.jpg
# Leo la imagen desde internet y la almaceno "localmente" como imagen.jpg
!wget {url_imagen} -O imagen.jpg
```

También es necesario instalar la librería de la GPU clickeando el botón de “play” de la celda que contiene el comando “!pip install pycuda”

```
!pip install pycuda
```

Luego de finalizar el armado de ambiente, se podrá comenzar a probar las ejecuciones. Para el caso de la ejecución Secuencial (CPU), dar click al botón de “play” que se encuentra en el apartado “Desarrollo CPU”.

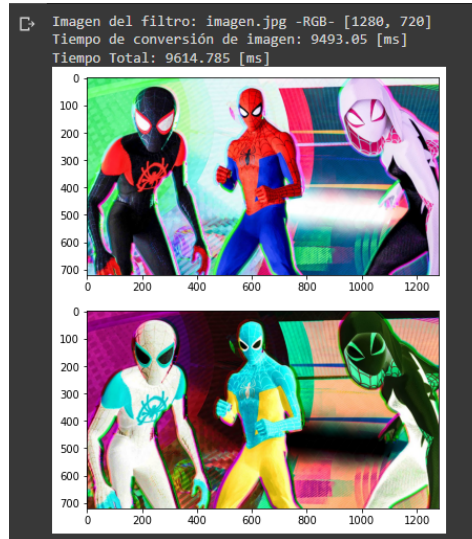
▼ Desarrollo CPU

```
%matplotlib inline
from datetime import datetime

tiempo_total = datetime.now()


import matplotlib.pyplot as plt
import numpy
from PIL import Image
```

Esto nos dara como resultado la imagen original, su inversion de color y los tiempos de ejecucion.



Para el caso de la ejecución paralela (GPU), dar click al botón de “play” que se encuentra en el apartado “Desarrollo GPU”

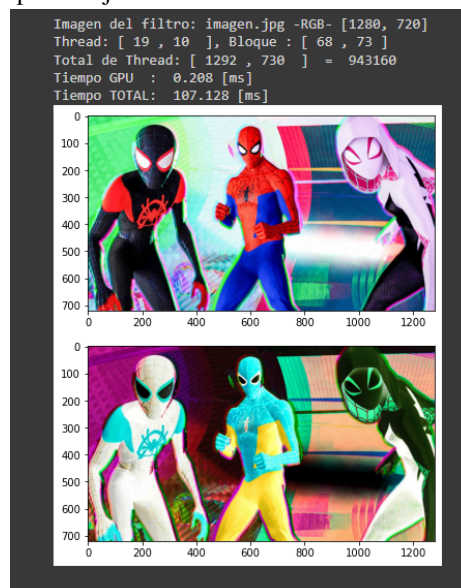
Desarrollo GPU



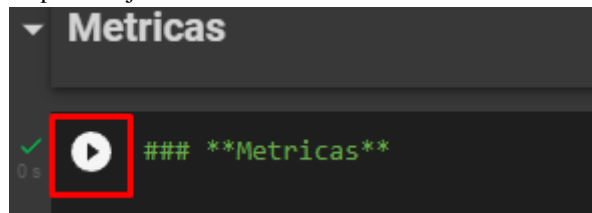
```
%matplotlib inline
from datetime import datetime
tiempo_total = datetime.now()

import matplotlib.pyplot as plt
import numpy
from PIL import Image
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
```

De forma similar a la ejecucion Secuencial, esto dará como resultado la imagen original, su inversion de color y los tiempos de ejecucion



Finalmente, en el apartado de “Métricas”, clickeando en el botón “play” se podrá ver la comparación de los tiempos de ejecución tanto de CPU como de GPU.



- Dirección del repositorio en GitHub:
[https://github.com/santiagogiasone/COVID-PersonLimiter/blob/master/HPC%20\(directorio\)/Cuaderno_1_grupo17_2021.ipynb](https://github.com/santiagogiasone/COVID-PersonLimiter/blob/master/HPC%20(directorio)/Cuaderno_1_grupo17_2021.ipynb)

- Explicar cómo funciona el ejercicio desde el punto de vista de la programación CPU-GPU.

Desde el punto de vista del CPU:

El ejercicio funciona comenzando por cargar la imagen original en memoria (convirtiéndola en un array). Luego se reserva memoria para la imagen resultado (del mismo tamaño que el array de la imagen original), y se procede a realizar el algoritmo de Inversión de color recorriendo el array original y calculando el complemento para cada pixel de dicho array, guardando el resultado de ese calculo en el array de la imagen de resultado.

Desde el punto de vista del GPU:

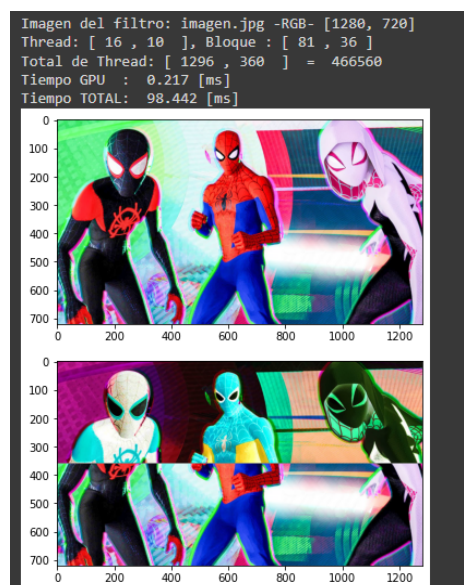
El ejercicio funciona comenzando por cargar la imagen original en memoria (convirtiéndola en un array) y se reserva memoria para la imagen resultante. A diferencia del procesamiento secuencial, para trabajar con el GPU se debe definir el kernel en el código y se debe definir la función que se ejecutará en la GPU (dicha función tendrá el algoritmo de Inversión de Color, pero se implementará a través de Threads). Luego la CPU deriva el procesamiento de la imagen a la GPU a través de la función desarrollada y una vez finalizada la ejecución se obtiene en la CPU el resultado del procesamiento que se realizó en la GPU.

- Debe experimentar respondiendo a las consignas:

¿Qué sucede si se planifican la mitad de los hilos-gpu?

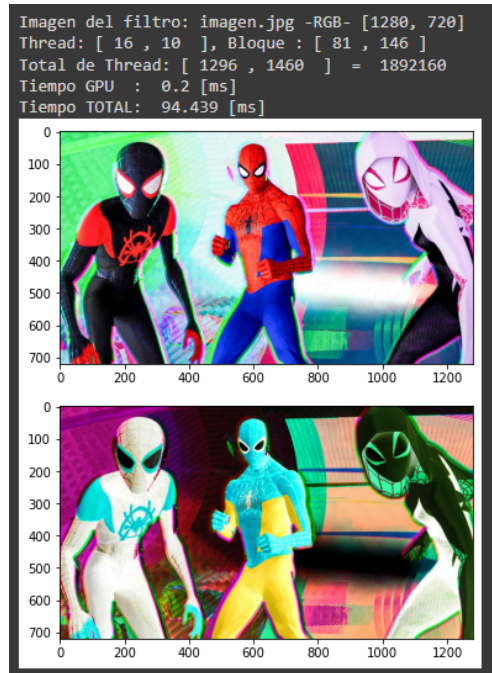
Se realizara inversion de color solo la primera mitad de la imagen luego se completara con la basura que se encuentre en la memoria global ya que al ser un vector grande sera lo que este almacenado en la primera reserva del resultado en este caso coincidio con la imagen de origen pero esto podria no ser asi.

Resultado:



¿Qué sucede si se planifican el doble de hilos-gpu?

Ocurriría la inversión de color a un menor tiempo de ejecución, pero no otorgaría beneficios extra.
Resultado:



¿Qué cambios hay que hacer para procesar la mitad inicial de la imagen de origen?

Se debe hacer este cambio en la verificación de las dimensiones de la imagen en la función de kernel para no perder la imagen en distintas ejecuciones.

```
if( idx < ancho && idy < (int)(alto/2) )
{
    //calculo el valor de cada pixel
    red=255-(img_O[(idx+(idy*ancho))*3 ]);// Componente Rojo del pixel.
    green=255-(img_O[((idx+(idy*ancho))*3)+1]);// Componente Verde del pixel.
    blue=255-(img_O[((idx+(idy*ancho))*3)+2]);// Componente Azul del pixel.

    //escribo el color de cada pixel.
    img_R[ (idx+(idy*ancho))*3 ] = red;
    img_R[((idx+(idy*ancho))*3)+1] = green;
    img_R[((idx+(idy*ancho))*3)+2] = blue;
}
else if(idx < ancho && idy < alto )
{
    img_R[ (idx+(idy*ancho))*3 ] = img_O[((idx+(idy*ancho))*3) ] ;
    img_R[((idx+(idy*ancho))*3)+1] = img_O[((idx+(idy*ancho))*3)+1] ;
    img_R[((idx+(idy*ancho))*3)+2] = img_O[((idx+(idy*ancho))*3)+2] ;
}
```

¿Qué cambios hay que hacer para procesar la mitad final de la imagen destino?

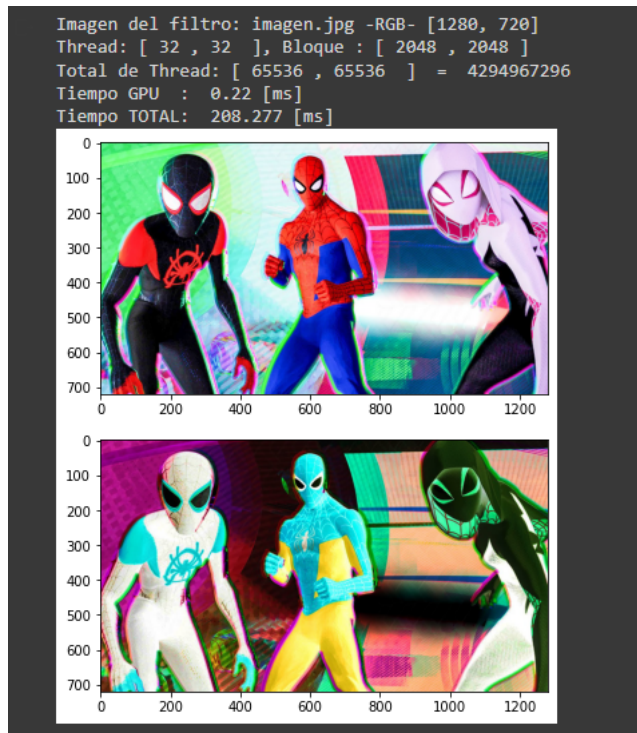
Se debe hacer este cambio en la verificación de las dimensiones de la imagen en la función de kernel para no perder la imagen en distintas ejecuciones.

```
if( idx < ancho && idy > (int)(alto/2) && idy < alto)
{
    //calculo el valor de cada pixel
    red=255-(img_O[(idx+(idy*ancho))*3  ]);// Componente Rojo del pixel.
    green=255-(img_O[((idx+(idy*ancho))*3)+1]);// Componente Verde del pixel.
    blue=255-(img_O[((idx+(idy*ancho))*3)+2]);// Componente Azul del pixel.

    //escribo el color de cada pixel.
    img_R[ (idx+(idy*ancho))*3  ] = red;
    img_R[ ((idx+(idy*ancho))*3)+1] = green;
    img_R[ ((idx+(idy*ancho))*3)+2] = blue;
}
else if(idx < ancho && idy < alto )
{
    img_R[ (idx+(idy*ancho))*3  ] = img_O[((idx+(idy*ancho))*3)  ] ;
    img_R[ ((idx+(idy*ancho))*3)+1] = img_O[((idx+(idy*ancho))*3)+1] ;
    img_R[ ((idx+(idy*ancho))*3)+2] = img_O[((idx+(idy*ancho))*3)+2] ;
}
```

¿Cuál es el resultado si se utilizan un tamaño de bloques igual a la cantidad máxima de hilos soportada por el GPGPU?

Continúa mostrando el resultado correcto pero al usar una cantidad de hilos excesiva el tiempo de ejecución aumenta más del doble se usaron dimensiones de hilos de 32 y 32 con bloques de 2048 y 2048. Con un total de 4.294.967.296 hilos.



Para el Ejercicio 2:

- Comentar cuales son los conceptos del tema (HPC) que están aplicando/experimentando:

Se está aplicando un Deepfake, se trata de una técnica de inteligencia artificial que permite editar vídeos falsos de personas que aparentemente son reales, utilizando para ello algoritmos de aprendizaje no supervisados, conocidos en español como RGAs (Red generativa antagónica), y vídeos o imágenes ya existentes. El resultado final de dicha técnica es un vídeo muy realista, aunque sea ficticio.

- Explicar detalladamente como utilizar el ejercicio

Análogo a lo que se hizo en el ejercicio 1, para poder utilizar el algoritmo es necesario configurar/armar el ambiente de trabajo. Para ello, lo primero que se debe verificar es que la configuración del cuaderno de Google Colab esté seteada para ejecutar con la GPU.

Luego de realizar dicha verificación, es necesario configurar un BOT de Discord. Para ello, se debe hacer click en el link de la invitación al BOT a un servidor de Discord que usted debe poseer (ser administrador del mismo).

2. Armado del ambiente

Primero que nada es indispensable que configure el BOT de Discord, si usted no sabe como crear un bot y obtener su TOKEN nosotros le proveemos el mismo y puede dejar el TOKEN configurado en la variable de entorno por defecto.

Link de Invitación al BOT (Por si quiere usar el default): https://discord.com/api/oauth2/authorize?client_id=912791690639704134&permissions=517544069184&scope=bot

Luego de esto, se deberá iniciar sesión en Discord y seleccionar el Servidor al cual se desea agregar al Bot. Luego de seleccionar el servidor y hacer click en continuar, aparecerá otra pantalla donde se deberá también otorgarle los permisos necesarios al Bot (click en autorizar).

FakeCord BOT

Conectarse a Discord

Sesión iniciada como **Giasito#9814** ¿No eres tú?

AÑADIR AL SERVIDOR:

Selecciona un servidor

Para esto tienes que tener permisos para **Gestionar servidor** en este servidor.


CancelarContinuar


Checkpoint que permite la reconstrucción facial en la nube y los videos para hacer el deepfake

```
# Video de Tobey
!wget --no-check-certificate "https://i.imgur.com/oBun5NF.mp4" -O "tobey.mp4"
```

Clonar el Repositorio de Deepfake

```
!git clone https://github.com/AliaksandrSiarohin/motion-cosegmentation motion-co-seg
```

 `cd motion-co-seg/`

 !pip install discord

Parcheo asyncio

```
## Este módulo parchea asyncio
import nest_asyncio
nest_asyncio.apply()
```

Dependencias de DeepFake

```
import imageio
from skimage.transform import resize
from part_swap import load_checkpoints
from part_swap import make_video
from skimage import img_as_ubyte
```

TOKEN: OTEyNzkxNjkwNjM5NzA0MTM0.YZ1FgQ.yxtJNLEFAASeXu8zdC476hAYkGw

[illegible]

TOKEN: ""

Una vez configurado el ambiente, se debe ejecutar el código que contiene la definición de la función de DeepFake, dándole click al botón “play”

Definimos nuestra funcion de Deepfake



```
async def get_video(img_url, file_name, video_path):  
    filename, file_extension = os.path.splitext(file_name)  
    r = requests.get(img_url)  
    image = imageio.imread(r.content, file_extension)  
    image = resize(image, (256, 256))[..., :3]  
    video = imageio.mimread(video_path)  
    video_final = []
```

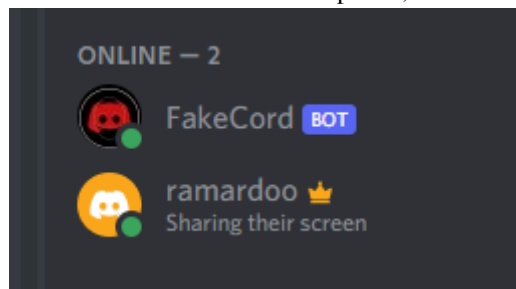
Y finalmente, correr el Servicio del Bot de Discord, dándole click al botón “play”

Corremos el servicio del Bot de Discord

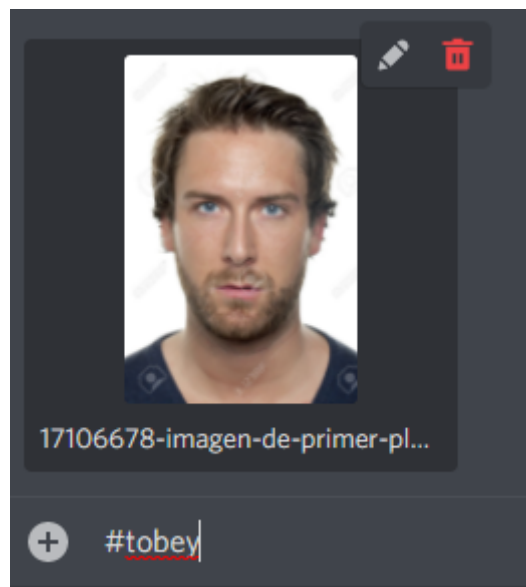
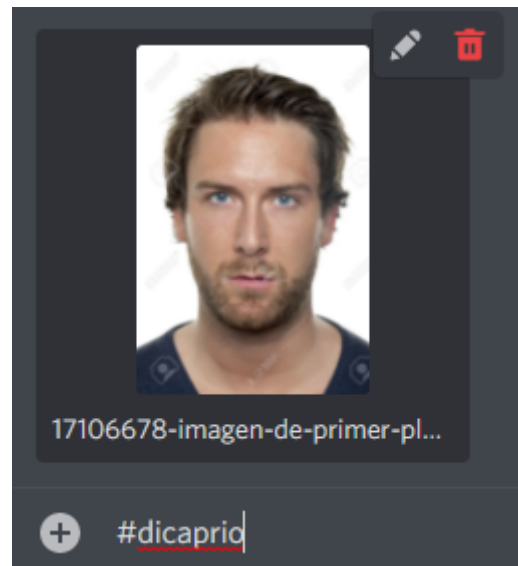


```
import discord  
from discord.ext import commands  
import requests  
import shutil  
import os
```

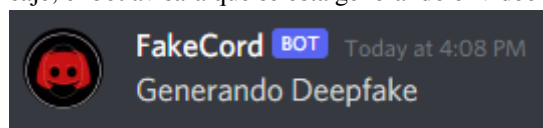
Una vez realizados todos los pasos anteriores, si ingresamos a nuestro Servidor de Discord (el cual administramos), deberíamos de ver activo al Bot de DeepFake, con el cual interactuamos.



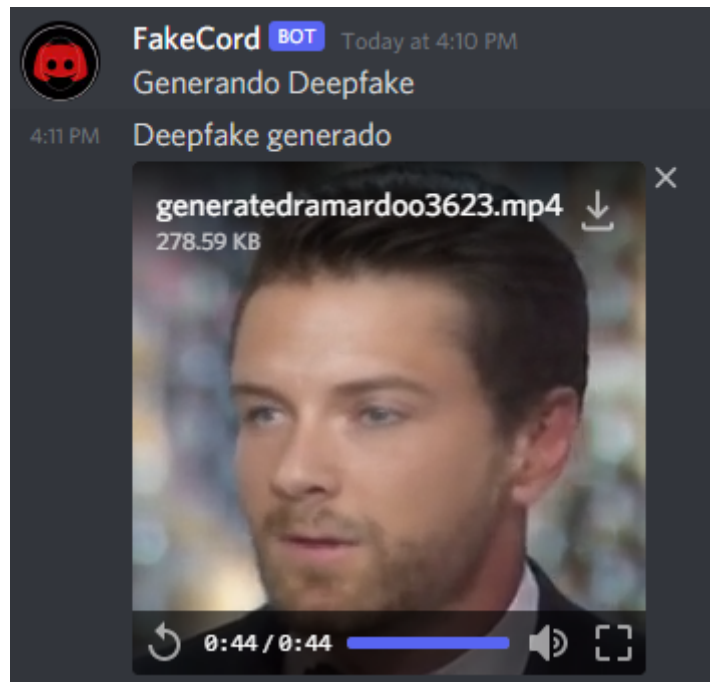
Para interactuar con el bot, se debe cargar una imagen y se debe escribir #tobey o #dicaprio. Según qué parámetro se le pase, se creará el DeepFake con un video u otro (que se tiene previamente configurado).



Una vez enviado el mensaje, el bot avisará que se esta generando el video



Luego de esperar unos pocos minutos, obtendremos el video ficticio.



El video y la imagen utilizados en este ejemplo pueden visualizarse en [https://github.com/santiagogiasone/COVID-PersonLimiter/tree/master/HPC%20\(directorio\)/](https://github.com/santiagogiasone/COVID-PersonLimiter/tree/master/HPC%20(directorio)/)

Imagen: primerplano.jpg

Video: Generado-Dicaprio.mp4

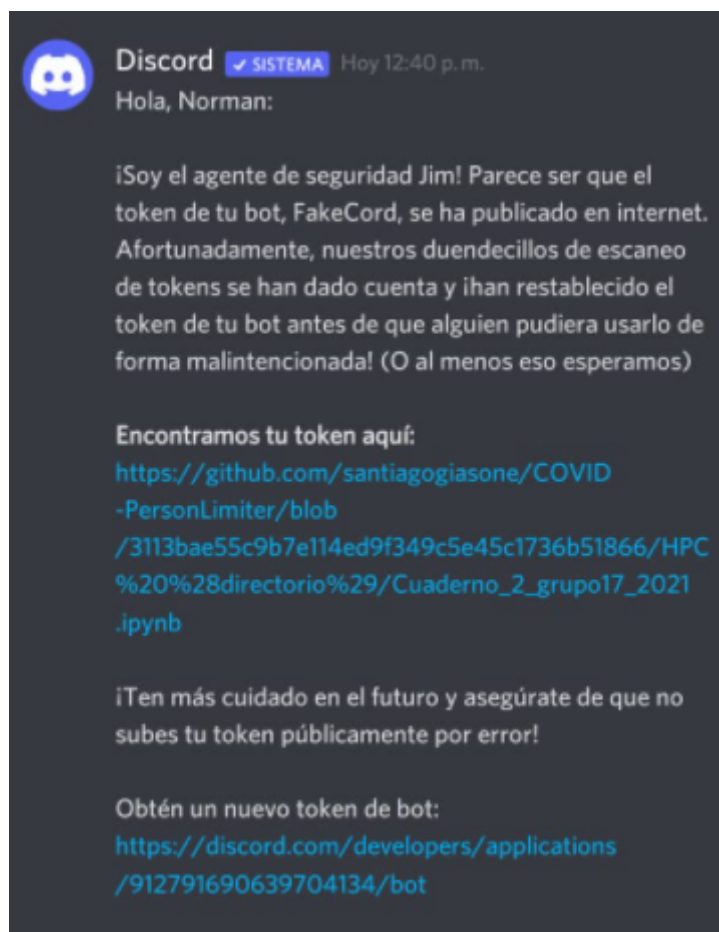
- Deben indicar la dirección web del ejercicio dentro del repositorio GitHub: [https://github.com/santiagogiasone/COVID-PersonLimiter/blob/master/HPC%20\(directorio\)/Cuaderno_2_grupo17_2021.ipynb](https://github.com/santiagogiasone/COVID-PersonLimiter/blob/master/HPC%20(directorio)/Cuaderno_2_grupo17_2021.ipynb)

3 Conclusiones

- Comente los problemas surgidos en el desarrollo de los ejercicios

Al momento del desarrollo del ejercicio 1 hubo un mal manejo de los vectores de la imagen en el gpu por lo que el resultado que se mostraba era incorrecto en la mayoría de los casos esto fue solucionado gracias a encontrar que siempre se trabajó con la imagen resultado y no con la de origen. Otro problema en este ejercicio fue al momento de trabajar con las mitades de imágenes que fue solucionado trabajando con los límites de la imagen ya que si esto no se tomaba en cuenta pasaba lo mismo que con el problema anterior mostraba segmentos de memoria con basura y no lo que queríamos mostrar realmente.

Por su parte, respecto al ejercicio 2, tuvimos un inconveniente con el Token que se genera para interactuar con el Bot. Al encontrarse el token hardcodeado en el Colab, los algoritmos de seguridad de Discord detectaron que el token se encontraba de forma pública en la web, por lo que procedieron a darlo de baja y enviarnos un aviso para generar uno nuevo. Por esta razón, implementamos que el Token se cargue manualmente a través de un parámetro e informar dicho token por medio de este documento. Se deja captura a continuación del mensaje de Discord.



- Ejercicio 1: Comente acerca de las comparaciones en las métricas obtenidas.

En el primer ejercicio se observa claramente que al trabajar con la GPU la velocidad del procesamiento de las imágenes es muchísimo más rápida que si se trabaja con la CPU

- Ejercicio 2: Comente sobre la experiencia.

Al realizar el ejercicio dos tuvimos la oportunidad de experimentar, informarnos, investigar e interactuar con tecnologías avanzadas como son las del Deepfake y acopladas en algo que nosotros usamos personal y cotidianamente como la plataforma de chat de discord. Pudimos aprender cómo funciona el algoritmo un poco más en detalle y como la GPU realiza los cálculos para generar los videos a partir de las imágenes brindadas

- Indique una conclusión general.

Este trabajo práctico nos fue de mucha utilidad ya que aprendimos diversos conceptos que no teníamos muy presentes. Tanto de la temática de Computación de Altas Prestaciones, sus diversos algoritmos existentes para el procesamiento de información, cómo la GPU hace el procesamiento de datos, cómo interactúan la CPU con la GPU mediante las instrucciones de kernel, el uso de la plataforma de Google Colab, la comparativa de las velocidades de procesamiento tanto del CPU como de la GPU y el algoritmo de auto-detección que tiene Discord ante un posible filtrado de Token de un Bot propio en la web.

4 Referencias

- [1] Algoritmo de inversion de color de imagen OpenCV. Anónimo. (s. f.) <https://programmerclick.com/article/1612326767/>
- [2] Valiente Waldo: Prueba 2 - Imagen - CPU. (27/06/2021). GitHub. https://github.com/wvaliente/SOA_HPC/blob/main/Ejercicios/Prueba%202%20-%20Imagen%20-%20CPU.ipynb
- [3] Valiente Waldo: Prueba 2 - Imagen - GPU. (27/06/2021). GitHub: https://github.com/wvaliente/SOA_HPC/blob/main/Ejercicios/Prueba%202%20-%20Imagen%20-%20GPU.ipynb
- [4] Valiente Waldo: Ejemplo Hola Mundo GPU. (23/06/2021). GitHub. https://github.com/wvaliente/SOA_HPC/blob/main/Ejercicios/Prueba%200%20-%20Hola%20Mundo%20GPU.ipynb
- [5] Linghu, L., Wu, J., Wu, Z. y Wang, X. (2018). Parallel Computation of EM Backscattering from Large Three-Dimensional Sea Surface with CUDA. *Sensors*, 18(11), 3656. <https://doi.org/10.3390/s18113656>
- [6] Zendran, M. y Rusiecki, A. (2021). Swapping Face Images with Generative Neural Networks for Deepfake Technology – Experimental Study. *Procedia Computer Science*, 192, 834–843. <https://doi.org/10.1016/j.procs.2021.08.086>
- [7] Nguyen, T. T., Nguyen, C. M., Nguyen, D. T., Nguyen, D. T. y Nahavandi, S. (2020). Deep Learning for Deepfakes Creation and Detection: A Survey. *Fellow*. https://www.researchgate.net/publication/336055871_Deep_Learning_for_Deepfakes_Creation_and_Detection_A_Survey
- [8] AliaksandrSiarohin: first-order-model. (30/06/2021). GitHub. <https://github.com/AliaksandrSiarohin/first-order-model>
- [9] AliaksandrSiarohin: motion-cosegmentation. (24/04/2021). GitHub. <https://github.com/AliaksandrSiarohin/motion-cosegmentation>
- [10] API Reference: <https://discordpy.readthedocs.io/en/stable/api.html>