INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

DOCUMENTACIÓN DE AVANCE DE PROYECTO I

ERICK ALFARO HERNÁNDEZ

ARIANA BERMÚDEZ VENEGAS

XIMENA BOLAÑOS FONSECA

NICOL MORICE

PROFESOR: SAÚL CALDERÓN RAMÍREZ

Cartago, agosto 2017

CONTENIDO

1 INTRODUCCIÓN **1**

2 DIAGRAMA DE COMPONENTES **2**

3 ANÁLISIS DE TROZOS **3**

3.1 TROZO 1 3

3.2 TROZO 2 4

3.3 TROZO 3 4

3.4 TROZO 4 5

4 PROBLEMAS **7**

5 RESULTADOS **8**

5.1 Pruebas Unitarias 8

5.1.1 Trozo 1 8

5.1.2 Trozo 2 9

5.1.3 Trozo 3 11

5.1.4 Trozo 4 12

5.2 Página Web 13

6 CONCLUSIONES **15**

Bibliografía **15**

1 INTRODUCCIÓN

Python es un gran lenguaje de programación de propósito general, es de tipo dinámico y nos permite expresar ideas muy poderosas en muy pocas líneas de código, a la vez siendo muy legible. Python con la ayuda de algunas bibliotecas bastante populares (numpy,matplotlib y opencv) es un entorno potente para el área de la computación científica.

Para este proyecto se va a hacer uso de la biblioteca NumPy, uno de los módulos más importantes de Python. La biblioteca es encargada de agregar un mayor soporte para el manejo de vectores y matrices, donde nos ofrece una serie de funciones matemáticas para poder operar este tipo de estructuras.

Otros módulos de Python como Open CV, ayudará a escribir códigos computacionalmente intensivos en C/C++. De este modo el código es tan rápido como el código de C/C++ y el código es más sencillo. Trabajar con NumPy y Open CV hace que el trabajo sea más sencillo, ya que cualquier operación que pueda hacer NumPy se puede combinar con Open CV, de esta manera se tiene una mayor flexibilidad y facilidad.

Django es un framework para aplicaciones web gratuito y open source que fue escrito en Python. El cual brinda un conjunto de componentes que ayudaron a crear el sitio web más fácil y rápidamente.

2 DIAGRAMA DE COMPONENTES

En esta sección se va a representar los componentes de alto nivel y se va a describir cada uno de estos. Representado en la Figura 2.1 se puede observar como un diagrama relativamente sencillo.

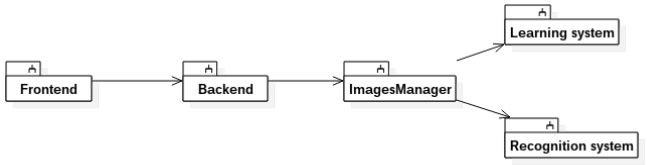

Figura 2.1: Diagrama de Componentes

Entre los componentes del diagrama se pueden ver:

- FrontEnd: es el componente encargado de la interacción directa con el usuario, lo que permite la carga de imágenes en el servidor para su posterior procesamiento.
- BackEnd: represente la primera capa del servidor y será el que recibe las peticiones a través del FrontEnd.
- ImagesManager: encargado del procesamiento de las imágenes.
- Learning System: encargado del entrenamiento de rostros para su posterior reconocimiento.
- Recognition System: encargado de reconocer un rostro por medio de la base de datos muestral almacenada en el sistema.

3 ANÁLISIS DE TROZOS

La siguiente sección consiste en el análisis profundo de cada uno de los primeros cuatro trozos implementados hasta el momento. Cada uno de estos trozos se implementaron a partir de las librerías OpenCv y Numpy. Al principio se intentó hacer la comparación entre la función de la librería y un código implementado por el grupo, pero aunque se obtuviera el mismo resultado las funciones de las librerías eran mucho más eficientes en tiempo de ejecución. Entre los trozos que se implementaron están: agregar una imagen y convertirla en matriz, calcular covarianza, sacar la transpuesta y convertir matriz en vector.

3.1 TROZO #1

```
def addImage(path):
    """
    @summary: This function reads the image

    Parameters
    ----------
    @param path: is the address of the file that needs to be read

    Returns
    ----------
    @return: the matrix of the image selected by the parameter path
    """

    return cv2.imread(path,0)
```

La función cv.imread() de la librería OpenCV es utilizada para poder leer una imagen. Esta función recibe dos parámetros, los cuales deben ser el path a la imagen que queremos leer

y el otro parámetro es una bandera que ayuda a especificar la manera en que se quiere que la imagen sea leída. Para que se pueda leer la imagen correctamente, la imagen debe estar en el directorio que se está trabajando, o tener el path completo. En este segundo parámetro se puede poner un 1 para leer la imagen a color, un 0 para leer la imagen en gris y un -1 para leer la imagen sin cambios. Para el proyecto, se utiliza leer la imagen en gris.

3.2 TROZO #2

```
def calculateCovarianceMatrix(samples):
    """
    @summary: This function calculates the covariance matrix of a given matrix
        containing
    the samples of the same image; where every column represents a sample and
        every row
    represents the different values for the same pixel.

    Parameters
    ----------
    @param samples: the matrix with all the samples of a given image. Which
        should have
    every sample as a column.

    Returns
    -------
    @return: the covariance matrix for the samples matrix of the image.
    """

    return np.cov(samples)
```

La función np.cov de la librería Numpy ayuda en lo que es obtener la coverianza de una matriz dada, en este caso las matrices de las imágenes. Como único parámetro se tienen lo que es la matriz, bajo el formato donde cada columna es una muestra y las filas representan sus pixeles. La misma retorna la matriz de covarianza respectiva a su entrada.

3.3 TROZO #3

```
def matrix2vector(self,matrix):
        """
        @summary: This function transforms a matrix that comes from an image to a
        vector ; every row goes consecutive in the vector, in the same order.

        Parameters
        ----------
        @param self: part of OOP syntax
        matrix: is the matrix of a given image.

        Returns
        ----------
        @return: the same matrix in an array format
        """

        return np.asarray(matrix).reshape(-1)
```

Básicamente, np.asarray de la librería Numpy sirve para convertir cualquier entrada como un array o vector en este caso, que es necesario para manejar los pixeles de las imágenes. Entonces, esta función facilita lo que es la transformación de la matriz obtenida de la imagen como un array o vector.

3.4 TROZO #4

```
def transpose():
    """
    @summary: This function transforms the matrix of images, puts each row as a
        column
    because each sample is as a row, but for the matrix of covariance

    Parameters
    ----------
    @param :  there are no parameters

    Returns
    ----------
    @return: the matrix of images transposed
```

```
    """

    return np.array(images).transpose()
```

NumPy también brindó la función np.array(nombre del array).transpose() que ayudó a acomodar a través de la transpuesta la matriz de manera que las filas se convirtieran en columnas y las columnas en filas, con el fin de convertirla a una matriz en el formato que se requiere para el calculo de la covarianza que se mostró anteriormente.

4 PROBLEMAS

Entre los problemas que se encontraron están la dificultad de encontrar la librería OpenCV en Eclipse usando el Sistema Operativo de Windows. A varios de los miembros del grupo que utilizaron Windows les pasó que la configuración de pyDev en Eclipse se desconfiguraba sin haber hecho ninguna modificación y se probaron diferentes soluciones brindadas por las comunidades de Stack Overflow, Github, entre otras, y ninguna funcionaba por lo que se tuvo que programar en Ubuntu o en Mac. A pesar de que programar en Eclipse suena como algo que iría a facilitar la tarea a los desarrolladores, se concluyo que fue menos productivo, por quitar mucho tiempo en buscar como solucionar este tipo de problemas.

Otro de los problemas fue el tamaño de matriz que generaba cada una de las imágenes por lo que por medio de la consola era muy difícil de observar el resultado y no se sabía si realmente era el resultado deseado. Para esto, se tuvo que implementar el manejo de archivos adicionales .txt para que estos se encargaran de mostrar la matriz completa y ver con los unit test si se obtuvo el resultado deseado. Sin embargo, uno de los archivos de texto que almacena el resultado esperado de la matriz de covarianza, quedó muy grande y por ello no se adjuntó al repositorio.

Por otro lado, la herramienta *Deoxygen* para la generación de documentación tipo html de la aplicación también mostró problemas, ya que los archivos referentes al *frontend* no los tomaba en cuenta (principalmente los html y javascripts), de manera que la documentación final generada no contaba con los archivos que deberían hacer referencias a los mismos.

Por último, uno de los problemas iniciales que se dieron al inicio fue el manejo de la herramienta zohoprojects, ya que ninguno de los miembros del grupo lo había usado antes se hizo complicada la tarea de entender en que parte se podían agregar las tareas.

5 RESULTADOS

Hasta el momento no se obtuvo ningún resultado negativo, ya que a partir de todas las pruebas unitarias se comprobó que si se obtenía el resultado deseado con lo que respecta a los trozos implementados. Además, si se logró la unión entre lo que es el front end y el back end con lo que fue la herramienta Django y se implemetó el paso de imágenes a través de esta.

5.1 PRUEBAS UNITARIAS

Se llevaron a cabo las pruebas unitarias con ayuda de la clase UnitTest que tiene Python, que trabaja junto a la herramienta eclipse. Esta clase contiene un conjunto de funciones que facilita realizar pruebas a los desarrolladores.Para este proyecto principalmente se usó assertEqual. Para el segundo trozo DEBE descargar la siguiente prueba: https://1drv.ms/u/s!AivfRA4YYnYCgcsfFKOWWW6QvarZ6Q. Y ubicarlo en Proyecto-Aseguramiento-II-S-2017 ->WebServer ->src ->ImagesManagement ->Pruebas.

5.1.1 Trozo #1

Para la probar la funcion addImages, creamos las respuestas en archivos de texto, para que pudieran ser leídos por la función loadtxt de numpy y así comparar con el resultado.

```
1     def test_addImages(self):
-         """
-         @summary: This function is an unit test for add images of the class
              Imagesmanager
-
5         Parameters
```

```
        ----------
        @param self: part of OOP syntax

        Returns
        ----------
        @return: void
        """
        self.G = 0
        imagesm = Imagesmanager()
        for path in self.path_list:
            result = Imagesmanager.addImage(imagesm, path)
            result = result.astype('float')

            self.assertEqual(result.all(), self.expected[self.G].all(), "No
                devuelve la misma matriz de la respectiva imagen")

            print(self.G)
            self.G = self.G + 1
```

Se probó con las 10 imágenes del sujeto 1. Los resultados esperados de las pruebas están en Proyecto-Aseguramiento-II-S-2017 ->WebServer ->src ->ImagesManagement ->Pruebas ->s1 matrizImg. A continuación se adjunta el resultado de las pruebas realizadas.

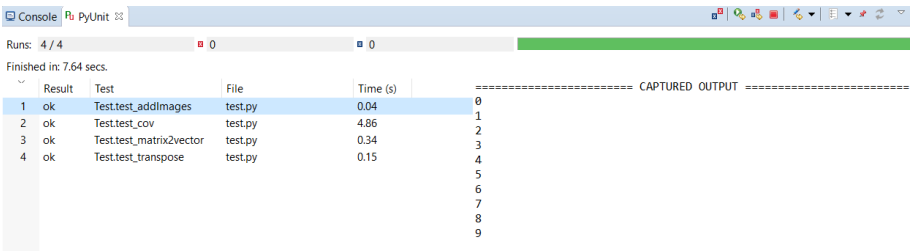

Figura 5.1: Resultado unit test 1

5.1.2 Trozo #2

Para la prueba unitaria del Trozo 2 el cual corresponde a la función de calcular la matriz de covarianza, el resultado generó una salida que es tan grande que dificulta mostrarla.El resultado fue guardado por medio de una función que lo guardaba en un archivo .txt y por esto se pudo comprobar que el resultado fue el mismo al que se esperaba.

```
    def test_cov(self):
        """
        @summary: This function is an unit test for calculating covariance matrix
             of the class Imagesmanager

        Parameters
        ----------
        @param self: part of OOP syntax

        Returns
        ----------
        @return: void
        """
        imagesm = Imagesmanager()

        self.G = 0


        result = imagesm.calculateCovarianceMatrix(np.loadtxt("Pruebas/muestra s1
            -10"))

        self.assertEqual(result.all(), self.expectedcov.all(), "No son la misma
            matriz")
```

Hicimos esta prueba con 10 sujetos, y el resultado fue correcto. Ubicado en Proyecto-Aseguramiento-II-S-2017 ->WebServer ->src ->ImagesManagement ->Pruebas. A continuación se adjunta el resultado de la prueba realizada.

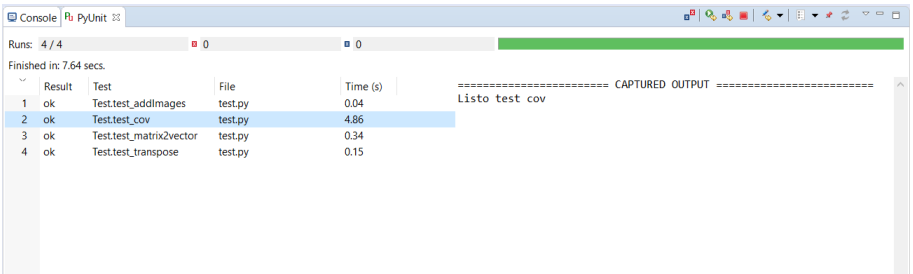

Figura 5.2: Resultado unit test 2

5.1.3 Trozo #3

Para la probar la funcion matrix2vector, creamos las respuestas en archivos de texto, para que pudieran ser leídos por la función loadtxt de numpy y así comparar con el resultado.

```
    def test_matrix2vector(self):
        """
        @summary: This function is an unit test for matrix2vector of the class
            Imagesmanager

        Parameters
        ----------
        @param self: part of OOP syntax

        Returns
        ----------
        @return: void
        """
        imagesm = Imagesmanager()
        self.G = 0
        for mat in self.expected:
            result = imagesm.matrix2vector(mat)
            result = result.astype('float')

            self.assertEqual(result.all(), self.expectedt2[self.G].all(), "El
                vector no coincide con el resultado esperado")

            print(self.G)
            self.G = self.G + 1
```

Se probó con las 10 matrices del sujeto 1 y se comprobó que se realizaran matrices transformadas en vectores de cada imagen del sujeto 1. Los resultados de las pruebas están ubicadas en Proyecto-Aseguramiento-II-S-2017 ->WebServer ->src ->ImagesManagement ->Pruebas ->s1 mxv. A continuación se adjunta el resultado de la prueba realizada.

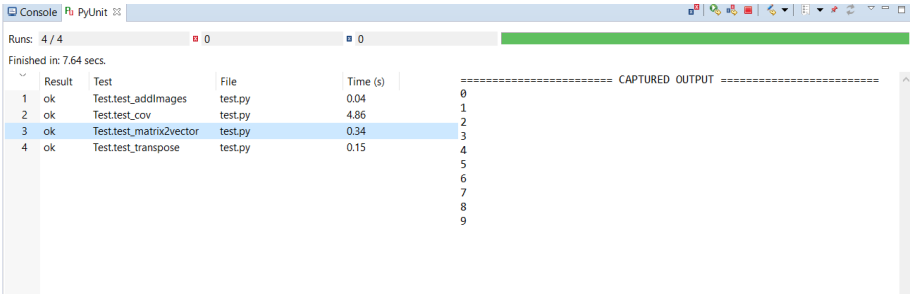

Figura 5.3: Resultado unit test 3

5.1.4 Trozo #4

Para la probar la funcion transpose, creamos las respuestas en archivos de texto, para que pudieran ser leídos por la función loadtxt de numpy y así comparar con el resultado.

```
    def test_transpose(self):
        """
        @summary: This function is an unit test for transpose of the class
            Imagesmanager

        Parameters
        ----------
        @param self: part of OOP syntax

        Returns
        ----------
        @return: void
        """
        imagesm = Imagesmanager()
        self.G = 0
        for muestra in self.samples:

            result = imagesm.transpose(muestra)
            result = result.astype('float')

            self.assertEqual(result.all(), self.samplesTranposed[self.G].all(), "
                La matriz dada no coincide con el resultado esperado")

            print(self.G)
            self.G = self.G + 1
```

Se probó con las muestras sin ser transpuestas de cada sujeto, y se comprobó que se realizaran las muestras transpuestas. A continuación se adjunta el resultado de la prueba realizada.

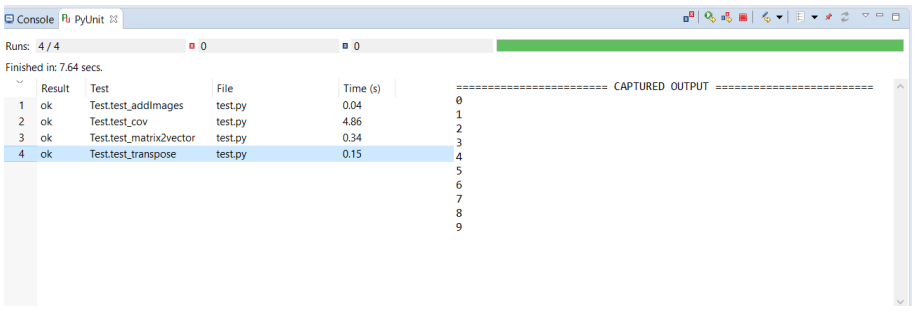

Figura 5.4: Resultado unit test 4

Los resultados de las pruebas están ubicadas en Proyecto-Aseguramiento-II-S-2017 ->WebServer ->src ->ImagesManagement ->Pruebas ->MuestrtasTranspuestas.

5.2 PÁGINA WEB

En está sección se mostrará el proceso que se sigue para cargar una imagen de la página web al servidor (figuras 5.5, 5.6, 5.7 y 5.8, en dicho orden) y la respuesta que se recibe del mismo, la cual es la matriz en escala de grises de la imagen que se ha cargado. En caso que se presiona el botón de enviar sin haber seleccionado una imagen, se mostrará de nuevo la página de inicio.

Figura 5.5: Página de Inicio

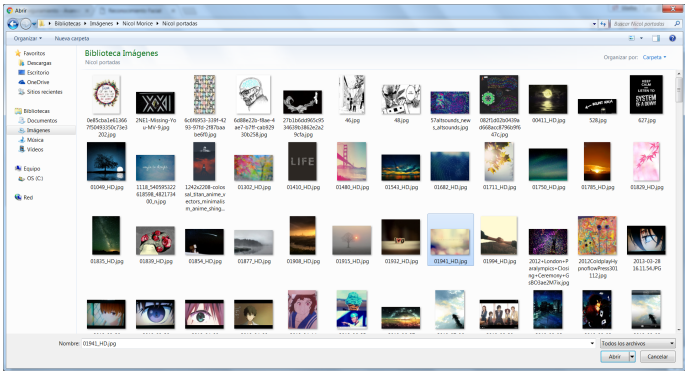

Figura 5.6: Página para abrir imagen

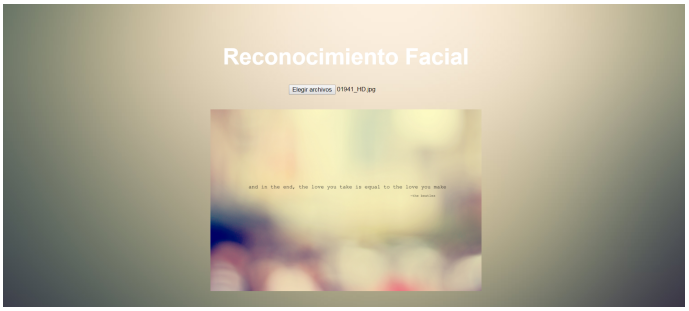

Figura 5.7: Página con imagen cargada

La imagen se ha procesado con éxito

[[255 255 255 ..., 255 255 255] [255 255 255 ..., 255 255 255] [255 255 255 ..., 255 255 255] ..., [255 255 255 ..., 255 255 255] [255 255 255 ..., 255 255 255] [255 255 255 ..., 255 255 255]]
Volver a inicio

Figura 5.8: Página que muestra el resultado de la carga

Como se puede observar la última figura (5.8) muestra el resultado de la imagen como una matriz en su escala de grises, debido a su tamaño, puede que solo se muestren algunas partes de la misma y otras serán obviadas mediante el uso de tres puntos (...), por lo que el resultado será distinto de acuerdo a la imagen que se escoja. Esto con el fin de mostrar que existe una conexión entre *Frontend*, *Backend* y las funciones de procesamiento de imágenes.

6 CONCLUSIONES

Finalmente, a pesar de los problemas que no eran muchos si consumieron gran parte del proceso de realización de cierta manera las librerías compensaron ese tiempo, ya que realmente fue bastante rápido la codificación de las matrices con estas herramientas. Pero no solo eso, el montar el server con Django no se volvió una tarea tediosa como lo son otras herramientas.

El trabajar con la herramienta zohoprojects al principio si fue un difícil porque no se sabía bien como era su funcionamiento a la hora de la creación de tareas porque no se sabía bien como iba a ser su distribución. Pero finalmente se puso realizar una buena organización y distribución de lo que era el trabajo.

Otra de las cosas que facilito mucho el trabajo fue el manejo de un repositorio con GitHub a través de GitKraken, ya que se podía ver quién hacia que y que cambios explícitos realizó la persona.

Por último, se logró cada una de las partes hasta el momento por lo cuál el PoC fue un pedazo del Proyecto exitoso.