	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		


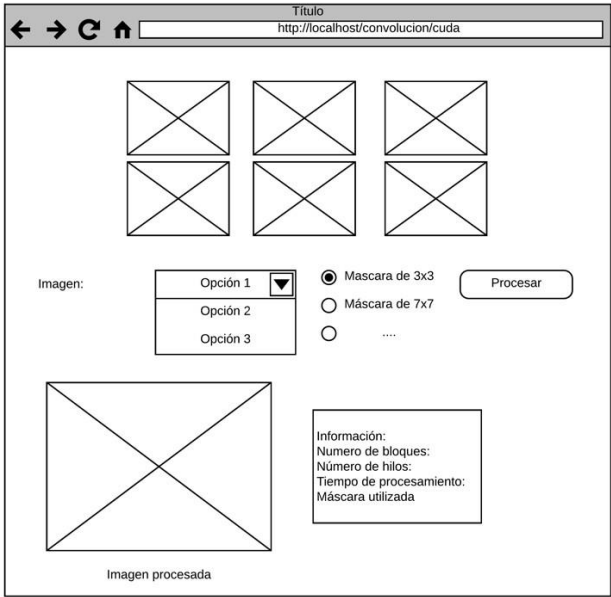
		PRÁCTICA DE LABORATORIO	
CARRERA: COMPUTACION		ASIGNATURA: COMPUTACION PARALELA	
NRO. PRÁCTICA:	4	TÍTULO PRÁCTICA: Desarrollo e implementación de aplicaciones de computo paralelo basado en CUDA en la nube	
OBJETIVOS <ul style="list-style-type: none"> Experimenta con soluciones basadas en: computación en la nube, cómputo de alto rendimiento, clústeres 			
INSTRUCCIONES		<p>Con base al desarrollo de la práctica Practica de laboratorio 03 - Desarrollo e implementación de aplicaciones de computo paralelo basado en CUDA. Se pide realizar los siguientes requerimientos:</p> <ol style="list-style-type: none"> Desarrollar una aplicación web usando django, flask, web2py o cualquier otro framework de Python para que permita a través de un servicio web realizar la convolución de una imagen usando PyCUDA. Dockerizar la solución para que sea desplegada como un servicio en la nube 	
			

Figura 1: Implementación de la aplicación web

ACTIVIDADES POR DESARROLLAR

1. Desarrollar un algoritmo usando PyCUDA para la convolución de imágenes

Para esta actividad, se realizó un código para ejecutar en GPU y a fin de mostrar una comparativa se creó también el código para ejecutar este algoritmo en CPU.

- Mostrar Disposición de carpetas y archivos:
 - Código para CPU
- Las librerías que se emplearon para el código son:

```
from django.db import models
from django.urls import reverse
from ConvolucionImagenGPU import models
from itertools import product
from PIL import Image
from numpy import dot, exp, mgrid, pi, ravel, square, uint8, zeros
import numpy as np
import pandas as pd
import math
import sys
import timeit
import signal
```

- Se crearon las clases, a. `GaussFilterCPU()`, con las siguientes funciones:

a) `def gaussFilter(x,y, sigma)`

Esta función corta, contiene la fórmula para la creación del Filtro de Gauss

```
def gaussFilter(x,y, sigma):
    #formula: w(x,y) = e^(-(x^2+ y^2)/(sigma^2))/(2*pi*sigma^2)
    return ( np.exp(-(x ** 2 + y ** 2) / (2 * sigma ** 2)) / (2 * np.pi * sigma ** 2))
```

b) `def gen_gaussian_kernel(kernel , sigma)`

En esta función se crea una matriz de numpy de tipo float32

Primero se crea una matriz vacía de NxN de tipo float

Se obtiene el valor central de la matriz ($5/2 = 2.5 = 2$)

Iteramos desde -2 hasta +3 teniendo 5 posiciones si el kernel es de 5 y creamos la matriz en la posición i, j, va desde 0 a 4 cuando el kernel es 5.

Posterior dividimos la matriz para el resultado y finalmente retorno `kernel_matrix`

```
def gen_gaussian_kernel(kernel , sigma):
    kernel_matrix = np.empty((kernel, kernel), np.float32)
    centro_del_kernel = kernel // 2

    for i in range(-centro_del_kernel, centro_del_kernel + 1):
        for j in range(-centro_del_kernel, centro_del_kernel + 1):
            kernel_matrix[i + centro_del_kernel][j + centro_del_kernel] = GaussFilterCPU.gaussFilter(i,j, sigma)
    kernel_matrix = kernel_matrix / kernel_matrix.sum()
    return kernel_matrix
```

c) `def filtroGauss (img_input_array, sigma, kernel, gauss_matriz)`

Crear la matriz de salida con la misma forma y tipo que la matriz de entrada. Posterior se crea de una matriz de ceros de este size de la imagen original

Se obtiene del alto y ancho de una imagen mediante el uso de un canal blue y obtenemos el valor central de la matriz $5/2 = 2.5 = 2$

```
def filtroGauss(img_input_array, sigma, kernel, gauss_matriz):
    result_array = np.empty_like(img_input_array)
    alto, ancho = img_input_array.shape[:2]
    centro_del_kernel = kernel // 2
```

Luego se hace un recorrido de cada píxel de la imagen de manera secuencial. El iterador *i* va a obtener los valores del alto de la imagen y los iteradores *j* los valores del ancho.

Y se crean las variables que van a almacenar la suma de producto de los canales rgb por cada valor de la matriz de gauss

```
    for i in range(0, alto):
        for j in range(0, ancho):
            red = 0.0
            green = 0.0
            blue = 0.0
```

En esta parte se crean bucles para recorrer la matriz de gauss

Después de 2 for para *k* y *L* se busca la posición "x" y "y" de un píxel en específico para manipularlo.

El *min* asegura no sobrepasar el ancho o alto de la imagen y *max* asegura obtener solo valores positivos, para evitar problemas con la dimensión de la imagen. Se obtiene la posición del píxel:

`img_input_array[y][x] = [0,72,166]` y los valores de la matriz de Gauss: `gauss_matriz[0][0] = 0.0032...`

Finalmente se multiplica un canal rgb por el valor de la matriz de gauss y se suma y almacena los resultados para obtener los valores reales para nuestro píxel. Se agrega el nuevo valor para el píxel con gauss aplicado, en los tres espacios de color. Por último retorna la variable `result_array`

```
def filtroGauss(img_input_array, sigma, kernel, gauss_matriz):
    result_array = np.empty_like(img_input_array)
    alto, ancho = img_input_array.shape[:2]
    centro_del_kernel = kernel // 2

    for i in range(0, alto):
        for j in range(0, ancho):
            red = 0.0
            green = 0.0
            blue = 0.0

            # Bucle para recorrer la matriz de gauss#
            for k in range(-centro_del_kernel, centro_del_kernel + 1):
                for l in range(-centro_del_kernel, centro_del_kernel + 1):

                    # Obtenemos la posición "x" y "y"
                    x = max(0, min(img_input_array.shape[1] - 1, j + l))
                    y = max(0, min(img_input_array.shape[0] - 1, i + k))
                    # posición del píxel: img_input_array[y][x] = [0,72,166], gauss_matriz[0][0] = 0.0032...
                    # Multiplicamos cada canal rgb por el valor de la matriz de gauss y los almacenamos
                    r, g, b = (img_input_array[y][x] * gauss_matriz[k + centro_del_kernel][l + centro_del_kernel])
                    # Se suma y almacena los resultados para pixel #
                    red += r
                    green += g
                    blue += b

            # gauss aplicado#
            result_array[i][j] = (red, green, blue)
    return result_array
```

d) `def gaussFilterCPU(path, kernelMascara, desviacionEstandar)`

Carga de imagen en RGB en la matriz y extraer sus canales de color y leer la imagen original.

Se la imagen a un array de numpy para obtener los canales.

```
def gaussFilterCPU(path, kernelMascara, desviacionEstandar):
    try:
        img = Image.open(path)
        img_input_array = np.array(img)

    except FileNotFoundError:
        sys.exit("No se pudo cargar la imagen")
```

Genera gaussian kernel (size of $N * N = 1234 \ 35$), y crea la matriz de salida con la misma forma y tipo que la matriz de entrada. Posterior creación de una matriz de ceros de este size de la imagen original, finalmente realiza una llamada a la función para la generacion de matriz de Gauss de $N \times N$.

```
kernel = kernelMascara
sigma = desviacionEstandar

img_output_array = np.empty_like(img_input_array)

gauss_matriz = GaussFilterCPU.gen_gaussian_kernel(kernel, sigma)
```

Aplicación de Filtro de Gauss

Toma de tiempo para el programa con `time_started` y se llama a la función de Gauss para su cálculo. Finalmente se muestra el tiempo total y se realiza la unión de cada canal rojo, azul y verde. Por último, se guarda imagen Resultados.

```
def gaussFilterCPU(path, kernelMascara, desviacionEstandar):
    try:
        img = Image.open(path)
        img_input_array = np.array(img)

    except FileNotFoundError:
        sys.exit("No se pudo cargar la imagen")

    kernel = kernelMascara
    sigma = desviacionEstandar

    img_output_array = np.empty_like(img_input_array)

    gauss_matriz = GaussFilterCPU.gen_gaussian_kernel(kernel, sigma)

    time_started = timeit.default_timer()

    img_output_array = GaussFilterCPU.filtroGauss(img_input_array, sigma, kernel, gauss_matriz)
    time_ended = timeit.default_timer()
    tiempoFinal = time_ended - time_started

    # Guardar imagen Resultados
    pathImgResultado = 'assets/images/resultado.png'
    Image.fromarray(img_output_array).save(pathImgResultado)
    return tiempoFinal, pathImgResultado
```

e) `def predict(imagen, mascara, desviacion)`

Como última función tenemos `predict`, que recibe parametros de imagen, mascara, desviacion. Para el trabajo se realiza un menu de imágenes (6), y cualquiera de estas puede ser seleccionadas. También se presenta un llamado a la función `GaussFilterCPU` con los parametros de path (path imagen seleccionada), mascara y desviación. Finalmente se retorna el tiempo final, la mascara y la desviacion.

```
def predict(imagen, mascara, desviacion):
    if( imagen == "Imagen1"):
        path = "ConvolucionImagenGPU/logica/images/ave.jpg"
    elif(imagen == "Imagen2"):
        path = "ConvolucionImagenGPU/logica/images/lago.jpg"
    elif(imagen == "Imagen3"):
        path = "ConvolucionImagenGPU/logica/images/paisaje-montanas.jpg"
    elif(imagen == "Imagen4"):
        path = "ConvolucionImagenGPU/logica/images/paisaje-nubes.jpg"
    elif(imagen == "Imagen5"):
        path = "ConvolucionImagenGPU/logica/images/paisaje.jpg"
    elif(imagen == "Imagen6"):
        path = "ConvolucionImagenGPU/logica/images/rombo.jpg"

    tiempoFinal, pathImgResultado = GaussFilterCPU.gaussFilterCPU(path, mascara, desviacion)
    return 0, 0, tiempoFinal, mascara, desviacion
```

- Código para GPU

Para la creación del algoritmo para GPU, las librerías que se emplearon fueron las siguientes. En las cuales pueden resaltar la utilización de pycuda.driver como la utilización de pycuda.compiler.

```
from ConvolucionImagenGPU.logica.GaussFilterCPU import GaussFilterCPU
from django.db import models
from django.urls import reverse
from ConvolucionImagenGPU import models
from itertools import product
from PIL import Image
from numpy import dot, exp, mgrid, pi, ravel, square, uint8, zeros
import numpy as np
import math
import sys
import timeit
import pandas as pd
#import pycuda.autoinit
import pycuda.driver as drv
from pycuda.compiler import SourceModule
```

De este modo, también se creó la clase con el nombre de GaussFilterGPU(). Y tiene ciertas funciones en las cuales tienen similitud a las anteriores, y en algunos casos no fueron modificadas.

a. `def gaussFilter(x,y, sigma):`

Esta función no tiene modificaciones, es la misma función que se utilizó en el algoritmo para GaussFilterCPU

b. `def gen_gaussian_kernel(kernel , sigma):`

Esta función no tiene modificaciones, es la misma función que se utilizó en el algoritmo para GaussFilterCPU

c. `def gaussFilterGPU(path, kernelMascara, desviacionEstandar):`

Esta función recibe los parámetros de path que es la imagen seleccionada de la función predict. Una

máscara y la desviación estándar.

Primero se carga una imagen RGB en la matriz y se extraen sus canales de color. Luego se pasa la imagen a un array de numpy para los canales. De este modo, se crean arrays de numpy para cada canal. En este caso son 3 (rojo, azul y verde). Y se genera un gaussian kernel (size of $N * N$) #1234 35, esto se hace mediante una llamada a la función para la generación de matriz de Gauss de $N \times N$, con parámetros de kernel y sigma.

```
def gaussFilterGPU(path, kernelMascara, desviacionEstandar):  
    try:  
        img = Image.open(path)  
        img_input_array = np.array(img)  
  
        red = img_input_array[:, :, 0].copy()  
        green = img_input_array[:, :, 1].copy()  
        blue = img_input_array[:, :, 2].copy()  
  
    except FileNotFoundError:  
        sys.exit("No se pudo cargar la imagen")  
    kernel = kernelMascara  
    sigma = desviacionEstandar  
    gaussian_kernel = GaussFilterGPU.gen_gaussian_kernel(kernel, sigma)
```

Siguiente paso tenemos el calculo de threads/blocks/grid basado en el ancho y altura de una imagen. Se busca obtener del alto y ancho de una imagen usando de un canal blue, la dimensión máxima por bloque es 32. Se presenta dimensiones de cuadrilla para "x" y "y", recordar que ceil nos devuelve un valor entero de la división obtenida.

```
# Obtencion del alto y ancho de una imagen hacemos uso de un canal blue.  
alto, ancho = img_input_array.shape[:2]  
dimension_por_bloque = 32  
drv.init()  
device = drv.Device(0) # enter your gpu id here  
ctx = device.make_context()  
tamano=(MATRIX_SIZE,MATRIX_SIZE)  
  
dim_grid_x = int(math.ceil(ancho / dimension_por_bloque))  
dim_grid_y = int(math.ceil(alto / dimension_por_bloque))
```

Posterior realizamos la llamada a función de pycuda para obtener respuesta. Leemos la función almacenada en el archivo gaussFilter.cu mediante mod = SourceModule(f'"" ""').

En el global_void aplicarFiltrosGauss como primer paso tenemos obtener las columnas resultantes de la multiplicación del número de hilos por el tamaño del bloque por dimensión del bloque todas en el espacio de X. Posterior se obtiene las filas resultantes de la multiplicación del número de hilos por tamaño del bloque por dimensión del bloque todas en el espacio de Y

```
mod = SourceModule(f"""
    global void aplicarFiltroGauss(const unsigned char *inputEspacioColor,
    unsigned char *outputEspacioColor,
    const unsigned int ancho,
    const unsigned int alto,
    const float *gausskernel,
    const unsigned int kernel) {{

        const unsigned int columnas = threadIdx.x + blockIdx.x * blockDim.x;
        const unsigned int filas = threadIdx.y + blockIdx.y * blockDim.y;
```

Se comprueba no se ha superado las dimensiones de la imagen mediante las siguientes líneas. Si cumple con la condición, entonces se obtiene el valor central de la matriz $5//2 = 2.5 = 2$. Se visualiza también la variable que van a almacenar la suma de producto de los canales rgb por cada valor de la matriz de gauss.

Se crean bucles para recorrer la matriz de gauss desde $(-2,2)$ y se obtiene la posición "x" y "y" de un pixel en específico para manipularlo. Con el min se evita sobrepasar del ancho o alto de la imagen y con max aseguramos obtener solo valores positivos y no se problemas con la dimensión de la imagen.

Recordamos que la matriz en este caso es una lista no una matriz seguida por lenguaje C, entonces solo se necesita una posición para el kernel que buscamos, después se multiplica el valor de la matriz de gauss por el pixel en la posición x,y .

De esta forma obtenemos la posición del píxel haciendo uso de columnas, filas y ancho, luego asignamos el valor del pixel modificado.

```
if(filas < alto && columnas < ancho) {{
    const int mitadSizeKernel = (int)kernel / 2;
    float pixel = 0.0;
    for(int i = -mitadSizeKernel; i <= mitadSizeKernel; i++) {{
        for(int j = -mitadSizeKernel; j <= mitadSizeKernel; j++) {{
            const unsigned int y = max(0, min(alto - 1, filas + i));
            const unsigned int x = max(0, min(ancho - 1, columnas + j));

            const float valorGauss = gausskernel[(j + mitadSizeKernel) + (i + mitadSizeKernel) * kernel];
            pixel += valorGauss * inputEspacioColor[x + y * ancho];
        }}
    }}
    outputEspacioColor[columnas + filas * ancho] = static_cast<unsigned char>(pixel);
}}
""")
```

Obtención de la función de CUDA.

Para la aplicacion de Filtro de Gauss primero realizamos el paso de parámetros para la función filtroGauss y se realiza la toma de tiempo para el programa en time_started.

Los parametros usados son:

- # 1. Input: canal que pasamos
- # 2. Output: canal que se recupera y se almacena en la misma variable
- # 3. ancho imagen
- # 4. alto imagen
- # 5. Matriz de Gauss
- # 6. Size de kernel

7. block

8. grid

Se crea la matriz de salida con la misma forma y tipo que la matriz de entrada, es una matriz de ceros del mismo size de la imagen original, posterior se da la unión de cada canal rojo, azul y verde. Por último, se guardan los resultados.

```
# Obtencion de la funcion de CUDA
filtroGauss = mod.get_function('aplicarFiltroGauss')
time_started = timeit.default_timer()
for espacioColor in (red, green, blue):

    filtroGauss(
        drv.In(espacioColor),
        drv.Out(espacioColor),
        np.uint32(ancho),
        np.uint32(alto),
        drv.In(gaussian_kernel),
        np.uint32(kernel),
        block=(dimension_por_bloque, dimension_por_bloque, 1),
        grid=(dim_grid_x, dim_grid_y)
    )
time_ended = timeit.default_timer()
tiempoFinal = time_ended - time_started
img_output_array = np.empty_like(img_input_array)

img_output_array[:, :, 0] = red
img_output_array[:, :, 1] = green
img_output_array[:, :, 2] = blue

pathImgResultado = 'assets/images/resultado.png'
Image.fromarray(img_output_array).save(pathImgResultado)
ctx.pop()
return tiempoFinal, pathImgResultado, dimension_por_bloque, dim_grid_x, dim_grid_y
```


d. `def predict(imagen, mascara, desviacion):`

Obtención de la función de CUDA.

Al igual que nuestra función predict anterior en la parte de algoritmo para CPU, se puede observar que las funciones son similares, pero no son las mismas. Primero tenemos el “menú” en el que se muestran 6 imágenes sobre paisajes.

Finalmente se retorna la dimensión por bloque, hilos (variable del resultado de `dim_grid_x + dim_grid_y`). El tiempo final, una máscara y la desviación.


```
def predict(imagen, mascara, desviacion):  
    if( imagen == "Imagen1"):  
        path = "ConvolucionImagenGPU/logica/images/ave.jpg"  
    elif(imagen == "Imagen2"):  
        path = "ConvolucionImagenGPU/logica/images/lago.jpg"  
    elif(imagen == "Imagen3"):  
        path = "ConvolucionImagenGPU/logica/images/paisaje-montanas.jpg"  
    elif(imagen == "Imagen4"):  
        path = "ConvolucionImagenGPU/logica/images/paisaje-nubes.jpg"  
    elif(imagen == "Imagen5"):  
        path = "ConvolucionImagenGPU/logica/images/paisaje.jpg"  
    elif(imagen == "Imagen6"):  
        path = "ConvolucionImagenGPU/logica/images/rombo.jpg"  
  
    tiempoFinal, pathImgResultado = GaussFilterCPU.gaussFilterCPU(path, mascara, desviacion)  
    #print("tiempoFinal: ", tiempoFinal)  
    #resultado = pd.DataFrame([], columns = ['Tiempo' , 'path'])  
    #resultado.loc[0]= [tiempoFinal, pathImgResultado]  
    return 0, 0, tiempoFinal, mascara, desviacion  
    #return 0, 0, 0.1, mascara, desviacion
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

2. Desarrollar una aplicación web para procesar el algoritmo del punto 1

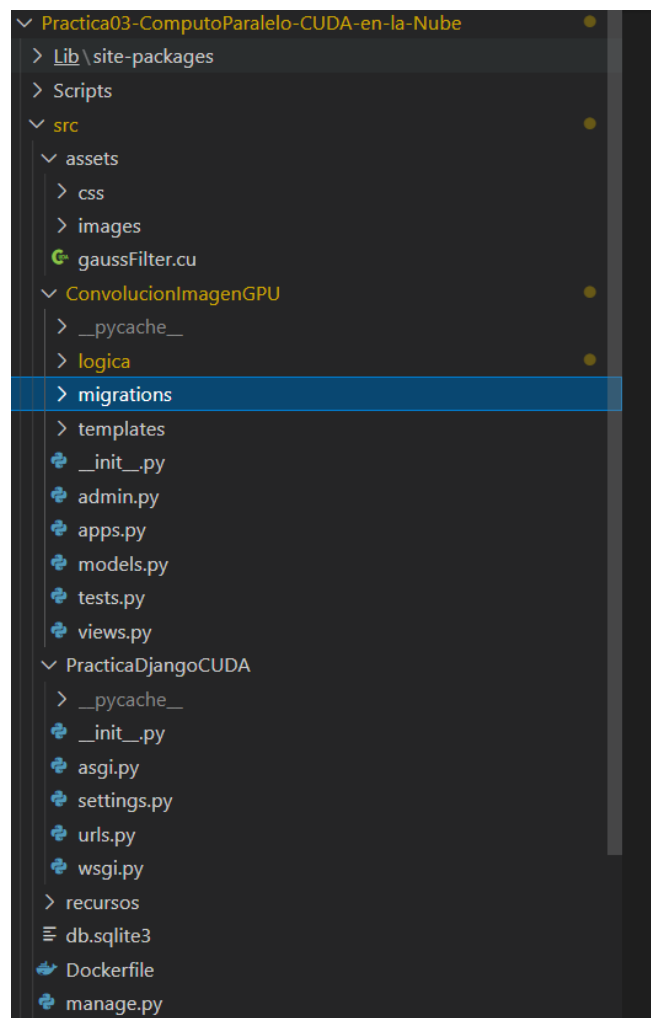
Para esta parte se trabajo utilizando el framework de django, en el proyecto se creo en Windows para posterior ser ejecutado en Linux con Docker.

A continuación, un listado de los comando para la creación de un proyecto django.


- crear virtual-enviroment = virtualenv .
- activar entorno buscar carpeta scripts = .\Scripts\activate
- instalar django = pip install django
- crear proyecto django = .\Scripts\django-admin startproject "nombreProyecto"
- correr en servidor = python manage.py runserver
- para actaulizar cambios = python manage.py makemigrations
- para actualizar base de datos = python manage.py migrate

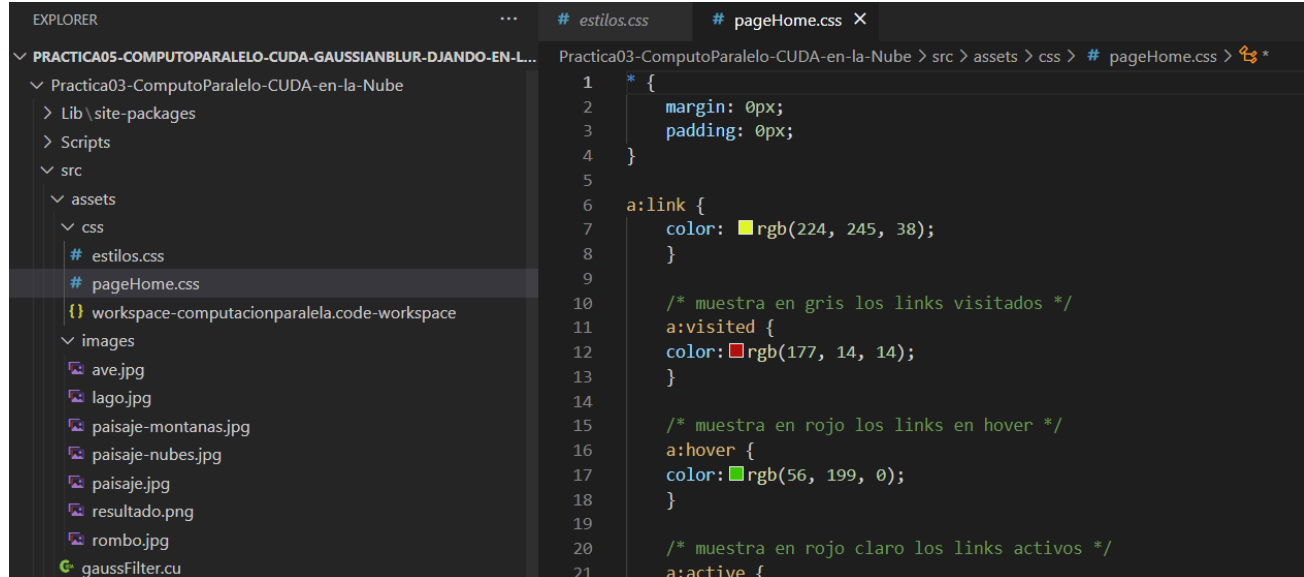
Una vez realizado estos cambios lo que se procede a modificar los archivos que existen dentro del proyecto.

La estructura del proyecto es la siguiente:



La carpeta assets/css y assets/miagenes, son directorios que contiene documentos estáticos.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

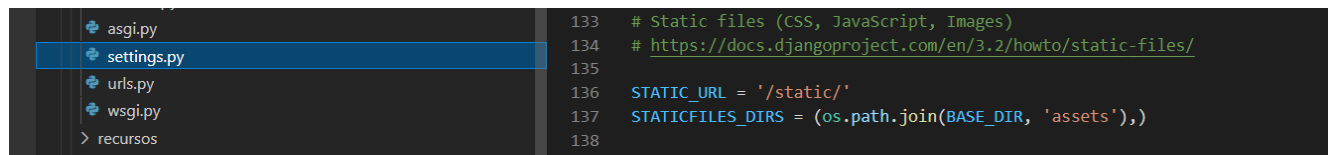


```

1  * {
2      margin: 0px;
3      padding: 0px;
4  }
5
6  a:link {
7      color: rgb(224, 245, 38);
8  }
9
10 /* muestra en gris los links visitados */
11 a:visited {
12     color: rgb(177, 14, 14);
13 }
14
15 /* muestra en rojo los links en hover */
16 a:hover {
17     color: rgb(56, 199, 0);
18 }
19
20 /* muestra en rojo claro los links activos */
21 a:active {

```

Estos documentos se los recupera con el siguiente path si necesitamos una imagen: /static/images/paisaje.jpg o si quiero una plantilla css = /static/css/pageHome.css. Para que se pueda ver los archivos de cualquier lado, tenemos que modificar el archivo settings.py

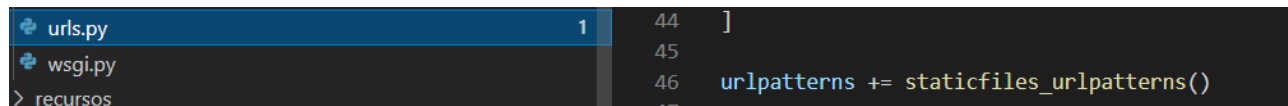


```

133 # Static files (CSS, JavaScript, Images)
134 # https://docs.djangoproject.com/en/3.2/howto/static-files/
135
136 STATIC_URL = '/static/'
137 STATICFILES_DIRS = (os.path.join(BASE_DIR, 'assets'),)
138

```

Además, dentro del archivo urls.py, tenemos que modificar lo siguiente:



```

44 ]
45
46 urlpatterns += staticfiles_urlpatterns()
47

```

Configuraciones de las modificaciones del proyecto.

Dentro de settings.py tenemos varias cosas que agregar.

- Habilitamos los hosts que pueden ver el proyecto.

```
ALLOWED_HOSTS = ['0.0.0.0', '127.0.0.1', 'proyectomlweb.uc.r.appspot.com', 'proyectomlweb.appspot.com']
```

- Agregamos lo siguiente dentro de INSTALLED_APPS

```

'ConvolucionImagenGPU.apps.ConvolucionimagengpuConfig', # ESTO SE AGREGO
'rest_framework', # ESTO SE AGREGO
'drf_yasg', # ESTO SE AGREGO
'polls', # ESTO SE AGREGO

```

- Además, se agrega esta configuración:

#ESTO SE AÑADIO

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ],
    #'DEFAULT_AUTHENTICATION_CLASSES': [
    #     'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'
    # ],
    'DEFAULT_SCHEMA_CLASS': 'rest_framework.schemas.coreapi.AutoSchema',
}
```

- Dentro de TEMPLATES agregamos la dirección en donde se van a encontrar las paginas HTML, en nuestro caso dentro del directorio templates

```
'DIRS': ['ConvolucionImagenGPU/templates'],
```


Configuración de URL's para el proyecto

- Agregamos nuestra información para el proyecto:

```
schema_view = get_schema_view(
    openapi.Info(
        title="Convolucion de GPU API",
        default_version='v1',
        description="Es el API de la Convolucion de imagenes",
        terms_of_service="https://www.google.com/policies/terms/",
        contact=openapi.Contact(email="jbarrerab1@est.ups.edu.ec"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=(permissions.AllowAny,),
)
```

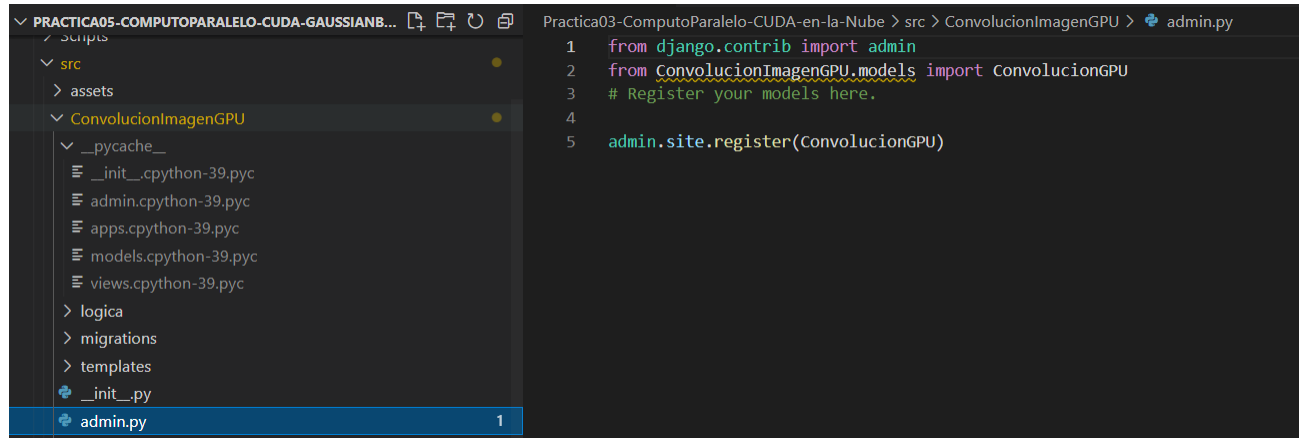
- Agregamos nuestras URLS para navegar por nuestra app, tenemos 3, la primera para la parte de la consola del administrador(admin), segundo tenemos a la url de inicio al momento de cargar nuestra app y por ultimo tenemos una url predecir en donde se encuentra la función del filtro de gauss.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    url(r'^$', views.LandingPage.inicio),
    url(r'^predecir/', views.LandingPage.predecir),
]
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Configuración de la parte del administrador

- Dentro del directorio ConvolucionImagenGPU, nos dirigimos hacia admin.py, aquí se debe encontrar lo siguiente:

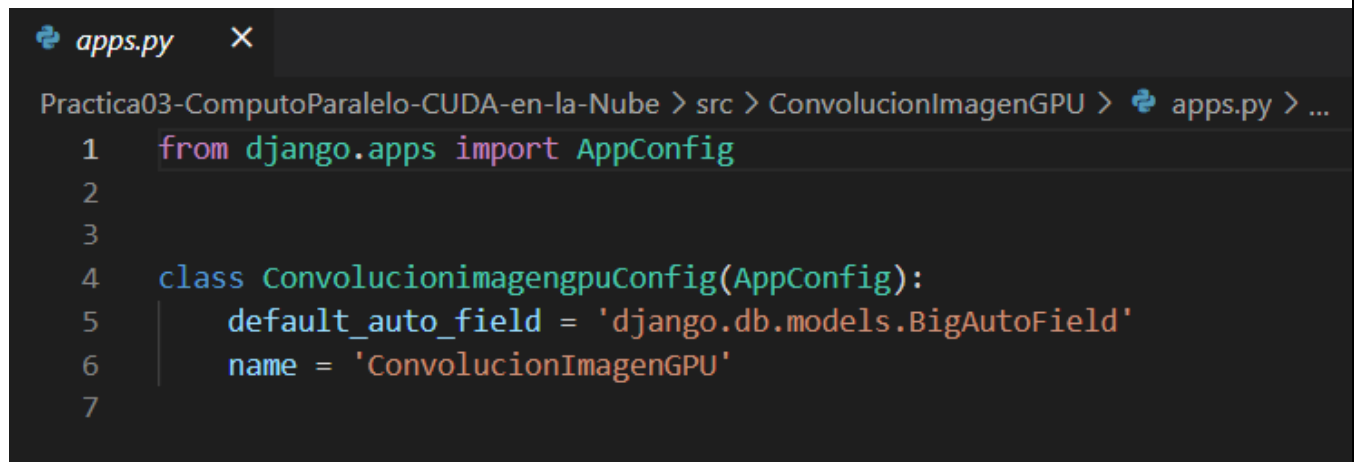


```

1 from django.contrib import admin
2 from ConvolucionImagenGPU.models import ConvolucionGPU
3 # Register your models here.
4
5 admin.site.register(ConvolucionGPU)

```

- Dentro del directorio ConvolucionImagenGPU, nos dirigimos hacia apps.py, aquí se debe encontrar el nombre de nuestra aplicación.




```

1 from django.apps import AppConfig
2
3
4 class ConvolucionimagengpuConfig(AppConfig):
5     default_auto_field = 'django.db.models.BigAutoField'
6     name = 'ConvolucionImagenGPU'
7

```

- Dentro del directorio ConvolucionImagenGPU, nos dirigimos hacia models.py, este archivo nos permitirá crear una tabla para la base de datos para ir almacenando resultados obtenidos, para esto se debe configurar los atributos de la tabla como en la imagen a continuación.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

models.py x
Practica03-ComputoParalelo-CUDA-en-la-Nube > src > ConvolucionImagenGPU > models.py > ...
1  from django.db import models
2
3  # Create your models here.
4  # Create your models here.
5  class ConvolucionGPU(models.Model):
6      codigo =models.AutoField(primary_key=True)
7      numeroBloques = models.IntegerField() # numero de Bloques GPU
8      numeroHilos = models.IntegerField() # numero de hilos GPU
9      tiempoProcesamiento = models.FloatField() # Tiempo de Procesamiento
10     mascara = models.IntegerField() # Mascara utilizada
11     sigma = models.IntegerField() # Sigma utilizada
12     def __str__(self):
13         return str(self.numeroBloques) + ':' + str(self.numeroHilos) + ':' , str(self.tiempoProcesamiento) +

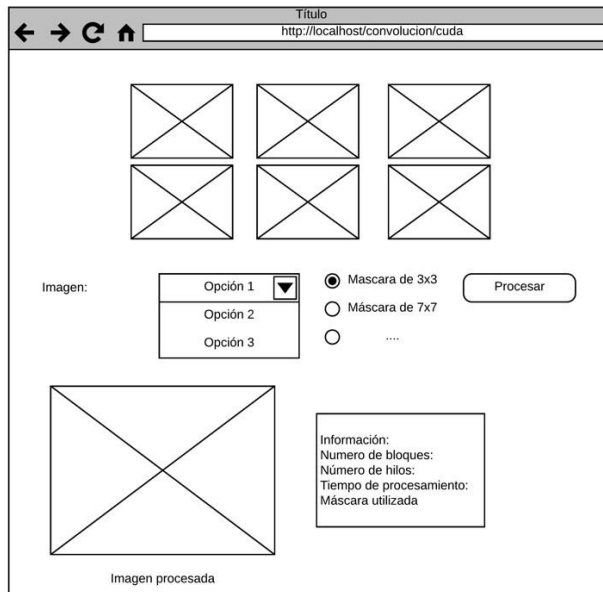
```

Con esta parte hemos configurado la parte del administrador, ahora nos centraremos en la parte pública.


Configuración de la parte publica

- Primero debemos crear una carpeta en donde vamos a crear nuestro HTML, en este caso, lo configuramos dentro de templates/inicio.html
- En este archivo vamos a crear una plantilla html para nuestro proyecto.

Se nos pidió algo como de la siguiente manera.



Entonces lo hemos adaptado y obtuvimos el siguiente resultado como para nuestra interfaz.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		



Todo el código para generar esta página se encuentra dentro de templates/inicio.html

- Ahora tenemos que unir la parte grafica inicio.html con views.py, para esto nuestra página html cuenta con un formulario, el cual tiene selección de imagen, de máscara y de desviación estándar. Todos estos parámetros serán enviados mediante evento POST, hacia la url /predecir/.

```
<form action="/predecir/" method="POST">

    <div>
        <h5>Seleccione imagen: </h5>
        <select name="imagen" id="imagen">
            <option value="Imagen1" selected>Imagen 1</option>
            <option value="Imagen2">Imagen 2 </option>
            <option value="Imagen3">Imagen 3</option>
            <option value="Imagen4">Imagen 4</option>
            <option value="Imagen5">Imagen 5</option>
            <option value="Imagen6">Imagen 6</option>
        </select>
    </div>

    <br>

    <div>
        <h5>Seleccione una máscara:</h5>
        <input type="radio" id="mascara5" name="mascara" value="5" checked>Máscara 5X5
        <input type="radio" id="mascara3" name="mascara" value="3"> Máscara 3X3 </h5>
    </div>

    <h5>Seleccione la Desviación Estandar:</h5>
    <input type="radio" id="1d" name="desviacion" value="1" checked> Sigma: 1
    <input type="radio" id="2d" name="desviacion" value="2">Sigma: 2
    <input type="radio" id="3d" name="desviacion" value="3">Sigma: 3
    <input type="radio" id="3d" name="desviacion" value="4">Sigma: 4</h5>
    {% csrf_token %}
    {{ form }}
    <input type="submit" id="butonProcesarImagen">Procesar Imagen</input>

</form>
```

- Una vez que el usuario de click en el botón, entonces el formulario se dirige hacia views.py en donde tenemos los siguientes métodos.

```
class LandingPage():
    def inicio(request):
        #print("Iniciooooooooooooooooooooo")
        return render(request, "inicio.html")

    def predecir(request):
        #print("Solicitud de prediccion")
        try:
            #print("Solicitud de prediccion")
            imagen = request.POST.get('imagen')
            mascara = request.POST.get('mascara')
            desviacion = request.POST.get('desviacion')

            except:
                imagen = ""
                mascara = ""
                desviacion = ""
            #resul=modeloSNN.modeloSNN.suma(num1,num2)

            if imagen != "":
                #resultado = GaussFilterGPU.GaussFilterGPU.predict(imagen, mascara, desviacion)
                #bloques, hilos, tiempoFinal, kernel, sigma = GaussFilterCPU.predict(imagen, int(mascara), int(desviacion))
                bloques, hilos, tiempoFinal, kernel, sigma = GaussFilterGPU.predict(imagen, int(mascara), int(desviacion))

            else:
                resultado = "Selecciona la imagen"

            return render(request, "inicio.html", {"bloques": bloques,
                                                    "hilos": hilos,
                                                    "tiempoFinal": tiempoFinal,
                                                    "kernel": kernel,
                                                    "sigma": sigma,
                                                    "path": "/static/images/resultado.png"})
```

- El primer método, tiene como finalidad redireccionar hacia la pagina de inicio cuando el usuario ingrese a nuestra aplicación utilizando 127.0.0.1:8000.
- El segundo método en cambio recibe lo que es el formulario, entonces se recupera cada uno de estos valores con POST.get, después hacemos la llamada al método GaussFilterGPU.predict, el cual recibe como parámetros, la imagen seleccionada, la mascara y la desviación estándar. Una vez que se ejecute la función entonces esta nos devolverá, los bloques, hilos, tiempo que se demoro en procesar la imagen, la mascara utilizada y la desviación estándar. Por último, todos estos resultados se envían de nuevo a la pagina de inicio mediante un diccionario clave – valor.


```
<div class="container5">
  {% csrf_token %}
  <h4>Resultado</h4>
  <h6>Numero de Bloques: {{bloques}} </h6>
  <h6>Numero de Hilos: {{hilos}}</h6>
  <h6>Tiempo de Ejecución: {{tiempoFinal}}</h6>
  <h6>Mascara Utilizada: {{kernel}}</h6>
  <h6>Sigma Utilizada: {{sigma}}</h6>
</div>

<div class="container6">
  {% csrf_token %}
  
</div>
```

Como se puede observar el inicio.html recuperamos estos resultados y se muestran al usuario

Resultado utilizando GPU

Convolucion de Imagenes con CUDA - Mozilla Firefox

Resultado x Convolucion de Imagenes x pycuda liberar memory x +

127.0.0.1:8000/resultado/

Imagenes para procesar






Imagen 1 Imagen 2 Imagen 3





Imagen 4 Imagen 5 Imagen 6

Seleccione imagen:
Imagen 1

Seleccione una mascara:
☒ Mascara 5X5 ☐ Mascara 3X3

Seleccione la Desviación Estandar:
☒ Sigma: 1 ☐ Sigma: 2 ☐ Sigma: 3 ☐ Sigma: 4

Resultado

Numero de Bloques: 32


Numero de Hilos: X = 16 Y = 16

Tiempo de Ejecución:
0.005719318985939026

Mascara Utilizada: 5

Sigma Utilizada: 4



	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

3. Dockerizar la solución del punto 1 y 2

- En este punto se creó un documento de tipo "Dockerfile" que contiene la lista de instrucciones a ser ejecutadas. En la ubicación del archivo **Practica05-ComputoParalelo-CUDA-GaussianBlur-Djando-en-la-Nube/Practica03-ComputoParalelo-CUDA-en-la-Nube/src/** con el nombre de DOCKERFILE con el siguiente contenido:

```

1 FROM nvidia/cuda:10.1-cudnn7-devel-ubuntu18.04
2 COPY . /app
3 WORKDIR /app
4 RUN apt-get -qq update && \
5 apt-get -qq install build-essential python3.8-dev python3-pip
6 RUN rm /usr/bin/python3 && ln -s python3.8 /usr/bin/python3
7 RUN pip3 install flask && \
8 pip3 install pycuda==2020.1
9 RUN pip3 install --upgrade pip && pip3 install -r requirements.txt
10 EXPOSE 8000
11 CMD python3 manage.py runserver 0.0.0.0:8000 --noreload

```

- Este archivo apunta a requirements.txt, en donde el archivo contiene una lista de las librerías que necesitamos instalar en nuestro contenedor de docker.

```


1 pillow
2 Django
3 django-rest-framework
4 drf-yasg
5 numpy
6 pandas
7 scikit-learn
8 gunicorn
9 polls
10 markdown
11 django-filter
12 pyrebase4

```

Para su configuración se consideraron los siguientes pasos:

- Abrimos una terminal de ubuntu y mandamos a correr los siguientes comandos:

Comando para crear contenedor de docker	sudo docker build --tag practica5barrerajk --network host /home/usuario/Documentos/BarreraJBarreraK-Django/Practica05-ComputoParalelo-CUDA-GaussianBlur-Djando-en-la-Nube-main/Practica03-ComputoParalelo-CUDA-en-la-Nube/src/
Comando para Correr el contenedor de docker	sudo docker run --gpus all -p 8000:8000 practica5barrerajk
Comando para listar procesos ejecutandose, ver si existe un proceso docker en ejecución.	sudo docker ps
Para ejecución de docker, finalizar	sudo docker stop "identificador proceso"

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

RESULTADO(S) OBTENIDO(S):

- Al momento de iniciar nuestro proyecto, ingresamos con el navegador a la direccion: <http://127.0.0.1:8000/>
- Aqui vemos la primera pagina de inicio.html
- Establecemos los parametros para nuestro filtro de Gauss: Mascara(Kernel), Desviación Estandar(Sigma)
- En la siguiente imagen podemos ver, algunas configuraciones realizadas:



En la siguiente imagen vemos el resultado de aplicar nuestro filtro

Convolucion de Imagenes con CUDA - Mozilla Firefox

Resultado x Convolucion de Imagenes : x pycuda liberar memory x +

127.0.0.1:8000/precdir/

Imagenes para procesar









Seleccione imagen:
Imagen 1

Seleccione una mascara:
☒ Mascara 5X5 ☐ Mascara 3X3

Seleccione la Desviación Estandar:
☒ Sigma: 1 ☐ Sigma: 2 ☐ Sigma: 3 ☐ Sigma: 4

Resultado

Numero de Bloques: 32

Numero de Hilos: X = 16 Y = 16

Tiempo de Ejecución:
0.005719318985939026

Mascara Utilizada: 5

Sigma Utilizada: 4



CONCLUSIONES:

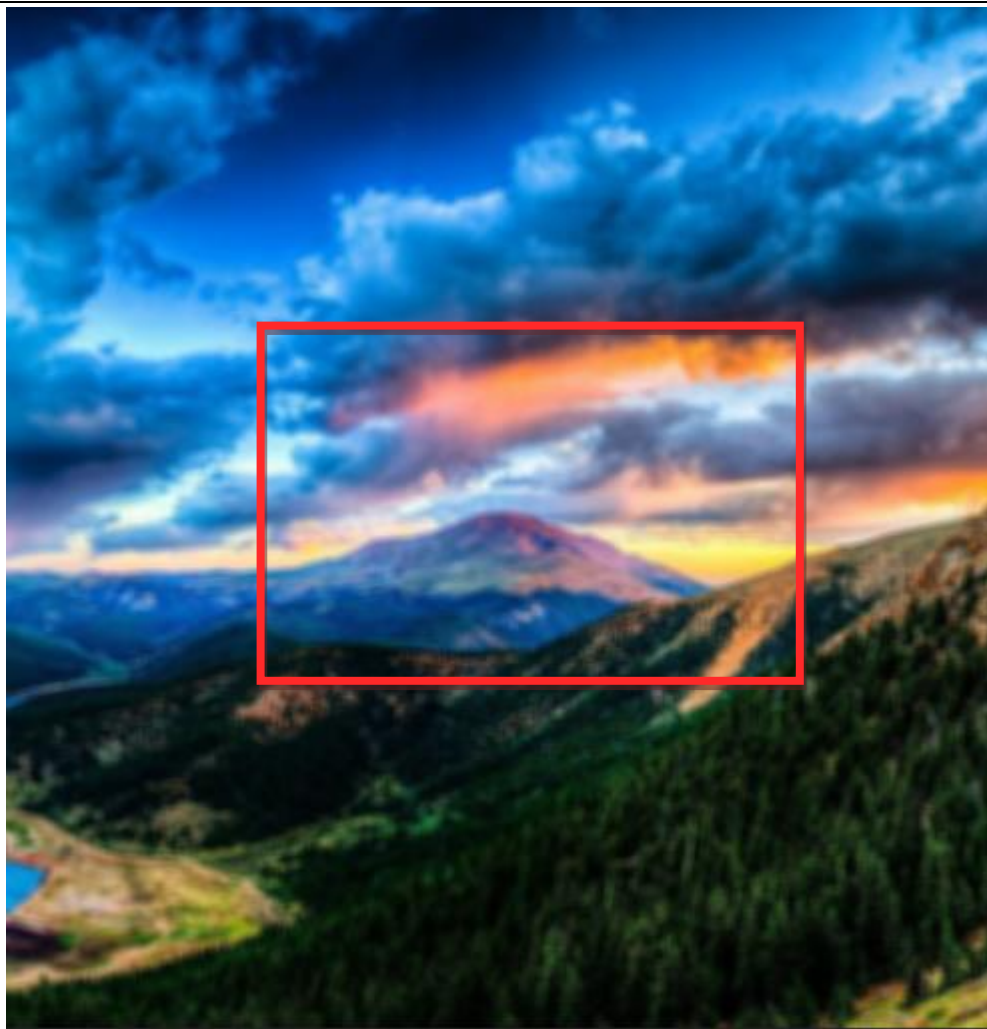
- Se logro unir tanto Docker, django, Python, Ubuntu y CUDA para poner llegar a obtener un servicio web, realizando las pruebas pertinentes se trabajo con CPU y GPU, y la diferencia entre procesamiento es abismal, cuando se realizo con CPU, la pagina demoraba entre 1 minutos para obtener un resultado, mientras que con la GPU el resultado llego a ser por poco 0 segundos. Por lo tanto el uso de GPU en la nube para servicios web llegan a ser muy satisfactorios

Comparacion de Imagenes

- Imagen Original:



- Imagen aplicando filtro de gauss



RECOMENDACIONES:

- Tomar el tiempo de ejecución de cada proceso por separado utilizando las funciones de tiempo de CUDA

Estudiantes: Juan Carlos Barrera Barrera y Katherine Michelle Barrera Barrera

Firma: