Reconocimiento Facial con HPC

Avalos Leonel, Callapiña López Juan, Crescente Maximiliano, Zurdo Misael

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
leonelavalos.95@gmail.com, guillekallap@gmail.com,
maxi.crescente10@gmail.com, misaelzurdo@gmail.com

Resumen. El siguiente Trabajo de Investigación que nuestro grupo va a presentar está basado en el estudio para mejorar el tiempo de respuesta y procesamiento de cálculos que presenta nuestro proyecto actual a presentar. Este estudio se desarrolla a partir de la tecnología HPC para paralelizar la distribución de datos y lograr un óptimo reconocimiento facial por medio del sistema biométrico planteado.

Palabras claves: Sistema Embebido, HPC, Android, Viola & Jones, OpenCV, Dlib, Árbol de Regresión.

1 Introducción

Nuestro objetivo consiste en mejorar una funcionalidad del sistema embebido como agregar una nueva a la aplicación.

El sistema embebido trabaja diferenciando entre rostros de personas para poder habilitar la puerta. Esto proceso se lleva a cabo tomando las diferencias entre una foto tomada en ese instante con los modelos de face code que se encuentra en dispositivo. El tiempo de procesamiento es considerable, por lo que se busca reducir el tiempo lo mínimo posible utilizando al máximo el dispositivo.

En la aplicación buscamos que el usuario tenga la posibilidad de entrenar el sistema con su propia cámara, pero pasando por una pequeña validación de que en las fotos que capture contenga verdaderamente un rostro el cual el sistema pueda utilizar para entrenar.

Para estos 2 casos utilizaremos el módulo de GPU del dispositivo para procesar una cantidad considerable de datos de forma paralela. En la actualidad, las aplicaciones móviles similares que se pueden utilizar con detección de rostros son variadas en el ámbito de redes sociales, en las cuales se realizan modificaciones sobre los rostros de sus usuarios. Algunos ejemplos de esto son Snapchat e Instagram. En ambas aplicaciones se utilizan este tipo algoritmos de detección de rostros para

cambiar las caras de las personas que aparecen en una imagen, o agregar diferentes tipos de filtros.

2 Desarrollo

En los dos casos utilizaremos OpenCV, el caso del dispositivo Android orientado a la utilización del algoritmo Viola & Jones el cual tiene un bajo costo de procesamiento gracias al trabajar en una escala de grises y permite una sencilla detección de rostro.

Para esto, se utiliza una característica por cada píxel, Para cada cálculo esta característica, necesitamos encontrar la suma de los píxeles debajo de los rectángulos blanco y negro. Una ventana de 24x24 da como resultado más de 160000 características. Por este motivo luego se empezó a utilizar una imagen integral lo cual reduce el cálculo solo a 4 píxeles. Igual sigue siendo una gran cantidad de datos a procesar, pero gracias a la GPU podemos realizar una gran cantidad de estos cálculos de forma paralela ya que estas son independientes entre sí.

En el otro caso necesitamos diferenciar un rostro del otro extraído de una captura tomada en ese momento. Esto consiste en los siguientes pasos: extraer la región de la imagen en que se encuentra el rostro, por cual podemos usar el mismo algoritmo mencionado anteriormente, aunque se podría utilizar otro.

Luego se detecta las estructuras faciales clave que se encuentra en el rostro como:

- Boca
- Ceja derecha
- Ceja izquierda
- Ojo derecho
- Ojo izquierdo
- Nariz
- Mandíbula

Para lograr esto utilizamos la librería Dlib el cual permite:

Un conjunto de entrenamiento de puntos de referencia faciales etiquetados en una imagen. Estas imágenes se etiquetan manualmente, especificando coordenadas x e y de las regiones que rodean a cada estructura facial. El cual permite una alineación del rostro mediante la utilizando la probabilidad de distancia entre pares de píxeles de entrada obtenido mediante árboles de regresión.

El resultado final es un detector de referencia facial que se puede utilizar para detectar puntos de referencia faciales de un rostro y luego poder diferenciarlo de otra referencia facial calculada con la foto tomada del dispositivo.

Para generar estos árboles de regresión se utiliza la siguiente ecuación “La impureza de Gini”:

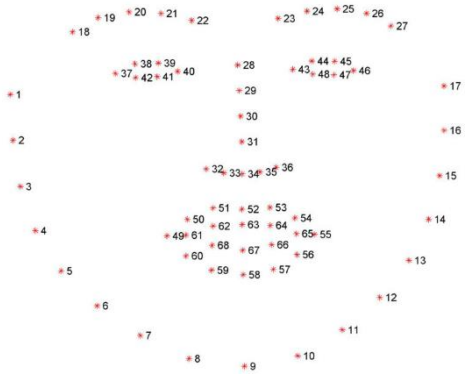
$$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m} (f_i - f_i^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i^2 = 1 - \sum_{i=1}^{m} f_i^2$$

La impureza de Gini se puede calcular sumando la probabilidad de cada elemento siendo elegido multiplicado por la probabilidad de un error en la categorización de ese elemento.

Lo importante para extraer es que esta sumatoria del árbol se puede paralelizar, además que se utilizar múltiples árboles y estos también son paralelizables. Utilizando la GPU podemos distribuir las operaciones sumatorias de cada árbol en bloques de hilos trabajando de manera paralela.

Luego de todo esto se puede estimar la ubicación de las estructuras faciales en la cara. Y se guardan en forma de face code.

Estos datos sirven tanto para entrenar al sistema con los datos de una persona como para compararla con otra.



3 Explicación del algoritmo

La utilización del algoritmo en celular consiste en una vez que se quiere entrenar al sistema se toma un conjunto de fotos con el dispositivo enviándolas al servidor para generar un entrenamiento con un promedio de los distintos face code obtenido de cada imagen.

Para esto se necesita los siguientes pasos:

Verificar en tiempo real que la cámara esté apuntando al rostro de una persona y solo en ese momento está habilitado la opción de capturar la imagen. Y solo tiene que ser un rostro.

PseudoCódigo

```
let cv = OpenCV();
int[][] MatrisRgb = CamaraStream();
int[][] MatrisRgbGry;
MatrisRgbGry = cvtColor(MatrisRgb, CV_RGBA2GRAY) // Pasa la imagen a una escala de grises
```

```
CascadeClassifier face_cascade;
int[] faces;

faces = detectInGreyScale(MatrisRgbGry, CV_RGBA2GRAY) // Detecta las caras en la matriz
if(faces.lenght == 1) { // Solo si existe una cara envia la imagen
return enviarImagen(MatrisRgb)
}
return false;
```

En la parte del servidor, una vez recibida todas las imágenes validadas para el entrenamiento se procede a generar un face code por cada una y sacar promedios de estos, generando un modelo con el id de la persona relacionado con el facecode: EJ:

```
{
id: NombrePersona,
face_code: [[-0.0816178499908, ……., 0.064642295259,0.0385133620224]]
}
```

Una vez almacenado el modelo se puede utilizar para comparar el los face_code de una persona con uno imagen capturada en ese momento. Aunque la respuesta no es verdadero o falso, sino la diferencia entre las relaciones de los puntos clave de la cara del modelo almacenado con la cara de la imagen capturada del sistema embebido.

4 Pruebas que pueden realizarse

Se pueden realizar las siguientes pruebas:
Una primer prueba estaría relacionada con la iluminación, demostrando que al exigir al sistema con una imagen con escasa luz puede realizar el reconocimiento.
Como segunda prueba, y más compleja, podríamos someter al sistema a una imagen con muy buena resolución (mayor cantidad de pixeles) exigiendo al algoritmo de esta manera.
En el caso de entrenar mayor cantidad de imágenes para autorizar el ingreso, va a requerir de mayor procesamiento computacional tanto en velocidad como para el resultado en tiempos óptimos.

5 Conclusiones

Es importante tener en cuenta el uso de la GPU a la hora de implementar algoritmos de procesamiento de imágenes e investigar sobre los procesos matemáticos que están detrás de ella para poder adquirir una performance mayor a como se venía trabajando.

La biometría presenta ciertos inconvenientes y limitaciones como la variación de las características biométricas a la hora de su adquisición o el rechazo social por violación a la privacidad.

Fue bastante útil el conocimiento previo a estas tecnologías durante nuestra cursada en la cátedra ya que nos facilitó el comienzo del camino a investigador tanto

por el Proyecto grupal y los temas vistos como HPC que nos resultaron interesantes ya que siempre vemos que el hardware potencia a un computador, pero nunca como funcionaba por dentro y las mejoras a realizar del mismo.

Quizás un próximo trabajo de investigación a pensar o implementar sería sobre algún algoritmo que pueda ser útil para el reconocimiento de iris y añadir a este Proyecto.

6 Referencias

1. Alpika Gupta, Dr. Rajdev Tiwari: Face Detection Using Modified Viola Jones Algorithm. In: Ultra Pradesh, India (2015).
2. Cazorla Martínez, R.: Software para la detección y el reconocimiento de rostros. In: Universitat Autònoma de Barcelona (UAB), España (2015).
3. Caballero. Franco Gabriel: Reconocimiento facial por el método de Eigenfaces. In: Pistas Educativas 127, México (CITEC 2017).