Procesamiento de imágenes en un dispositivo movil

Aranda Luciana, Carnovale Pablo, Perez Luis, Pichetti Gonzalo

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
aranda.luciana.f@gmail.com, pecarnovale@yahoo.com.ar, luisrperez2006@gmail.com,
gpichetti@gmail.com

Resumen. Esta investigacion se basa en las implicancias que puede tener para el dispositivo movil el procesar imágenes dentro de la aplicación del Vaso Inteligente para la cual el usuario deberia acceder a la cámara de su dispositivo movil, capturar la imagen de una bebida y automáticamente conectarse a la API de Google mediante la cual se traerá información relacionada a la bebida.

Palabras claves: <<Android, GPU, CUDA, OpenCV, ORB, SURF, SIFT>>.

1 Introducción

A partir del acelerado avance de la tecnología, en el mercado se pueden encontrar diversos dispositivos los cuales cuentan con cámaras de última generacion, mayor velocidad de internet y mayor poder de procesamiento.

Sin embargo a pesar de estos avances, los dispositivos tienen recursos limitados en cuanto a CPU, RAM y batería, con lo cual es importante tener en cuenta el uso eficiente de los recursos, sobre todo en un caso como el propuesto que conlleva un procesamiento importante a la hora de reconocer imágenes.

El propósito principal de la investigacion es conocer la forma en que se puede paralelizar el trabajo y cual es la mejor forma de realizar esta tarea para afectar lo menos posible los recursos del dispositivo móvil al procesar imágenes.

Existen aplicaciones capaces de buscar objetos a través de la cámara como Google Goggles[1] sin embargo una desventaja que posee es que no siempre retorna lo esperado. Otro ejemplo es la aplicación Similfy[2] que permite, a hombres y mujeres, encontrar y comprar moda a partir de una foto.

Estos ejemplos son aplicaciones exclusivas para la búsqueda a través de imágenes, nuestra propuesta es una funcionalidad extra dentro de nuestra aplicación del Vaso Inteligente, con lo cual no será necesario que el usuario deba utilizar otra aplicación.

[1] https://play.google.com/store/apps/details?id=com.google.android.apps.unveil&hl=es_AR
[2] https://berepublic.com/es/blog/berepublic-lanza-la-version-en-android-para-la-app-similify/

2 Desarrollo

Para el procesamiento de imágenes se pueden utilizar diversos algoritmos que permiten detectar diferentes características. La forma en la que se procese tendrá diferentes consecuencias en la administración de recursos.
Para obtener la imagen a procesar en la aplicación propuesta, en primer lugar, se tomará una imagen de entrada desde la cámara, y a esta se le aplicará el algoritmo ORB debido a que, según la investigación realizada, es el más óptimo para nuestras necesidades.
Debido a que la API de Google no permite búsqueda por imágenes, el usuario deberá introducir una palabra descriptiva, por ejemplo "Vodka", "Cerveza", "Vino" la cual acompañara a la imagen para de esta forma poder realizar la búsqueda en la API de Google.
El procesamiento de imágenes no aumenta la información que se puede extraer de ellas, solo realza ciertas características.
Con las técnicas de procesamiento de imágenes se puede modificar una imagen y de esta forma, al resaltar ciertos aspectos de la imagen, se puedan aplicar después algoritmos para obtener información a partir de ella.
Para la realización del Proyecto es necesario contar con las bibliotecas de visión artificial ya que brindan la posibilidad de realizar el análisis y clasificación de las imágenes. En este caso se propone utilizar OpenCV[3], que incluye soporte específico para Android.
OpenCV está diseñado para la eficiencia computacional con un fuerte enfoque en aplicaciones en tiempo real[4].
Debido a que necesitamos paralelizar el trabajo para agilizar los tiempos de procesamiento, será necesario aplicar la arquitectura CUDA[5] para la cual se deberá utilizar el lenguaje C++. Esto implica que también debamos utilizar el NDK[6] para Android ya que es un conjunto de herramientas que permiten usar el código C++ en la aplicación de Android.
Existen varios algoritmos para el reconocimiento de imágenes, para este caso analizaremos los siguientes: SIFT, SURF y ORB.
El algoritmo SIFT[7] transforma una imagen a un vector característico invariante a escalados, traslaciones, rotaciones y es parcialmente invariante a cambios de iluminación y afinidades.
El algoritmo SURF[8] es otro algoritmo de detección de características de imágenes. Es invariante a rotaciones, cambios de escala y cambios en la iluminación de las mismas. SURF propone una caracterización de las imágenes de manera más ágil pero menos robusta que SIFT.

[3] https://opencv.org/
[4] Ver sección Referencias [3]
[5] http://www.nvidia.es/object/cuda-parallel-computing-es.html
[6] https://developer.android.com/ndk/guides/?hl=es-419
[7] https://docs.opencv.org/3.0.0/da/df5/tutorial_py_sift_intro.html
[8]https://docs.opencv.org/3.0-alpha/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

ORB[9] es el algoritmo más eficiente para procesar imágenes, utiliza los algoritmos FAST[10] y BRIEF[11]. Otra característica es que es gratis mientras que SIFT y SURF están patentados.
De acuerdo con la investigación[12] realizada, según las distintas características de los algoritmos se encontró que el tiempo de computo empleado por el descriptor SIFT es mayor que el tiempo empleado por el descriptor SURF y ORB. Esto conduce a determinar que el descriptor SIFT no es una buena opción para aplicaciones que realicen reconocimiento en tiempo real.
El descriptor ORB es una mejor alternativa para aplicaciones de tiempo real.
En cuanto al consumo de batería, el porcentaje de batería utilizado por cada descriptor para procesar una imagen es proporcional al tiempo de ejecución analizado anteriormente.
Con respecto a la cantidad de memoria que necesitan los descriptores para almacenar sus vectores de características, sabemos que tanto SIFT como ORB son los que menos uso de memoria tienen y que el vector de características generado por el descriptor SURF necesita un promedio de casi 8 veces el peso del vector generado por ORB.

3 Explicación del algoritmo.

Haciendo referencia a [2], el procesamiento de imágenes digitales realizado a través de algoritmos implementa operaciones de matrices que contienen pixeles, aquí es donde necesitamos las ventajas del procesamiento paralelo ya que cuanto mayor sea el tamaño de la matriz, mayor será el tiempo de cálculo.
Las librerías de OpenCV proveen diferentes tipos de variables para almacenar y operar con imágenes. La clase utilizada para almacenar los frames de entrada de la cámara, así como las imágenes obtenidas tras realizar las operaciones pertinentes en dichos frames, es la clase Mat[13]. Estas matrices almacenan el valor de cada pixel de la imagen en sus filas y columnas correspondientes
Para la obtención de los frames de la cámara se utiliza una clase de las librerías OpenCV llamada JavaCameraView[14]. Esta clase es una implementación de un puente entre la cámara de Java y OpenCV.

El siguiente fragmento de código ejemplifica la parte secuencial para tomar una foto:

```
if (True){
//Definición de Parámetros para utilizar la cámara
        mOpenCvCameraView = (CameraBridgeViewBase)
findViewById(R.id.camera_surface_view);
     mOpenCvCameraView.setVisibility(SurfaceView.VISIBLE);
     mOpenCvCameraView.setCvCameraViewListener(this);
```

[9] https://docs.opencv.org/3.0-alpha/doc/py_tutorials/py_feature2d/py_orb/py_orb.html
[10] https://www.aprenderpython.net/algoritmo-fast-la-deteccion-esquinas/
[11] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_brief/py_brief.html
[12] Ver sección Referencias [4]
[13]https://docs.opencv.org/3.0.0/d3/d63/classcv_1_1Mat.html
[14] https://docs.opencv.org/java/3.0.0/org/opencv/android/JavaCameraView.html

```
        //Parámetros para capturar la foto
     ImageButton capture =
(ImageButton)findViewById(R.id.imageButton);
     capture.setOnClickListener(new View.OnClickListener() }
```

El uso del color en el procesamiento de imágenes simplifica la identificación de los objetos de una imagen ya que es un descriptor fundamental para la diferenciación entre unos objetos y otros. Sin embargo, conlleva un aumento en la memoria utilizada y en el coste computacional. Trabajar con imágenes en escala de grises, minimiza la memoria a utilizar y los tiempos y costos de computación.

El siguiente código ejemplifica el uso de CUDA con el algoritmo ORB:

```
//Se pasa la imagen a escala de grises y se sube a la GPU
cv::Mat img = cv::imread("image.jpg",
CV_LOAD_IMAGE_GRAYSCALE);
cv::cuda::GpuMat gpu_img;
gpu_img.upload(img);

//Se reduce el tamaño de la imagen leída tanto al eje x como y
para reducir el tiempo de procesado
cuda::resize(input, output, Size(), .25, 0.25, CV_INTER_AREA);

//Se define el descriptor ORB y el vector para los puntos
característicos

cv::Ptr<cv::cuda::ORB> orb = cv::cuda::ORB::create();
std::vector<cv::KeyPoint> keypoints;

//Se ejecuta el método de detección de puntos característicos
orb->detect(gpu_img, keypoints);
```

En este ejemplo se aplica paralelismo al trabajar con la imagen y luego se utiliza el algoritmo ORB para detectar los keypoints necesarios para realizar la comparación con otra imagen.

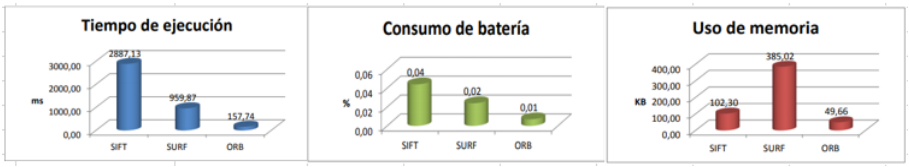
4 Pruebas que pueden realizarse

Las pruebas que podrían realizarse para verificar el correcto funcionamiento implican la captura de la imagen de una botella de cualquier bebida de la cual pueda obtenerse el nombre, a partir de esto se debería reconocer la imagen y obtener información relevante para el usuario.

5 Conclusiones

A modo de resumen, en esta investigacion se propuso principalmente conocer las consecuencias de incorporar el procesamiento de imágenes a nuestra aplicación del Vaso Inteligente incorporando la posibilidad de encontrar informacion de la bebida que el usuario requiera a partir de la camara del dispositivo movil. Para lo cual es imprescindible utilizar la tecnología GPU.

De acuerdo con lo expuesto en la sección 2 y basándonos en los datos obtenidos de [4], se ha comprobado que comparado con los descriptores SIFT y SURF, el descriptor ORB es el que tiene mejor efectividad en cuanto al uso de memoria tiempo de procesamiento y consumo de batería.



A partir de esto creemos que se debe utilizar el descriptor ORB si se quiere optimizar el uso de recursos del dispositivo móvil.

Para futuros trabajos, partiendo de un análisis como el de [1], se propone analizar estos mismos algoritmos en el reconocimiento de imágenes desde un video lo cual supone un mayor procesamiento.

Además, sería una gran mejora incorporar a la búsqueda de información de la aplicación, una reducción del espectro de búsqueda a través del procesamiento de imágenes ya que es sabido que muchas veces al utilizar estas búsquedas se obtiene información irrelevante respecto de las intenciones del usuario.

6 Referencias

1. Yu-Doo Kim, Jin-Tae Park, Il-Young Moon and Chang-Heon Oh: Performance Analysis of ORB Image Matching Based on Android. International Journal of Software Engineering and Its Applications. Vol.8, No.3 (2014)
2. Batuhan Hangün, Önder Eyecioğlu: Performance Comparison Between OpenCV Built in CPU and GPU Functions on Image Processing Operations. International Journal of Engineering Science and Application. Hangun and Eyecioglu, Vol.1, No.2, 2017
3. K. B. Neelima, Dr. T. Saravanan: Image Detection and Count Using Open Computer Vision (Opencv). K. B. Neelima Int. Journal of Engineering Research and Applications, September 2014.
4. Miguel Ñauñay Ilbay, Luis Tipantuña Córdova, Geovanny Raura, Tatiana Gualotuña: Análisis de eficiencia en algoritmos de reconocimiento de imágenes digitales aplicables a dispositivos móviles bajo la plataforma Android. (2013)