INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

SOFTWARE DE APOYO AL ANÁLISIS RADIOLÓGICO DE TOMOGRAFÍAS AXIALES COMPUTARIZADAS

TT 2016-B018

Manual técnico

Alumnos:

León Díaz Raúl Alberto

Osnaya Gómez Alexis Alan

Ríos López José Alberto

Santiago Nieves Edgar Augusto

Directores:

Franco Martínez Edgardo

Rosas Trigueros Jorge Luis

Contenido

Lista de tablas

1. Introducción

Este documento contiene la información necesaria para poder realizar modificaciones o actualizaciones en el Trabajo Terminal Software de Apoyo al Análisis Radiológico de Tomografías Axiales

Computarizadas. En él se encuentran tanto los requerimientos mínimos que se necesitan para la ejecución del programa, como las clases principales que lo componen.

1.1. Requerimientos mínimos de hardware

1.1.1. Visual Studio 2017

Para poder realizar modificaciones al sistema, ejecutarlo o añadir más herramientas se necesita el IDE Visual Studio 2017. Por lo tanto el equipo en donde se instale Visual Studio 2017 debe cumplir con los siguientes requisitos del sistema marcado en la tabla 1.1:

Requisitos	Descripción
Procesador	Procesador 1.8 GHz. Dual-Core recomendado.
RAM	Memoria RAM de 2GB. 4GB recomendado.
Disco Duro	Disco duro con 130GB de espacio disponible, dependiendo de las características que se instalen.
Tarjeta de video	Tarjeta de video con una resolución mínima de 720p.
Sistema Operativo	Windows 7 o posterior.

Tabla 1.1 Requerimientos para instalar Visual Studio. *[1]*

1.2. Requerimientos de software

1.2.1. Instalación de Visual Studio 2017

La instalación de Visual Studio 2017 se puede hacer mediante la descarga desde la página oficial de Microsoft (https://www.visualstudio.com/downloads/) [1]. Para poder realizar la instalación se deberán seguir los siguientes pasos:

- Descargar los archivos que componen el sistema.
- Ejecutar el archivo de instalación como se ve en la figura 1.1.

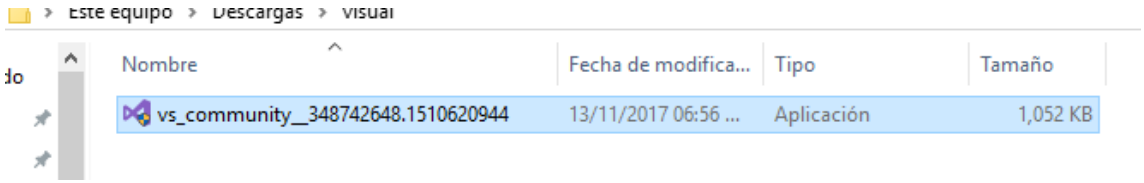

Figura 1.1 Ejecución de archivo de instalación.

- Se tienen que leer y aceptar los términos de privacidad como se ve en la figura 1.2.

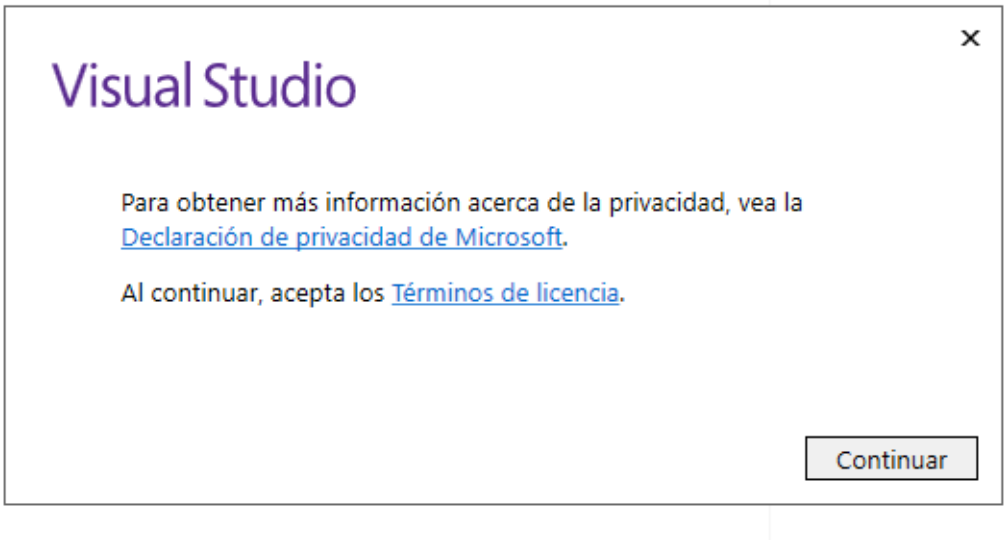

Figura 1.2 Archivos de privacidad.

- El instalador Visual Studio te proporciona algunas herramientas extras a las que incluye la instalación, escogemos las herramientas que vamos a utilizar como se puede observar en la figura 1.3.

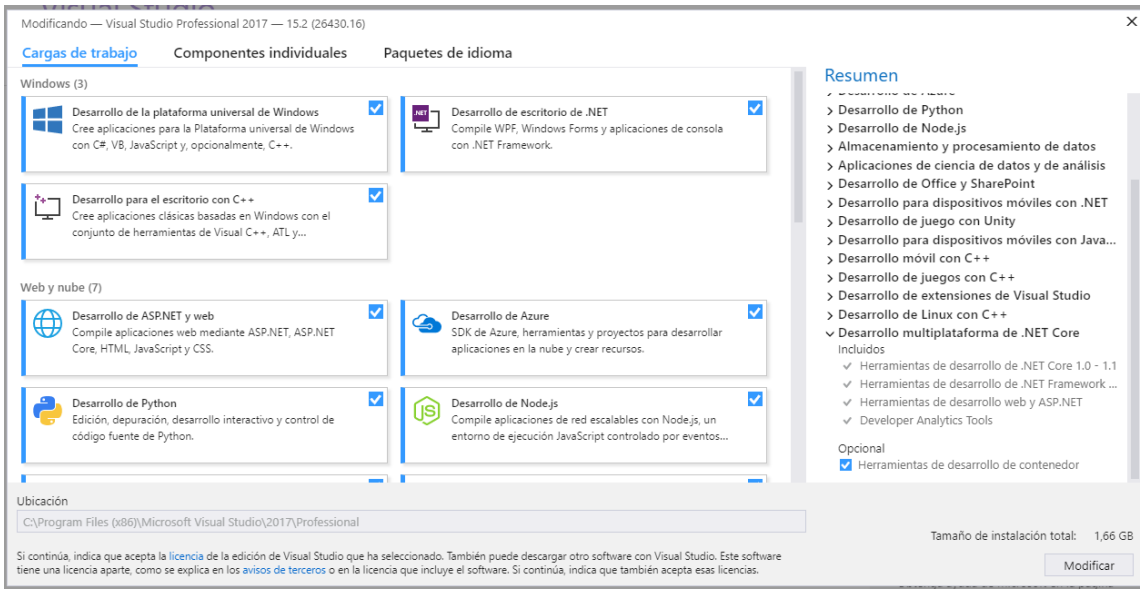

Figura 1.3 Elección de herramientas que se pueden descargar.

- Finalmente le daremos click en la opción que dice “Modificar” como se puede ver en la figura 1.4.

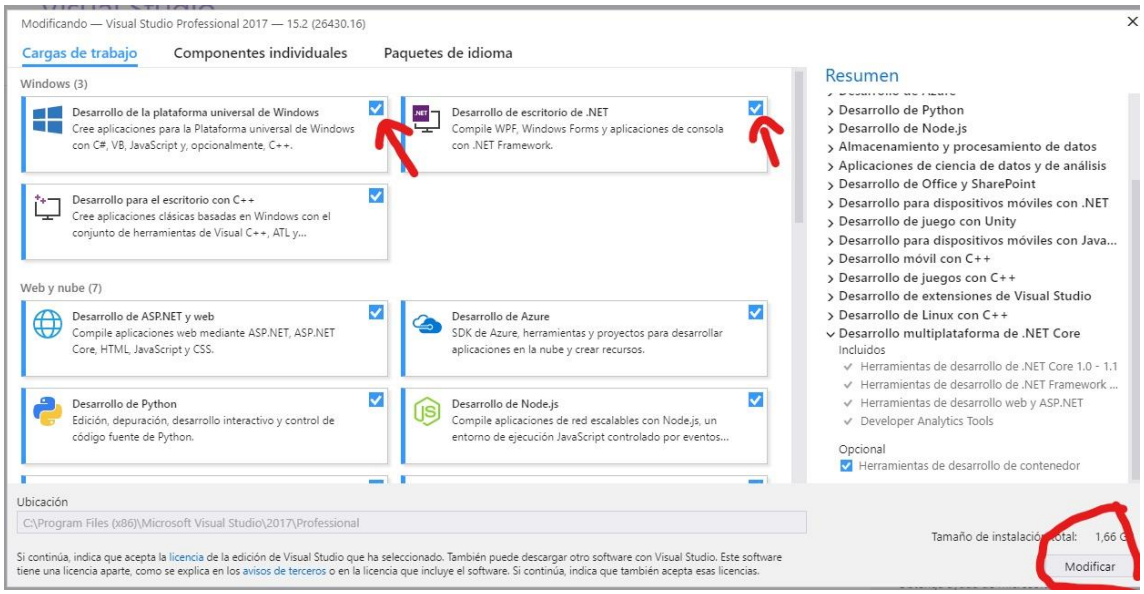

Figura 1.4 Finalización de instalación.

1.2.2. Instalación de Python

Si se desea hacer modificaciones en el sistema es necesario contar con Python en tu computadora, debido a que realizara el proceso de obtener los datos a partir de un archivo DICOM.

La descarga de Python para Windows se puede hacer desde su página oficial (https://www.python.org/download/releases/2.7/) [2]. Para poder realizar la instalación se deberán seguir los siguientes pasos:

- Ejecutar el archivo de instalación como se puede observar en la figura 1.5.

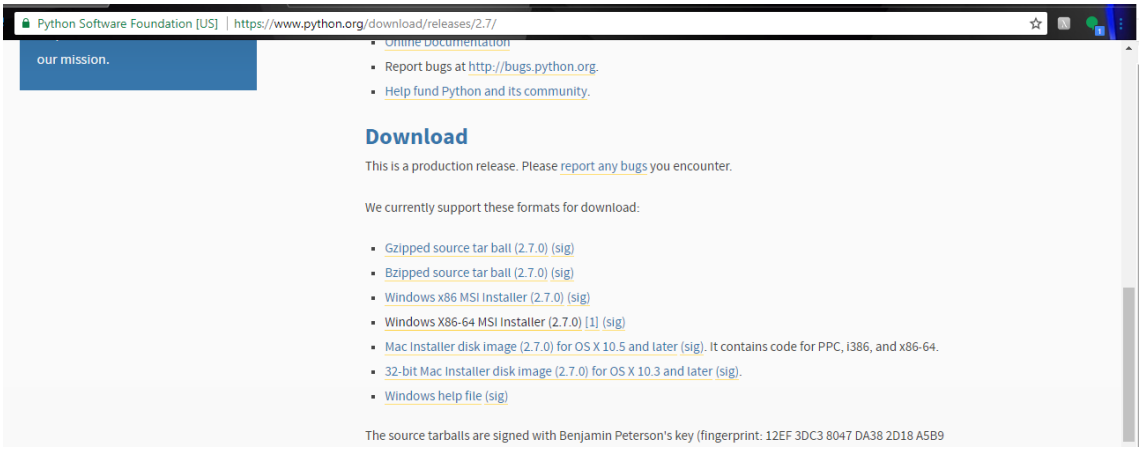

Figura1.5 Selección de archivo de instalación.

- Ejecutar el archivo de instalación de Python como se puede observar en la figura 1.6.

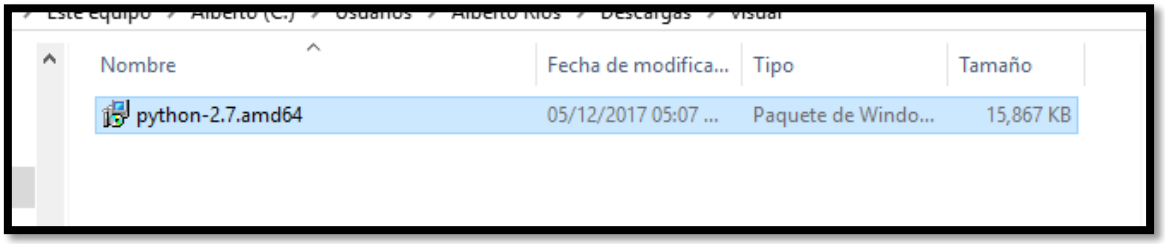

Figura 1.6 Ejecución de archivo de instalación.

- Seleccionar para que usuarios en el sistema será instalado Python como se puede ver en la figura 1.7.

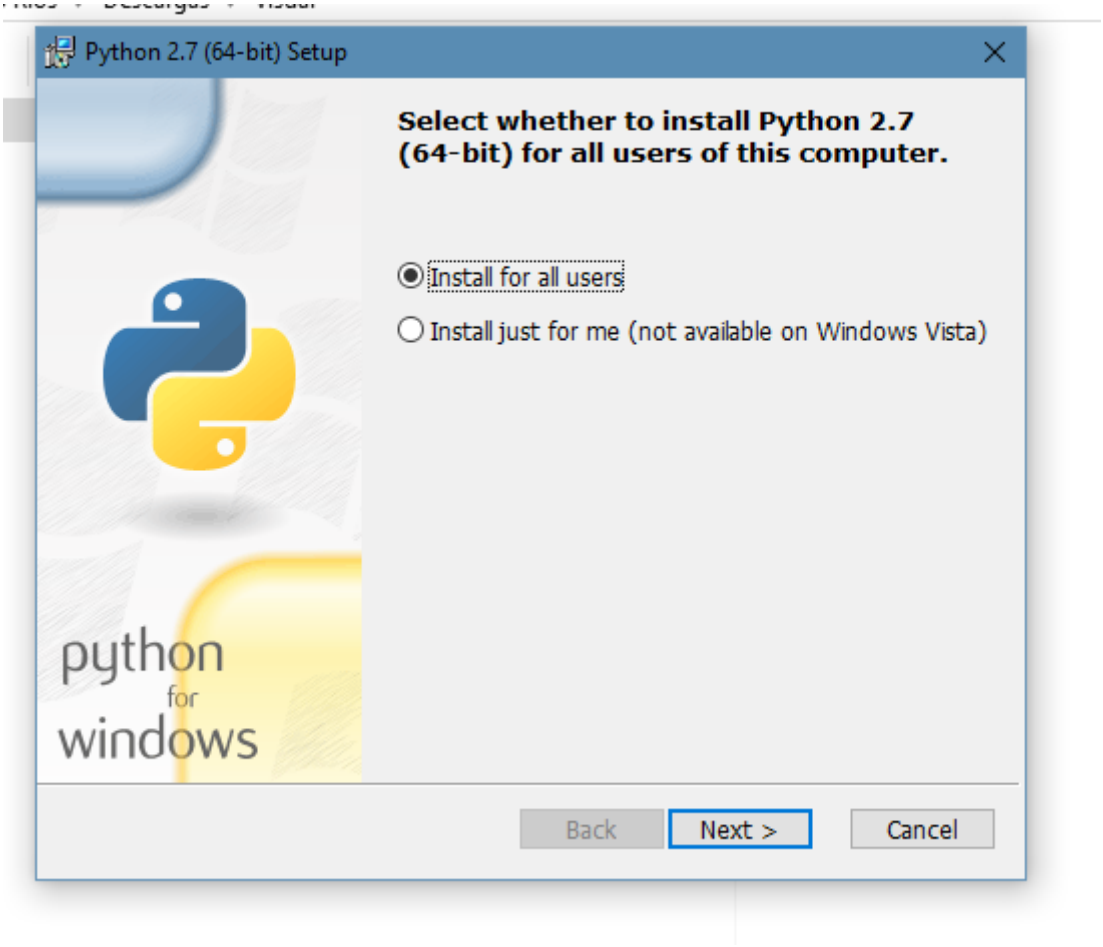

Figura 1.7 Selección de usuarios Python.

- Escoger carpeta en donde será instalado Python como se puede ver en la figura 1.8.

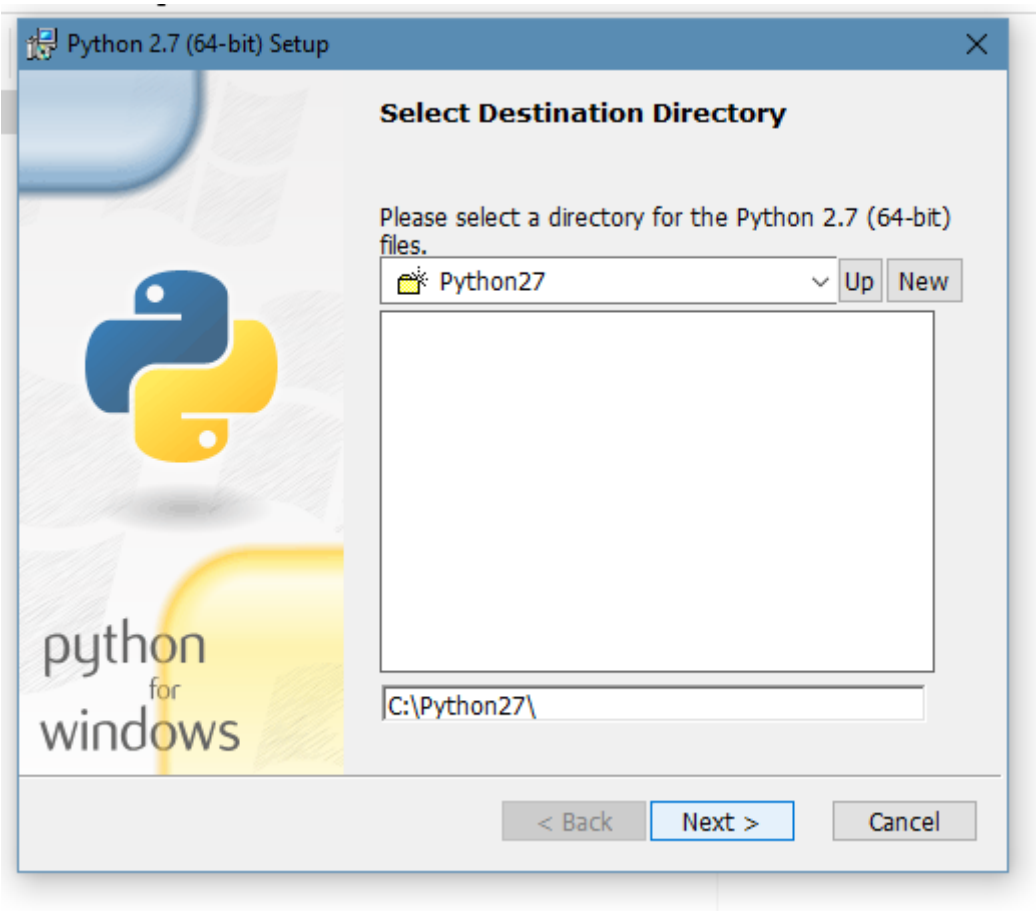

Figura 1.8 Selección de carpetas de almacenamiento.

- Para acabar con la instalación le damos click a la opción “Finalizar” como se observa en la figura 1.9.

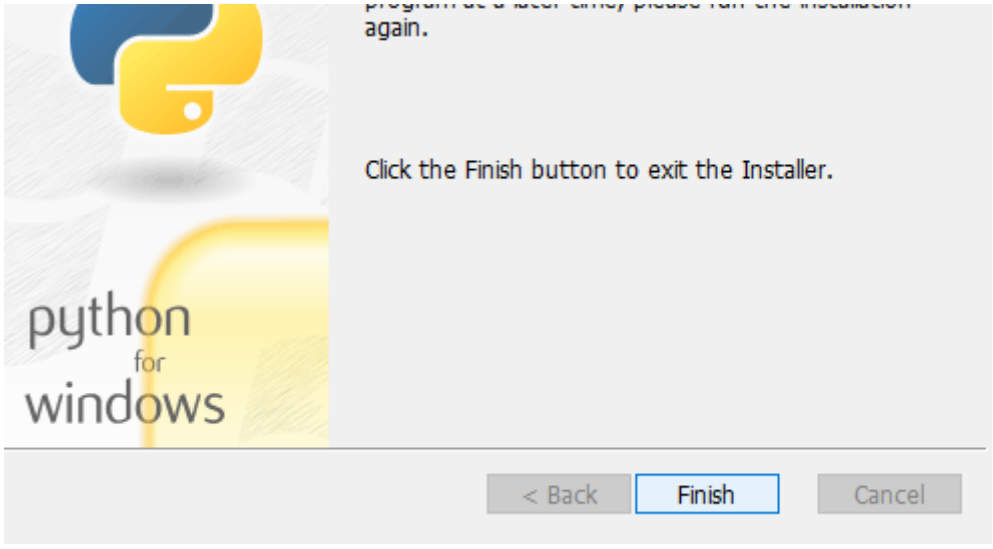

Figura 1.9 Finalización de instalación Python.

1.2.3. Instalación de Pydicom

Se usa una librería especial de Python para poder manipular archivos DICOM. Se puede descargar esta librería desde la página http://pydicom.readthedocs.io/en/stable/getting_started.html [3]. Para poder realizar la instalación se deberán seguir los siguientes pasos:

- Seleccionar el archivo de instalación de Pydicom adecuado a las características de tu ordenador como se puede observar en la figura 1.10.

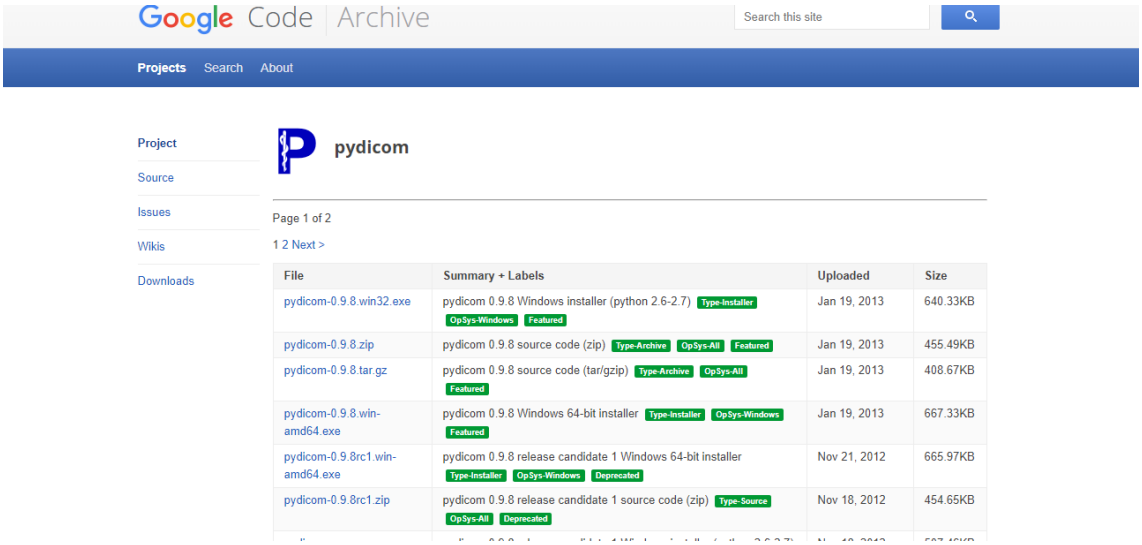

Figura 1.10 Selección de archivo de instalación Pydicom.

- Ejecutamos el archivo de instalación que descargamos previamente como se puede ver en la figura 1.11.

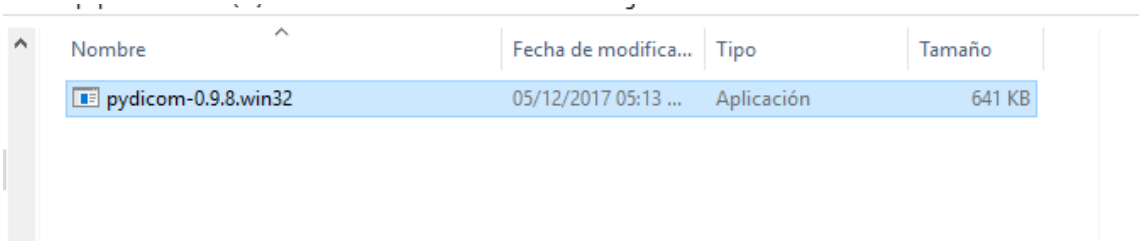

Figura 1.11 Ejecución de archivo de instalación Pydicom.

- Leer y aceptar los términos y condiciones como en la figura 1.12.

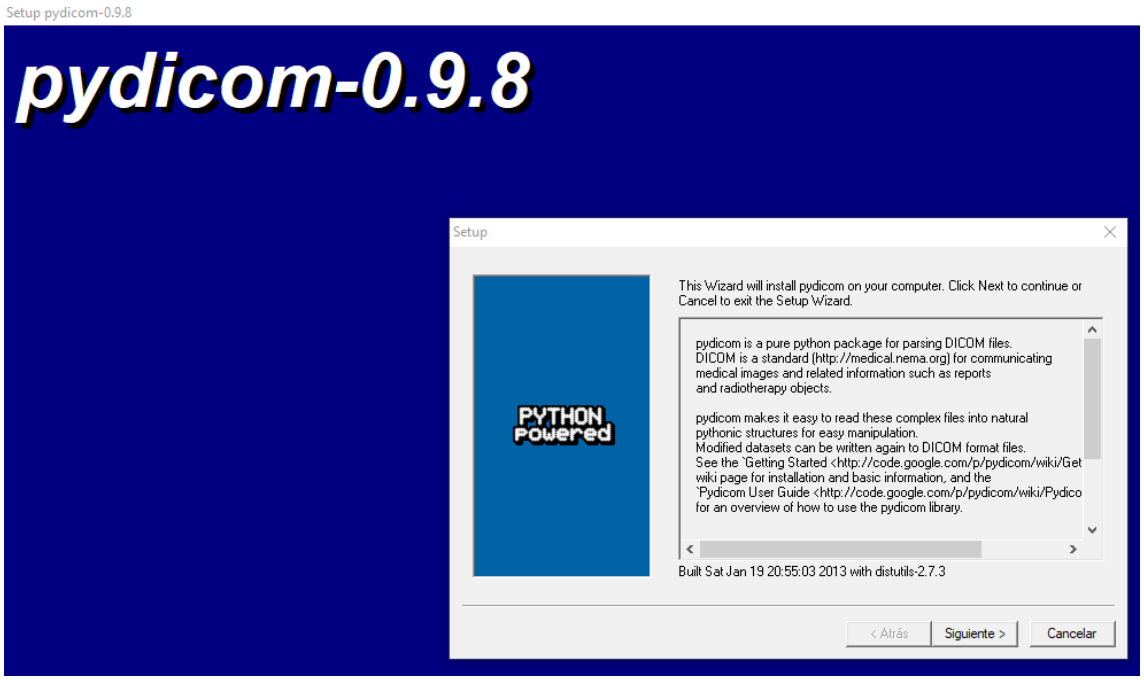

Figura 1.12 Términos y condiciones de Pydicom.

- Escoger carpeta donde se instalará Pydicom como se puede observar en la figura 1.13.

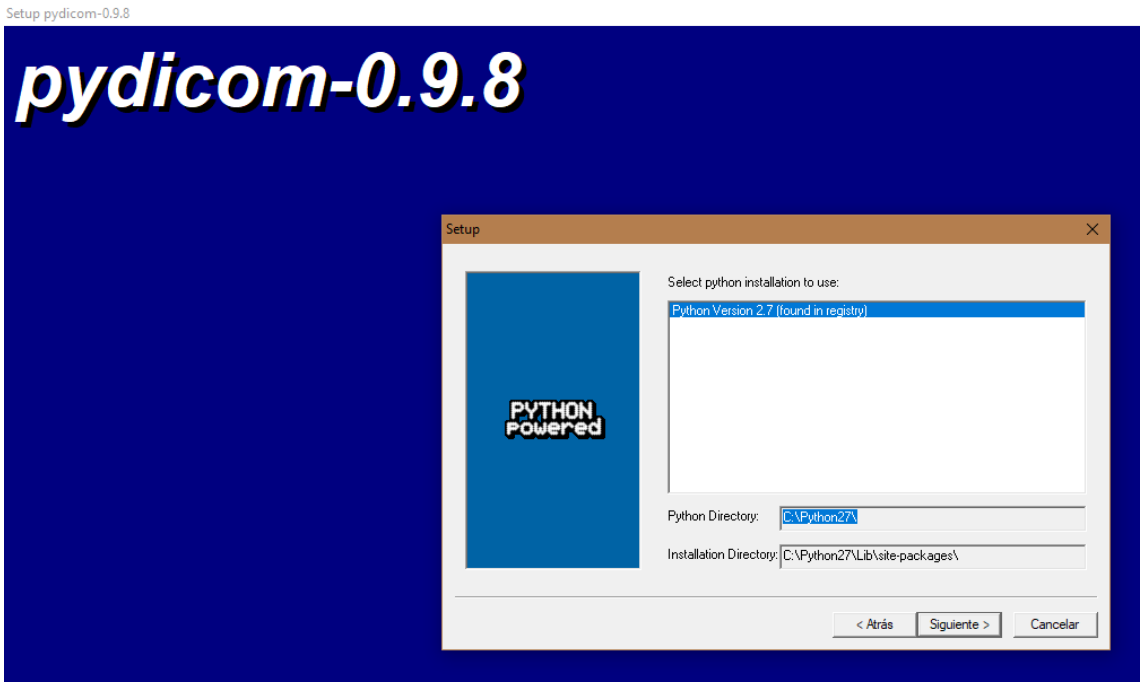

Figura 1.13 Selección de carpetas de almacenamiento de Pydicom.

- Finalmente le damos click a la opción de finalizar figura 1.14.

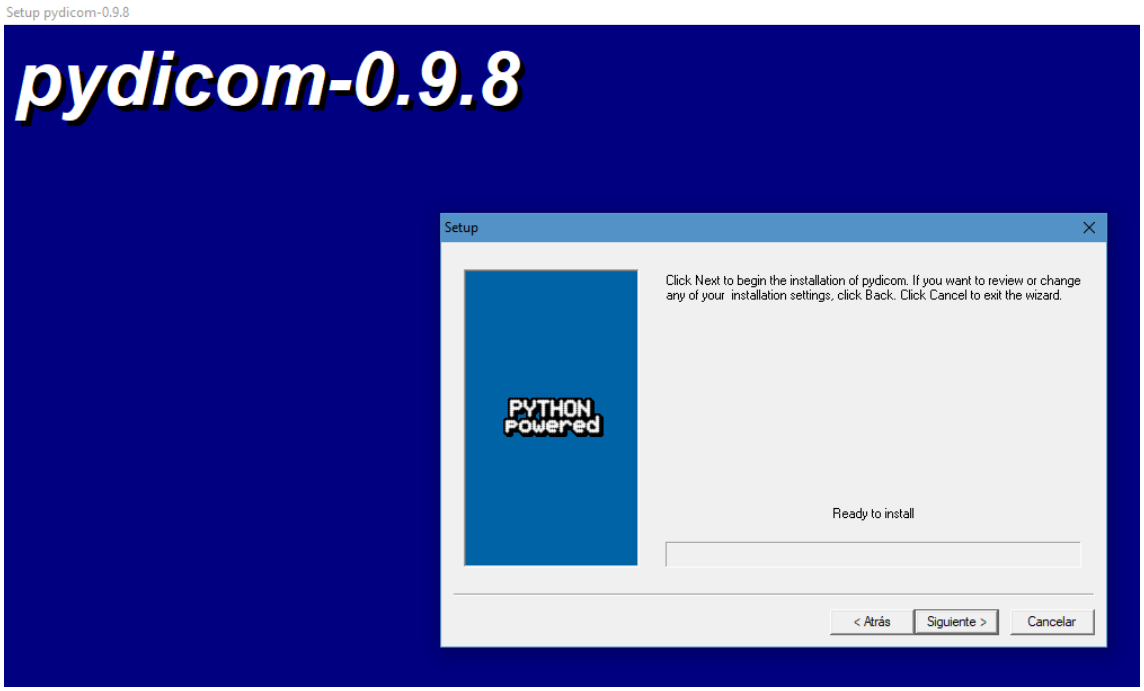

Figura 1.14 Finalización de instalación de Pydicom.

2. Estructura del sistema

2.1. Diagrama de clases

El anexo A contiene la figura A.1 tal en la que se muestra el diagrama de clases que son utilizadas en el sistema. Dichas clases son:

- **Contraste:** Esta clase nos ayuda a poder realizar el mapeado de los datos obtenidos del archivo DICOM a una escala de grises para así, visualizarla en pantalla.

- **FuzzyCMeans:** En esta clase se implementa el algoritmo de clasificación Fuzzy C-Means que agrupa regiones con base a una distancia euclidiana y a un valor de pertenencia.

- **kMeans:** Esta clase implementa el algoritmo de clasificación K-Means que logra agrupar regiones con base a una distancia euclidiana.

- **LecturaArchivosDicom:** Clase que crea proceso el cual, mediante el programa sum.py logra obtener los datos de un archivo en formato DICOM.

- **MatrizDicom**: Esta clase contendrá los datos obtenidos a partir de LecturaArchivosDicom y que contiene métodos de manipulación de dicha matriz.

- **RegionCreciente:** Clase que implementa el algoritmo de región creciente para poder segmentar una imagen, de tal manera que el usuario pueda colocar una semilla inicial.

- **Regla:** Esta clase crea permite poder tomar mediciones en la imagen a partir de dos puntos en la misma.

- **Seccion:** En esta clase se permite al usuario seleccionar una región específica de la imagen, en donde se almacenara la matriz delimitada.

- **SplitMerge:** Clase que implementa el método de segmentación de Split and Merge en donde de una forma se segmenta la imagen.

- **Umbralizacion:** En esta clase se almacenan los valores predefinidos por la escala de Hounsfield y en la cual se implementa una umbralizacion por definición.

- **Ventana:** Esta clase será la interfaz en donde el usuario podrá visualizar y manipular archivos DICOM como diferente tipos de métodos.

3. Funciones

3.1. Decodificación de archivos DICOM

Se crea un proceso en el cual mediante el lenguaje Python y las librerías usadas en el (Pydicom) se accederán a varios campos del archivo DICOM con el fin de obtener los datos que se utiliza el sistema. Dichos campos son:

- **DimensionesPixel:** Campo del archivo DICOM que nos dice las dimensiones de los pixeles que se manejarán.
- **Nombre:** Campo que proporciona el nombre completo del paciente.
- **Edad:** Campo que proporciona la edad del paciente.
- **Sexo:** Campo que proporciona el sexo del paciente.
- **Fecha:** Campo que proporciona la fecha en la cual fue realizada el estudio.
- **Espesor Rebanada:** Campo que proporciona, en milímetros la distancia que hay entre un corte axial y el otro.
- **Hospital:** Campo que proporciona el nombre del hospital donde fue tomada la tomografía.
- **Ventana UH:** Campo que proporciona la ventana recomendada para que se puede visualizar la tomografía mediante una escala de grises.
- **Matriz DICOM:** Campo que nos proporciona los datos que son presentados en forma de matriz del archivo DICOM.

3.2. Umbralización

A pesar de que la umbralización es una técnica de segmentación se muestra separada debido a la importancia que tiene en este sistema. La técnica que se usó para la umbralización es una umbralización por definición tomando como base los valores de la escala de Hounsfield que se encuentran en un rango de -1000 a 1000 o 2000 dependiendo la tomografía. Los tejidos y/o sustancias considerados para esta umbralización son mostrados a continuación:

- **Agua:** Con un centro de 0 y una ventana de 10.
- **Aire:** Con un centro de -1000 y una ventana de 150.
- **Fluido CerebroEspinal:** Con un centro de 11 y una ventana de 6.
- **Sustancia Cerebral Blanca:** Con un centro de 40 y una ventana de 6.
- **Grasa:** Con un centro de -75 y una ventana de 25.
- **Hígado:** Con un centro de 65 y una ventana de 5.

- **Páncreas:** Con un centro de 40 y una ventana de 10.
- **Riñón:** Con un centro de 30 y una ventana de 10.
- **Hueso compacto:** Con un centro de 1000 y una ventana de 750.
- **Hueso esponjoso:** Con un centro de 130 y una ventana de 100.
- **Pulmones:** Con un centro de -700 y una ventana de 200.
- **Sangre:** Con un centro de 55 y una ventana de 5.
- **Sangre Coagulada:** Con un centro de 60 y una ventana de 5.

3.3. Clasificación

Los métodos clasificadores son técnicas de reconocimiento de patrones que buscan una partición a una característica espacial dada por un una imagen usando datos con etiquetas conocidas. Una característica espacial es un rango de espacio de cualquier función de una imagen, siendo la característica espacial más común las intensidades de la imagen. Se usaron dos métodos para esta etapa de clasificación.

3.3.1. KMeans

En el clustering de K-medias, particiona una colección de datos en k números de grupos de datos. Clasifica un conjunto de datos ya dados en k números de clúster desarticulados. El algoritmo de K-medias es separado dos fases. En la primer fase se calcula el centro k y en la segunda fase se traslada cada punto al clúster en el cual tiene el centro más cercano desde su respectivo punto de dato. Hay diferentes métodos que definen la distancia entre el centro más cercano y uno de los métodos más utilizados es la distancia Euclidiana. Una vez que el agrupamiento ya se realizó se recalcula un nuevo centro para cada clúster y basado en ese centro, una nueva distancia Euclidiana es calculada entre cada centro y cada punto y asigna los puntos al clúster que tiene la mínima distancia Euclidiana. Usando este concepto se toma en cuenta la similitud de cada pixel con base al valor UH con su centroide más cercano.

En el anexo B se encuentra el código correspondiente a este método.

3.3.2. Fuzzy CMeans

Método en donde se toma en cuenta la distancia euclidiana que tiene un pixel con su centroide siendo esta la similitud del valor de UH con el del centroide y fungiendo como segundo filtro un valor de pertenencia en donde se valúa este valor que tiene el pixel con sus respectivos centroides tomando en el mayor valor o por decirlo en otras palabras, al cual se encuentra más pertenecido.

En el anexo C se encuentra el código correspondiente a este método.

3.4. Segmentación

La segmentación de imágenes está definida como el particionado de una imagen en regiones constituyentes que no se solapan y las cuales son homogéneas con respecto a alguna característica como intensidad o textura. Se usaron dos técnicas de segmentación.

3.4.1. Región Creciente

El método de región creciente es una técnica para extraer una región de la imagen que está conectada basada en un criterio predefinido. En este caso este criterio es su similitud que tiene el pixel con sus 8 vecinos. En su forma más simple, la región creciente requiere una semilla que es seleccionada manualmente por el operador y extrae todos los pixeles conectados a la semilla inicial basado en la similitud del pixel.

En el anexo D se encuentra el código correspondiente a este método.

3.4.2. Split and Merge

Las técnicas de Split and Merge intentan solucionar los problemas de ruido y los falsos bordes usando una medida controlada de homogeneidad. El objetivo es segmentar automáticamente la imagen en un mínimo número de regiones que intentan representar áreas de uniformidad, produciendo bordes con características relacionadas con la resolución de la imagen.

En el anexo E se encuentra el código correspondiente a este método.

3.5. Herramientas de análisis

Con el fin de apoyar al usuario para poderle brindar diferentes perspectivas de la imagen que se está visualizando, se contempló añadir herramientas para hacer una sencilla manipulación a la imagen en donde desplegaremos algunos datos básicos. Estas herramientas de análisis se listarán a continuación:

- **Rotar:** Aquí el usuario podrá rotar la imagen 90 grados, sea a la izquierda o a la derecha.
- **Distancia:** Se seleccionan dos puntos en la imagen y mediante el teorema de Pitágoras se obtiene la distancia real entre esos dos puntos basándose en la distancia contenida en archivos DICOM.
- **UH por Pixel:** Mediante las coordenadas que se tiene en la matriz, se obtiene el valor UH del pixel conformado por esas coordenadas, mismo que se saca a partir de un archivo DICOM.

- **UH por región:** Seleccionando una región dentro de la imagen se obtienen todos los pixeles contenidos en la misma.

- **Seleccionar región:** Al seleccionar un área en específico de la imagen se delimitarán los tratamientos posteriores a dicha área.

- **Exportar imagen:** Herramienta que convierte, ya sea la imagen original o la imagen tratada a un formato jpg.

- **Reconstrucción 3D:** A partir del archivo DICOM ingresado se reconstruye una imagen en tercera dimensión.

Anexos

Anexo A

Diagrama de clases

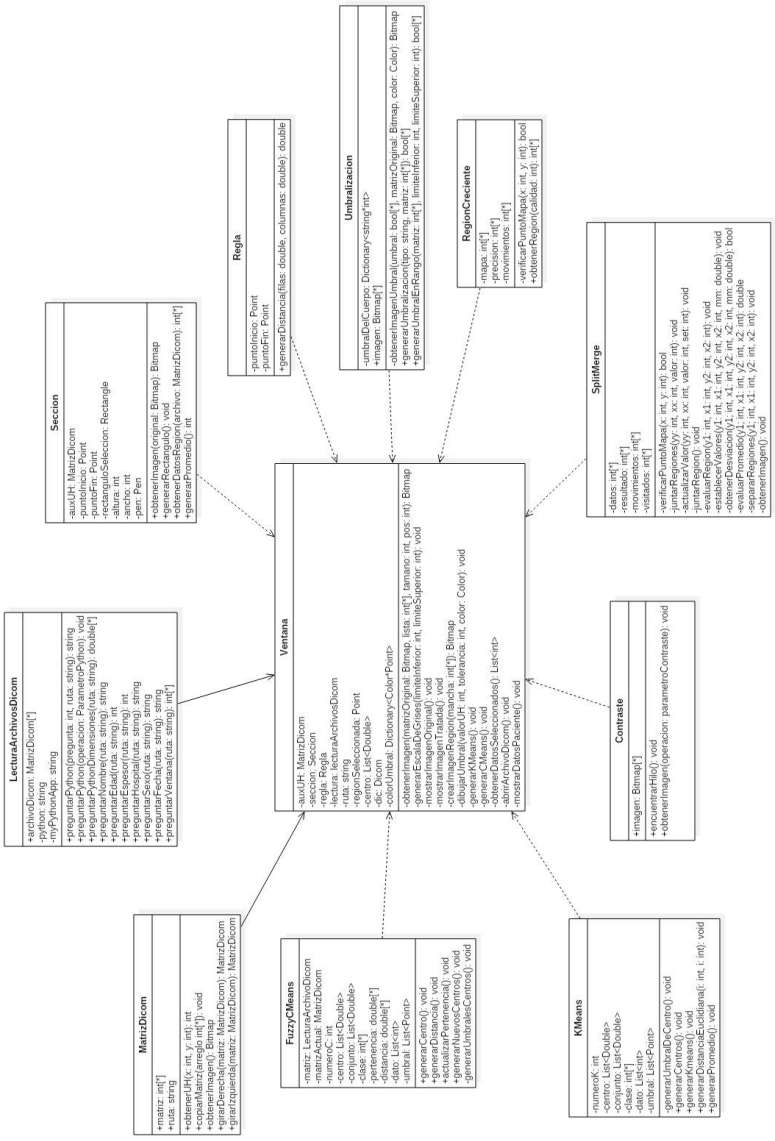

Figura A.1 Diagrama de clases.

Anexo B

Código K Means

Esta función genera centros aleatorios.

```
public void generarCentros(){
            centros = new List<Double>();
            for (int i = 0; i < numerosK; i++) {
                int numero = rnd.Next((IgnorarAire ? IGNORAR : -1000),
1400);
                centros.Add((double)numero);
            }
        }
```

Esta función Euclidiana entre dos densidades.

```
public void distanciaEuclidiana(int i, int j){
            if (IgnorarAire && datos[i][j] < IGNORAR) return;
            int indc = 0;
            conjunto.Clear();
            foreach (Double indice in centros){
                Double resta = Math.Abs(datos[i][j] - indice);
                conjunto.Add(resta);
            }
            for (int k = 1; k < conjunto.Count; k++)
                if (conjunto[indc] > conjunto[k])
                    indc = k;
            clases[i, j] = indc + 1;
        }
```

La función promedio saca el promedio de las densidaes en el clúster y genera nuevos centros con base en el promedio calculado.

```
public void promedio(){
            centros.Clear();
            double[] sumas = new double[numerosK];
            double[] contador = new double[numerosK];
            for (int i = 0; i < numerosK; i++)
                sumas[i] = contador[i] = 0;
            for (int i = 0; i < datos.Count; i++){
                for (int j = 0; j < datos[i].Length; j++){
                    if (IgnorarAire && datos [i] [j] < IGNORAR)
                        continue;
                    sumas[clases[i, j] - 1] += datos[i][j];
                    contador[clases[i, j] - 1]++;
                }
                if (reporte_progreso.CancellationPending)
                    return;
                operaciones_cargando++;
                reporte_progreso.ReportProgress((90 *
operaciones_cargando) / operaciones_total);
            }
            centros.Clear();
            for (int i = 0; i < numerosK; i++) {
                if (contador [i] == 0) {
                    centros.Add(ObtenPixelRandom());
                } else {
                    centros.Add(sumas [i] / contador [i]);
                }
            }

        }
```

Anexo C

Código Fuzzy C-Means

Se generan los centros aleatorios.

```
public void generarCentros(){
            centros = new List<Double>();
            rnd = new Random();
            for (int i = 0; i < numerosK; i++) {
                int numero = rnd.Next((IgnorarAire ? IGNORAR : -1000),
1400);
                centros.Add((double)numero);
            }
        }
```

Se calcula la distancia con base a la diferencia de densidades.

```
        public void GenerarDistancias(){
            for (int i = 0; i < N; i++){
                if (reporte_progreso.CancellationPending)
                    return;
                for (int j = 0; j < M; j++){
                    if (IgnorarAire && datos [i] [j] < IGNORAR)
                        continue;
                    for (int k = 0; k < numerosK; k++){
                        double dist = datos[i][j] - centros[k];
                        distancias[i, j, k] = dist * dist;
                    }

                }
            }
        }
```

Se crea la matriz de pertenencia, dando un valor de pertenencia a cada densidad para cada clúster.

```
public void ActualizarPertenencia(){
            for (int i = 0; i < N; i++){
                operaciones_cargando++;
                reporte_progreso.ReportProgress((operaciones_cargando
* 90) / operaciones_total);
                if (reporte_progreso.CancellationPending)
                    return;
                for (int j = 0; j < M; j++){
                    if (IgnorarAire && datos [i] [j] < IGNORAR)
                        continue;
                    for (int k = 0; k < numerosK; k++){
                        double sum = 0.0;
                        for (int l = 0; l < numerosK; l++)
                            sum += Math.Pow(distancias[i, j, k] /
distancias[i, j, l], 2.0 / (m - 1.0));
                        pertenencia[i, j, k] = 1.0 / sum;

                    }

                }
            }
        }
```

Se generan los nuevos centros para las iteraciones siguientes.

```
public void GeneraNuevosCentros(){
            for (int k = 0; k < numerosK; k++){
                long aa = 0;
                long bb = 0;
                for (int p = 0; p < N; p++){
                    if (reporte_progreso.CancellationPending)
                        return;
                    for (int i = 0; i < M; i++){
                        if (IgnorarAire && datos [p][i] < IGNORAR)
                            continue;
                        double valor =
Math.Round(Math.Pow(pertenencia[p, i, k], m), 5);
                        if (valor <= 0.00001)
                            continue;
                        aa += (long)(Math.Round(valor * datos[p][i],
5) * 100000);
                        bb += (long)(valor * 100000);

                    }
                }
                if (bb <= 0.0001) {
                    centros [k] = ObtenPixelRandom();
                } else {
                    centros [k] = (double)aa / (double)bb;
                }
            }
        }
```

Anexo D

Región Creciente

Con esta función se genera la segmentación del objeto con base a una semilla dada por el usuario.

```
public int[,] ObtenerRegion(int calidad = 0) {
            int [,] salida = new int [N, M];
            Queue busqueda = new Queue();
            busqueda.Enqueue(new Tuple<int, int>(origenY, origenX));
            salida [origenY, origenX] = 1;
            while(busqueda.Count > 0) {
                Tuple<int, int> actual = (Tuple <int, int>)
busqueda.Dequeue();
                for(int i = 0; i < 8; i++){
                    int y = actual.Item1 + movimientos [i, 0];
                    int x = actual.Item2 + movimientos [i, 1];
                    if(DentroMapa(y, x)) {
                        if (salida [y, x] == 1 ||
Math.Abs(mapa[origenY, origenX] - mapa[actual.Item1, actual.Item2]) >
precision[calidad])
                            continue;
                        salida [y, x] = 1;
                        busqueda.Enqueue(new Tuple<int, int>(y, x));
                    }
                }
            }
            return salida;
        }
```

Anexo E

Split and Merge

Esta función verifica para cada región si exsite similutud en ella, si no se cumple la condición, las regiones se separan.

```
private void EvaluacionRegion(int y1, int x1, int y2, int x2) {
            if (y1 > y2 || x1 > x2)
                return;
            double promedio = EvaluacionPromedio(y1, x1, y2, x2);
            if (Desviacion(y1, x1, y2, x2, promedio)) {
                SeparaRegiones(y1, x1, y2, x2);
            } else {
                EstablecerValores(y1, x1, y2, x2, promedio);
            }

        }
```

Esta función separa la imagen en 4 regiones para su evaluación.

```
private void SeparaRegiones(int y1, int x1, int y2, int x2) {
            int PMY = (y1 + y2) / 2;
            int PMX = (x1 + x2) / 2;
            EvaluacionRegion(y1, x1, PMY, PMX);
            EvaluacionRegion(y1, PMX + 1, PMY, x2);
            EvaluacionRegion(PMY + 1, x1, y2, PMX);
            EvaluacionRegion(PMY + 1, PMX + 1, y2, x2);
        }
```

Esta función junta las regiones separadas.

```
private void JuntaRegiones(int yy, int xx, int valor) {
            Queue busqueda = new Queue();
            busqueda.Enqueue(new Tuple<int, int>(yy, xx));
            visitados [yy, xx] = 1;
            while (busqueda.Count > 0) {
                Tuple<int, int> actual = (Tuple<int,
int>)busqueda.Dequeue();
                cuadrosTotales++;
                sumaTotal += resultado [actual.Item1, actual.Item2];
                for (int i = 0; i < 4; i++) {
                    int y = actual.Item1 + movimientos [i, 0];
                    int x = actual.Item2 + movimientos [i, 1];
                    if (DentroMapa(y, x)) {
                        if (visitados [y, x] == 1 ||
Math.Abs(resultado [y, x] - resultado [actual.Item1, actual.Item2]) >
EPS_merge)
                            continue;
                        visitados [y, x] = 1;
                        busqueda.Enqueue(new Tuple<int, int>(y, x));
                    }
                }
            }
```

Bibliografía

[1] MSDN, Microsoft, 2017, Online: https://www.visualstudio.com/

[2] Python, Python, 2017, Online: https://www.python.org/download/releases/2.7/

[3] Github, Pydicom,2013, Online: https://code.google.com/archive/p/pydicom/downloads