Introducción a la Visión Computacional

Tarea 2

Fecha de Entrega: Viernes 29, Abril 2022.

1. Diseñar un flujo de procesamiento a nivel abstracto: adquisición de imágenes, pre-procesamiento, procesamiento de imágenes, cálculo de la medición, almacenamiento.

2. Con la base de datos seleccionada, implementar un sistema de reconocimiento y/o procesamiento en el cual se realice una medida o inferencia sobre imagen. Puede ser la detección de un objeto, medición de distancias entre píxeles de alguna característica, medición del tamaño de algún objeto (en cantidad de píxeles) clasificación de la imagen, entre otros.

3. Para procesar el conjunto completo de imágenes, medir los tiempos de ejecución del proceso completo. Realizar la medición para el 50% y el 100% de la imágenes varias veces, de manera de obtener algunas figuras estadísticas: tiempo máximo, tiempo mínimo, tiempo medio, desviación estándar del tiempo.

4. Hacer resumen de los resultados del procesamiento. Para ello debe decidir una métrica para indicar si el procesamiento es satisfactorio. Para esto último algunas veces es necesario hacer la evaluación en forma manual, por lo tanto, si fuera este el caso, seleccionar un sub-conjunto de las imágenes (por ejemplo, unas 50) y comparar el desempeño del algoritmo en estas imágenes.

5. Escribir un informe o entregar el notebook con comentarios y documentación, incluyendo el análisis (comentando) los resultados.

Nota: de percatarse que el set de imágenes propuesto en la Tarea 1 no es el adecuado, se puede realizar un cambio en el dataset. Justificar el cambio. Este tipo de decisiones puede ocurrir en un proyecto.

En esta ocasión continuaremos con el código que se trabajó en la tarea 1. Eso sí, complementamos el dataset con mas imágenes, tanto de imágenes con fuego y sin fuego, para comparar el resultado. Esto no solucionara el problema de los falsos positivos, lo cual solo seria solucionado mediante técnicas de redes neuronales, lo que se busca en este momento medir al algoritmo actual en sus tiempos de respuesta y su precisión.

In [428…

```
import glob
from pathlib import Path
from PIL import Image, ImageFilter, ImageStat
import numpy as np
import cv2
import pandas as pd
import time
```

In [506…

```
_RESIZE_PERCENT = 0.3 #tamaño de la imagen al procesarla
_THRESHOLD = 200 #luminocidad valida del pixel al momento de procesar
_TYPES_IMG = ('*.jpg', '*.png') #

firePxlsMinPercent = 0.01 # mas de 1% de "fuego" o "posible fuego" dara como resultado OK.
```

In [375…

```
def isFireinImg(img, firePxlsMinPercent):
    img_processed = img.convert('L')
    img_processed = img_processed.filter(ImageFilter.GaussianBlur(5))
    img_processed = img_processed.point(lambda px1: 255 if px1 > _THRESHOLD else 0)
    img_processed = img_processed.convert('1')
    img_numpy = np.array(img_processed)
    if np.sum(img_numpy!=False) / img_numpy.size > firePxlsMinPercent:
        return True
    return False
    #return img_processed, np.sum(img_numpy!=False) / img_numpy.size

def resize_img(img, _RESIZE_PERCENT):
    x = img.size[0] * _RESIZE_PERCENT
    y = img.size[1] * _RESIZE_PERCENT
    img_processed = img.resize((int(round(x,0)),int(round(y,0))))
    return img_processed
```

In [671…

```
def read_all_imgs(resize=True):
    classes = ['non_fire_images','fire_images']
    retorna = []
    for c in classes:
        images=[]
        for img_path in glob.glob(r'.\fire_dataset\{0}\*.png'.format(c)):
            print('.', end='')
            try:
                img = Image.open(img_path) #leemos la imagen
                if resize:
                    img = resize_img(img, _RESIZE_PERCENT) #cambiamos tamaño para mejorar tiempos de procesamie
                images.append(img) # se guarda en memoria imagen con tamaño cambiado
            except Exception as e:
                print('\nError de lectura con {0}'.format(e), end='\t')
                pass
        retorna[c] = images
    return retorna
```

In [681…

```
datasets = read_all_imgs()
```

```
.....................................................................................................
```

In [387…

```
q_non_fire = len(datasets.get('non_fire_images'))
q_fire     = len(datasets.get('fire_images'))
print('Existen {} imagenes que NO contienen fuego.'.format(q_non_fire))
print('Existen {} imagenes que SI contienen fuego.'.format(q_fire))
```

```
Existen 244 imagenes que NO contienen fuego.
Existen 755 imagenes que SI contienen fuego.
```

In [493…

```
# obtencion de metricas de tiempo imagenes
def obtener_metricas_repeticiones(clase, porcentaje=1, q_repeticiones=50):
    fire_results_epoch = []
    time_epoch         = []
    if clase == 'fire_images':
        q_imagenes = int(q_fire*porcentaje)
        result_value = True
    if clase == 'non_fire_images':
        q_imagenes = int(q_non_fire*porcentaje)
        result_value = False

    for epoch in range(q_repeticiones):
        start = time.time()
        resultado = []
        for img in datasets.get(clase)[:q_imagenes]:
            resultado.append(isFireinImg(img, firePxlsMinPercent))
        end = time.time()
        time_process       = (end - start)
        resultado_process = np.sum(np.array(resultado)==result_value)/q_imagenes
        fire_results_epoch.append(resultado_process)
        time_epoch.append(time_process)
        print('.', end='')

    df_result = pd.DataFrame(tuple(zip(time_epoch,fire_results_epoch)),  columns=['time','accuracy'])
    df_result = df_result.reset_index().rename(columns={'index':'epoch'})
    df_result.insert(0,'percentage', porcentaje)
    df_result.insert(0,'class', clase)
    return df_result
```

In [494…

```
%%time

df_final = pd.DataFrame()

df_temp  = obtener_metricas_repeticiones('fire_images', 0.5, 100)
df_final = pd.concat([df_final,df_temp])
df_temp  = obtener_metricas_repeticiones('fire_images', 1, 100)
df_final = pd.concat([df_final,df_temp])
df_temp  = obtener_metricas_repeticiones('non_fire_images', 0.5, 100)
df_final = pd.concat([df_final,df_temp])
df_temp  = obtener_metricas_repeticiones('non_fire_images', 1, 100)
df_final = pd.concat([df_final,df_temp])

df_final.head()
```

```
.....................................................................................................
```

Out[494…

```
Wall time: 8min 54s
```

	class	percentage	epoch	time	accuracy
0	fire_images	0.5	0	1.218469	0.498674
1	fire_images	0.5	1	1.132993	0.498674
2	fire_images	0.5	2	1.218817	0.498674
3	fire_images	0.5	3	1.156069	0.498674
4	fire_images	0.5	4	1.149642	0.498674

In [539…

```
# se calcular metricas de tiempo
q_rows = {'non_fire_images': q_non_fire, 'fire_images': q_fire}
for clase in ['non_fire_images','fire_images']:
    for percent in [0.5,1.0]:
        q_imagenes = int(q_rows.get(clase)*percent)
        print('Resultados para {} registros correspondiente al {:.1%} de ().'.format(q_imagenes, percent, clase
        mean_time    = df_final[(df_final['class']==clase) & (df_final['percentage']==percent)].time.mean()
        min_time     = df_final[(df_final['class']==clase) & (df_final['percentage']==percent)].time.min()
        max_time     = df_final[(df_final['class']==clase) & (df_final['percentage']==percent)].time.max()
        std_time     = df_final[(df_final['class']==clase) & (df_final['percentage']==percent)].time.std()

        print('Tiempo medio {0}\t\t: {1:.4f} segundos'.format(clase, mean_time))
        print('Tiempo mínimo {0}\t\t: {1:.4f} segundos'.format(clase, min_time))
        print('Tiempo maximo {0}\t\t: {1:.4f} segundos'.format(clase, max_time))
        print('Desviacion Estandar Tiempo {0}: {1:.4f} segundos'.format(clase, std_time), end='\n\n')
```

```
Resultados para 122 registros correspondiente al 50.0% de non_fire_images.
Tiempo medio non_fire_images        : 0.4494 segundos
Tiempo mínimo non_fire_images       : 0.4322 segundos
Tiempo maximo non_fire_images       : 0.5325 segundos
Desviacion Estandar Tiempo non_fire_images: 0.0121 segundos

Resultados para 244 registros correspondiente al 100.0% de non_fire_images.
Tiempo medio non_fire_images        : 1.0941 segundos
Tiempo mínimo non_fire_images       : 1.0617 segundos
Tiempo maximo non_fire_images       : 1.2875 segundos
Desviacion Estandar Tiempo non_fire_images: 0.0379 segundos

Resultados para 377 registros correspondiente al 50.0% de fire_images.
Tiempo medio fire_images            : 1.1429 segundos
Tiempo mínimo fire_images           : 1.1048 segundos
Tiempo maximo fire_images           : 1.3234 segundos
Desviacion Estandar Tiempo fire_images: 0.0338 segundos

Resultados para 755 registros correspondiente al 100.0% de fire_images.
Tiempo medio fire_images            : 2.6616 segundos
Tiempo mínimo fire_images           : 2.6012 segundos
Tiempo maximo fire_images           : 2.9405 segundos
Desviacion Estandar Tiempo fire_images: 0.0607 segundos
```

In [555…

```
# se calculan metricas de precision
q_rows = {'non_fire_images': q_non_fire, 'fire_images': q_fire}
for clase in ['non_fire_images','fire_images']:
    for percent in [0.5,1.0]:
        q_imagenes = int(q_rows.get(clase)*percent)
        print('Resultados para {} registros correspondiente al {:.1%} de ().'.format(q_imagenes, percent, clase
        mean_accu = df_final[(df_final['class']==clase) & (df_final['percentage']==percent)].accuracy.mean()
        min_accu = df_final[(df_final['class']==clase) & (df_final['percentage']==percent)].accuracy.min()
        max_accu = df_final[(df_final['class']==clase) & (df_final['percentage']==percent)].accuracy.max()
        std_accu = df_final[(df_final['class']==clase) & (df_final['percentage']==percent)].accuracy.std()
        print('Precisión media {}: {:.2%}'.format(clase, mean_accu))
        print('Precisión mínima {}: {:.2%}'.format(clase, min_accu))
        print('Precisión maxima {}: {:.2%}'.format(clase, max_accu))
        print('Desviacion estandar de la Precisión {}: {:.5f}'.format(clase, std_accu), end='\n\n')
```

```
Resultados para 122 registros correspondiente al 50.0% de non_fire_images.
Precisión media non_fire_images: 66.39%
Precisión mínima non_fire_images: 66.39%
Precisión maxima non_fire_images: 66.39%
Desviacion estandar de la Precisión non_fire_images: 0.00000

Resultados para 244 registros correspondiente al 100.0% de non_fire_images.
Precisión media non_fire_images: 63.93%
Precisión mínima non_fire_images: 63.93%
Precisión maxima non_fire_images: 63.93%
Desviacion estandar de la Precisión non_fire_images: 0.00000

Resultados para 377 registros correspondiente al 50.0% de fire_images.
Precisión media fire_images: 49.87%
Precisión mínima fire_images: 49.87%
Precisión maxima fire_images: 49.87%
Desviacion estandar de la Precisión fire_images: 0.00000

Resultados para 755 registros correspondiente al 100.0% de fire_images.
Precisión media fire_images: 44.50%
Precisión mínima fire_images: 44.50%
Precisión maxima fire_images: 44.50%
Desviacion estandar de la Precisión fire_images: 0.00000
```

Conclusiones:

En total se procesaron 244 imágenes que NO contienen fuego y 755 imágenes que, SI contienen fuego, esto da un total de 999 imágenes, las cuales pasaron por 100 ciclos de procesamiento para obtener las métricas, dando 99.900 repeticiones para el total del dataset y de 49.900 repeticiones para el 50% del dataset, sumando 149.800 repeticiones en total, todo esto en 8min 54s.

Cuando analizamos los resultados por clúster de trabajo, dentro de los resultados de las métricas de medición podemos ver que los tiempos de trabajo no varían demasiado entre cada repetición, esto se ve demostrado en las desviaciones estándares bajas, esta desviación estándar tiende a subir al procesar mayor cantidad de registros (50% vs 100%). De igual manera no superan el 0.1. (Más arriba se ven los resultados en detalle)

Cuando revisamos los resultados de la precisión, esta tiene una desviación estándar de 0, teniendo los porcentajes de precisión entre el mínimo el máximo igual, esto quiere decir en cada uno de las 100 repeticiones el resultado fue el mismo, lo que confirma la estabilidad del algoritmo. Aunque el algoritmo este estable su precisión de igual manera no es la esperada ya que si bien cumplió con el 63,93% de aciertos con el dataset de imágenes con fuego, este solo llego al 44,5% en las imágenes que no tienen fuego (si non_fire_image contiene fuego es falso lo considera como cumplimiento). Este resultado nos demuestra que no es suficiente la técnica para tener un clasificador de imágenes de fuego eficiente.

Link Codigo e Imagenes: https://github.com/pssotosa/MDS/tree/main/VISION_COMPUTACIONAL/TAREA_2

In [ ]:

```

```