UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Paralelización de una aplicación de contorneado automático de imágenes médicas 3D

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pedro Salguero García

Tutor: Víctor Manuel García Mollá

2016-2017

Resumen

Trabajar con grandes cantidades de datos de una manera inadecuada, el acceso a ellos y la manera de realizar los cálculos a la hora de programar puede afectar enormemente y de manera negativa al tiempo de ejecución de cualquier aplicación.

Cuando se trata con imágenes médicas para la segmentación automática debemos hacer frente a todos estos problemas, y para solucionarlos vamos a hacer uso tanto de la paralelización de la aplicación entre distintos núcleos de procesamiento, como de la reorganización de código y uso correcto de los datos para conseguir una mejora en el tiempo de cómputo del programa.

Por otro lado, se ha intentado abordar el problema del contorneado automático de órganos usando redes neuronales artificiales, prescindiendo de la anterior aplicación para buscar un mayor índice de acierto y proporcionar mejores resultados usando esta tecnología.

Palabras clave: paralelización, imagen médica, segmentación, redes neuronales.

Abstract

Working with large amounts of data in an inadequate form, the access to it and the way to perform calculations when programming can affect enormously and negatively to the time of execution of any application.

When it comes to automatic segmentation of medical images we must face all the problems mentioned and solve them, to do that we need to use both the parallelization of the application between different processing cores, and the reorganization of code and the correct usage of data to get an improvement in the computing time of the program.

On the other hand, we have tried to solve the problem about automatic segmentation of organs using artificial neural networks without the previous application to get a higher success rate and better results using this technology.

Key words: paralellization, medical image, segmentation, neuronal networks.

Agradecimientos

Este año ha sido muy duro para mí, desde el comienzo lo pasé muy mal y más adelante volvieron a ocurrir malas noticias, pero, aun así, también puedo decir que ha sido un año donde he aprendido a darle la importancia que se merece a muchas cosas que antes no valoraba como debía, muchas de ellas demasiado importantes para haberlas olvidado durante demasiado tiempo. En definitiva, este año me ha enseñado a ser mejor persona conmigo mismo y con los demás.

En primer lugar, me gustaría agradecer el apoyo que he recibido durante toda mi vida, a mis padres y a mi hermana. Me han ayudado en todo momento, siempre han estado ahí, aunque ellos mismos lo estuvieran pasando mal. Si no llega a ser por ellos, mi vida sería completamente diferente a lo que es ahora y me arrepentiría de que así fuera.

También me gustaría dar las gracias al resto de mi familia, nunca dudaron de mí y sabían que iba a seguir en mi línea.

Gracias también a dos amigos muy especiales que tengo desde hace ya seis años, Ana Reguilón y Jose Antonio Espinosa. Una de las pocas cosas que no cambiaría en la vida sería el haberos conocidos en aquel Campus Científico. Sois los que más lejos estáis y aun así sé que siempre voy a poder contar con vosotros para cualquier cosa, os considero amigos de verdad, esos que son para toda la vida pase lo que pase.

También quiero mostrar mi agradecimiento a mi profesor y a mis compañeros de tenis, aunque sea de manera indirecta y sin que lo supieran, jugar y aprender con ellos es una de las mejores cosas que he podido hacer durante este año. Ha sido mi desconexión de todo y he disfrutado cada momento que he pasado en la pista; no podía no nombrarles.

Respecto a la universidad, citar a mi tutor del TFG, Víctor Manuel García Mollá que me dio total libertad para realizar experimentaciones fuera de la temática principal, me ayudó cuando era necesario y nunca me puso ninguna pega. Por otro lado, también debo citar a mi grupo de amigos que me ha acompañado durante estos cuatro años, hemos pasado experiencias de todo tipo que nunca se olvidarán, gracias a todos los "koalas".

Por último, me gustaría agradecer a tres amigas muy especiales que me han aconsejado y ayudado siempre que lo he necesitado. A Yaiza Quiñones por su forma de ser, sus tonterías, sus borderías y hacer que los viajes a la universidad fueran mucho más amenos. A Belén Martínez por su apoyo, por hacerme desconectar visitando su Sevilla preciosa, por haberme acogido como lo hizo a pesar de conocernos desde hace relativamente poco en ese campus de inglés. Gracias porque sé que tengo una amiga más en quien confiar. Y finalmente a Ana Paes por miles de cosas. Has sido una de las personas que más me ha ayudado para superarlo todo, desde el principio, desde esa charla en tu portal no has hecho más que ayudarme y sigues haciéndolo. Me alegro muchísimo de tenerte como amiga, gracias por todo, aunque digas que no hace falta.

Tabla de contenidos

Listado de abreviaturas

Abreviatura	Significado
RMN	Resonancia Magnética Nuclear
TFG	Trabajo de Final de Grado
TAC	Tomografía Axial Computarizada
RNA	Redes Neuronales Artificiales
JPEG	*Joint Photographic Experts Group*
PNG	*Portable Network Graphics*
BMP	*Bits Maps Protocole*
DICOM	*Digital Imaging and Communication in Medicine*
MATLAB	*Matrix laboratory*
RAM	*Random Acces Memory*
CT	*Computed Tomography*
SSH	Secure SHell
API	*Application Programming Interface*

Listado de figuras

Listado de tablas

1. Introducción

El tratamiento de imágenes digitales ha sido caso de estudio desde su aparición. Si nos centramos en las imágenes de mapas de bits, se tratan de representaciones bidimensionales en forma de matriz donde cada celda representa con un valor numérico la intensidad de luz en ese punto de la imagen [1]. De esta manera, se puede decir que, a mayor número de celdas, la matriz podrá representar un mayor número de puntos de luz, y, por ende, de información visual. Estas celdas o puntos son denominados píxeles cuando los representamos en una pantalla.

Debemos aclarar que no solo por el hecho de tener más píxeles en la imagen, vamos a obtener mejores resultados, ya que esto significa poseer más información que debe procesarse para cada imagen y puede demorar el tiempo según el tratamiento que realicemos a la matriz.

Cuando nos trasladamos al sector médico y realizamos las pruebas necesarias tenemos otro problema. En este caso, nos interesa obtener la mayor información posible del paciente. Si realizamos una resonancia magnética nuclear (RMN) o una tomografía axial computarizada (TAC) queremos ver con el máximo detalle posible si el paciente sufre algún problema en la parte inspeccionada. Lo ideal sería capturar en una gran matriz todos los datos necesarios para poder observar los tejidos y realizar el tratamiento de la imagen para obtener los mejores resultados posibles por mucho tiempo de cálculo que lleve realizarlo. Pero la realidad es otra, la toma de imágenes digitales también viene restringida por la tecnología empleada para la captura de esos puntos de información.

En el caso de una máquina de RMN la toma de esa información es diferente si la comparamos a como se realiza una fotografía con nuestro teléfono inteligente o nuestra cámara de fotos. Los dos últimos realizan la captura basándose en la luz que captan con el objetivo de la cámara y de ahí realizan la conversión a la imagen de mapas de bits. En el caso de la RMN se utiliza un complejo sistema basado en la captación de ondas de radiofrecuencia procedentes de la estimulación de la materia, de los átomos del cuerpo humano, excitados mediante campos magnéticos [2]. Este es uno de los casos donde la tecnología nos limita y dependemos del número de sensores usados para la construcción de la matriz de información, y no solo en el número de celdas, sino también en la tonalidad obtenida, restringiendo la imagen a una escala de grises.

El funcionamiento y las dependencias de la RMN solo es un ejemplo, en este trabajo se realiza la aplicación con pacientes que han experimentado un TAC y se trabaja con este tipo de imágenes basadas en los rayos X.

Por otro lado, no solo nos interesamos la captación de una imagen y su transformación a una matriz, lo realmente interesante es el tratamiento de la imagen captada. Una vez disponemos de la matriz podemos realizar ciertas modificaciones o trabajar con ella para lograr nuestros objetivos. Al tratarse de una matriz numérica podemos realizar cualquier tipo de operación, y tras modificarla podemos representar la imagen y observar los

cambios sufridos [3]. Un ejemplo de ello, intentando seguir con el sector de las imágenes médicas, sería ampliar el contraste entre distintos elementos de la imagen para poder realizar una mejor distinción entre ellos como podemos observar en el siguiente gráfico (ver Figura 1).

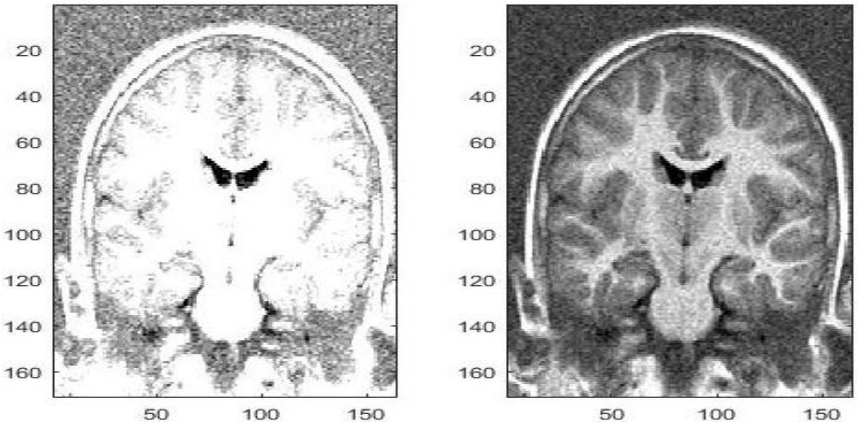

Figura 1 - Imagen de RMN antes y después de aplicar una mejora del contraste con un factor de 0.5

Esto podría realizarse aumentando la diferencia entre los diversos números de cada celda, es decir, multiplicando cada valor de cada celda por un factor. Como podemos observar en la figura superior, esta técnica nos permite apreciar con más detalle las variaciones captadas e intensificar su contraste.

El tratamiento de imágenes médicas no se va a realizar trabajando con la imagen que todos podemos visualizar, este tratamiento se realizará según la matriz donde se guarda la información de cada pixel de la imagen. Y no solo podemos usar estos valores para la modificación de la propia imagen como aumentar el contraste como hemos visto, sino que también podemos usarlos para otras funciones como realizar búsquedas de imágenes similares como se usan en este trabajo.

Por otro lado, el trabajo a realizar también se centrará en la tarea de la paralelización. La paralelización de una aplicación consiste en el aprovechamiento de los recursos disponibles de una máquina, específicamente los núcleos de procesamiento, para poder agilizar la ejecución del algoritmo [4]. Para ello se deben realizar las modificaciones oportunas para que el resultado de la aplicación no varíe, pero sí el tratamiento y distribución de los cálculos y de los datos en el propio código.

Al realizar los cambios oportunos se podrán ejecutar varias instrucciones al mismo tiempo en diferentes núcleos del procesador. Con ello, lo que se pretende es disminuir el tiempo de ejecución del programa comparado a ejecutarlo de forma secuencial, es decir, sin paralelización y realizando todas las operaciones de una en una en un solo núcleo del procesador.

Así pues, el trabajo se va a centrar en el refinamiento y desarrollo de una aplicación desarrollada en el lenguaje de programación MATLAB que sirva para la segmentación

automática de la prostata de un paciente (tratamiento de imagen médica) y paralelizarla para que se aproveche los procesadores actuales con multiples núcleos de procesamiento.

1.1 Justificación

He seleccionado este trabajo de final de grado (TFG) por dos motivos principales. El primero de ellos tiene que ver sobre la utilidad. Se va a trabajar con una aplicación de segmentación de órganos automática que en el caso de obtener buenos resultados puede llegar a usarse por los propios médicos para agilizar su trabajo eliminando una de las partes más tediosas como es la segmentación anteriormente mencionada.

La segmentación sobre imágenes médicas es una técnica que se utiliza para delimitar el contorno de un órgano, un segmento de tejido u otro cuerpo en específico o un conjunto de ellos y poder observar su estructura de una manera más clara. Esta tarea se realiza inspeccionando cada una de las imágenes producidas por una RMN, TAC u otra técnica de obtención de imágenes médicas y delimitando manualmente el contorno de cada objeto de estudio que se aprecie en la imagen, en la siguiente figura se puede observar la segmentación de un cerebro humano en ocho tejidos distintos (ver Figura 2).

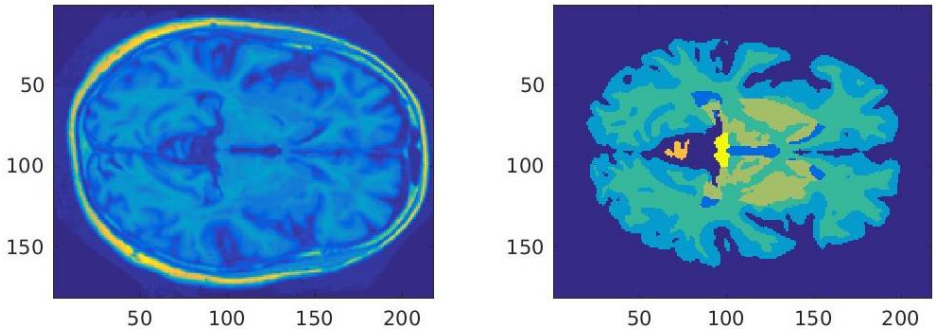

Figura 2 - Segmentación de una imagen cerebral en siete tejidos diferentes mediante redes neuronales artificiales

Esta tarea no puede realizarla cualquier persona sin experiencia porque se necesita saber interpretar de manera correcta las imágenes obtenidas e identificar los elementos a delimitar aun con la dificultad de vislumbrar en tonalidad de grises y a veces con baja calidad de imagen.

El segundo de ellos tiene que ver con el tema principal del TFG, la paralelización de la aplicación. Un mismo programa o aplicación puede desarrollarse de infinidad de maneras diferentes, y como es obvio, algunas mejores que otras. La paralelización de una aplicación te ayuda a encontrar una de esas maneras que puede ser mejor que las demás. Al realizar una paralelización de una aplicación no estamos perjudicando en ningún momento ni modificando los resultados que ofrece, es decir, si disponemos de una aplicación que para cierto paciente devuelve una segmentación de la próstata con un acierto del 60 %, la misma aplicación paralelizada devolverá el mismo porcentaje de acierto solo que realizará los cálculos en menor tiempo, siempre que se haya programado una paralelización de forma correcta.

1.2 Objetivos

El siguiente TFG poseerá los siguientes objetivos de carácter general:

- Aumentar la efectividad de la aplicación a la hora de la segmentación automática de próstatas.
- Disminuir el tiempo de ejecución de la aplicación sin afectar a los resultados.
- Paralelizar la aplicación para hacer uso de ordenadores con capacidad de multiprocesamiento.

De una manera más concreta se realizarán los siguientes objetivos de carácter específico:

- Estudio del código de la aplicación y comprensión del funcionamiento.
- Estudio de los datos de los pacientes y cómo funciona la carga de la información.
- Reajuste del código para poder incluir posibles mejoras u optimizaciones.
- Estudio de la aplicación para ver que partes del código pueden ser las más interesantes para la paralelización.
- Realizar la paralelización de las partes seleccionadas.
- Traspaso de parte del código de ejecución a otro lenguaje de programación para intentar aumentar la velocidad de procesamiento.
- Paralelizar la parte de código realizada en otro lenguaje si es posible para aumentar aún más la eficiencia.
- Contrastar y comentar los diferentes resultados obtenidos en las pruebas y experimentos llevados a cabo.

1.3 Estructura

El siguiente TFG se desarrollará de la siguiente manera: primero se tratará el estado del arte actual sobre el desarrollo de aplicaciones de segmentación automática y que técnicas se están utilizando para ello. Seguidamente se abordarán los fundamentos teóricos del trabajo y la metodología empleada en el mismo identificando los datos utilizados para la experimentación.

A ello continuará el desarrollo del trabajo mencionado y como trabajo extra la programación de una aplicación alternativa basada en redes neuronales artificiales para la segmentación automática de la próstata. Una vez tratadas las dos aplicaciones se pasará a la comparación de resultados, conclusiones finales y por último la bibliografía empleada.

2. Estado del arte

Actualmente (2017), existen dos grupos principales para la realización de la segmentación de imágenes médicas. El primero de ellos es la manera tradicional, técnicas asistidas por el hombre y el segundo técnicas basadas en la inteligencia artificial [5].

2.1 Técnicas asistidas por el hombre

En este grupo se encuentran las técnicas dependientes del usuario. Son aquellas que para lograr un resultado debe haber un usuario especializado que intervenga para la obtención de la segmentación deseada, ya sea de manera completa o parcial. Podemos encontrar algunas técnicas como las siguientes:

2.1.1 Perfilado a mano

El perfilado a mano o contorneado a mano alzada es una técnica básica usada por los especialistas. Es la técnica más simple y se basa en el uso del ratón como pincel para ir trazando el contorno del objeto a seleccionar [5]. También es un proceso lento y algo tedioso. Siempre se puede hacer de manera rápida pero no se obtendrá el mismo resultado que si se hiciera con detenimiento pixel por pixel.

Además, si se quiere poseer un número de casos de estudio para experimentar con otras técnicas, la segmentación resultante debe ser lo más perfecta posible para poder comparar los resultados obtenidos de manera automática o semiautomática y los generados por un experto con perfilado a mano.

2.1.2 Crecimiento de regiones

Este método es uno de los más populares, donde la selección del área de estudio es parte del trabajo del usuario y la parte de la delimitación se encarga un algoritmo [5].

La manera de funcionar es la siguiente: se establecen unos criterios para inclusión de vóxeles en la estructura. Los vóxeles son áreas de tejido humano representadas en la imagen médica, se pueden comparar con los píxeles de una imagen solo que estos son áreas de tejido en las tres dimensiones del espacio. El siguiente paso sería la selección de uno o más vóxeles semilla del interior de la estructura a seleccionar, por parte del humano, usando el ratón.

A continuación, esos vóxeles se añaden a una cola y se aplican los criterios establecidos previamente a los vóxeles cercanos (normalmente los 6, 18 o 26 más cercanos). Si cumplen los criterios se encolan para realizar este mismo paso y si no son descartados. Una vez no queda ningún elemento que procesar acaba el algoritmo y devuelve toda el área que cumple con el criterio según la semilla seleccionada.

2.1.3 Conectividad borrosa

Este es un algoritmo similar en concepto al de crecimiento de regiones. En este caso el humano vuelve a seleccionar un objeto y el proceso de segmentación es realizado por el algoritmo. En este caso se basa en la aceptación o no de vóxeles según una función de afinidad entre ellos en la que interviene la función distancia y la intensidad de la escena, aparte de otra función denominada función de fuerza o de conectividad [5].

Básicamente dos vóxeles pertenecen al mismo objeto cuando cumplen con una afinidad alta, y porque su fuerza de conectividad es mayor que un umbral establecido y además mayor a la fuerza de conectividad entre cualquier vóxel del interior de la región con otro vóxel no perteneciente.

2.2 Técnicas basadas en inteligencia artificial

Estás técnicas son totalmente independientes de la ayuda humana y pueden generar contornos o segmentaciones automáticamente dado un nuevo paciente. Se pueden diferenciar en técnicas basadas en el conocimiento y técnicas basadas en atlas. Las redes neuronales pertenecen al grupo basado en conocimiento a la hora de trabajar con imágenes médicas, pero son capaces de no disponer de ningún conocimiento previo y generar ellas mismas las segmentaciones que vean oportunas e ir aprendiendo con el tiempo.

2.2.1 Redes neuronales

Las redes neuronales artificiales (RNA) son un modelo computacional que intentan imitar al modo de trabajar de nuestro propio cerebro. Se basan en el comportamiento de la transmisión de información de las neuronas cerebrales usando sus axones. Las neuronas artificiales también están conectadas entre ellas y permiten incrementar o inhibir el estado de las neuronas adyacentes [6].

Las neuronas suelen estar constituidas en varias capas, las cuales forman la red completa. De las que podemos distinguir la capa de entrada y de salida, y luego tantas capas ocultas como hayan sido diseñadas para el sistema (ver Figura 3).

La capa de entrada es la que proporciona los datos de entrada de las variables que han sido seleccionadas para entrenar la red, esta es una elección complicada porque se debe intuir o saber mediante experimentación cuales son los parámetros adecuados a usar para lograr una red satisfactoria.

Por otro lado, la capa de salida puede funcionar de dos maneras diferentes según el tipo de red neuronal, una asistida donde el usuario le pasa la salida que debe adquirir dados unos ciertos parámetros de entrada y de esta manera la red empieza a entrenarse y modificar y redistribuir los pesos para que se consiga el error mínimo en la clasificación, o bien, existe un tipo de redes neuronales que pueden predecir cuál debería ser la salida para los datos de entrada establecidos.

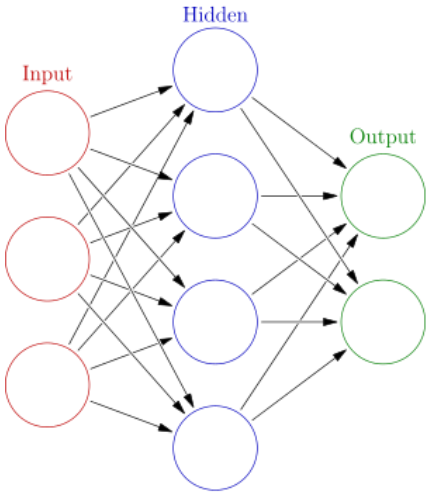

Figura 3 - Diagrama base de la estructura de una red neuronal donde se muestra la capa de entrada, la capa oculta y la capa de salida

Las RNA son bastante apropiadas para problemas en los que no se dispone de un modelo apropiado inicial, pero se dispone de un conjunto de ejemplos ya estén o no clasificados previamente. Entre sus características además disponen de un alto grado de robustez frente al ruido y facilitan su uso en problemas relacionados con clasificación, reconocimiento de patrones, imágenes, señales…

Dentro de estos campos entra el terreno de la informática médica y la clasificación de tejidos y su segmentación entre otros campos como puede ser el reconocimiento de tumores u otras patologías de manera automática.

3. Fundamentos teóricos

Para ser capaces de entender el funcionamiento de la aplicación desarrollada es importante familiarizarnos con unos conocimientos básicos sobre las imágenes médicas.

Igual que nos son habituales los formatos de imágenes *JPEG* (*Joint Photographic Experts Group*), PNG (*Portable Network Graphics*) o *BMP* (*Bits Maps Protocole*) las imágenes médicas poseen su propio formato por el siguiente motivo. Una imagen de un TAC o de una RMN no se puede comparar con una fotografía que sacamos con nuestra cámara de fotografía. En ellas no solo se encuentra el conjunto de imágenes generadas sino también información sobre el paciente, el doctor, la máquina usada, características o variables que se han usado en la toma de la imagen, cuantos cortes se han realizado y de que dimensiones y un largo etcétera.

Toda esta información se guarda en un formato de imagen denominado DICOM (*Digital Imaging and Communication in Medicine*). Se trata de un estándar reconocido mundialmente para el manejo de pruebas médicas [7].

Para el desarrollo del TFG se ha usado una colección de pacientes debidamente anonimizados tanto ellos como los doctores que se han encargado de las pruebas y toda la información personal que haya podido guardarse en los ficheros.

Por otro lado, la aplicación está desarrollada con el entorno de programación MATLAB (*matrix laboratory*) desarrollado por *MathWorks* y usa su propio lenguaje de programación [8]. A parte de ello, vamos a hacer uso de un *toolbox* que viene incorporado denominado *Parallel Computing Toolbox* que nos permitirá realizar la paralelización del código de una manera mucho más cómoda [9].

También se usará el lenguaje de programación C para el desarrollo de una pequeña parte del código dada su eficiencia para cálculos simples dentro de varios bucles.

4. Material

Para nuestro trabajo y experimentación contamos con un conjunto de 156 pacientes diferentes donde poseemos un total de 71 imágenes transversales del cuerpo y con una resolución de 200x300 px (tras realizar una normalización que se explicará en el desarrollo del proyecto), separadas entre ellas de 3 a 5 milímetros de grosor según el paciente y el modelo de máquina usada. Además, poseemos las segmentaciones de diversos tejidos, órganos o huesos que se pueden observar en el TAC. De esta manera contamos con las segmentaciones o contornos realizados por un especialista que nos serán de utilidad para poder hacer las comparaciones necesarias y comprobar la calidad y eficacia de nuestra aplicación.

Por otro lado, contamos con el núcleo de la aplicación a desarrollar que podréis encontrar en la sección de anexos. Se trata de un algoritmo de comparación de segmentos de imágenes. Dado un paciente y una imagen entre las 71 disponibles, seleccionamos un segmento y nos busca el segmento más parecido al seleccionado de entre los otros 155 pacientes.

Y sobre el material informático poseemos un usuario del supercomputador *kempes* que cuenta con el sistema operativo Ubuntu 14.04.3 LTS x86_64, posee dos procesadores de la gama Intel Xeon E5-2697 con 12 núcleos cada uno a una velocidad de 2,70 GHz y dos tarjetas gráficas Tesla K20Xm (Kepler) con 192 núcleos y 6GB de memoria global dedicada. Por último, posee un total de 128GB de memoria RAM (*random access memory*).

5. Desarrollo

En el siguiente apartado se explicará de manera detallada el procedimiento realizado y las modificaciones aportadas al programa para, tanto realizar la paralelización de la aplicación como para mejorar el índice de acierto. También se desarrollará la aplicación sobre redes neuronales y que pruebas se llegó a realizar con ella.

5.1 Aplicación para la segmentación de próstata

En los siguientes apartados nos centraremos en la aplicación principal del TFG, tanto en el estudio, como en la implementación de mejoras, optimización de código y la paralelización de la misma.

5.1.1 Experimentación con ficheros DICOM

Como ya hemos comentado, el formato estándar para el intercambio de imágenes médicas es el formato *DICOM* con la extensión "*.dcm"*. En un primer momento se disponía de un total de 125 pacientes en este formato. Cada paciente estaba dispuesto en una carpeta con un número arbitrario de archivos, entre 71 y 159 ficheros. Entre todos ellos se encuentra un gran grupo con el prefijo CT (*Computed Tomography*) que indican que las imágenes del paciente han sido realizadas por una tomografía computarizada.

Por otro lado, se disponía también de otros ficheros con los prefijos RD, RP y RS, los cuales, hacen referencias a distintos datos del paciente, como son la distribución de la dosis, el fichero con el plan de actuación y las estructuras segmentadas por los especialistas para cada tejido, órgano u objeto de estudio.

En nuestro caso, los únicos ficheros de importancia son las imágenes y las estructuras de los tejidos, pero para poder trabajar de una manera adecuada debemos disponer de la misma estructura para todos los ficheros, es decir, que todos contengan exactamente el mismo número de archivos o tomografías realizadas para poder realizar comparaciones entre ellos.

Por eso mismo se llegó al acuerdo de establecer un total de 71 imágenes para cada paciente. Se decidió así porque en todas las tomografías realizadas se trataba de dejar la próstata, el órgano de estudio, en el corte central. Así que se extrajeron las 35 secciones anteriores y posteriores a cada corte central para que todos los pacientes poseyeran la misma cantidad de cortes y más adelante poder compararlos entre sí.

5.1.2 Núcleo del programa

En un primer momento, se desarrolló un script que te permitía seleccionar un pequeño fragmento de una imagen leída de los ficheros DICOM y leyendo uno por uno los demás te devolvía el segmento más similar encontrado en otro paciente tras acabar la búsqueda.

La idea del programa era dividir la imagen de un paciente dado en un número determinado de bloques y realizar una reconstrucción de la imagen con segmentos de pacientes de los que sí conociéramos la ubicación de la próstata, ya que poseíamos la información en los datos de las estructuras de los ficheros DICOM.

Pero al realizar pequeñas pruebas nos percatamos de dos problemas que podían influir tanto en el tiempo como en la calidad del resultado. El primero de ellos era la toma de los TAC en los distintos pacientes, aparte de tener distinto número de cortes tomados y que ya solucionamos, también observamos que la separación entre ellos podía variar de paciente a paciente, y el segundo problema era la lectura desde MATLAB de los ficheros DICOM, era un proceso lento, así que decidimos convertir los ficheros a otro tipo de formato con el que se pudiera trabajar de manera más óptima.

5.1.3 Interpolación y conversión de DICOM a matrices

Los datos de los pacientes estaban divididos en dos tipos diferentes. Por un lado, los TAC que poseen una separación entre cada corte de 3 mm y por otro los que poseen una de 5 mm.

Para solucionar el primer problema se realizó una interpolación, que es el proceso de trasformación de los datos para emular su obtención con otro tipo de separación entre cortes en este caso [10], sobre las imágenes de 5 mm para convertirlas o emular cortes cada 3 mm. En un principio, esto no debería afectar en mayor o menor medida a la manera de trabajar de esta aplicación, pero se realizó la interpolación pensando en el futuro y en el desarrollo de la nueva aplicación basada en redes neuronales donde si podía afectar que la entrada de los datos fuera distinta para cada paciente y provocar resultados equívocos. De esta manera podríamos trabajar con una base de datos mayor que si solo eligiéramos los pacientes de 3 mm para el uso de redes neuronales.

Y para agilizar el tratamiento de la información se realizó la transformación de las imágenes del formato "*.dcm*" a matrices con las que poder trabajar en MATLAB. Estas matrices iban a poseer tres dimensiones, las primeras dos representan la imagen de cada corte, y la tercera dimensión el conjunto de los 71 cortes generados por el TAC (ver Figura 4). De esta manera por cada paciente obtuvimos dos ficheros, uno contenía la imagen del TAC y otro poseía la región donde se encontraba la próstata de ese paciente.

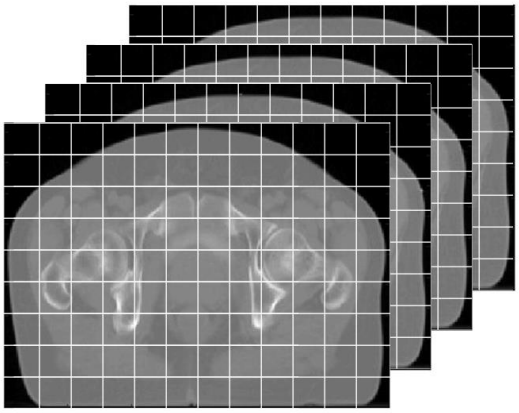

Figura 4 - Representación de un TAC en tres dimensiones

Aun así, para cargar todos los pacientes en la aplicación debíamos leerlos de uno en uno, y el tiempo de acceso a cada carpeta, seguido de la lectura del fichero y moverte a otro directorio para leer el siguiente no era eficiente. Por esa razón se realizó un *script* que recopilaba todos los datos anteriores en una matriz de una dimensión mayor. Esta nueva dimensión, la cuarta, se utiliza para la identificación del paciente. Y se realizó tanto para las imágenes del TAC como para el contorno de la próstata de cada uno. Se puede observar una representación de la cuarta dimensión la siguiente figura (ver Figura 5):

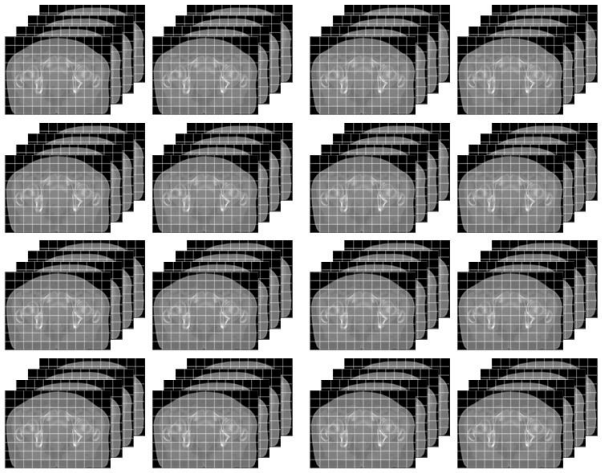

Figura 5 - Representación de los pacientes en una matriz de cuatro dimensiones

Para hacer una representación más clara, vamos a suponer que el nombre de la matriz de cuatro dimensiones es “array_imagen”, si deseáramos cargar la imagen del corte 35 del paciente número 10 y visualizarla, como podemos observar en la en la imagen inferior, se realizaría la siguiente subrutina (ver Figura 6):

```
imagen_objetivo=array_imagen(:,:,35,10);
colormap(gray);
imagesc(imagen_objetivo);
```

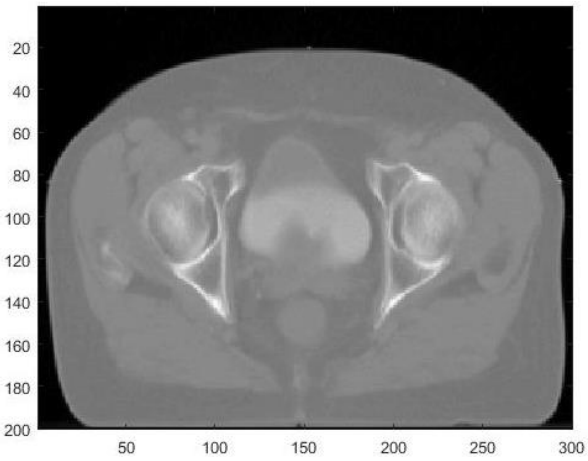

Figura 6 - Imagen TAC del paciente nº 10, corte nº 35

En este caso seleccionamos todos los pixeles que se encuentran en todas las filas y todas las columnas con el identificativo ":" que representan la imagen y seleccionamos en la tercena dimensión el corte 35 y en la cuarta el paciente número 10. Los otros comandos nos permiten interpretar la imagen en escala de grises y poder visualizarla correctamente.

5.1.4 Primera versión del programa

Tras el procesamiento de los datos y conversión de los pacientes a una matriz común se desarrolló una primera versión del programa. En ella, se realizaba la reconstrucción de un TAC a partir de segmentos de máxima similitud encontrados en la base de datos. Se puede ver el código completo en el Anexo 1.

Es decir, dado un paciente en el que deseamos segmentar su próstata, realizamos una división del TAC en un número de bloques determinados en tres dimensiones con un tamaño específico, se usaron bloques de 40x30x35 px. Podemos observar una representación de estos bloques en la siguiente figura (ver Figura 7).

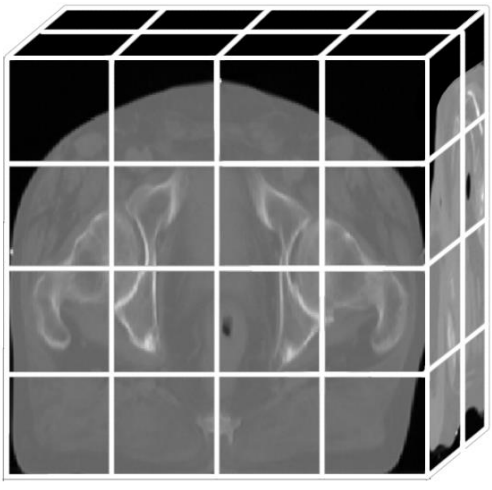

Figura 7 - Representación de una imagen de TAC dividida en bloques para su tratamiento

Seguidamente creamos una nueva variable del mismo tamaño que el TAC del paciente en las tres dimensiones, en la cual se irán colocando en la posición adecuada los segmentos más parecidos buscados en los demás pacientes de la base de datos.

Para realizar esta búsqueda se han determinado unos pequeños márgenes por los que buscar las imágenes similares, significa que para la búsqueda de un segmento de imagen, el paciente de estudio no va a buscar en todas las posiciones disponibles de todos los pacientes de la base de datos, sino que se centrará en buscar en la misma zona con un pequeño margen de variabilidad, tanto en el eje vertical como el horizontal, porque sabemos que es alrededor de esa zona donde se encontrará el segmento con más parecido ya que es donde se ha seleccionado en la imagen objetivo.

Por cada paciente de la base de datos que cargamos realizamos la búsqueda en su zona correspondiente (incluyendo los márgenes de búsqueda mencionados) y vamos calculando el valor de la diferencia entre la zona del paciente de estudio, y la zona por la que estamos buscando. Como hemos dicho anteriormente, estamos representando imágenes como valores numéricos y podemos realizar operaciones con ellas. Esta diferencia será el error y se va guardando siempre que vaya mejorando respecto a búsquedas anteriores. Por lo que al final tendremos el valor del error mínimo junto con el paciente y la zona exacta donde se ha encontrado ese parecido. Después de eso lo único que nos queda es añadir ese segmento a la variable creada para ir guardando las imágenes encontradas y pasar con el siguiente bloque para obtener el próximo segmento de imagen más similar.

Podemos representar de una manera más gráfica el proceso con la siguiente imagen (ver Figura 8). En un primer momento se selecciona un bloque del TAC del paciente original como en la Figura 7. Esa selección se realiza también en una variable tridimensional vacía donde se va a ir reconstruyendo con los segmentos más parecidos respecto el bloque original, viene representado por la primera imagen de la figura 8, el bloque seleccionado para la reconstrucción se ha marcado en color gris, seguidamente se van recorriendo secciones de todos los pacientes disponibles en la base de datos almacenando únicamente el que mejore el error, es decir, aquellos segmentos que se parezcan más que el actualmente seleccionado, se representa con la imagen central de la figura 8 donde se simula el recorrido de búsqueda en la matriz de cuatro dimensiones y por último, tras analizar todos los pacientes, el mejor segmento se recoloca en el lugar adecuado para generar el TAC artificial, última imagen de la figura.

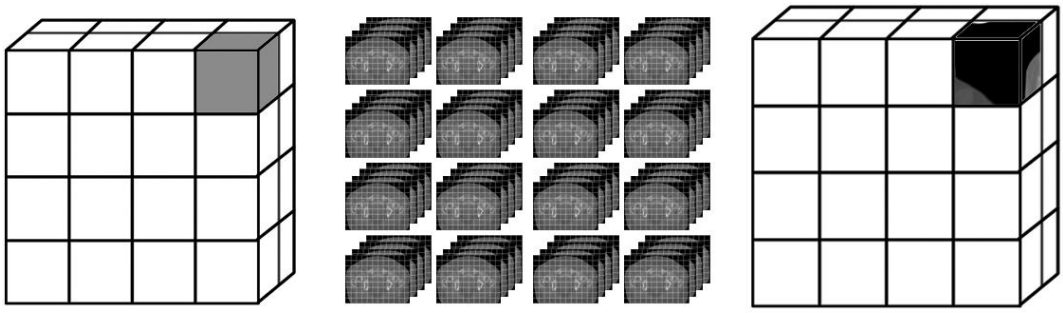

Figura 8- Representación de la búsqueda del mejor segmento para la reconstrucción de la nueva imagen

Tras acabar el proceso nos quedará una imagen como la siguiente que representa la reconstrucción de un corte del TAC de un paciente según las zonas más parecidas comparadas con la base de datos (ver Figura 9).

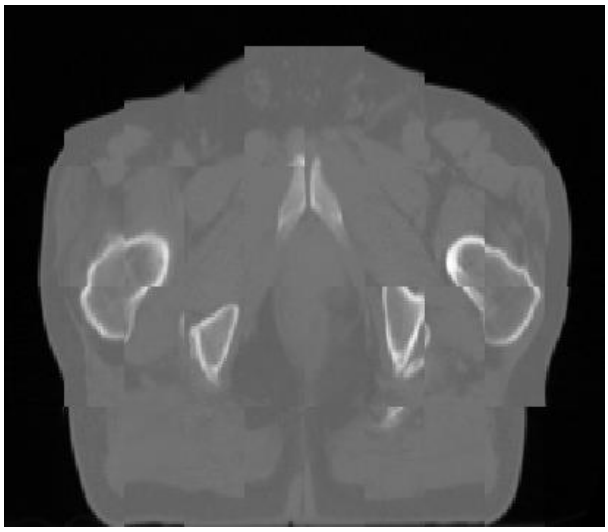

Figura 9 - Reconstrucción de la sección 35 del TAC del paciente 10 basándose en los demás pacientes

Todo este proceso nos sirve para la reconstrucción de la imagen, pero en realidad, lo que nos interesa es la reconstrucción o la segmentación de la próstata del paciente de estudio. Pero como tenemos la reconstrucción del paciente con imágenes de nuestra base de datos, la cuales tienen su estructura de la próstata segmentada por un especialista, lo que podemos hacer a continuación es realizar el mismo procedimiento, pero en vez de rellenando la variable con los segmentos de imagen del paciente más similar, rellenarla con los segmentos de la próstata del paciente con mayor similitud.

De esta manera podemos reconstruir la próstata de un paciente desconocido basándonos en el conocimiento que poseemos de otros pacientes. Podemos ver la reconstrucción de la próstata de la figura anterior y en el mismo corte del TAC a continuación (ver Figura 10):

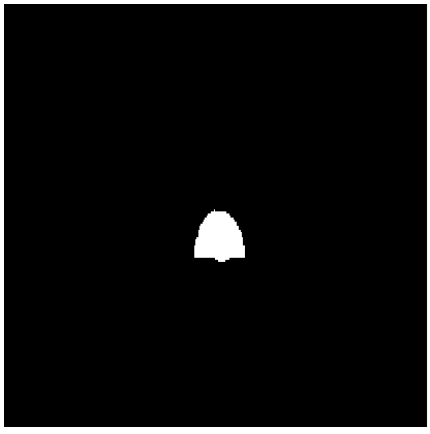

Figura 10 - Ejemplo de próstata reconstruida dada la figura anterior (Figura 9)

5.1.5 Estudio del programa y revisión del código

Podemos decir que la estructura principal del programa se basa en siete bucles que recorren tanto la imagen a reconstruir como los pacientes. Vamos a explicar cada uno de ellos sin entrar en mucho detalle en las operaciones que se realizan en cada apartado.

Antes de encontrar los primeros tres bucles, lo primero que realiza el programa es cargar tanto la base de datos de los pacientes como la de las próstatas, que ya comentamos que se encuentran en matrices de cuatro dimensiones en dos ficheros separados. Y por otro lado también se le pasa como parámetro la imagen del paciente del que deseamos obtener su próstata segmentada que será la imagen objetivo a conseguir.

Seguidamente encontramos los tres primeros bucles que hacen referencia a la reconstrucción de la nueva imagen basada en los datos de los pacientes que tenemos en la base de datos. Como estamos trabajando en tres dimensiones, para recorrer todos los bloques necesitamos tres bucles. El primero de ellos hace el recorrido en el eje horizontal o de las abscisas representado por la variable **ix**, por otro lado, tenemos el eje vertical o el de las ordenadas, representado por la variable **jy** y por último el eje de profundidad que se usa para seleccionar cada corte producido por el TAC con la variable **kz**.

```
13 for ix=1:indblx:200-indblx+1
14    for jy=1:indbly:300-indbly+1
15        for kz=1:indblz:70-indblz+1
```

Las variables **indblx**, **indbly** y **indblz** hacen referencia a que tamaños se van a usar para crear los bloques de la imagen. Por defecto se usaron bloques de 40x30x35 px como se comentó anteriormente.

Las acciones siguientes son determinar los valores márgenes para realizar la búsqueda de los segmentos de imagen más parecidos. Además de extraer de la imagen objetivo el segmento apropiado en la iteración del bucle actual que es guardada en una variable para poder compararla con los segmentos que se extraerán de los pacientes de la base de datos.

El siguiente bucle hace referencia al recorrido por toda la base de datos, recorremos todos los pacientes disponibles para buscar en cada uno de ellos el segmento del tamaño anterior más parecido al primer segmento de la imagen objetivo.

```
33 for paciente=1:muestras-10
```

En este bucle se realiza un recorrido entre toda la base de datos menos los diez últimos pacientes que se tomaron como pacientes de test para realizar los experimentos y comprobaciones.

Por último, dentro de este bucle de paciente recorremos búsquedas de bloques con un margen establecido para buscar el segmento de imagen más parecida a cada segmento objetivo de ese momento como se ha explicado.

```
35 for i=i_ini : i_fin
36     for j=j_ini : j_fin
37         for k=z_ini : z_fin
```

Los valores de las variables ya se establecieron anteriormente, pero su funcionamiento es igual que el primer bucle explicado al inicio solo que en vez de realizar saltos según el tamaño de bloque, aquí se hace un recorrido de pixel a pixel porque puede marcar la diferencia entre encontrar el segmento más similar o no.

Tras estos bucles se realiza la selección de cada posible imagen con máximo parecido al segmento de imagen objetivo y el cálculo de su error. Si el error es mejorado en alguna iteración seleccionamos el paciente actual como el mejor candidato y guardamos en qué posición de los márgenes seleccionamos hemos encontrado el mejor parecido.

Al finalizar todos los bucles obtenemos dos ficheros, la imagen reconstruida con partes de otros pacientes y la próstata reconstruida en base a estos segmentos.

5.1.6 Métodos de validación

Con el programa anterior obteníamos la próstata reconstruida según la base de datos y los parecidos encontrados, pero no teníamos ninguna manera de comprobar el índice de efectividad del programa.

Así que se desarrollaron varios *scripts* que nos permitían realizar esta medición y dotar al programa de unos valores que indicarían el tanto por ciento de acierto. Recordamos que, de todos los pacientes, incluidos los seleccionados para las pruebas (pacientes test), tenemos la próstata segmentada por un especialista y podemos basarnos en esta estructura para poder hacer la comparación corte por corte.

Así pues, se realizaron tres métodos diferentes pero los dos primeros se descartaron por no realizar una aproximación real para la comparación de las próstatas, la próstata ideal realizada por el especialista y la próstata reconstruida por la aplicación.

Los dos primeros métodos se basaban en la diferencia de imágenes de igual manera que calculamos el error entre un paciente y el paciente objetivo pero el problema principal era la estructura de la matriz de las próstatas.

Como sabemos, la matriz se compone de cuatro dimensiones donde la cuarta nos sirve para identificar el paciente y las otras tres seleccionar el pixel del espacio entre las diferentes imágenes de cada corte.

En ella, se encuentran ceros si no hay próstata identificada en ese píxel o bien un uno si el especialista ha indicado que en esa zona hay parte de la próstata. Al realizar un

método basado en la diferencia, como en todas las matrices había un número mayor de ceros que de unos, como es obvio y como se puede observar, si cogemos diferentes imágenes de próstatas en la figura inferior, ya se reconstruyera de una manera adecuada o no la próstata objetivo, a la hora de comparar los valores con un cero, hacía que el índice de acierto subiera hasta alcanzar valores del 99% aunque no se hallara ningún parecido en la próstata (ver Figura 11).

Figura 11 - Representación y comparación de las próstatas. La primera imagen representa la próstata reconstruida por un especialista, la segunda es la próstata del mismo corte y paciente reconstruida por el programa y la tercera imagen representa las diferencias

Entonces se decidió realizar un cálculo diferente, en este caso no realizamos una comparación de píxel por píxel, sino que identificamos la posición de los unos (valores donde se encuentra parte de la próstata) en la próstata ideal y los comparamos con la posición de los unos en la próstata reconstruida. De esta manera al comparar estas posiciones sí que nos centramos en el material relevante y podemos realizar el cálculo de acierto según número de posiciones coincidentes dividido entre el número de posiciones totales que se encuentran en la imagen del especialista.

$$acierto = \frac{x \cap y}{y}$$

Donde x representa los índices de la matriz de la próstata tridimensional generada automáticamente donde se encuentran los valores con un uno, es decir, en ese pixel hay parte de la próstata, e y representa los mismos índices, pero de la próstata seleccionada por el especialista. De esta manera al realizar la intersección entre los índices de la próstata generada y la del especialista podemos obtener la fracción de pixeles acertados en total. Sin realizar ninguna penalización por los píxeles generados de más que no forman parte de la próstata objetivo. Se puede encontrar el código en el Anexo 2.

5.1.7 Paralelización del bucle "Paciente"

Una vez ya podemos identificar la eficacia del programa base podemos realizar las modificaciones oportunas para mejorar si es posible tanto el porcentaje de acierto como la velocidad de ejecución de la aplicación a la hora de reconstruir la próstata de un paciente. De nuevo, el código completo puede observarse, en este caso, en el Anexo 3.

La primera idea que se realizó fue la paralelización del bucle de pacientes. Se busca el procesar varios pacientes a la vez para que el cálculo del error sea más rápido y se pueda obtener el segmento de imagen más parecido de una manera mucho más ágil.

Para realizar este procedimiento se ha hecho uso del *toolbox* de MATLAB, en concreto el uso del bucle paralelizado ***parfor***. Para poder usar este tipo de bucle se debe rehacer parte del código y trato de la información para que sea consistente.

Debemos tener en cuenta que al realizar el bucle en paralelo se realizará el cálculo de error mínimo en varios pacientes a la vez, si usamos el supercomputador *kempes* se procesarán 12 pacientes al mismo tiempo. Esto implica que debemos guardar el valor de error mínimo de cada paciente para poder elegir el mínimo de entre todos ellos. Por eso se han creado tres nuevas variables que representan vectores o matrices en la que se guarda: el valor de error de cada paciente (valmin), la imagen con el menor error (im_minima) y la imagen de la próstata que le corresponde a esa imagen elegida (prostata_minima) donde todas ellas se guardan según el paciente procesado en ese momento.

```
46 valmin(paciente)=err;
47 im_minima(:,:,:,paciente)=seccion;
…
52 prostata_minima(:,:,:,paciente)=seccion_prostata;
```

De esta manera realizando una búsqueda del mínimo error en la primera variable nos devuelve el índice de la posición que debemos buscar en las otras dos variables para elegir la imagen que le corresponde a ese bloque de la imagen original.

5.1.8 Reestructuración de los bucles iniciales

Como ya se ha dicho, los tres primeros bucles recorren los bloques de la imagen y de la próstata a reconstruir. Los bucles estaban diseñados de tal manera que los incrementos no eran de unidad en unidad, sino que dependían del tamaño de bloque.

Una manera para aclarar el código y que posiblemente ayudara al tiempo de ejecución fue desarrollar los bucles de otra manera, calculando las pasadas que se realizan en ellos antes de ejecutarlos para usar variables que vayan de unidad en unidad y no con incrementos arbitrarios. Con lo consiguiente encontramos una interpretación más clara y simple del código, aunque se realice en más de una línea. A continuación, se pueden observar las dos partes de código, a la izquierda el código original y a la derecha la modificación.

```
13 for ix=1:indblx:200-indblx+1
14 for jy=1:indbly:300-indbly+1
15 for kz=1:indblz:70-indblz+1
```

```
15 for i_ori=1:200/indblx
16 ix=1+(i_ori-1)*indblx;
17 ixfin=ix+indblx-1;

18 for k_ori=1:70/indblz
19 kz=1+(k_ori-1)*indblz;
```

```
17 ixfin=ix+indblx-1;
18 jyfin=jy+indbly-1;
19 kzfin=kz+indblz-1;
```

```
20 kzfin=kz+indblz-1;

21 for j_ori=1:300/indbly
22 jy=1+(j_ori-1)*indbly;
23 jyfin=jy+indbly-1;
```

Para que la programación se mantuviera se ha tenido que realizar unos pequeños ajustes en las variables y precálculos para saber cuántas iteraciones se realizaban según el tamaño de los bloques por defecto. El código completo está disponible en el Anexo 4.

5.1.9 Paralelización del bucle inicial del eje Y

Tras haber paralelizado el bucle de pacientes y reorganizar los bucles principales del programa se me ocurrió paralelizar esta vez un bucle más externo. Tenía tres opciones, el bucle del eje horizontal o x, el vertical o el de profundidad o cortes, el eje z. Me decidí por realizar una ordenación diferente para los bucles y dejar el bucle y el más interno de los tres y paralelizar este porque era el que más iteraciones realizaba de ellos.

De nuevo usé el *toolbox* de MATLAB y el *for* paralelo solo que esta vez el tratado de los datos era más complicado al poseer mucha más información internamente. De nuevo se modificaron varias variables necesarias para mantener la consistencia del código y que no afectara a los resultados.

```
13 for ix=1:indblx:200-indblx+1
14 for jy=1:indbly:300-indbly+1
15 for kz=1:indblz:70-indblz+1

17 ixfin=ix+indblx-1;
18 jyfin=jy+indbly-1;
19 kzfin=kz+indblz-1;
```

```
16 for i_ori=1:200/indblx
17 ix=1+(i_ori-1)*indblx;
18 ixfin=ix+indblx-1;

19 for k_ori=1:70/indblz
20 kz=1+(k_ori-1)*indblz;
21 kzfin=kz+indblz-1;

22 parfor j_ori=1:300/indbly
23 jy=1+(j_ori-1)*indbly;
24 jyfin=jy+indbly-1;
```

En este caso la paralelización lo que realizaba era la reconstrucción en paralelo de la imagen y de la próstata resultante. Se calculaban a la vez diversos bloques de la imagen en cada núcleo del procesador por lo que cada uno lee todos los pacientes y calcula el error mínimo en cada caso.

Esto conlleva a un menor intercambio de información entre procesos, en la paralelización del bucle de búsqueda de paciente mínimo cada núcleo escribía en la misma variable la información. En este caso solo se escribe la información en la imagen resultado y no tiene que tratarse nuevamente porque cada dato escrito es el correcto para ese bloque. Por lo que proporciona un mejor tiempo de ejecución que el método de paralelización

anterior, sin embargo, se apreciarán mejor los resultados en el apartado correspondiente. El código completo se encuentra en el Anexo 5.

5.1.10 Desarrollo del cálculo del error en C

Otra manera de mejorar tiempos es realizar cálculos pesados en el lenguaje de programación C. MATLAB no está pensado para realizar muchos bucles por lo que pierde eficiencia en ellos, pero se puede traspasar parte de código al lenguaje C, es decir, bucles que den problemas, y seguir con el programa principal desarrollado en MATLAB.

Estos ficheros tienen la extensión ".mex" y están programados en C pero siguiendo una arquitectura y una librería de MATLAB para poder hacer el emparejamiento [11].

El primer bucle que se intentó solucionar fue el del cálculo del error de la imagen. Si nos fijamos, el cálculo del error se compone de un bucle triple para sumar el error en las tres dimensiones de la matriz de la imagen seleccionada.

```
43 err=sum(sum(sum(abs(im_target-seccion)))); %error
```

Para habituarnos al intercambio de información entre MATLAB y C decidimos empezar por un solo cálculo, aunque ya estuviéramos trabajando con matrices de tres dimensiones en C. Los principales problemas de usar C en estos casos es la interpretación que se dan sobre las matrices, en MATLAB los *arrays* se almacenan por columnas, pero a la hora de pasar la información a un fichero *mex*, este tratamiento específico de MATLAB no se puede usar y debes tenerlo en cuenta. Por otro lado, en C, no se tienen *arrays* multidimensionales de una manera clara, se tiene un *array* de una dimensión que representa las tres, por lo que tienes que tener cuidado a la hora de llamar los datos adecuados en cada bucle para el cálculo correspondiente.

La manera de hacer referencia a un dato en un array de dos dimensiones en un fichero *mex* desarrollado en C es con la siguiente expresión:

`a[i][j]` se obtendría como a: `[i +j*dimx]`

Análogamente si trabajamos con un fichero de tres dimensiones deberíamos poseer un bucle para cada dimensión y suponiendo que las variables que recorre cada uno se denominan *i, j, k* la manera de acceder al valor del *array[i][j][k] es:*

`array[i][j][k]` sería equivalente a: `array[i+j*dimy+k*dimy*dimx]`

Donde la variable *dimy* y *dimz* son los valores máximos, es decir, el tamaño de las dimensiones *y* y *z* que hacen referencia a las variables de los bucles *j* y *k*. El código completo se encuentra en el Anexo 6.

5.1.11 Implementación del código C en MATLAB

Una vez tenemos el código en el lenguaje C desarrollado con la función correspondiente para vincular los datos de MATLAB al programa definidos correctamente se debe compilar como un archivo "*.mex*".

Para ello debemos usar el comando *mex* + *nombre_del_fichero.c* desde la consola de MATLAB. Tenemos que tener en cuenta que si trabajamos en una máquina con Linux debemos compilarlo para ese tipo de sistema operativo y sino para Windows. La extensión del archivo resultante variará, en nuestro caso como usamos el supercomputador *kempes* basado en una distribución de Linux nuestro fichero resultante fue: *calculo_error.mexa64*.

Y para llamarlo dentro del código de MATLAB de nuestro programa debemos cambiar el antiguo cálculo del error por la llamada a la función de C desarrollada.

```
err=calculo_error(seccion,im_target);
```

De esta manera a la hora de realizar el cálculo se llamará a la función diseñada y su cálculo se añadirá a la variable *err* como indica la igualdad.

5.1.12 Desarrollo de los bucles de márgenes y error en C

Después de desarrollar una primera versión del programa en C que nos calculaba el error de cada imagen, desarrollamos una versión algo más compleja que además de calcular el error incluía el cálculo de todas las posibles secciones posibles de la imagen para cada paciente. Esto mejora el código en puntos importantes, eliminamos tres bucles de MATLAB como son la holgura de los márgenes y por otro lado se llama menos a esta función reduciendo el tiempo de comunicación y ejecución de llamadas.

En este caso debemos pasar más parámetros a la función desarrollada en C, concretamente la imagen objetivo (*im_target*), el TAC completo de cada paciente de donde sacaremos las posibles secciones (*seccion*), y luego todos los valores iniciales y finales realizar las secciones oportunas, al igual que el tamaño de cada una (*i_ini, i_fin, j_ini, j_fin, k_ini, k_fin, indblx, indbly, indblz*).

```
[err,im,jm,km]=margen_paciente(Vqimd,im_target,[i_ini,i_fin,j_ini,j_fin,
z_ini,z_fin],[indblx,indbly,indblz]);
```

Como resultado devuelve tres variables, el error de la mejor sección encontrada y sus coordenadas para localizarla.

De igual manera se han realizado dos pruebas con este nuevo programa, una versión secuencial y otra en paralelo para observar los nuevos resultados. Como siempre podéis encontrar el código del fichero en C en el Anexo 7.

5.1.13 Implementación del nuevo código C en MATLAB

Para realizar la llamada correcta con esta nueva función, el bucle paciente debe quedar de la siguiente forma:

```
for paciente=1:muestras-10
    Vqimd=double(array_imagen(:,:,:,paciente));

[err,im,jm,km]=margen_paciente(Vqimd,im_target,[i_ini,i_fin,j_ini,j_fin,
z_ini,z_fin],[indblx,indbly,indblz]);

    if (err<valmin)
        valmin=err;
        pac_min=paciente;
        i_mejor=im;
        j_mejor=jm;
        k_mejor=km;
    end
end %bucle pacientes
```

Aclarar que la comparación que se realiza con el *if* es necesario porque la función devuelve la mejor sección de un único paciente, pero debemos elegir entre todos los pacientes de la base de datos cual es el mejor. Esa búsqueda se realiza con el *if* que se muestra en el código de arriba.

5.2 Redes neuronales artificiales para segmentación

En una asignatura optativa de la carrera vi el término que hacía referencia a las redes neuronales artificiales y sus diferentes usos en los problemas informáticos actuales.

En uno de ellos vi el potencial que podían alcanzar para problemas de segmentación y a pesar de no tener mucha idea y de apenas trabajado con ellas intenté desarrollar una aplicación que segmentara la próstata basándome en el código de la segmentación de un cerebro visto en la asignatura optativa del grado, informática médica, impartida por José Vicente Manjón Herrera. Por lo que la mayoría del código fue desarrollado por él y me limité a hacer pequeñas modificaciones que vi oportunas.

5.2.1 Idea principal

Como ya se comentó al inicio, las RNA tienen diversas partes características. Las entradas, número de capas ocultas y sus neuronas correspondientes y, por último, la capa de salida.

Las redes neuronales son capaces de aprender por sí mismas, es decir, dada una entrada y una salida pueden reestructurarse internamente para que dada una entrada similar ofrezca una salida similar. Al poseer una base de datos de 156 pacientes de los que disponemos el TAC correspondiente podíamos usarlos como características de entrada y como salida que nos devolviera las próstatas que disponíamos de ellos.

El problema de las redes neuronales es que el número de características de entrada que debes ofrecer no debe ser un número muy grande por lo que no podemos pasar todos los píxeles disponibles en cada TAC, un total de 4 260 000 pixeles por cada paciente.

Inicialmente los pacientes contenían un número diverso tanto de cortes como de tamaño de imágenes. En su momento se decidió normalizar todos estos datos para poder trabajar mejor con ellos como matrices y porque para redes neuronales venía mejor.

De esta manera se redimensionaron todas las imágenes a un tamaño de 200x300 px y a un total de 71 cortes por TAC, la imagen central y 35 imágenes a cada extremo para asegurarnos de que se encontrara la próstata en esta selección.

5.2.2 Características de entrada y salida

Como hemos comentado, no podemos pasar toda la matriz tridimensional del TAC a la red neuronal, o no es aconsejable. De igual manera tampoco se le puede proporcionar como salida toda la matriz de la próstata así que usé una librería que se usaba en el experimento de la segmentación del cerebro que devolvía 27 puntos o pixeles de interés dentro de un cubo de las proporciones que desearas.

Me explico mejor, si seleccionamos un cubo de 3x3x3 px obtenemos un total de 27 pixeles o características. Si en vez de ese tamaño seleccionamos uno mayor, como 5x5x5 px, la función de la librería nos proporciona 27 características de nuevo, eligiendo los pixeles centrales y los de las esquinas del cubo como se puede observar en la figura marcados de una tonalidad grisácea (ver Figura 12).

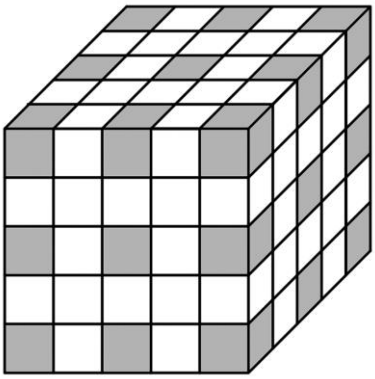

Figura 12 - Representación de la selección de características de un bloque de tamaño 5x5x5

Así seleccionamos pixeles clave de la imagen sin seleccionar toda ella y reduciendo la cantidad de datos de entrada muy significativamente. En total se usaron cinco tamaños diferentes de bloques, tamaños de 3, 7, 9 y 11 y 13 pixeles respectivamente.

Por otro lado, como características de entrada también se generó una matriz de probabilidad calculada con todas las próstatas disponibles donde se indicaba la probabilidad de que en ese punto de la matriz hubiera o no un segmento de la próstata.

En cambio, para los datos de salida se realizó algo parecido, pero de una manera mucho más simple, solamente se utilizó un bloque de tamaño 3x3x3 junto con la librería para seleccionar puntos clave de la próstata.

5.2.3 Estructura de la red neuronal

Normalmente una red neuronal, una vez es entrenada con todos los datos de entrada y salida especificados es funcional y ya es capaz de realizar clasificaciones o en este caso, contorneado de próstatas.

En cambio, hemos seguido un modelo algo más complejo que una simple red neuronal en este caso, y en el entrenamiento de la red se ha usado una técnica llamada *boosting* que consiste en el entrenamiento de varias redes neuronales, cada una partiendo de la anterior para al final realizar una votación ponderada y elegir la que mejores resultados ofrezca [12].

En este caso se han utilizado un total de cinco redes neuronales para el método de *boosting* donde el primer entrenamiento parte con unas características y valores aleatorios y los siguientes siempre se basan en la red ya entrenada anterior.

5.2.4 Experimento realizado

Al principio se introdujeron valores aleatorios para ver los resultados que producía la red neuronal entrenada y al ver que podía llegar a obtenerse un resultado decente intentamos hallar los valores óptimos para nuestro experimento.

Las redes neuronales que resuelven problemas de este estilo no se caracterizan por tener más de dos capas de neuronas ocultas, diferentes a la capa de entrada o salida, así que el experimento que se realizó fue el cálculo del número de neuronas óptimo para que se obtuvieran los mejores resultados de la primera capa.

El experimento se realizó sobre una única capa oculta, a pesar de que la red objetivo tuviera dos, para detectar a partir de que número de neuronas la red se sobreentrenaba, que es un concepto que indica que la red neuronal solo sirve para detectar los casos de entrenamiento y no los generales que sería lo ideal. El número de neuronas que se usan a medida que profundizas en cada capa suele disminuir, así que, si encontramos el mejor número de neuronas para la primera capa, para las siguientes el número tendrá que ser menor que el obtenido.

Para realizar esta prueba construimos un *script* que nos permitiera lanzar el entrenamiento de la red neuronal un total de 45 veces, usando los valores impares del 1 al 89 (se iba a lanzar hasta el valor 100, pero suponía mucho tiempo de coste computacional a medida que se aumentaba el número de neuronas) para el número de neuronas en la primera capa oculta. Para cada valor, se entrenaban dos redes neuronales y se guardaba cada una de las redes. Entonces se pasaba un test para comprobar la eficacia de cada una de ellas mostrando el tanto por ciento de acierto y una imagen de la reconstrucción de la

próstata en un corte clave del TAC de un paciente como podemos observar en la siguiente figura (ver Figura 13).

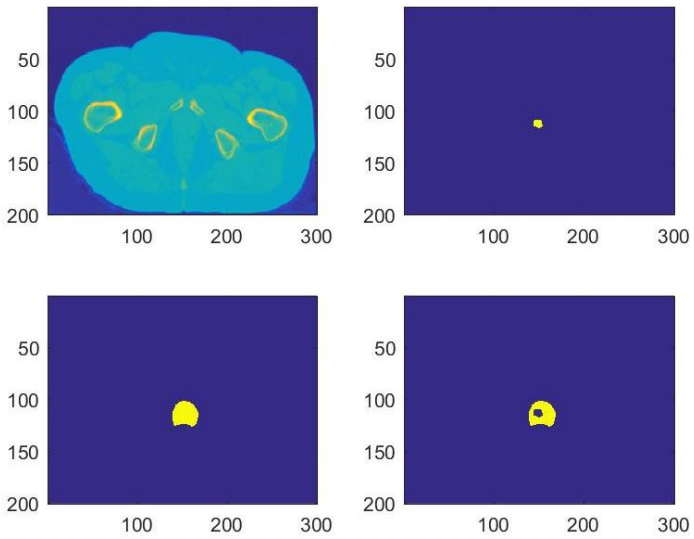

Figura 13 - Visualización de los resultados de la reconstrucción con RNA. La primera imagen representa un cortee del TAC, la de la derecha la próstata generada, la inferior izquierda la próstata del especialista y la última la diferencia entre ellas

De esta manera al final del experimento obtuvimos 45 redes entrenadas diferentes, donde cada una de ellas estaba entrenada y probada dos veces. Esto se debe a que pueden existir pequeñas diferencias en los resultados según que valores aleatorios iniciales cogió la red para empezar el entrenamiento de la misma. Al realizar lo mismo dos veces se podía promediar y conseguir unos valores de acierto algo más precisos, aunque lo ideal sería promediar entre más redes con el test pasado y el tanto por ciento de acierto medio para los pacientes de test.

A partir de aquí intentamos varios experimentos rápidos como una vez elegido el mejor número de neuronas para la primera capa de los estudiados probar con dos capas ocultas, variar el tamaño de los bloques que seleccionan las características de cada imagen, pero debido a problemas de tiempo y que se trataba de un trabajo opcional no se avanzó más en él.

5.3 Implementación de scripts para *kempes*

Centrándonos en otro tema, a la hora de realizar todas las pruebas mencionadas con anterioridad hemos hecho uso del supercomputador *kempes*. La conexión se realiza mediante el protocolo *Secure SShell (SSH)* que nos permite conectarnos a otra máquina de manera remota. Para manejar el supercomputador se hace uso del intérprete de comandos basado en el sistema operativo instalado.

Desde este podemos ejecutar las aplicaciones instaladas y hacer uso de ellas de forma remota y sin interfaz gráfica, al menos, de conectándose de una manera tradicional y sin programas externos que nos den esta opción.

Así pues, para ejecutar el programa desarrollado en MATLAB debemos iniciar la aplicación desde el intérprete y lanzar el código que previamente debimos copiar en el directorio adecuado del supercomputador. En mi caso, se ha usado un programa que posibilita la conexión *SSH* desde el sistema operativo Windows y además incluye un gestor FTP para el traspaso de archivos de una manera más sencilla.

Aun así, decidí desarrollar varios *scripts* en MATLAB para lanzar las distintas versiones de una manera más controlada y automática. Tenemos distintas versiones que podemos denominar las que trabajan en forma secuencial y las que trabajan de manera paralela, y entre estos dos grupos variaciones en cada una de ellas.

Para conseguir resultados de una manera más eficaz dependiendo del paciente de test que se esté ejecutando podemos realizar un bucle *for* para cada paciente y en cada iteración del bucle medir todos los resultados que nos pueden ofrecer estas ejecuciones como puede ser tiempo de ejecución, tanto por ciento de acierto, entre otros valores, podemos ver el script en el Anexo 8.

De igual manera, para el apartado de redes neuronales se usó un script que me permitía cambiar los parámetros de la red de una manera más sencilla y lanzar el programa de continuo ofreciéndome los resultados a medida que se iban calculando y no tener que realizar cada modificación de forma manual y estando pendiente de cuando terminaba cada iteración del programa. El script de la red neuronal se puede ver en el Anexo 9 y Anexo 10 según sea para el entrenamiento o el test.

5.4 Implementación de scripts para la visualización de los resultados

Al igual que se necesitaba unos scripts para poder realizar de una manera más cómoda distintas ejecuciones con variaciones del programa, también diseñé unos scripts que me permitían leer de una manera más eficaz los datos que generaba, se pueden obtener en el Anexo 11.

En estos podía interpretar tanto el tiempo como el acierto de cada prueba sobre cada paciente, y podía trabajar con todas las imágenes reconstruidas al igual que con sus próstatas.

Si trabajaba con todos los datos a la vez podía buscar máximos y mínimos en cualquier apartado para poder localizar cual podía llegar a ser el mejor método de los planteados.

Por otro lado, para la visualización de las matrices tridimensionales se usó la función de MATLAB *dcm3DViewer* desarrollada por Eric Johnston [13].

6. Resultados

Podemos dividir el apartado de resultados según los avances que hemos ido realizando en la aplicación principal, también se tratará el modelo con redes neuronales y una comparación conjunta con todos los datos.

En el programa principal siempre se han usado los diez últimos pacientes como pacientes de test y no se ha incluido en la base de datos para realizar las comparaciones, de manera que 146 pacientes pertenecían a la base de datos y por otro lado tenemos los diez que hemos comentado.

En las redes neuronales se han usado un total de 80 pacientes de entrenamiento y 17 de test que son los pacientes originales con una separación de 3mm por corte. Se esperaba poder incorporar la interpolación realizada a los pacientes de cortes cada 5mm, pero como se comentó anteriormente el tiempo fue un problema para este experimento extra.

6.1 Aplicación para la segmentación de próstata

Primero nos vamos a centrar en el programa principal del TFG como es la paralelización de la aplicación y detallaremos los resultados obtenidos de igual manera que se han ido explicando en el punto de desarrollo.

6.1.1 Primera versión del programa

La primera versión del programa es la versión que se me facilitó sin ninguna mejora implementada en ella. En este código, la selección del segmento adecuado de la próstata contenía un pequeño error y provocaba un acierto bastante bajo respecto las próstatas segmentadas por un especialista.

En la siguiente tabla (ver Tabla 1) se puede comprobar el tanto por ciento de acierto y el tiempo que se tardó en procesar cada paciente de test, esto último en segundos.

Paciente	Acierto (%)	Tiempo de ejecución (s.)
147	17.71 %	1576.7
148	30.14 %	1498.6
149	14.51 %	1577.9
150	19.62 %	1548.1
151	4.61 %	1490.4
152	8.32 %	1488.5

153	29.72 %	1501.7
154	34.89 %	1495
155	0.00 %	1492.1
156	6.05 %	1493.6

Tabla 1 - Representación del tanto por ciento de acierto y tiempo de ejecución por paciente de la primera versión del programa

Se puede comprobar que el acierto es bajo debido a que en esta primera versión había un fallo a la hora de recalcular la próstata óptima como hemos comentado. Podemos ver de manera gráfica el error y el tiempo de cada paciente en las siguientes dos figuras (ver Figura 14y Figura 15):

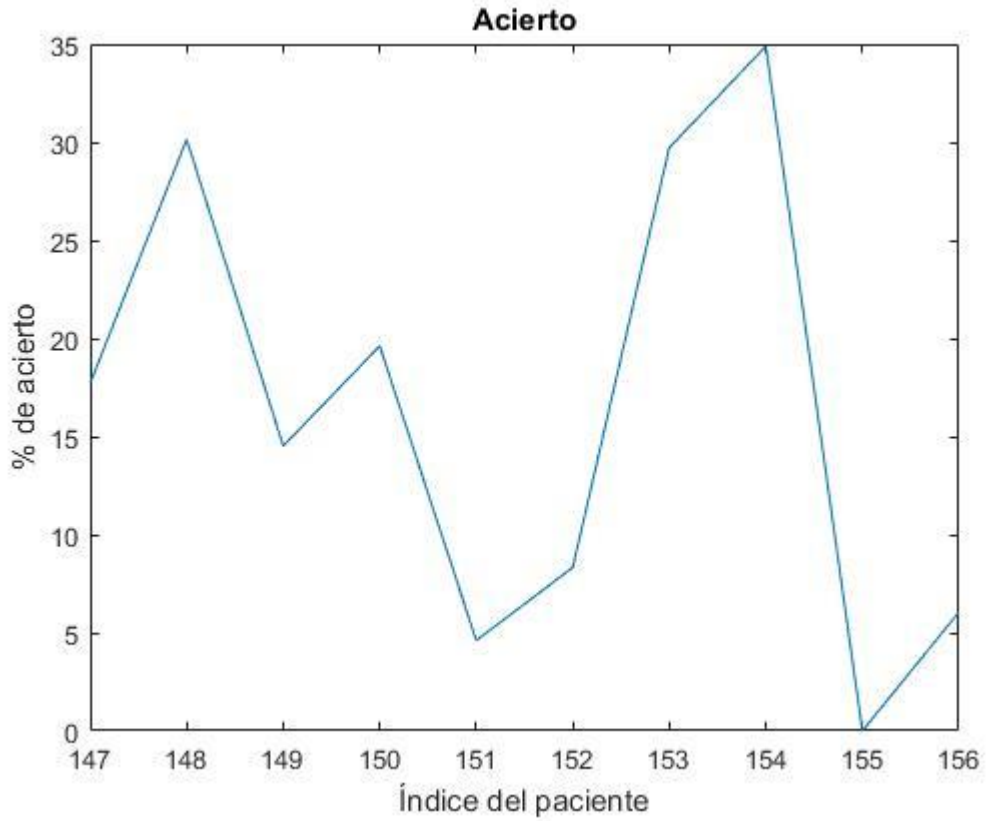

Figura 14 - Representación del acierto en el conjunto de pacientes de test para la primera versión del programa

En la figura relacionada con el acierto, podemos destacar ese paciente 155 que recibe un 0% de acierto, aunque la reconstrucción no sea la mejor en esta primera versión todos los otros pacientes si que llegaron a mostrar algún tanto por ciento, incluso el paciente154 alcanzó un 34,89 %.

Esto puede deberse a la estructura de la próstata de este paciente, puede poseer una forma o una posición distinta respecto a los demás pacientes y no puede encontrar el mejor segmento. También si se tuviera una base de datos mayor esto podría ser diferente y que se encontrara un paciente con las mismas características.

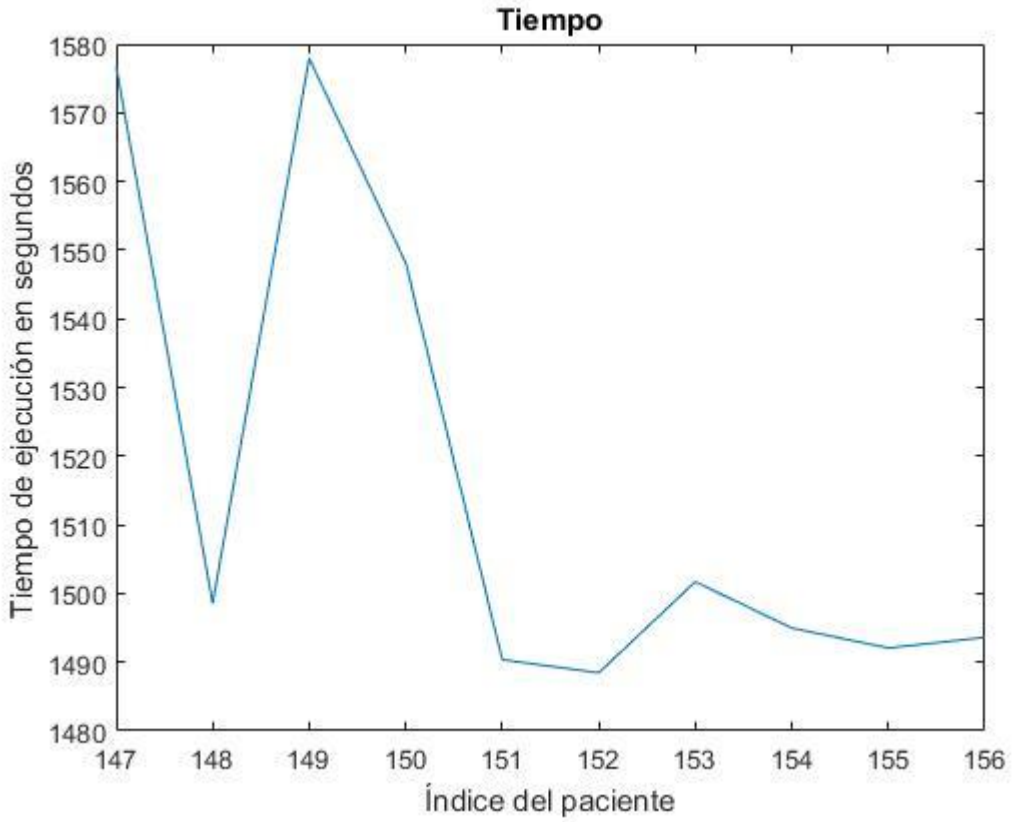

Figura 15 - Representación del tiempo de ejecución en el conjunto de pacientes de test para la primera versión del programa

Si nos centramos en el tiempo, podemos observar unos máximos tanto en el paciente 147, como en el 149 y en menor medida el 150. La diferencia entre estos y los otros pacientes en el peor de los casos llega a ser de hasta 89,4 segundos.

Si nos centramos en datos más generales podemos calcular los mínimos, máximos, medias y medianas de cada apartado mostrando el paciente en el que se han obtenido esos mejores y peores resultados. Se puede observar en las siguientes dos tablas que hacen referencia tanto al acierto como al tiempo consumido (ver Tabla 2 y Tabla 3):

	Acierto (%)	**Paciente**
Acierto mínimo	0.00 %	155
Acierto máximo	34.89 %	154
Acierto medio	16.56 %	-

Mediana	16.11 %	-

Tabla 2 - Representación de datos del tanto por ciento de acierto en la primera versión del programa

Como hemos comentado los resultados no son especialmente buenos, sobre todo nos podemos percatar si estudiamos la media o mediana de estos pacientes donde se llega a un 16.56% de acierto entre ellos.

	Tiempo (s.)	**Paciente**
Tiempo mínimo	1488.5	152
Tiempo máximo	1577.9	149
Tiempo medio	1516.2	-
Mediana	1496.8	-

Tabla 3 - Representación de datos del tiempo en la primera versión del programa

Si nos centramos en el tiempo, estamos consumiendo aproximadamente unos 25 minutos y 16 segundos por paciente para reconstruir la próstata según la base de datos.

6.1.2 Paralelización del bucle paciente

Una vez se realizó la primera paralelización se pudo comprobar como el tiempo disminuyó de una manera notable. Además, se corrigió el error sobre la selección de la próstata adecuada y se pueden observar unos mejores resultados de acierto en la siguiente tabla de datos (ver Tabla 4).

Paciente	**Acierto (%)**	**Tiempo de ejecución (s.)**
147	46.21 %	765.76
148	71.52 %	806.06
149	64.04 %	646.65
150	50.86 %	730.00
151	60.23 %	688.60
152	27.62 %	684.91
153	53.10 %	703.90
154	55.07 %	839.81

155	0.10 %	700.65
156	61.55 %	647.47

Tabla 4 - Representación del tanto por ciento de acierto y tiempo de ejecución por paciente de la versión del bucle "paciente" paralelizado

Si los representamos gráficamente de la misma manera que el modelo anterior conseguimos las siguientes gráficas (ver Figura 16 y Figura 17):

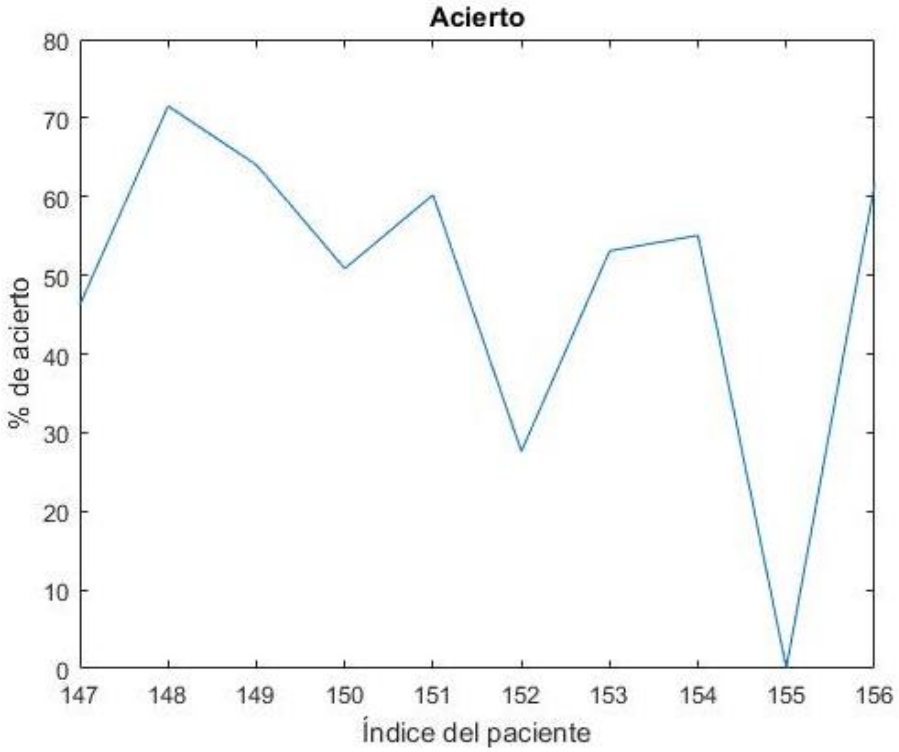

Figura 16 - Representación del acierto en el conjunto de pacientes de test para la versión con la reconstrucción corregida

De nuevo se puede observar un acierto cercano al 0 % en el paciente 155. Como ya se ha explicado antes, puede ser que el paciente tenga la próstata en una posición o tenga una forma distinguida. En todo caso, los demás pacientes han aumentado el acierto drásticamente respecto la anterior versión. El mejor caso anterior, cercano a un 35 %, es casi el peor caso de la mejora sin contar el paciente mencionado, en este caso el paciente 152 obtiene un 27,62% de acierto, por lo que se ha mejorado notablemente.

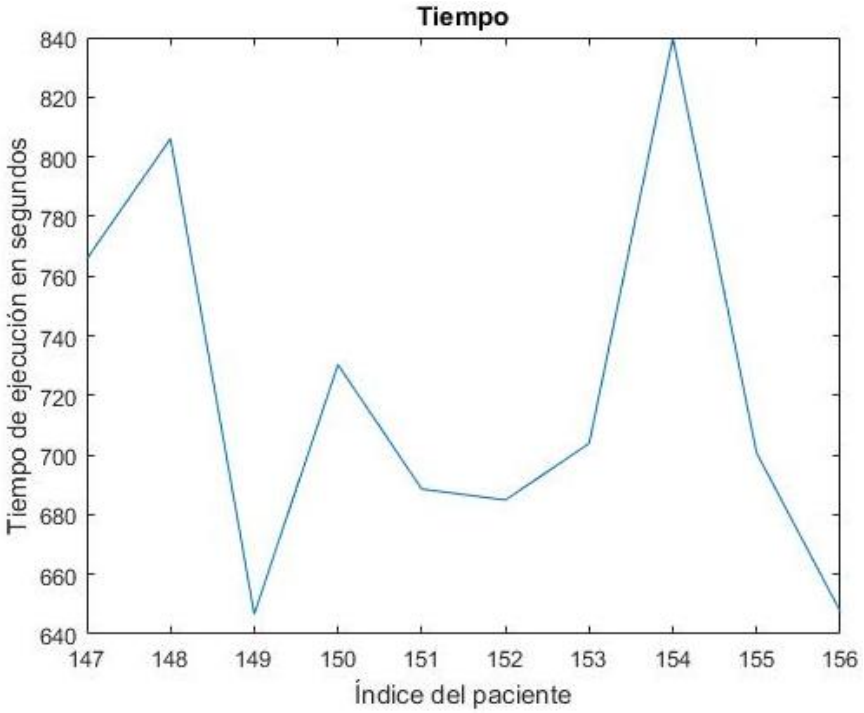

Figura 17 - Representación del tiempo en el conjunto de pacientes de test para la versión con el bucle "paciente" paralelizado

Respecto a la gráfica relacionada con el tiempo encontramos varios máximos y mínimos en diversos pacientes, en este caso la diferencia entre el peor y el mejor caso ha aumentado hasta los 193,16 segundos de diferencia.

Y si de nuevo recalculamos los máximos y mínimos, media y mediana de los resultados tenemos los siguientes datos (ver Tabla 5 y Tabla 6):

	Acierto (%)	**Paciente**
Acierto mínimo	0.01 %	155
Acierto máximo	71.52 %	148
Acierto medio	49.03 %	-
Mediana	54.09 %	-

Tabla 5 - Representación de datos del tanto por ciento de acierto para la versión con el bucle "paciente" paralelizado

Es en la tabla mostrada donde podemos ver con datos la mejora de la que hablábamos al solucionar el error del cálculo de la mejor próstata. Hemos pasado de una media de un 16,56 % de acierto hasta un 49,03 %. Estos resultados, aunque mejores, tampoco lo son como para sustituir a un especialista, pero sí que pueden ser de utilidad para un ahorro del trabajo del mismo como se comentará en el apartado de conclusiones.

	Tiempo (s.)	Paciente
Tiempo mínimo	646.65	149
Tiempo máximo	839.81	154
Tiempo medio	721.41	-
Mediana	702.28	-

Tabla 6 - Representación de datos del tiempo para la versión del bucle "paciente" paralelizado

El tiempo se ha reducido de manera drástica ya que se ha paralelizado un bucle con bastantes cálculos, hemos pasado de los 1516,6 segundos de media a unos 721,41 segundos, que equivalen a 12 minutos aproximadamente.

6.1.3 Reestructuración de bucles

Después de realizar la primera paralelización se decidió reestructurar los bucles, y para comprobar si este cambio podía aportar algo en velocidad realicé de nuevo las pruebas de una manera secuencial para comparar los resultados con la primera versión del programa sin tener en cuenta el tanto por ciento de acierto, la parte que nos interesaba era ver si se producía una mejora en el tiempo con esta reestructuración (ver Tabla 7).

Paciente	Acierto (%)	Tiempo de ejecución (s.)
147	46.21 %	1533.0
148	71.52 %	1516.8
149	64.04 %	1514.8
150	50.86 %	1505.9
151	60.23 %	1490.5
152	27.62 %	1492.4
153	53.10 %	1485.8
154	55.07 %	1496.9
155	0.10 %	1510.2
156	61.55 %	1517.6

Tabla 7 - Representación del tanto por ciento de acierto y tiempo de ejecución por paciente de la versión con los bucles reestructurados

Dado que no se realiza ninguna mejora más a la hora de realizar el cálculo del error para la búsqueda de la mejor próstata, omitiremos los resultados del acierto porque son exactamente los mismos para que nueva versión de la aplicación. La variable que se va desarrollando y variando es el tiempo de ejecución por lo que nos centraremos en ella (ver Figura 18).

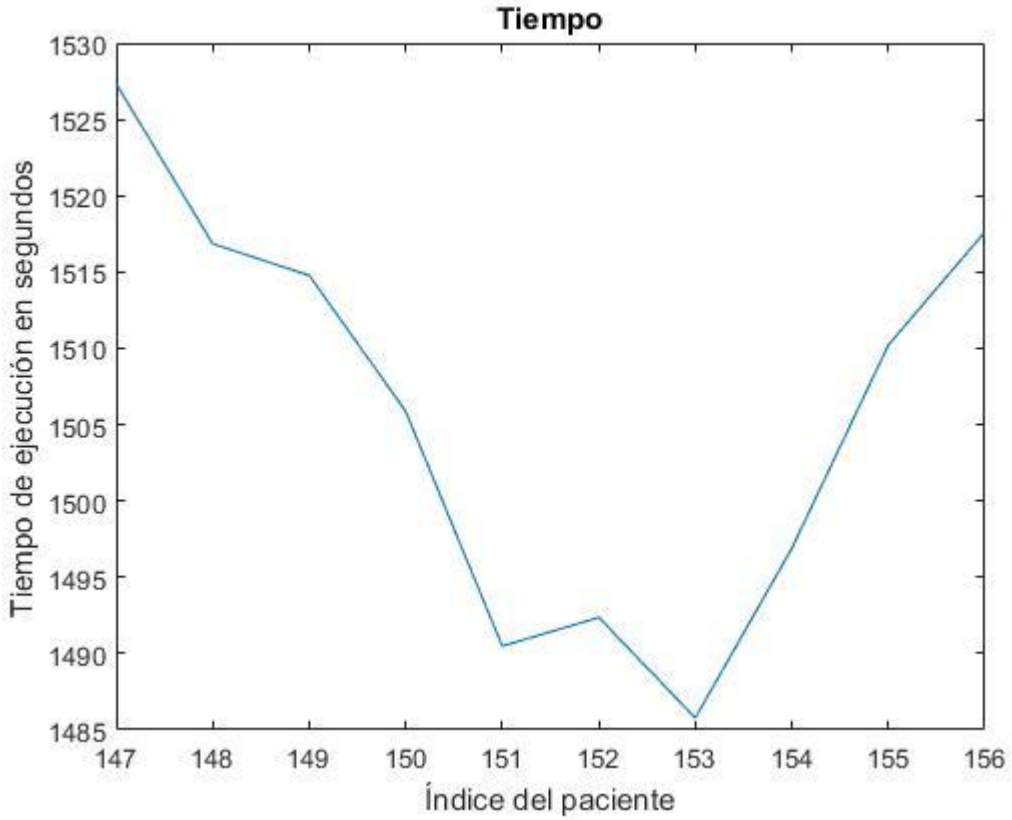

Figura 18 - Representación del tiempo en el conjunto de pacientes de test para la versión con los bucles reestructurados

En este caso, si nos fijamos, aparentemente devuelve unos resultados muy similares a la primera versión del programa. Si estudiamos estos valores de la misma manera que en los apartados anteriores podemos obtener la siguiente tabla (ver Tabla 8):

	Tiempo (s.)	**Paciente**
Tiempo mínimo	1485.8	153
Tiempo máximo	1533.0	147
Tiempo medio	1505.8	-
Mediana	1508	-

Tabla 8 - Representación de datos del tiempo de la versión con los bucles reestructurados

Prácticamente se puede decir que no hay una mejora sustancial a la hora de reestructurar los bucles, se puede mencionar que se ha mejorado el tiempo medio en unos 10 segundos, pero no es algo demasiado relevante como para indicar que si que se ha producido una mejora.

6.1.4 Paralelización del bucle exterior del eje Y

A este experimento siguió la paralelización de un bucle más externo, se decidió por el bucle del eje Y al poseer un mayor número de iteraciones que podrían ser paralelizables. Se modificó el orden de los tres bucles externos y se realizó la paralelización con lo que se obtuvieron los siguientes resultados (ver Tabla 9):

Paciente	Acierto (%)	Tiempo de ejecución (s.)
147	46.21 %	381.81
148	71.52 %	377.30
149	64.04 %	374.63
150	50.86 %	374.85
151	60.23 %	375.39
152	27.62 %	375.40
153	53.10 %	373.14
154	55.07 %	374.88
155	0.10 %	374.18
156	61.55 %	373.00

Tabla 9 - Representación del tanto por ciento de acierto y tiempo de ejecución por paciente de la versión con el bucle exterior paralelizado

Como se puede ver en la columna de acierto, se obtienen los mismos datos que las anteriores versiones ya corregidas del programa, por lo que de nuevo mencionar que no se estudiarán estos resultados.

En cambio, si nos fijamos en la gráfica generada sobre el tiempo de procesamiento para cada paciente observamos ya resultados notables (ver Figura 19).

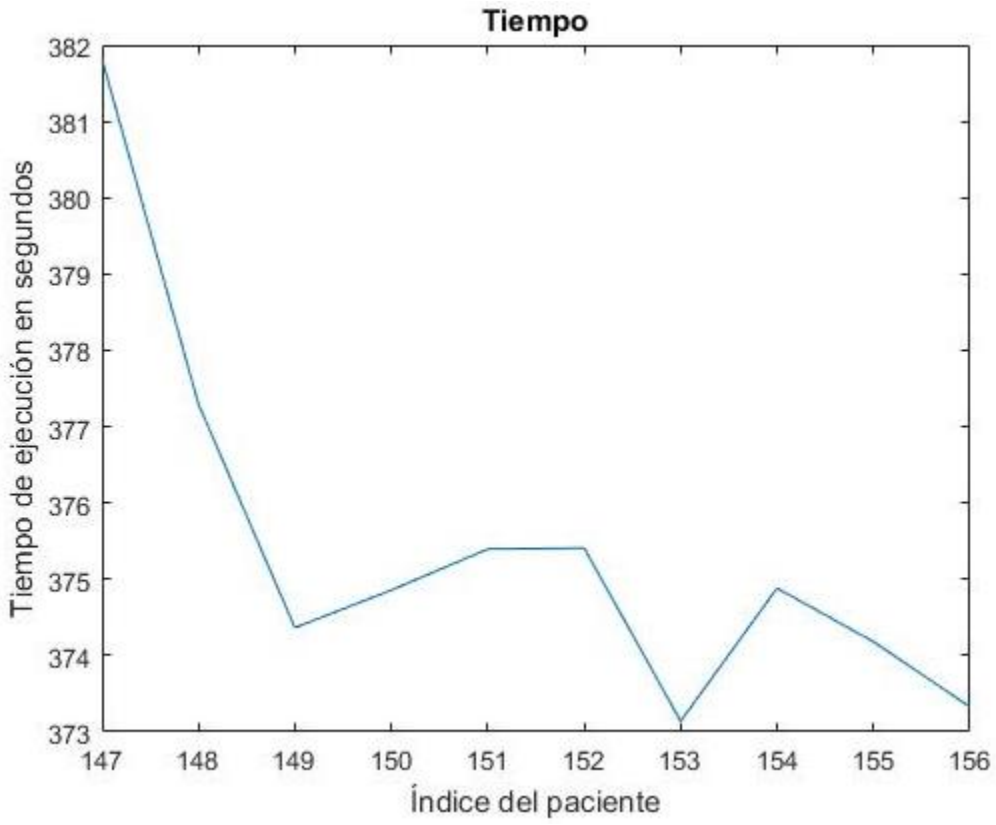

Figura 19 - Representación del tiempo en el conjunto de pacientes de test para la versión paralelizada del bucle más externo

En este caso ya hemos disminuido el tiempo por debajo de los 400 segundos. En este caso se puede apreciar un máximo en el primer paciente bastante notorio en la escala de tiempo respecto a los demás que produce una diferencia entre el mejor y peor caso de apenas 8,67 segundos.

De igual manera si nos centramos en datos que podemos generar (ver Tabla 10):

	Tiempo (s.)	**Paciente**
Tiempo mínimo	373.14	153
Tiempo máximo	381.81	147
Tiempo medio	375.46	-
Mediana	374.87	-

Tabla 10 - Representación de datos del tiempo de la versión paralelizada del bucle más externo

Observamos ya una media que se sitúa en los 375 segundos, que equivalen a 6 minutos y 15 segundos. De momento una mejora de 19 minutos respecto a la primera versión del programa.

6.1.5 Implementación del error en el lenguaje C (secuencial)

Para seguir avanzando en el proyecto se intentó realizar parte del código en el lenguaje de programación C como se comentó en el apartado 5.1.10. En este caso se realizó de nuevo una primera prueba secuencial por si afectaba de manera negativa al tiempo de ejecución el uso de tantas llamadas externas a la función implementada. En este caso los resultados que obtuvimos fueron los siguientes (ver Tabla 11):

Paciente	Acierto (%)	Tiempo de ejecución (s.)
147	46.21 %	2108,6
148	71.52 %	2072,1
149	64.04 %	2096,2
150	50.86 %	2078,3
151	60.23 %	2080,6
152	27.62 %	2068,8
153	53.10 %	2066,3
154	55.07 %	2077,6
155	0.10 %	2071
156	61.55 %	2070,2

Tabla 11 - Representación del tanto por ciento de acierto y tiempo de ejecución por paciente de la versión con el cálculo del error en C (secuencial)

Se puede observar que el tiempo es mucho mayor que en los otros experimentos realizados, y esto se puede deber al mayor número de llamadas a funciones externas comparadas con la versión anterior. En este caso, cada vez que se intenta calcular el error se hace una llamada, aumentando el tiempo debido al intercambio de información entre el código en MATLAB y el código en C. Podemos ver la gráfica de ello en la siguiente figura (ver Figura 20):

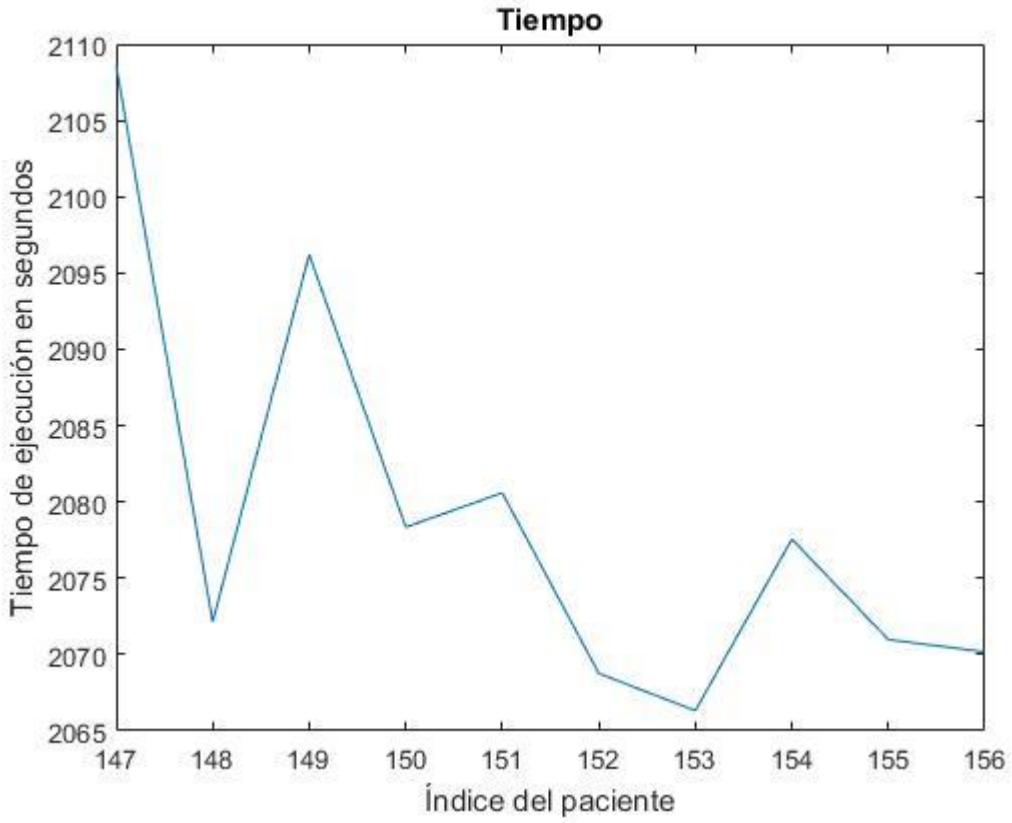

Figura 20 - Representación del tiempo en el conjunto de pacientes de test para la versión secuencial con la implementación del error en el fichero .mex

Igual que en los apartados anteriores podemos mostrar los siguientes resultados tras el estudio del tiempo de cada paciente de una manera más general (ver Tabla 12):

	Tiempo (s.)	**Paciente**
Tiempo mínimo	2066.3	153
Tiempo máximo	2108.6	147
Tiempo medio	2079	-
Mediana	2074.8	-

Tabla 12 - Representación de datos del tiempo de la versión secuencial con la implementación del error en el fichero .mex

Y podemos indicar que se produce un empeoramiento del tiempo de ejecución por los motivos mencionados anteriormente. Alcanzamos una media de 34 minutos y 39 segundos

de media para obtener los mismos resultados en las versiones anteriores. Como estamos tratando la versión secuencial del programa y empeora, también podemos suponer que la versión paralela del mismo obtendría resultados peores comparada con otras versiones en paralelo, aun así, decimos realizar la prueba para comprobarlo.

6.1.6 Implementación del error en el lenguaje C (paralelo)

En este segundo modelo se realizó la comparación con el modelo paralelo del bucle más externo. Como podemos suponer, debería ser algo más costoso en tiempo que la versión original sin la llamada al código en C por el mismo problema que sucede en la versión secuencial. Podemos ver los resultados y la gráfica en la tabla y figura siguiente (ver Tabla 13 y Figura 21).

Paciente	Acierto (%)	Tiempo de ejecución (s.)
147	46.21 %	422,92
148	71.52 %	418,59
149	64.04 %	418,29
150	50.86 %	418,23
151	60.23 %	422,43
152	27.62 %	423,99
153	53.10 %	419,06
154	55.07 %	419,05
155	0.10 %	420,17
156	61.55 %	419,45

Tabla 13 Representación del tanto por ciento de acierto y tiempo de ejecución por paciente de la versión con el cálculo del error en C (paralelo)

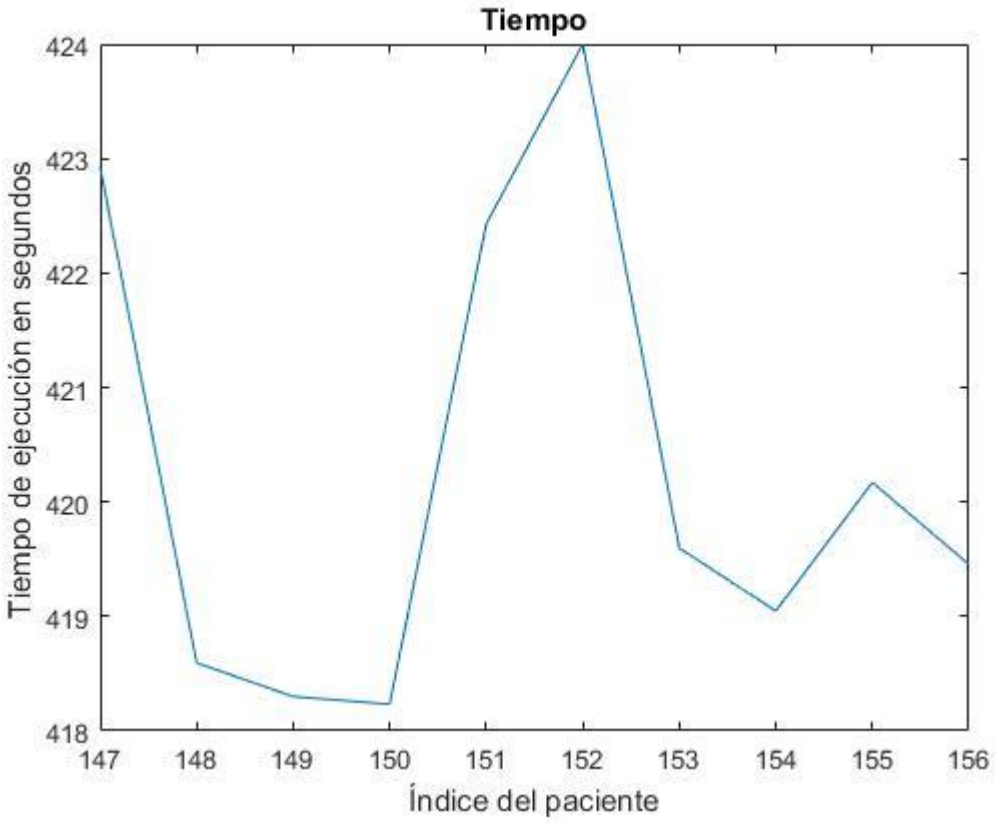

Figura 21 - Representación del tiempo en el conjunto de pacientes de test para la versión paralela con la implementación del error en el fichero .mex

De nuevo para estudiar el coste temporal de una manera más general vamos a centrarnos en los datos generados teniendo en cuenta todos los pacientes y que podemos ver en la siguiente tabla (ver Tabla 14):

	Tiempo (s.)	**Paciente**
Tiempo mínimo	418,23	150
Tiempo máximo	423,99	152
Tiempo medio	420,27	-
Mediana	419,52	-

Tabla 14 - Representación de datos del tiempo de la versión paralela con la implementación del error en los ficheros .mex

Como era de esperar, aunque hemos obtenido unos resultados buenos comparados con la versión secuencial, son peores en comparación a otras versiones paralelas del programa, aunque sí que es verdad que la diferencia entre ellas no es tan amplia como la diferencia entre distintas versiones secuenciales.

6.1.7 Implementación del código en C del mejor error de un paciente (secuencial)

En el siguiente experimento traspasamos una cantidad mayor de código de MATLAB a C, en especial tres bucles *for*, por lo que el rendimiento debería mejorar respecto al anterior código. Además, se le suma que, al realizar menos iteraciones a una llamada a un código externo, la función que desarrollamos, disminuimos el tiempo de intercambio de mensajes agilizando y optimizando el tiempo de cómputo. En la siguiente tabla podemos ver los resultados obtenidos de este experimento en su versión secuencial (ver Tabla 15).

Paciente	**Acierto (%)**	**Tiempo de ejecución (s.)**
147	46.21 %	624,392
148	71.52 %	566,9436
149	64.04 %	567,018
150	50.86 %	567,045
151	60.23 %	567,83
152	27.62 %	567,3282
153	53.10 %	565,8438
154	55.07 %	567,3945
155	0.10 %	565,6619
156	61.55 %	563,1942

Tabla 15 - Representación del tanto por ciento de acierto y tiempo de ejecución por paciente de la versión con el cálculo del mejor error en C (secuencial)

En este caso realizamos el mismo procedimiento que en el anterior experimento, primero conseguimos los datos de la versión en secuencial y después con la versión en paralelo, podemos ver la gráfica del tiempo en la siguiente figura (ver Figura 22):

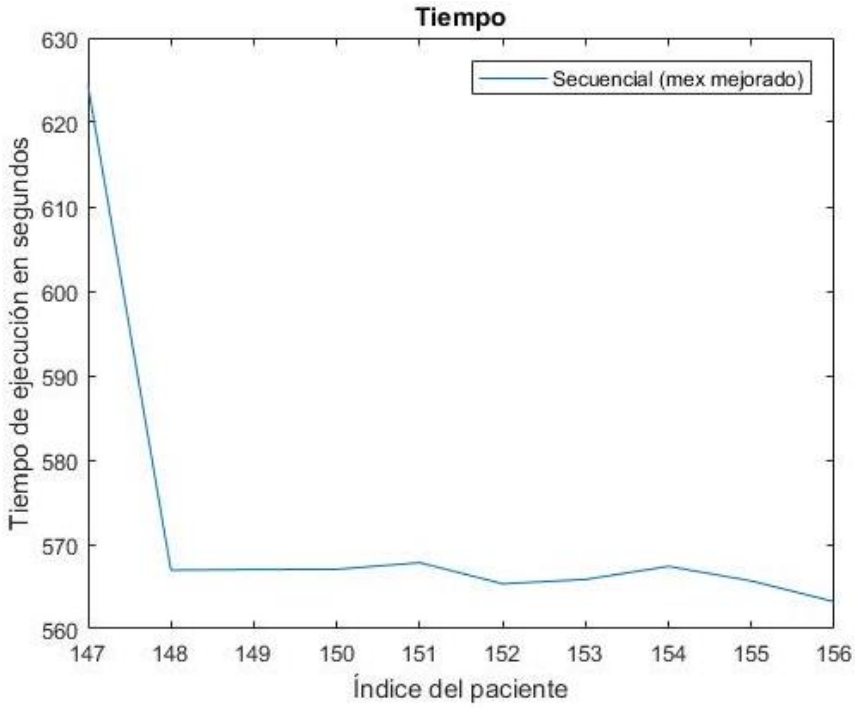

Figura 22 - Representación del tiempo en el conjunto de pacientes de test para la versión secuencial con la implementación del mejor error en el fichero .mex

Si nos centramos en el cálculo de los datos sobre el tiempo mínimo, máximo y su media y mediana obtenemos lo siguientes valores (ver Tabla 16):

	Tiempo (s.)	**Paciente**
Tiempo mínimo	563,1942	156
Tiempo máximo	624,33924	147
Tiempo medio	572,0652	-
Mediana	566,9808	-

Tabla 16 - Representación de datos del tiempo de la versión secuencial con la implementación del mejor error en el fichero .mex

En este caso, obtenemos una gran mejoraría del tiempo tratándose de la versión secuencial del programa. De igual manera que las versiones secuenciales anteriores rondaban los 25 minutos de computo por paciente, esta ha disminuido el tiempo hasta los 9 minutos y medio.

De esta manera se puede demostrar que el tratamiento de la información y la velocidad de cómputo son fácilmente alterables no solo paralelizando, sino, también usando un lenguaje de programación que se especialice en ello. Como dijimos al principio,

MATLAB no se trabaja demasiado bien con los bucles *for* por lo que dejando estas partes de código al lenguaje C podemos obtener resultados mucho mejores.

6.1.8 Implementación del código en C del mejor error de un paciente (paralelo)

De igual manera que en el apartado anterior vamos a estudiar el comportamiento del programa tras realizar más cálculos en la función implementada e C del programa, en este caso incluyendo la paralelización del bucle más externo y del que estamos familiarizados, aquel que mejores resultados dio comparado con la otra paralelización. Podemos observar los detalles en la siguiente tabla y en la siguiente gráfica (ver Tabla 17 y Figura 23):

Paciente	Acierto (%)	Tiempo de ejecución (s.)
147	46.21 %	278,7252
148	71.52 %	273,0438
149	64.04 %	274,3435
150	50.86 %	273,9904
151	60.23 %	273,9594
152	27.62 %	274,2478
153	53.10 %	273,5822
154	55.07 %	272,461
155	0.10 %	273,8423
156	61.55 %	273,7597

Tabla 17 - Representación del tanto por ciento de acierto y tiempo de ejecución por paciente de la versión con el cálculo del mejor error en C (paralelo)

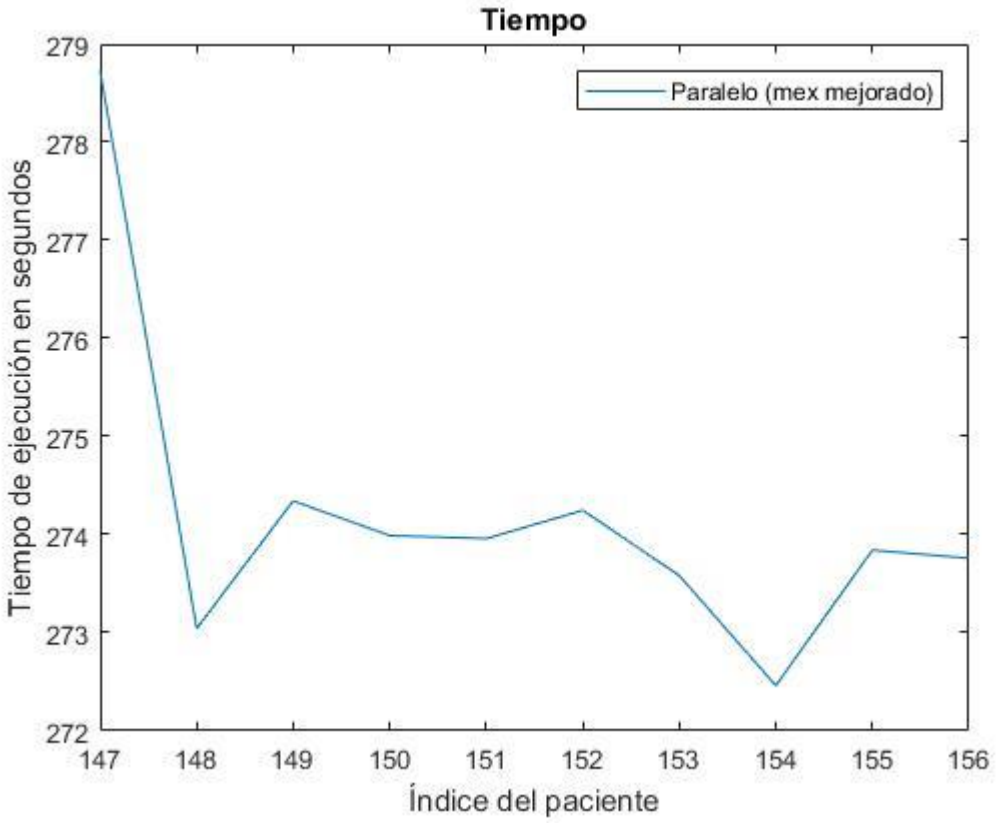

Figura 23 - Representación del tiempo en el conjunto de pacientes de test para la versión paralela con la implementación del mejor error en el fichero .mex

Si nos centramos directamente en los datos generales extraídos de la tabla anterior podemos construir la siguiente tabla de información (ver Tabla 18):

	Tiempo (s.)	**Paciente**
Tiempo mínimo	272,4610	154
Tiempo máximo	278,7252	147
Tiempo medio	274,1956	-
Mediana	273,9008	-

Tabla 18 - Representación de datos del tiempo de la versión paralela con la implementación del mejor error en el fichero .mex

Como era de esperar, teniendo una versión secuencial tan eficiente, si la paralelizábamos íbamos a obtener unos resultados mejores que los anteriores, y así ha sido el caso. Con esta versión hemos obtenido un tiempo medio de cómputo por paciente de 4 minutos y medio aproximadamente. Siendo la diferencia entre el mejor y el peor caso de apenas 6 segundos.

Y como es normal, consiguiendo los mismos resultados de acierto que la versión corregida, por lo que este es el mejor programa obtenido para la reconstrucción de la próstata.

6.1.9 Comparaciones entre las versiones anteriores

Si tenemos en cuenta todos los datos mostrados anteriormente podemos mostrar nuevas gráficas con la superposición de los acierto y tiempos de todas las pruebas.

Respecto al acierto el único cambio importante que se realiza es la corrección del código inicial, todas las siguientes versiones consiguen el mismo tanto por ciento de acierto, por lo que la comparación tras solucionar el error en las versiones y la versión original quedaría de la siguiente manera (ver Figura 24):

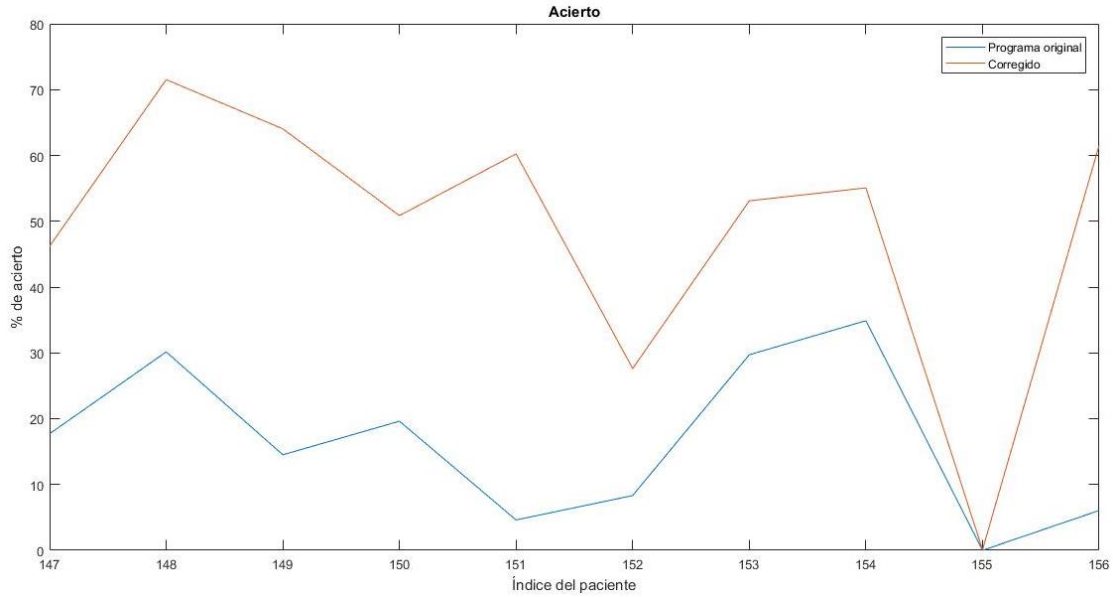

Figura 24 - Gráfica comparativa entre el tanto por ciento de acierto conseguido con la primera versión del programa y con la modificación del error

A excepción del paciente número 155, del que ya se hablado y supuesto los posibles problemas, todos los demás mejoran una media de 32,47 % respecto a la primera versión del programa. La máxima mejora se produce en el paciente 151 con 55,62 % de diferencia entre el mejor y el peor acierto.

Si vamos ahora a la comparación respecto el tiempo, en este caso podemos comparar todas las anteriores pruebas simultáneamente entre ellas. Siguiendo la mecánica del ejemplo anterior podemos realizar una gráfica con todos los tiempos de las pruebas como se muestra a continuación (ver Figura 25):

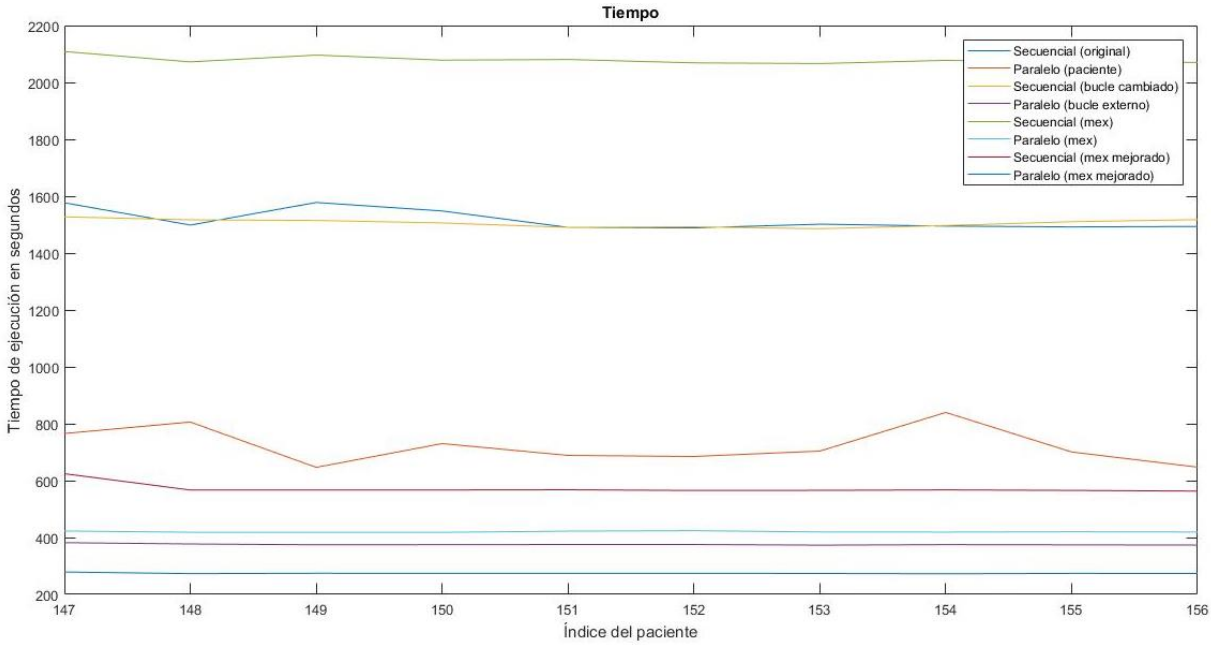

Figura 25 - Gráfica comparativa de tiempos para cada versión generada del programa

Se puede observar de una manera clara que la versión que mejores resultados ofrece es la última desarrollada. La versión con el fichero C que incluye el cálculo del mejor segmento en el propio código. Seguido por la versión con la paralelización del bucle más externo del programa como la misma versión con el cálculo del error en programación C, siendo la primera versión nombrada más rápida.

Destacar que la versión sin paralelizar de la última prueba realizada obtiene el cuarto puesto (versión con el cálculo del mejor segmento en C) superando incluso a la paralelización del bucle paciente, lo que nos lleva a pensar que el lenguaje usado y la manera de desarrollar el código de una aplicación puede influir más que la paralelización de la misma.

Respecto a las otras versiones, la peor es la versión secuencial del programa con el cálculo del error en el archivo *mex*. Es más lenta porque al realizar tantas llamadas a un fichero externo ralentiza la ejecución. La siguen las dos versiones secuenciales del programa, la original y con los bucles cambiados dando prácticamente los mismos resultados con pequeñas variaciones. Y por último la versión paralelizada en el bucle del paciente, que como hemos comentado es peor que la secuencial del último fichero *mex*.

6.2 Redes neuronales artificiales

Respecto a las redes neuronales podemos decir de una manera resumida que los resultados obtenidos no fueron los esperados.

Si nos fijamos en la prueba realizada, buscábamos determinar cuál era el número de neuronas óptimo para usar en la primera capa neuronal y de ahí ir mejorando el resultado con nuevas capas neuronales ocultas (ver Figura 26).

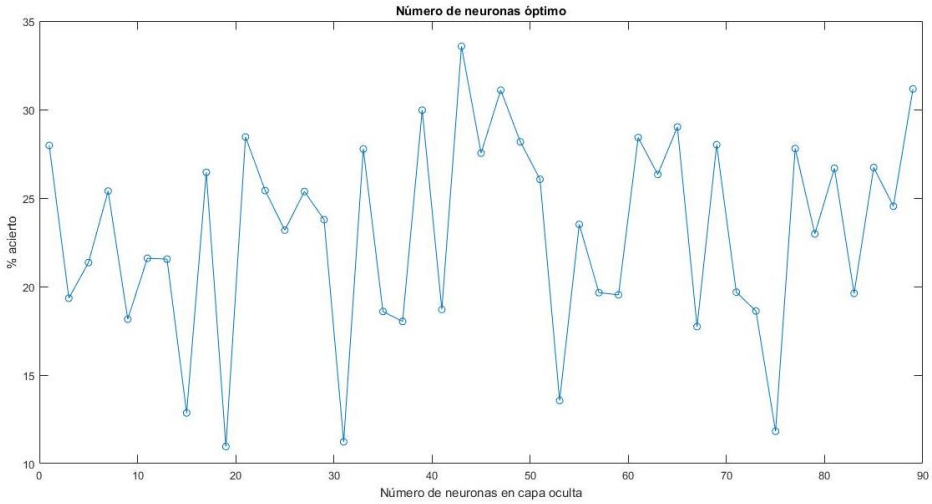

Figura 26 - Gráfica comparativa que muestra el tanto por ciento de acierto según el número de neuronas en la capa oculta

A pesar de encontrar uno que proporciona mayor índice de acierto que los demás, en este caso 43 neuronas en la primera capa, el valor de acierto conseguido sigue siendo muy bajo para competir con el programa original del TFG. En este caso estamos obteniendo en el mejor de los casos un acierto medio del 33,57 % trabajando con 43 neuronas en una única capa, que, si lo comparamos con el acierto medio del programa, 49,03 % es menor en un 15,46 %.

Esto puede deberse a las características elegidas para entrenar la red, tanto las de entrada como los puntos de salida. Se basó en una red neuronal para la segmentación de cerebros, realizada por el profesor J. V. Manjón Herrera, como ya se comentó y en ella sí que se obtenían muy buenos resultados. Por lo que la selección de las características para un TAC tiene que ser diferente de la empleada para RMN del cerebro. Cosa que nos dificultó el proyecto porque no podíamos investigar sobre cuál es la información más relevante de un TAC para un especialista que sirva para la detección y posterior segmentación de la próstata.

Aun así, cada cálculo se basa en la media entre varias redes neuronales entrenadas con las mismas características y distintos pacientes como se detalló en su apartado, y pesar de que es una media, en algunos casos se logró aciertos de hasta el 67 % respecto a la próstata segmentada por el especialista como se puede observar en la siguiente figura (ver Figura 27).

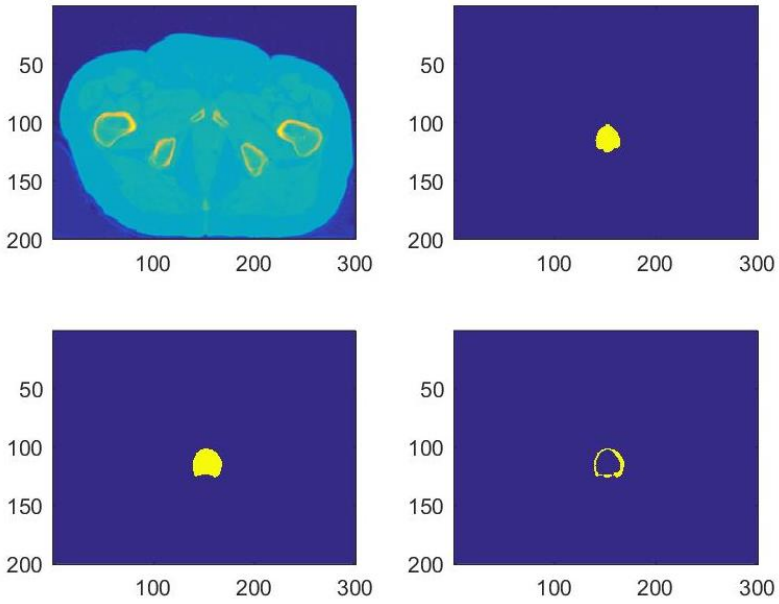

Figura 27 - Visualización del mejor segmento con una capa oculta de 43 neuronas

Donde la primera imagen es el corte 42 del TAC del último paciente de test, la superior derecha es la próstata generada por la red neuronal, la inferior izquierda la próstata segmentada por el especialista y la última la diferencia entre las dos próstatas.

7. Conclusión

En este último apartado se hará una síntesis de los resultados obtenidos, valorando el cumplimiento de los objetivos del proyecto. Además, se ofrecerán posibles mejoras o trabajos futuros que pueden mejorar la aplicación para alcanzar resultados mejores que los aquí conseguidos.

7.1 Análisis del trabajo realizado

El objetivo principal del TFG era la mejora de la eficacia de la aplicación, su paralelización para hacer uso de ordenadores con capacidad de multiprocesamiento en diferentes núcleos y la reducción de coste temporal que está ligada a la paralelización.

En nuestro caso, hemos obtenido unos resultados que han mejorado el porcentaje de acierto respecto el programa original en un 32,47 % de media con la corrección de la reconstrucción de las secciones de la mejor próstata para un paciente dado.

Por otro lado, hemos mejorado el coste temporal reduciéndolo desde los 25 minutos que tardaba la primera versión secuencial del programa, hasta los 9 minutos y medio que se obtienen de la versión secuencial con la implementación del cálculo de la mejor sección de un paciente en el fichero *mex*. Si incluimos la paralelización del programa, que era otro de los objetivos, hemos logrado tiempos medios de 4 minutos y medio para cada paciente procesado, lo que supone una reducción respecto al tiempo inicial del 82 %.

Con los resultados mencionados podemos concluir que sí se han logrado los objetivos planteados.

7.2 Posibles mejoras y trabajos futuros

El siguiente paso para mejorar aún más la aplicación no es otro que desarrollar todo el bucle *for* de pacientes e implementarlo en el fichero *mex* en C. Esto debería mejorar el tiempo todavía más al realizar menos bucles en MATLAB, pero añade la dificultad de trabajar con matrices de cuatro dimensiones en el fichero *mex*.

Otro tipo de mejora que se puede desarrollar para esta aplicación es, no solo paralelizar el fichero MATLAB, sino paralelizar el fichero *mex* con la interfaz de programación de aplicaciones (API) *OpenMP*.

Por supuesto también se podría desarrollar un estudio variando el tamaño de los bloques en cada dimensión y modificando el número de pacientes en la base de datos para ver cuál es la curva de porcentaje de acierto según se use un número u otro de pacientes y que número debería ser el óptimo para que el acierto y el tiempo de cómputo al recorrer toda la base de datos sea eficiente.

Po último, el trabajo con más posibilidades de mejora es el correcto desarrollo de las redes neuronales. En este TFG se intentó un experimento basado en esta tecnología, pero no se consiguieron resultados comparables a la versión programaba de MATLAB. A pesar de ello, con el tiempo suficiente para profundizar y estudiar las RNA se podría desarrollar y entrenar una red que aprendiera a medida que se le suministraran pacientes de test y fuera evolucionando con el tiempo adaptándose a las posibles evoluciones que surjan en el cuerpo humano en los próximos años.

Este tipo de redes se denominan redes neuronales incrementales y sería la aplicación ideal para temas médicos por la sencilla razón de que, como especie, evolucionamos [14]. No obstante, siempre y cuando se lograra un entrenamiento de la red que proporcionara un resultado de la reconstrucción de la próstata equiparable al de un especialista.

8. Bibliografía

[1] Wikipedia (2017, May 21). Imagen de mapa de bits [Online]. Disponible: https://es.wikipedia.org/wiki/Imagen_de_mapa_de_bits

[2] J. V. Manjón Herrera, Tema 6. Resonancia Magnética Nuclear, 2014, Feb 14. Informática Médica.

[3] J. V. Manjón Herrera, Tema 8. Procesamiento digital de señales, 2014, Mar 10. Informática Médica

[4] J. E. Román Moltó, 2015. Computación Paralela

[5] I. García. Fenoll, Segmentación de Imágenes Médicas (Cap. 3), [Online]. Disponible: http://bibing.us.es/proyectos/abreproy/11854/fichero/Volumen+1%252FCapitulo+3.pdf

[6] Wikipedia (2017, Jun 28). Red neuronal artificial [Online]. Disponible: https://es.wikipedia.org/wiki/Red_neuronal_artificial

[7] Wikipedia (2017, Jun 6). DICOM [Online]. Disponible: https://es.wikipedia.org/wiki/DICOM

[8] MathWorks, Documentation [Online]. Disponible: https://es.mathworks.com/help/index.html

[9] MathWorks, Parallel Computing Toolbox [Online]. Disponible: https://es.mathworks.com/help/distcomp/index.html

[10] Wikipedia (2017, Mar 27). Interpolación [Online]. Disponible: https://es.wikipedia.org/wiki/Interpolaci%C3%B3n

[11] MathWorks, Introducing MEX Files [Online]. Disponible: https://es.mathworks.com/help/matlab/matlab_external/introducing-mex-files.html

[12] H. Schwenk and Y. Bengio (2000). Boosting neural networks [Olline]. Disponible: https://www.ncbi.nlm.nih.gov/pubmed/10953242

[13] MathWorks File Exchange, Interactive DICOM 3D Viewer [Online]. Disponible: http://es.mathworks.com/matlabcentral/fileexchange/28353-interactive-dicom-3d-viewer?focused=5245726&tab=function

[14] F. Casacuberta Nolla, Tema 6. Redes Neuronales Multicapa, 2016, Sep 06. Aprendizaje Automático.

9. Anexo 1 – Primera versión del programa

```
function [imagen_reconstruida,
pros_reconstruida]=reconstruccion_secuencial(array_imagen, database_p,Vq)

muestras=size(array_imagen,4);
im3d_target=Vq;

imagen_reconstruida=zeros(200,300,70);
pros_reconstruida=zeros(200,300,70);

indblx=40;indbly=30;indblz=35;
extra_blx=4;extra_bly=4;extra_blz=10;

for ix=1:indblx:200-indblx+1
    for jy=1:indbly:300-indbly+1
        for kz=1:indblz:70-indblz+1

            ixfin=ix+indblx-1;
            jyfin=jy+indbly-1;
            kzfin=kz+indblz-1;

            i_ini=max(1,ix-extra_blx);
            i_fin=min(ix+extra_blx,200-indblx);
            j_ini=max(1,jy-extra_bly);
            j_fin=min(jy+extra_bly,300-indbly);
            z_ini=max(1,kz-extra_blz);
            z_fin=min(kz+extra_blz,70-indblz);

            im_target=double(im3d_target(ix:ixfin,jy:jyfin,kz:kzfin));
            im_minima=zeros(size(im_target));

            valmin=100000000;

            for paciente=1:muestras-10
                Vqimd=double(array_imagen(:,:,:,paciente));
                for i=i_ini: i_fin
                    for j=j_ini : j_fin
                        for k=z_ini: z_fin
                            seccion=Vqimd(i:i+indblx-1,j:j+indbly-1,k:k+indblz-
1);
                            err=sum(sum(sum(abs(im_target-seccion))));
                            if (err<valmin)
                                valmin=err;
                                pac_min=paciente;
                                im_minima=seccion;
                                i_mejor=i;
                                j_mejor=j;
                                k_mejor=k;
                            end
```

```
                        end
                    end
                end
            end
imagen_reconstruida(ix:ixfin,jy:jyfin,kz:kzfin)=im_minima;
pros_reconstruida(ix:ixfin,jy:jyfin,kz:kzfin)=database_p(i_mejor:i_mejor+indblx-
1,j_mejor:j_mejor+indbly-1,k_mejor:k_mejor+indblz-1,pac_min);
        end
    end
end

end
```

10. Anexo 2 – Método de validación

```
posicion_r=find(pros_reconstruida==1);
posicion_o=find(prostata_objetivo==1);
interseccion=intersect(posicion_r,posicion_o);
acierto=numel(interseccion)/numel(posicion_o);
```

11. Anexo 3 – Paralelización del bucle paciente

```
function [imagen_reconstruida,
pros_reconstruida]=reconstruccion_paralelo(array_imagen,info_prostata,Vq)

muestras=size(array_imagen,4);
im3d_target=Vq;

imagen_reconstruida=zeros(200,300,70);
pros_reconstruida=zeros(200,300,70);

indblx=40; indbly=30;indblz=35;
extra_blx=4;extra_bly=4;extra_blz=10;

for ix=1:indblx:200-indblx+1
    for jy=1:indbly:300-indbly+1
        for kz=1:indblz:70-indblz+1
            ixfin=ix+indblx-1;
            jyfin=jy+indbly-1;
            kzfin=kz+indblz-1;

            i_ini=max(1,ix-extra_blx);
            i_fin=min(ix+extra_blx,200-indblx);
            j_ini=max(1,jy-extra_bly);
            j_fin=min(jy+extra_bly,300-indbly);
            z_ini=max(1,kz-extra_blz);
            z_fin=min(kz+extra_blz,70-indblz);

            im_target=double(im3d_target(ix:ixfin,jy:jyfin,kz:kzfin));
            im_minima=zeros(size(im_target));
            prostata_minima=zeros(size(im_target));
            for i=2:muestras-10
                im_minima(:,:,:,i)=0;
            end

            valmin=zeros(muestras-10,1);
            valmin(:)=100000000;

            parfor paciente=1:muestras-10
                Vqims=double(array_imagen(:,:,:,paciente));
                for i=i_ini:i_fin
                    for j=j_ini:j_fin
                        for k=z_ini:z_fin
                            seccion=Vqims(i:i+indblx-1,j:j+indbly-1,k:k+indblz-
1);
                            err=sum(sum(sum(abs(im_target-seccion))));
                            if (err<valmin(paciente))
                                valmin(paciente)=err;
                                im_minima(:,:,:,paciente)=seccion;

```

```
                                Vprostata=double(info_prostata(:,:,:,paciente));
seccion_prostata=Vprostata(i:i+indblx-1,j:j+indbly-1,k:k+indblz-1);
prostata_minima(:,:,:,paciente)=seccion_prostata;
                            end
                        end
                    end
                end
            end
            minimo=min(valmin);
            paciente_min=find(valmin==minimo);

imagen_reconstruida(ix:ixfin,jy:jyfin,kz:kzfin)=im_minima(:,:,:,paciente_min);

pros_reconstruida(ix:ixfin,jy:jyfin,kz:kzfin)=prostata_minima(:,:,:,paciente_min
);
        end
    end
end

end
```

12. Anexo 4 – Reestructuración de bucles iniciales

```
function [imagen_reconstruida,
pros_reconstruida]=reconstruccion_exp(array_imagen,info_prostata,im3d_target)

muestras=size(array_imagen,4);

imagen_reconstruida=zeros(200,300,70);
pros_reconstruida=zeros(200,300,70);

segmento_imagen=zeros(40,30,35,10);
segmento_prostata=zeros(40,30,35,10);

indblx=40;indbly=30;indblz=35;
extra_blx=4;extra_bly=4;extra_blz=10;

for i_ori=1:200/indblx
   ix=1+(i_ori-1)*indblx;
   ixfin=ix+indblx-1;
   for k_ori=1:70/indblz
       kz=1+(k_ori-1)*indblz;
       kzfin=kz+indblz-1;
       for j_ori=1:300/indbly
            jy=1+(j_ori-1)*indbly;
            jyfin=jy+indbly-1;

            i_ini=max(1,ix-extra_blx);
            i_fin=min(ix+extra_blx,200-indblx);
            j_ini=max(1,jy-extra_bly);
            j_fin=min(jy+extra_bly,300-indbly);
            z_ini=max(1,kz-extra_blz);
            z_fin=min(kz+extra_blz,70-indblz);

            im_target=double(im3d_target(ix:ixfin,jy:jyfin,kz:kzfin));

            valmin=100000000;

            for paciente=1:muestras-10
                Vqimd=double(array_imagen(:,:,:,paciente));
                for i=i_ini: i_fin
                    for j=j_ini : j_fin
                        for k=z_ini: z_fin
                            seccion=Vqimd(i:i+indblx-1,j:j+indbly-1,k:k+indblz-
1);
                            err=sum(sum(sum(abs(im_target-seccion))));
                            if (err<valmin)
                                valmin=err;
                                pac_min=paciente;
                                im_minima=seccion;
                                i_mejor=i;
```

```
                                j_mejor=j;
                                k_mejor=k;
                            end
                        end
                    end
                end
            end
        imagen_reconstruida(ix:ixfin,jy:jyfin,kz:kzfin)=im_minima;
pros_reconstruida(ix:ixfin,jy:jyfin,kz:kzfin)=info_prostata(i_mejor:i_mejor+indb
lx-1,j_mejor:j_mejor+indbly-1,k_mejor:k_mejor+indblz-1,pac_min);

        end
    end
end

end
```

13. Anexo 5 – Paralelización del bucle *y* externo

```
function [imagen_reconstruida,
pros_reconstruida]=reconstruccion_exp_par(array_imagen,info_prostata,im3d
_target)

muestras=size(array_imagen,4); %10 últimas test

imagen_reconstruida=zeros(200,300,70);
pros_reconstruida=zeros(200,300,70);

segmento_imagen=zeros(40,30,35,10);
segmento_prostata=zeros(40,30,35,10);

indblx=40;indbly=30;indblz=35;
extra_blx=4;extra_bly=4;extra_blz=10;

for i_ori=1:200/indblx
   ix=1+(i_ori-1)*indblx;
   ixfin=ix+indblx-1;
   for k_ori=1:70/indblz
       kz=1+(k_ori-1)*indblz;
       kzfin=kz+indblz-1;
       parfor j_ori=1:300/indbly
            jy=1+(j_ori-1)*indbly;
            jyfin=jy+indbly-1;


            i_ini=max(1,ix-extra_blx);
            i_fin=min(ix+extra_blx,200-indblx);
            j_ini=max(1,jy-extra_bly);
            j_fin=min(jy+extra_bly,300-indbly);
            z_ini=max(1,kz-extra_blz);
            z_fin=min(kz+extra_blz,70-indblz);

            im_target=double(im3d_target(ix:ixfin,jy:jyfin,kz:kzfin));
            valmin=100000000;

            pac_min=1;
            i_mejor=1;
            j_mejor=1;
            k_mejor=1;

            for paciente=1:muestras-10
                Vqimd=double(array_imagen(:,:,:,paciente));
                for i=i_ini:i_fin
                    for j=j_ini:j_fin
                        for k=z_ini:z_fin
```

```
                                seccion=Vqimd(i:i+indblx-1,j:j+indbly-
1,k:k+indblz-1);
                                err=sum(sum(sum(abs(im_target-seccion))));
                                if (err<valmin)
                                    valmin=err;
                                    pac_min=paciente;
                                    i_mejor=i;
                                    j_mejor=j;
                                    k_mejor=k;
                                end
                            end
                        end
                    end
                end

segmento_imagen(:,:,:,j_ori)=array_imagen(i_mejor:i_mejor+indblx-
1,j_mejor:j_mejor+indbly-1,k_mejor:k_mejor+indblz-1,pac_min);

segmento_prostata(:,:,:,j_ori)=info_prostata(i_mejor:i_mejor+indblx-
1,j_mejor:j_mejor+indbly-1,k_mejor:k_mejor+indblz-1,pac_min);
        end
        for aux=1:size(segmento_imagen,4)
             ini_aux=1+(aux-1)*indbly;
             fin_aux=aux*indbly;

imagen_reconstruida(ix:ixfin,ini_aux:fin_aux,kz:kzfin)=segmento_imagen(:,
:,:,aux);

pros_reconstruida(ix:ixfin,ini_aux:fin_aux,kz:kzfin)=segmento_prostata(:,
:,:,aux);
        end
    end
end

end
```

14. Anexo 6 – Calculo de error en C

```
#include <math.h>
#include <matrix.h>
#include <mex.h>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    const mwSize *dims;
    const mwSize *im_dims;
    double *im_target, *seccion, *error; /*seccion (0) e im_target (1) son 3D*/
    int dimx, dimy, dimz, numdims;
    int im_dimx, im_dimy, im_dimz, im_numdims;
    int i,j,k;

    dims = mxGetDimensions(prhs[0]);
    numdims = mxGetNumberOfDimensions(prhs[0]);

    dimy = (int)dims[0]; dimx = (int)dims[1]; dimz = (int)dims[2];

    im_dims = mxGetDimensions(prhs[1]);
    im_numdims = mxGetNumberOfDimensions(prhs[1]);

    im_dimy = (int)im_dims[0]; im_dimx = (int)im_dims[1]; im_dimz =
(int)im_dims[2];

    plhs[0] = mxCreateDoubleMatrix(1,1,mxREAL); /*no tiene porque ser matriz*/
    error =mxGetPr(plhs[0]);

    seccion = mxGetPr(prhs[0]);
    im_target = mxGetPr(prhs[1]);

    error[0]=0.0;
    for(k=0;k<dimz;k++) /*cambia la profundidad*/
    {
        for(i=0;i<dimx;i++)
        {
            for(j=0;j<dimy;j++) /*va por columnas*/
            {
            error[0] += fabs(im_target[j+i*dimy+k*dimy*dimx]-
seccion[j+i*dimy+k*dimy*dimx]);
            }
        }
    }
    return;
}
```

15. Anexo 7 – Calculo del mejor segmento en C

```
#include <math.h>
#include <matrix.h>
#include <mex.h>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{

/*declare variables*/
    const mwSize *dims;
    const mwSize *im_dims;
    double *im_target, *seccion, error;
    double *lims, *limblocks;
    int dimx, dimy, dimz, numdims;
    int im_dimx, im_dimy, im_dimz, im_numdims;
    int i,j,k,i_ini,i_fin,j_ini,j_fin,z_ini,z_fin,indblx,indbly,indblz;
    int i_out,j_out,k_out;
    double *i_mejor,*j_mejor,*k_mejor;
    double *valmin;

    dims = mxGetDimensions(prhs[0]);
    dimx = (int)dims[0]; dimy = (int)dims[1]; dimz = (int)dims[2];

/*figure out dimensions*/
    seccion = mxGetPr(prhs[0]);
    im_target = mxGetPr(prhs[1]);
    im_dims = mxGetDimensions(prhs[1]);
    im_dimx = (int)im_dims[0]; im_dimy = (int)im_dims[1]; im_dimz =
(int)im_dims[2];

    lims=mxGetPr(prhs[2]);
    limblocks=mxGetPr(prhs[3]);
    i_ini=(int)lims[0];
    i_fin=(int)(lims[1]);
    j_ini=(int)(lims[2]);
    j_fin=(int)(lims[3]);
    z_ini=(int)(lims[4]);
    z_fin=(int)(lims[5]);
    indblx=(int)(limblocks[0]);
    indbly=(int)(limblocks[1]);
    indblz=(int)(limblocks[2]);
/*associate outputs*/
    /*mexPrintf("Asociamos salida... ");*/
    plhs[0] = mxCreateDoubleMatrix(1,1,mxREAL);
    valmin =mxGetPr(plhs[0]);
    plhs[1] = mxCreateDoubleMatrix(1,1,mxREAL);
    i_mejor =mxGetPr(plhs[1]);
    plhs[2] = mxCreateDoubleMatrix(1,1,mxREAL);
    j_mejor =mxGetPr(plhs[2]);
```

```
49      plhs[3] = mxCreateDoubleMatrix(1,1,mxREAL);
50      k_mejor =mxGetPr(plhs[3]);
51      i_mejor[0]=1;
52      j_mejor[0]=1;
53      k_mejor[0]=1;
54
55  /*do something*/
56      /*mexPrintf("Empezamos cálculo... ");*/
57      valmin[0]=100000000;
58
59      for (i_out=i_ini-1;i_out<i_fin;i_out++)
60      for (j_out=j_ini-1;j_out<j_fin;j_out++)
61      for (k_out=z_ini-1;k_out<z_fin;k_out++)
62      {
63               error=0.0;
64      for(k=0;k<im_dimz;k++){
65          for(j=0;j<im_dimy;j++){
66              for(i=0;i<im_dimx;i++){
67              error += fabs(im_target[i+j*im_dimx+k*im_dimy*im_dimx]-
68  seccion[i+i_out+(j+j_out)*dimx+(k+k_out)*dimy*dimx]);
69              }
70          }
71      }
72      if (error<valmin[0])
73      {valmin[0]=error;
74       i_mejor[0]=i_out+1;
75       j_mejor[0]=j_out+1;
76       k_mejor[0]=k_out+1;
77      }
78       }
79      /*mexPrintf("OK\n");*/
80      return;
81  }
```

16. Anexo 8 – Script Kempes (Segmentación)

```
%Script-SC
clear
clc
%%
P=parpool
%%
load('array_imagen.mat')
load('info_prostata.mat')
%%
for i=1:10
muestras=size(array_imagen,4) %10 últimas test
imagen_objetivo=array_imagen(:,:,:,muestras-i+1);
prostata_objetivo=info_prostata(:,:,:,muestras-i+1);
prostata_objetivo(:,:,end)=[]; %eliminamos la última seccion para dejar 70

eval(sprintf('save imagen_objetivo_%d imagen_objetivo', i));
eval(sprintf('save prostata_objetivo_%d prostata_objetivo', i));

%%

 disp('Secuencial')
 tic
 [imagen_reconstruida,
pros_reconstruida]=reconstruccion_secuencial(array_imagen,info_prostata,imagen_o
bjetivo);
 tiempo=toc

 eval(sprintf('save imagen_reconstruida_sec_%d imagen_reconstruida', i));
 eval(sprintf('save pros_reconstruida_sec_%d pros_reconstruida', i));
 eval(sprintf('save tiempo_sec_%d tiempo', i));

 %Basandome solo en próstata (1)
 posicion_r=find(pros_reconstruida==1);
 posicion_o=find(prostata_objetivo==1);

 interseccion=intersect(posicion_r,posicion_o);
 acierto=numel(interseccion)/numel(posicion_o)

 eval(sprintf('save acierto_sec_%d acierto', i));

%%

disp('EXP_Secuencial')
tic
```

```
[imagen_reconstruida,
pros_reconstruida]=reconstruccion_exp(array_imagen,info_prostata,imagen_objetivo
);
tiempo=toc

eval(sprintf('save imagen_reconstruida_exp_sec_%d imagen_reconstruida', i));
eval(sprintf('save pros_reconstruida_exp_sec_%d pros_reconstruida', i));
eval(sprintf('save tiempo_exp_sec_%d tiempo', i));

%Basandome solo en próstata (1)
posicion_r=find(pros_reconstruida==1);
posicion_o=find(prostata_objetivo==1);

interseccion=intersect(posicion_r,posicion_o);
acierto=numel(interseccion)/numel(posicion_o)

eval(sprintf('save acierto_exp_sec_%d acierto', i));

%%

 disp('Paralelo')
 tic
 [imagen_reconstruida,
pros_reconstruida]=reconstruccion_paralelo(array_imagen,info_prostata,imagen_obj
etivo);
 tiempo=toc

 eval(sprintf('save imagen_reconstruida_par_%d imagen_reconstruida', i));
 eval(sprintf('save pros_reconstruida_par_%d pros_reconstruida', i));
 eval(sprintf('save tiempo_par_%d tiempo', i));

 %Basandome solo en próstata (1)
 posicion_r=find(pros_reconstruida==1);
 posicion_o=find(prostata_objetivo==1);

 interseccion=intersect(posicion_r,posicion_o);
 acierto=numel(interseccion)/numel(posicion_o)

 eval(sprintf('save acierto_par_%d acierto', i));

%%

disp('EXP Paralelo')
tic
[imagen_reconstruida,
pros_reconstruida]=reconstruccion_exp_par(array_imagen,info_prostata,imagen_obje
tivo);
tiempo=toc

eval(sprintf('save imagen_reconstruida_exp_par_%d imagen_reconstruida', i));
eval(sprintf('save pros_reconstruida_exp_par_%d pros_reconstruida', i));
eval(sprintf('save tiempo_exp_par_%d tiempo', i));
```

```
97
98  save imagen_reconstruida_exp_par imagen_reconstruida
99  save pros_reconstruida_exp_par pros_reconstruida
100
101 Basandome solo en próstata (1)
102 posicion_r=find(pros_reconstruida==1);
103 posicion_o=find(prostata_objetivo==1);
104
105 interseccion=intersect(posicion_r,posicion_o);
106 acierto=numel(interseccion)/numel(posicion_o)
107
108 eval(sprintf('save acierto_exp_par_%d acierto', i));
109
110 %%
111 disp('PAR MEX')
112 tic
113 [imagen_reconstruida,
114 pros_reconstruida]=reconstruccion_par_mex(array_imagen,info_prostata,imagen_obje
115 tivo);
116 tiempo=toc
117
118 eval(sprintf('save imagen_reconstruida_par_mex_%d imagen_reconstruida', i));
119 eval(sprintf('save pros_reconstruida_par_mex_%d pros_reconstruida', i));
120 eval(sprintf('save tiempo_par_mex_%d tiempo', i));
121
122 %Basandome solo en próstata (1)
123 posicion_r=find(pros_reconstruida==1);
124 posicion_o=find(prostata_objetivo==1);
125
126 interseccion=intersect(posicion_r,posicion_o);
127 acierto=numel(interseccion)/numel(posicion_o)
128
129 eval(sprintf('save acierto_par_mex_%d acierto', i));
130
131 %%
132
133  disp('SEC y MEX')
134  tic
135  [imagen_reconstruida,
136 pros_reconstruida]=reconstruccion_sec_mex(array_imagen,info_prostata,imagen_obje
137 tivo);
138  tiempo=toc
139
140  eval(sprintf('save imagen_reconstruida_sec_mex_%d imagen_reconstruida', i));
141  eval(sprintf('save pros_reconstruida_sec_mex_%d pros_reconstruida', i));
142  eval(sprintf('save tiempo_sec_mex_%d tiempo', i));
143
144  %Basandome solo en próstata (1)
145  posicion_r=find(pros_reconstruida==1);
146  posicion_o=find(prostata_objetivo==1);
147
148  interseccion=intersect(posicion_r,posicion_o);
```

```
149     acierto=numel(interseccion)/numel(posicion_o)
150
151     eval(sprintf('save acierto_sec_mex_%d acierto', i));
152   %%
153   end
154   %%
155   delete(P)
```

17. Anexo 9 – Script Kempes (Red Neuronal - Train)

```
clc
clear

nm=50000;                %muestras
NC=80;                   % 80 train y 17 test

M=[5];                  %num_redes

HL=[];                   %capas ocultas
for i=1:100
HL=[HL,[i]];
end

ps1=3;                   % patch1 size
ps2=5;                   % patch2 size
ps3=7;                   % patch3 size
ps4=9;                   % patch4 size
ps5=11;                  % patch5 size

funcion_train='trainscg';           %funcion de entrenamento
funcion_train_ocultas='trainscg';   %funcion de entrenamento de las capes
ocultas
epoch=1000;                         %epoch

prefijo='test_num_HL_';

%Load data
load('array_imagenes.mat'); %array_imagen
load('info_prostatas.mat'); %info_prostata (0 fondo - 1 prostata)
%load lista_clases

disp('¡Empieza el script!')

for n=67:2:length(HL)
    for i=1:2 %para promediar

script_train(nm,NC,M,HL(n),ps1,ps2,ps3,ps4,ps5,funcion_train,funcion_ocult
as,epoch,strcat(prefijo,num2str(HL(n)),'_',num2str(i)),array_imagen,info_prostat
a)
    end
end

disp('¡Script Finalizado!')
```

18. Anexo 10 – Script Kempes (Red Neuronal - Test)

```
clc
clear

nm=50000;                %muestras
NC=80;                   % 80 train y 17 test

M=[5];                   %num_redes

HL=[];                   %capas ocultas
for i=1:100
HL=[HL,[i]];
end

ps1=3;                   % patch1 size
ps2=5;                   % patch2 size
ps3=7;                   % patch3 size
ps4=9;                   % patch4 size
ps5=11;                  % patch5 size

funcion_train='trainscg';           %funcion de entrenamento
funcion_train_ocultas='trainscg';   %funcion de entrenamento de las capes
ocultas
epoch=1000;                         %epoch

prefijo='test_num_HL_';

%Load data
load('array_imagenes.mat'); %array_imagen
load('info_prostatas.mat'); %info_prostata (0 fondo - 1 prostata)
load('lista_clases.mat');

disp('¡Empieza el script!')

for n=1:2:length(HL)
    for i=1:2 %para promediar

script_test(NC,ps1,ps2,ps3,ps4,ps5,strcat(prefijo,num2str(HL(n)),'_',num2str(i))
,array_imagen,info_prostata,lista_clases)
    end
end

disp('¡Script Finalizado!')
```

19. Anexo 11 – Script Visualización de resultados

```
array_prefijos={'sec','par','exp_sec','exp_par','sec_mex','par_mex','exp_sec_mex
','exp_par_mex'};
nom_leyenda={'Secuencial (original)','Paralelo (paciente)','Secuencial (bucle
cambiado)','Paralelo (bucle externo)','Secuencial (mex)','Paralelo
(mex)','Secuencial (mex mejorado)','Paralelo (mex mejorado)'};
nom_leyenda_acierto={'Programa original','Corregido'};

array_dif=zeros(1,10);
array_dif(:,:,2)=zeros(1,10);
for i=1:8
%acceder a la carpeta
prefijo=array_prefijos(i);
leyenda=nom_leyenda(i);
%leyenda_acierto=nom_leyenda_acierto(i);

cd(prefijo{1})

array_acierto=zeros(1,10);
array_tiempo=zeros(1,10);

%cargar datos
for j=1:10
    nom_acierto=strjoin(strcat('acierto_',prefijo,'_',num2str(j),'.mat'));
    nom_tiempo=strjoin(strcat('tiempo_',prefijo,'_',num2str(j),'.mat'));

    load(nom_acierto);
    load(nom_tiempo);
    array_acierto(j)=acierto;
    array_tiempo(j)=tiempo;
end

% array_dif(:,:,i)=array_acierto

% if(i==1)
%    array_acierto
%    array_dif
% end
%
% if(i==2)
%     array_dif
%     res=array_dif(:,:,2)-array_dif(:,:,1)
% end

ind_pacientes=[147:156];

```

```
% plot(array_acierto*100,'DisplayName',leyenda_acierto{1}); title('Acierto');
xlabel('Índice del paciente'); ylabel('% de acierto');
set(gca,'XTickLabel',ind_pacientes);
% legend('show');
% hold on

plot(array_tiempo,'DisplayName',leyenda{1}); title('Tiempo'); xlabel('Índice del
paciente'); ylabel('Tiempo de ejecución en segundos');
set(gca,'XTickLabel',ind_pacientes);
legend('show');
hold on

[min_acierto,ind]=min(array_acierto)
[max_acierto,ind]=max(array_acierto)
[mean_acierto]=mean(array_acierto)
[median_acierto]=median(array_acierto)

[min_tiempo,ind]=min(array_tiempo)
[max_tiempo,ind]=max(array_tiempo)
[mean_tiempo]=mean(array_tiempo)
[median_tiempo]=median(array_tiempo)

array_acierto
array_tiempo

cd ..
%clear
end
```