Automatización del análisis de imágenes para experimentos de ensayos de herida.

Sebastián Andrés Urrejola Barrios

Proyecto de Grado entregado a la Facultad de Ingeniería para optar al grado académico de Magíster en Data Science

Profesora guía: Ph.D Daniela Opitz (Facultad de Ingeniería, UDD)

Profesora co-guía: Ph.D Lorena Lobos (Facultad de Medicina, UDD)

Facultad de Ingeniería
Universidad del Desarrollo
Chile

04 de enero de 2022

Índice

Agradecimientos

Queremos agradecer al Team Mama - Ovario, equipo de investigación del centro de medicina regenerativa de la Universidad del Desarrollo, y en particular a Lorena Lobos por darnos la oportunidad de hacer este trabajo.

Resumen

Un ensayo que aborda el comportamiento tumoral de una célula muy establecido en las ciencias biológicas es el ensayo de migración celular. De este tipo de ensayo uno de los más comunes y simples es el ensayo de migraciones en dos dimensiones, donde lo que se hace es evaluar en el plano eje X como un área no ocupada de células, puede ser repoblada por células que se movilizan. A partir de estos experimentos se obtienen fotografías desde donde se analizan las propiedades migratorias de grupos colectivos de células. Hacer estos análisis de forma manual es una tarea que consume considerables cantidades de tiempo para las personas haciendo estas investigaciones y genera un resultado sujeto al ejecutor. En este trabajo presentamos dos métodos distintos para la automatización de este proceso. El primer método se basa en detección de contornos para así identificar la región libre de células, mientras que el segundo método se basa en detección de texturas pasa clasificar los píxeles de las imágenes. Hasta el momento ejecutando el segundo método se logra procesar imágenes en aproximadamente un 80 % del tiempo que le toma a una persona. En el caso particular del método de detección de texturas, mientras los grupos de imágenes sean similares entre sí en términos del brillo y las gamas de colores que tengan, los resultados de las mediciones de áreas de las heridas serán efectivos y replicables, en donde esta última característica es un aspecto esencial en el proceso de hacer ciencia a nivel mundial.

1. Introducción

1.1. Contexto Biológico

Las células son las unidades biológicas esenciales de la vida, cada tipo de células de un organismo cumple con su función de acuerdo a su estructura molecular y a su comportamiento. En ese contexto hay células que tienen como misión ser reparadoras del tejido en caso de daños como son los fibroblastos: células que proliferan y migran para cerrar heridas de la piel y/o de otras lesiones como aneurisma. Otras células como los macrófagos también migran a tejidos que están en un contexto inflamatorio. Así hay muchas ejemplos fisiológicos donde la migración celular es una función y acción reparatoria.

La migración celular también es la podemos hallar en procesos patológicos. Cuando las células migran para regenerar o reparar ciertos tejido, hay un gran control de señalización para que esta migración sea adecuada para reparar el daño, y se frenan para no generar fibrosis. Sin embargo, hay células normales que adquieren características atípicas que le dan a la célula potenciales que no son controlados, como la hiperproliferación, la inmortalización, evasión del sistema inmune y migración e invasión descontrolada desencadenando invasiones (repoblación) en un tejido sano.

Esto ha sido descrito como cáncer. Así una célula tumoral tiene la capacidad migratoria desatada. En la investigación a la fecha de esta enfermedad esta es una capacidad comúnmente evaluada en modelos *in vitro* especialmente en cultivos celulares, donde se observa como las células se movilizan y ocupan espacios [1].

1.2. Ensayos de Herida

Siendo experimentos sencillos y accesibles, ya que, no se requieren de equipamiento altamente especializado para ejecutarse, los ensayos de herida son un enfrentamiento común para estudiar las capacidades migratorias de grupos colectivos de células, en particular en estudios sobre cáncer, ya que, la migración celular es fundamental en etapas como la metástasis y la invasión a tejidos sanos [12, 8]. En estos experimentos se cultivan células en placas de cultivo hasta que las células formen una mono capa confluente, es decir, que las células ocupan todo el espacio disponible de la placa. Luego se hace una herida sobre esta mono capa bajo condiciones determinadas, comúnmente se utiliza la punta de una pipeta esterilizada para generar el espacio libre de células. En la figura 1 mostramos un esquema del procedimiento de este experimento.

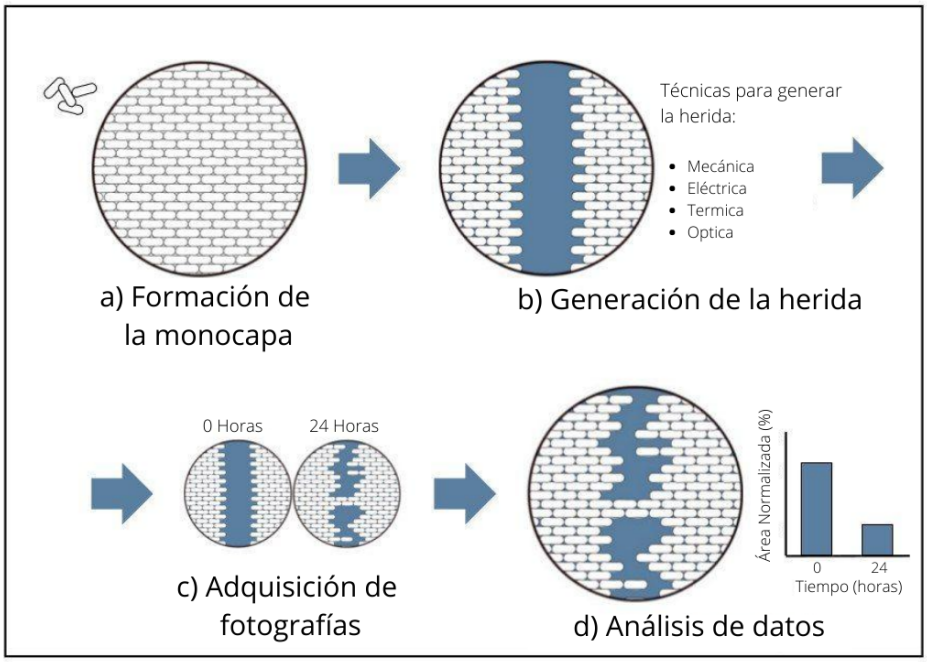


Figura 1: "Migración 2D: Ensayos de Herida." Esquema modificado de Regnard et al[4] en donde se muestran las etapas de un ensayo de herida. a) Primero se cultivan células hasta que forme una monocapa confluente. b) Se procede a generar la herida formando un espacio libre de células. c) Documentación de los estados del cultivo celular a través de fotografías. d) Cuantificación del área de las heridas en las imágenes.

Una vez que se ha generado la herida, se procede a documentar la herida a través de fotografías que se obtengan con equipo de microscopía como un microscopio óptico o uno de contraste de fases y finalmente, una vez que se hayan conseguido todas las imágenes se hace el análisis del cambio del área de la herida en el tiempo.

Como mencionamos anteriormente, distintos tipos de células tienen la capacidad de migrar de forma colectiva a distintas locaciones. Además en el contexto del laboratorio se puede alterar por ejemplo la expresión de genes y/o el funcionamiento de proteínas, generando así distintas condiciones en donde se promuevan o inhiban comportamientos de las células. Otro aspecto importante del contexto del laboratorio es que se pueden generar o estudiar lineas celulares. Si bien no todas las células tienen funciones para replicarse de forma indefinida, a través de mutaciones, dichas células pueden llegar a tener la capacidad de mantener la división celular de forma sostenida en el tiempo. Así estas mutaciones permite que se puedan inmortalizar tipos específicos de células que luego se pueden cultivar *in vitro* y así hacer investigación replicable. A estos tipos de células inmortalizadas se les llama linea celular. Puesto a que son sumamente útiles para la investigación en áreas como la biología celular o bioquímica se han generado múltiples lineas celulares de distintos tipos de células que se pueden conseguir de forma comercial, como por ejemplo, se pueden obtener lineas celulares de células sanas de ovario, o de células tumorales de ovario. Así, se pueden tener distintas lineas celulares

(linea tumoral y linea sana) del mismo tipo de células (células de ovario) lo que permite estudios más completos y complejos en relación a dichas células [1].

Con estos ensayos se pueden estudiar las influencias de distintos fármacos, alteraciones genéticas, estímulos, etc, con la finalidad de comparar y conocer los comportamientos de la migración celular de las distintas condiciones que se están investigando [8]. Por ejemplo, podríamos investigar como es que dos lineas celulares distintas se ven afectadas por dos fármacos distintos, lo que esquematizamos de forma simplificada en la tabla 1.

Placa	**Linea Celular**	**Fármaco**
1	A	fármaco$_1$
2	A	fármaco$_2$
3	B	fármaco$_1$
4	B	fármaco$_2$

Tabla 1: "Condiciones a Estudiar." En este ejemplo en cada fila tenemos distintas condiciones desde donde se obtendrían fotografías para procesar.

En las filas de la tabla 1 tenemos cuatro distintas condiciones que se están evaluando. En este escenario, la documentación de la de migración celular se podría hacer en un periodo de 24 horas documentando las heridas cada 6 horas y luego, una vez que se tengan las fotografías se procedería a cuantificar las áreas de las heridas en cada imagen. Para la primera condición de la tabla 1 (Linea Celular A - Fármaco$_1$) se podría ejemplificar el resultado de la obtención de las fotografías y el resultado del análisis de las áreas posterior con el cuadro 2.

Condición	**Tiempo**	**Medición**
A - fármaco$_1$	tiempo 0h	Área$_1$
A - fármaco$_1$	tiempo 6h	Área$_2$
A - fármaco$_1$	tiempo 18h	Área$_3$
A - fármaco$_1$	tiempo 24h	Área$_4$

Tabla 2: "Imágenes de una Condición." En este ejemplo en cada fila tenemos la información que se obtendría al procesar las imágenes de la condición de la linea celular A con el Fármaco$_1$.

Como se aprecia en el cuadro 2, la herida es documentada a lo largo de un intervalo de tiempo. Luego al procesar las fotografías se obtienen las medidas del área de las heridas y finalmente se repite este procedimiento con cada condición que se este evaluando.

La adquisición de imágenes se puede hacer de forma manual o automática dependiendo del equipo de laboratorio al que se tenga acceso [12]. La adquisición manual de las fotografías el procedimiento es llevar la placa con los cultivos de células al microscopio y así obtener las imágenes. En el caso automático hay dos enfrentamientos posibles. En primer lugar se puede usar un microscopio capaz de funcionar dentro de una incubadora sin perturbar a los cultivos celulares. Por otro lado, se pueden

usar incubadoras portátiles en donde se dejan las placas con las células y luego esta incubadora es colocada en el microscopio. Estas formas automáticas de adquisición de imágenes tienen importantes beneficios en comparación a la forma manual, puesto a que, al hacer la documentación manual es probable que no coincidan los puntos de vista en las imágenes de tiempos consecutivos de la misma placa, lo que se traduce en una fuente de ruido a la hora de hacer el análisis de las áreas de las heridas.

Otro aspecto que se puede automatizar de este experimento es la forma en la que se procesan las imágenes. Analizar las imágenes de forma manual es un proceso que consume una considerable cantidad de tiempo, ya que, normalmente se estudian distintas condiciones y se obtienen múltiples imágenes por condición [5]. Comúnmente el procedimiento para hacer el análisis de forma manual es trazar el contorno de la herida y medir su área, en cada imagen a través de algún software de procesamiento de imágenes (como *ImageJ*), por lo tanto, los resultados de este procedimiento están ligados a la persona que los obtenga, debido a la dificultad que implica replicar exactamente los mismos contornos si se propone analizar la misma imagen dos veces. Este es el enfoque que le daremos a nuestro trabajo con el objetivo de generar un algoritmo abierto para la automatización del análisis de las imágenes que se obtengan en los ensayos de herida.

1.3. Revisión Bibliográfica

Aunque en los últimos años se haya visto una tendencia al aumento en el uso de los ensayos de herida como herramienta para evaluar propiedades migratorias de células en distintos escenarios, en muchos casos aun podemos encontrar que el procesamiento se ejecuta de forma semi automática o totalmente manual [7]. Como mencionamos anteriormente distintos trabajos se han hecho para automatizar aspectos de este experimento. En esta sección mencionaremos trabajos que se han desarrollado en los últimos años enfocándonos en particular en literatura asociada a la automatización del análisis y procesamiento de las imágenes.

CellProfiler es uno de los primeros avances en el área de automatización del procesamiento de imágenes en experimentos biológicos [9]. Este software fue el primero de su clase, siendo de uso libre y *open-source*, capaz de analizar imágenes de células en donde se pueden ejecutar distintos análisis dependiendo del las imágenes y experimentos con los que se trabaje (por ejemplo, conteos de células, mediciones del tamaño de las células, análisis de la forma de las células u organelos). Con CellProfiler se pueden analizar las imágenes que se obtienen a partir de múltiples experimentos y entre ellos están los experimentos de ensayos de herida en donde se detectan las heridas basados la diferencia del brillo de los píxeles en las regiones de herida (región libre de células) y las regiones con células. Por otro lado, otro de los trabajos pioneros en esta área, es un software que llamaron TScratch [6]. En este caso, el programa es especializado para el análisis de las imágenes que se obtienen en los experimentos de ensayos de herida y se basa en transformadas discretas de curvelets, así se extraen información de los gradientes de los cambios de intensidad de los píxeles y luego se binariza la imagen comparando los valores de los píxeles a un threshold seleccionado de forma automática.

Aunque ambos modelos recién mencionados fueron pioneros en esta área, los múltiples parámetros que se deben manejar y la sensibilidad al tipo y calidad de imágenes que se trabajen son problemas [14, 9] que han llevado a más investigación e innovación en búsqueda de modelos más robustos y simples de manejar.

Así nos encontramos con MultiCellSeg [14], algoritmo especializado para la segmentación de imágenes de ensayos de herida. En este caso se usan maquinas de vectores de soporte (SVM por sus siglas en ingles) para clasificar los píxeles de la imagen como píxeles que corresponden a regiones con células o regiones de la herida. Primero se debe pasar por una fase de entrenamiento de las SVM's en donde se selecciona una imagen y se divide en distintos parches que el usuario debe etiquetar de forma manual como regiones con células o libre de células. Una vez finalizada la fase de entrenamiento se usa este modelo entrenado para segmentar el resto de las imágenes. Notablemente con este algoritmo no se deben ajustar parámetros pero el proceso de entrenar el modelo repetidas veces con las imágenes de las distintas condiciones que se estén estudiando es una tarea que consume tiempo.

Otro trabajo reciente y creado en Python es PyScratch [5]. En este proyecto las imágenes primero se pasan a escala de grises y se les aplican un filtro gaussiano y un filtro laplaciano para disminuir el ruido y acentuar los bordes de las estructuras de la imagen respectivamente. Luego se comparan los valores de los píxeles de la imagen a un threshold para binarizar la imagen. Finalmente se detectan los contornos en la imagen binaria y se guarda el valor del área del contorno más grande. Todas estas funciones son implementadas desde la conocida librería de visión computación OpenCV y el software se puede usar desde una interfaz gráfica para el usuario, facilitando el uso de este algoritmo para personas que no sean programadores con experiencia.

Comúnmente estos métodos llegan a un punto en donde se obtienen una imagen binaria a partir de la comparación de los valores de los píxeles a un threshold. Un enfrentamiento distintos a este lo podemos encontrar en [10] en donde la segmentación se lleva a cabo a partir del modelo LBL de detección de contornos [11] y una serie de filtros aplicados en paralelo. Por un lado los filtros se usan para distinguir grupos grandes de células, mientras que por el otro lado se aplican los filtros para distinguir grupos pequeños de células, para finalmente unir los resultados de ambos caminos paralelos en una imagen final donde se han segmentado las regiones con células y las regiones sin células, obteniendo así una imagen binaria desde donde se obtiene la información del comportamiento migratorio de las células.

Hasta el momento nos hemos enfocado en trabajos que hayan desarrollado software *stand-alone* o implementaciones de codigos en Python o MatLab, existen varios trabajos generados en torno al software de procesamiento de imágenes *ImageJ* que es comúnmente utilizado para efectuar el análisis de las imágenes de estos experimentos. Por lo tanto a continuación mencionaremos tres de estos trabajos en donde se desarrollaron *plug ins* para *ImageJ*.

El primer *plug in* que destacaremos fue llamado MiToBo [7]. En este trabajo los autores generaron un modelo a partir del método numérico de conjuntos de nivel y el modelo de energía de una imagen de Chan-Vese [3]. Primero las imágenes son pasadas a escala de grises y se les aplica un filtro gaussiano. Luego se consigue una imagen entrópica a partir de la entropía local de cada píxel de la imagen. Una vez que se tiene esta imagen entrópica, se segmentan las imágenes en regiones con células y regiones de herida. El siguiente paso es una etapa de post-procesamiento en donde se usan métodos de cierre y rellenado de los espacios vacíos. Con este enfrentamiento se impone la existencia de ambas clases de píxeles (células y herida) en la imagen final, lo que es un problema en las situaciones en que la herida cerró por completo a lo largo del experimento, ya que, el modelo clasifica de forma arbitraria a los píxeles en ambas clases para estos casos. Para solucionar este problema, antes de obtener y guardar los datos de las áreas asociadas a cada tipo de región segmentada, se pasa la imagen por una maquina de vectores de soporte para clasificarla como una imagen con herida o como una imagen sin herida, logrando así un modelo en donde solo dos parámetros se deben ajustar para lograr hacer

la segmentación. En particular, los parámetros que se pueden variar son la desviación estándar en el filtro gaussiano y el tamaño de la ventana para el calculo de la entropía local.

Continuando, un *plug-in* de *ImageJ* que no fue diseñado para trabajar exclusivamente en imágenes de ensayos de herida pero que de todas formas se puede usar para este propósito, es el *plug-in Trainable Weka Segmentation* o TWS descrito en [2]. En este trabajo los investigadores presentan un *plug-in* en donde el usuario demarca regiones de interés y selecciona uno o más métodos para la extracción de características y luego se procede a entrenar un modelo de machine learning, también seleccionado por el usuario, para segmentar las regiones demarcadas. Este *plug-in* se basa en las herramientas disponibles en la plataforma *Weka* para aprendizaje automático de la universidad de Waikato (Nueva Zelanda) que esta implementada en Java. El computo se ejecuta localmente en el computador que se este usando para trabajar, por lo que dependiendo del tamaño de las imágenes y la cantidad de estas, ejecutar la segmentación puede consumir una considerable cantidad de memoria y tiempo.

Finalmente el último modelo que destacaremos es el trabajo hecho en [13] donde los investigadores generaron otro *plug-in* para *ImageJ*. En este caso para cuantificar el cambio del área en los ensayos de herida, primero pasan las imágenes a escala de grises y se les aumenta el contraste para que así la varianza de las intensidades de los píxeles también aumente. La varianza en la intensidad de los píxeles de las regiones de heridas normalmente es menor a la varianza en las regiones de células, por lo que al aumentar el contraste esta diferencia se hace aun más notable, lo facilita y mejora el resultado de la binrazicación a través de la comparación de cada píxel a un threshold. Comúnmente en estos experimentos quedan células individuales o grupos de células en la región de la herida, inclusive restos celulares arrastrados en la placa. Para que esto no sea una problema en la cuantificación del área de las heridas, los investigadores emplean un método de reconstrucción morfológica por erosión, también conocido como *hole filling*. Aplicando este método en la región de la herida en la imagen binarizada logran remover pequeños grupos de células que aportarían con ruido a la medición del área. Además, pueden haber espacios sin células en las regiones de la imagen que correspondan a la capa de células, por lo que al hacer la segmentación, el algoritmo podría retornar más de una región etiquetada como la región de la herida. Para evitar este problema, el algoritmo discrimina que región etiquetar como la herida manteniendo solo la región más grande.

Así hemos visto que la mayoría de los trabajos tienen un enfrentamiento común, en donde las imágenes se pasan a escala de grises, luego se aplican filtros para después obtener una imagen binaria a partir de la comparación de cada píxel de la imagen a algún threshold. También vimos que en algunos casos se implementan métodos de detección de contornos para refinar las mediciones finales del área de la herida. Con todo esto en mente nosotros proponemos dos algoritmos, en el primero nos basamos en detección de contornos y binarización de la imagen a través del método de thresholds, y en el segundo implementamos un modelo de *Random Forest* para clasificar los píxeles de la imagen entre píxeles de región con células o de regiones libres de células (la herida), para entrenar el clasificador el usuario tendrá que usar una imagen de referencia en donde indique regiones con los píxeles de ambas clases que se quieren segmentar.

2. Datos y Metodología

2.1. Descripción de los Datos

Este trabajo será realizado en conjunto con un equipo de investigación del Centro de Medicina Regenerativa (*CMR*) de la Universidad Del Desarrollo (UDD) en donde estudian los cánceres de mama y ovario. Uno de los experimentos que hacen en el *CMR* son los ensayos de herida, en donde han trabajado con lineas celulares de cáncer de mama y ovario. En particular, en el *CMR* las heridas fueron fotografiadas de forma manual usando un microscopio óptico en aumento de x4.

Desde el momento en que comenzó este trabajo, el laboratorio ha generado un total de 1696 imágenes en tres sesiones de experimentos distintas. En sesiones distintas, ya que, desde el punto de vista del laboratorio, ha sido un proceso de estandarización para la obtención de imágenes para utilizar con nuestro algoritmo. Así, la calidad de las imágenes fue mejorando a medida que repitieron el experimento, por lo que las imágenes de la tercera sesión de experimentos son las imágenes de mejor calidad, mientras que las del primer set son las de peor calidad. Esto nos permitirá trabajar con distintos escenarios para así generar un algoritmo robusto que sea capaz de funcionar a partir de distintas formas de obtener las imágenes Todas las imágenes se guardaron en formato *.TIF*, en tres bandas de colores (RGB) con dimensiones de 2592 píxeles de ancho y 1936 píxeles de alto.

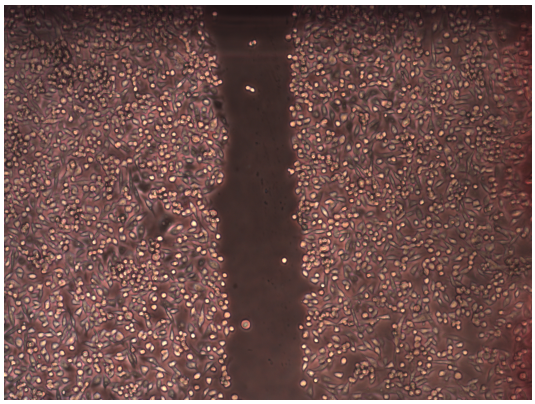


Figura 2: "Imagen tiempo 0h" Fotografía de la mono capa celular inmediatamente después de la generación de la herida.

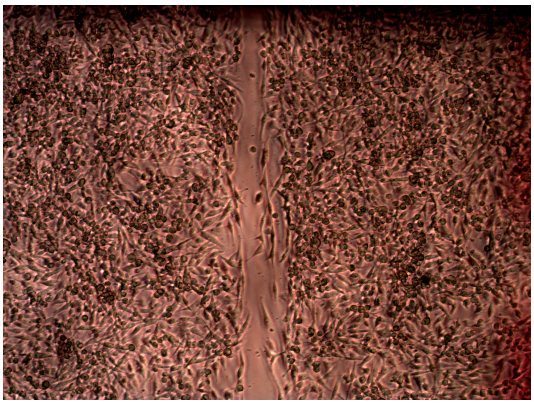


Figura 3: "Imagen tiempo 24h" Fotografía de la mono capa celular pasadas 20 horas desde que se efectuó la herida.

En las figuras 2 y 3 se presentan dos ejemplos de las imágenes que se obtienen en estos experimentos. La figura 2 es de una fotografía obtenida inmediatamente después de que se hizo la herida, mientras que la figura 3 es una imagen de la misma condición pero 20 horas después de que se hizo la herida. En especifico estas imágenes son de la misma placa, en donde se cultivaron células de la linea celular de cáncer de mama metastásico "MDA - MB - 231". Estas células fueron estimuladas con "Antilacta + Exo" que hace referencia a que a la placa con las células, una vez que se generó la herida se agregan anticuerpos anti lactaderina y exosomas, o microvesículas, que son secretados por las células metastásicas, estos fueron proporcionados por el grupo de investigación.

2.2. Procesamiento de las imágenes

A través de distintos módulos de Python, generamos dos modelos para hacer el análisis de las imágenes. El primer método consiste generar máscaras a partir de las imágenes de los tiempos 0h y luego binarizar el resto de las imágenes para finalmente usar las máscaras sobre las imágenes binarizadas y así obtener una medida normalizada del cambio del área de las heridas. Por otro lado, el segundo método consiste clasificar cada píxel de las imágenes dentro de dos clases de texturas o tipos de regiones que previamente fueron definidas por el usuario.

Ambos métodos pueden ser divididos en dos etapas, en donde la primera consiste en obtener información necesaria para poder procesar todas las imágenes en la segunda etapa de cada algoritmo. En la figura 4 presentamos un diagrama de flujo de cada etapa de ambos algoritmos.

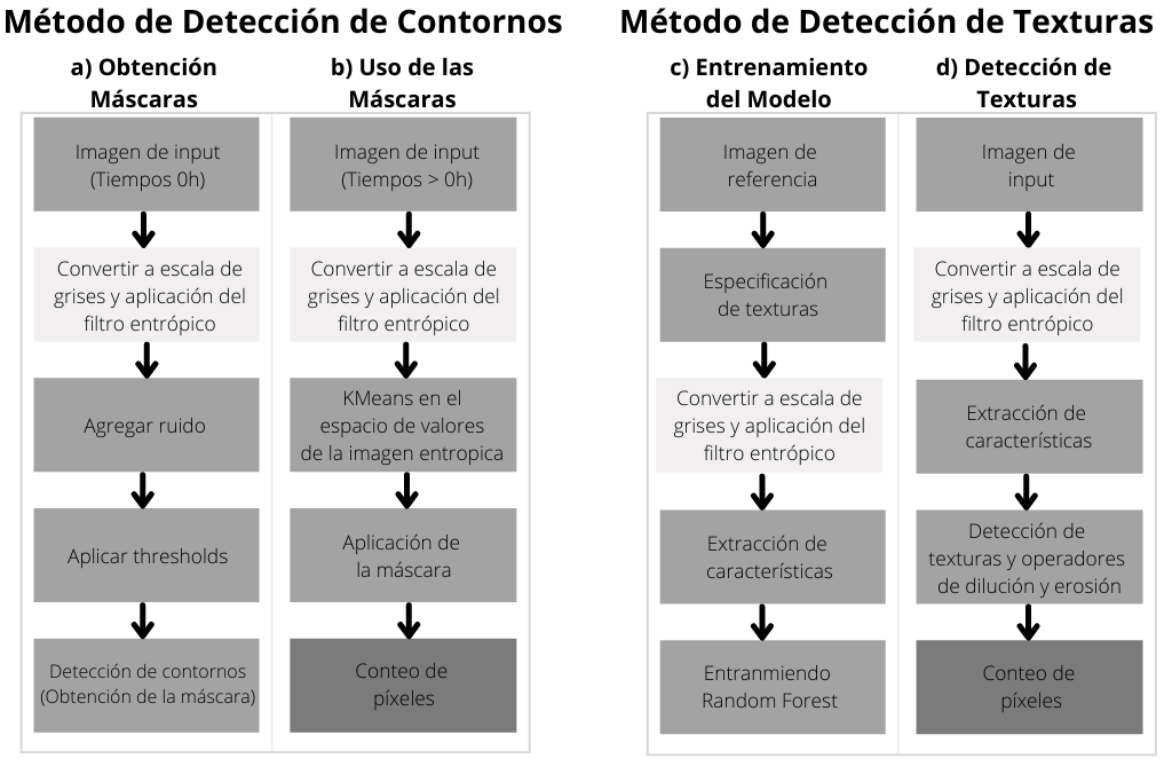


Figura 4: "Diagrama de Flujo de los Métodos Propuestos para el Análisis de Imágenes Automáticos." a) Primera etapa del método de detección de contornos en donde se obtienen máscaras de las heridas a partir de las imágenes del tiempo 0h. b) Segunda etapa del método de detección de contornos que consiste segmentar las imágenes de tiempos superiores a través de *KMeans* y aplicarles las máscaras de las heridas. c) Primera etapa del método de detección de texturas donde se definen las regiones con las texturas de interés y se entrena el modelo de *Random Forest*. d) Segunda etapa del método de detección de texturas donde se procede a segmentar el resto de las imágenes usando el modelo entrenado en la etapa anterior.

Como se aprecia en la figura 4, repetimos en cada algoritmo los bloques de procesamiento de a convertir las imágenes a escala de grises y luego aplicar un filtro entrópico. Estos bloques son fundamentales para el procesamiento de las imágenes ya que nos permiten llevar a las imágenes a escalas similares de valores. Al aplicar el filtro entrópico a las imágenes en escala de grises, se le asigna a cada píxel la medida de la entropía local dada una ventana de observación centrada en dicho píxel, en otras palabras, para cada píxel_i se considera un vecindario W_i. Al ser un espacio finito, se tiene N_i valores únicos x_j de píxeles dentro del vecindario W_i, así, podemos calcular la portabilidad $P(x_j)$ de hallar el valor x_j en W_i dividiendo las frecuencias de cada x_j por el tamaño del vecindario. Una vez que se obtienen estas probabilidades se calcula la entropía 1 y se le asigna este resultado al píxel_i. Finalmente se repite este proceso con todos los píxeles de la imagen.

$$H(W_i) = -\sum_{j=1}^{N_i} P(x_j) * \log_2 P(x_j) \tag{1}$$

2.2.1. Método de Detección de Contornos

En este método primero detectamos los contornos de las heridas y luego ejecutamos la segmentación de las imágenes, proceso esquematizado respectivamente en "a)" y "b)" de la figura 4. En la primera etapa del algoritmo se seleccionan las imágenes de tiempos 0h y luego se convierten a escala de girses y se les aplica el filtro entrópico. Luego se agrega ruido a la imagen para poder aproximar de forma robusta el contorno de la herida. Continuando, se calculan los promedios de los *thresholds* de Li y de la media para después binarizar la imagen en dos colores haciendo una comparación entre cada píxel y el promedio de los *thresholds*. Finalmente con la imagen binarizada procedemos a detectar los contornos de las heridas con los que generaremos máscaras que aplicaremos en las imágenes de tiempos superiores a las 0 horas.

En la segunda etapa del algoritmo primero se selecciona el resto de las imágenes que no se usaron en la primera etapa, es decir, todas las imágenes de tiempos mayores al de cero horas y luego estas imágenes son convertidas a escala de grises y se les aplica el filtro entrópico. Una vez que se ya tenemos las imágenes entrópicas, procedemos clusterizar en dos grupos el espacio de valores que tienen estas imágenes usando *KMeans*. Finalmente agregamos las máscaras que obtuvimos de las imágenes de los tiempos 0h y contamos los píxeles de cada clase que quedaron dentro de la máscara.

2.2.2. Método de Detección de Texturas

En las imágenes de ensayos de herida podemos considerar dos tipos de texturas de interés, estas corresponden a las regiones que tienen células y a las regiones libres de células, es decir, regiones que son de la herida. Así se podría entrenar algún modelo de machine learning que reciba una imagen y que nos retorne la imagen segmentada según las regiones de células y de herida. El problema de este enfrenamiento es que las características de las regiones, como las tonalidades de colores o el brillo, pueden cambiar considerablemente entre distintas imágenes debido a condiciones experimentales, por lo que tener un solo modelo que pueda segmentar de forma correcta cualquier imagen de estos experimentos se vuelve una tarea bastante compleja.

El modelo TWS planteado por Arganada et al. [2] es una buena solución a este problema pero hasta la fecha no existe una implementación directa del algoritmo en Python, por lo que decidimos implementarlo nosotros desde Python y así poder facilitarle al laboratorio un método en donde el experimentador ejecute el entrenamiento del modelo con una de sus imágenes y así el modelo segmente correctamente las regiones en el resto de las imágenes que tenga a disposición el experimentador.

La primera etapa de este método consiste en entrenar un modelo de *Random Forest* para clasificar píxeles dentro de dos clases de texturas definidas por el usuario en una imagen de referencia y luego, en la segunda etapa se procede a usar el modelo entrenado para segmentar todas las imágenes. Ambas etapas esquematizadas respectivamente en 'c)" y "d)" de la figura 4.

El primer paso corresponde a seleccionar una imagen de referencia y luego especificar las dos texturas que queremos segmentar encuadrando en la imagen regiones solo de células y regiones que solo tengan herida. Luego la imagen de referencia se convierte a escala de grises y se le aplica el filtro entrópico.

Para la extracción de características a la imagen entrópica se le aplican en paralelo cinco filtros de desenfoque gaussianos, cada uno con su propia desviación estándar, obteniendo así cinco versiones de la imagen con distintos grados de desenfoque. Después se aplica detección de crestas en cada una de las cinco imágenes, consiguiendo un total de diez nuevas imágenes, dos por cada una de las imágenes con desenfoque gaussianos. Así, a partir de la imagen entrópica se obtiene un arreglo de 15 nuevas imágenes.

Volviendo a las regiones especificadas por el usuario, para cada píxel tenemos la etiqueta de a que tipo de textura corresponde y a su vez tenemos un vector de 15 entradas que representan al píxel bajo las distintas transformaciones que se le hicieron a la imagen. Finalmente, se entrena el modelo de *Random Forest* con los vectores de características de cada píxel de las regiones especificada junto a sus etiquetas correspondientes, obteniendo así un modelo que recibe de input un vector de 15 entradas y nos retorna la clase de textura (herida o células) a la que el vector representa. Recordemos que en la etapa de extracción de características obtuvimos el vector de características para cada píxel de la imagen, por lo tanto, con el modelo entrenado se puede clasificar cada píxel de la imagen dentro de alguna de las dos texturas que fueron especificadas.

La segunda etapa de este método corresponde a la detección de texturas en el resto de las imágenes usando el modelo previamente entrenado. Así, cada imagen es convertida a escala de grises y se le aplica el filtro entrópico. En el siguiente paso se ejecuta la extracción de características en la imagen entrópica obteniendo un vector de características de 15 entradas para cada píxel de la imagen. Luego se alimenta el modelo de *Random Forest* con los vectores de características consiguiendo así las texturas predichas (herida o células) para cada píxel. Después reconstruimos la imagen con las predicciones del modelo. Finalmente le aplicamos operadores de erosión y dilatación a la imagen reconstruida para disminuir ruido y nos quedamos con la información de la cantidad de píxeles de cada textura. Este proceso se repite hasta que ya se hayan segmentado todas las imágenes y luego se procede a guardar un archivo CSV con la información de la cantidad de píxeles de cada textura en cada imagen segmentada.

2.2.3. Toolkit

Para el procesar y analizar las imágenes utilizaremos distintas librerías de Python como *SciKit-Learn*, *SciKit-Image*, *OpenCV* y *NumPy*, entre otras. Para poder implementar estos métodos es im-

portante considerar las versiones de los paquetes con los que trabajamos, ya que dichas librerías se actualizan y cambian en el tiempo.

Librería	**Versión**
SciKit-Image	0.18.3
SciKit-Learn	1.0.1
NumPy	1.19.5
OpenCV	4.1.2
Matplotlib	3.2.2
Plotly	4.4.1
Pandas	1.1.5

Tabla 3: "Versiones de las librerías" En esta tabla se presentan las versiones de las librerías que usamos para los dos métodos desarrollados en este trabajo.

En la tabla 3 presentamos las versiones con las que trabajamos. Por último, vale la pena mencionar que para que para poder instalar y trabajar con la versión 0.18.0 o superiores de *SciKit-Image*, es necesario tener Python 3.7 o versiones más modernas.

3. Hipótesis y Objetivos

Hipótesis:

- Usando herramientas de programación en Python se generarán algoritmos con un mínimo de parámetros de control, automatizando el procesamiento de imágenes de monocapas celulares en proceso de cierre de herida agilizando y normalizando este ensayo de la dependencia del operador.

Objetivo Principal:

- Generar un algoritmo de segmentación para el análisis automatizado de imágenes de experimentos de ensayos de herida, de uso abierto y orientado a usuarios que no sean programadores con experiencia.

Objetivo Secundarios:

- Lograr un algoritmo para analizar imágenes obtenidas bajo distintas condiciones sin tener que hacer mayores ajustes en los parámetros de operación del código.
- Acortar los tiempos de análisis de las imágenes de monocapas celulares de forma tradicional, permitiendo un uso más eficaz del tiempo de los usuarios de este algoritmo.
- Validar nuestros métodos de medición de las áreas de las heridas comparando nuestros resultados con mediciones hechas de forma manual usando *ImageJ*.

4. Resultados

Tenemos dos clases de resultados con nuestros algoritmos. Primero mencionaremos los resultados de la segmentación de las imágenes. El segundo tipo de resultado lo obtenemos a partir de las imágenes segmentadas, en donde contamos los píxeles de cada tipo de región, para así obtener una medida para el área de las heridas en las imágenes.

4.1. Procesamiento de Imágenes

Comenzando con el método de detección de contornos, primero se generaron máscaras de las heridas desde cada imagen del tiempo 0h a través de thresholds y filtros.

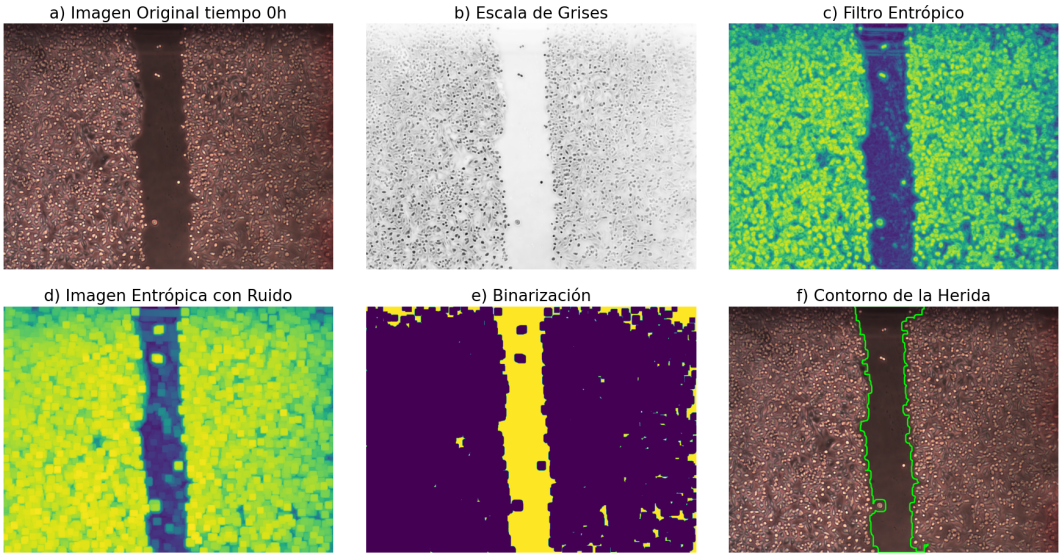


Figura 5: "Primera etapa del Método de Detección de Contornos, Obtención de los Contornos." a) Imagen del tiempo 0h. b) Transformación a escala de grises. c) Aplicación del filtro entrópico en la imagen convertida a escala de grises. d) Ruido agregado a la imagen entrópica. e) Binarización de la imagen con ruido a partir de los thresholds de la media y de Li. f) Imagen original del tiempo 0h con el contorno detectado a partir de la imagen binarizada.

En la figura 5 se aprecian las etapas por las que pasan las imágenes del tiempo 0h al ser procesadas. El siguiente paso en este método es obtener una imagen segmentada en dos colores a través de *KMeans* en las imágenes de tiempos posteriores al tiempo 0h y luego usar la máscara sobre las imágenes desde donde podremos contar los píxeles de cada color dentro de la máscara, obteniendo una medida del área de las heridas en las imágenes.

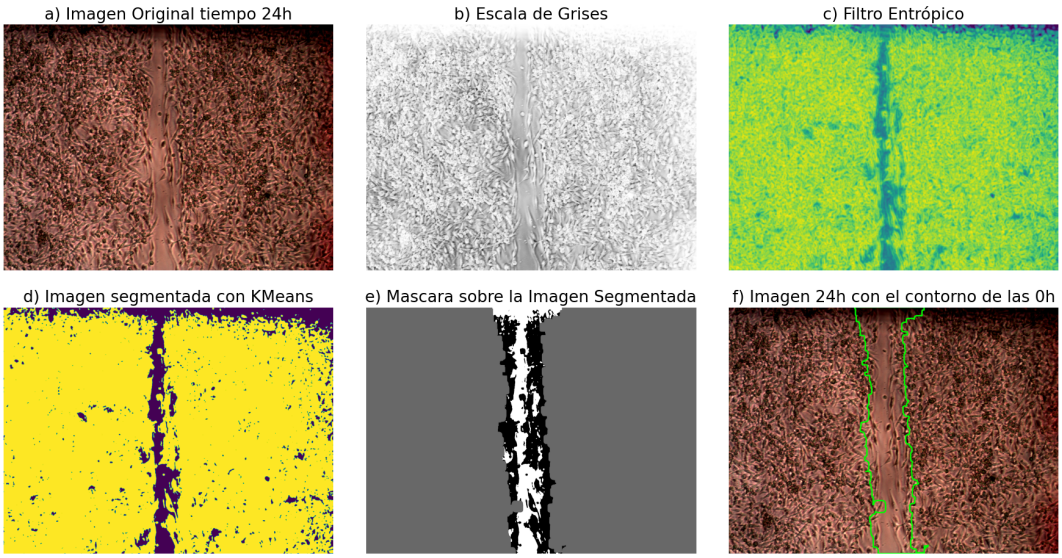


Figura 6: "Segunda etapa del Método de Detección de Contornos, Obtención de las Áreas." a) Imagen del tiempo 24h. b) Transformación a escala de grises. c) Aplicación del filtro entrópico en la imagen convertida a escala de grises. d) Segmentación de la imagen entrópica a través de *KMeans*. e) Aplicación de la máscara de la herida del tiempo 0h sobre la imagen segmentada. Los píxeles blancos corresponden a regiones sin células, mientras que los píxeles negros corresponden a regiones con células y los píxeles grises corresponden al área con células de la imagen del tiempo 0h. f) Imagen para referenciar el contorno de la herida de la imagen del tiempo 0h en la imagen del tiempo 24h.

Segmentar con *KMeans* tiene ventajas y desventajas, ya que, al exigirle dos clusters en el espacio de valores de la imagen entrópica se logra una buena segmentación de las dos clases de regiones que tienen la imagen, las regiones con células y las regiones sin células. Por otro lado, en las situaciones en que la imagen es de una herida que cerró, al exigirle dos clusters a *KMeans* se obtendrán resultados incorrectos para las medidas de las áreas, ya que en la imagen solo hay un tipo de región.

Como se aprecia en la figura 7, en las imágenes de placas donde la herida cerró, dentro de la imagen solo tenemos una de las dos clases, ya que, no hay grandes regiones sin células, pero de todas formas le estamos exigiendo dos grupos en el espacio de valores de la imagen entrópica. Esto ultimo nos muestra una desventaja de procedimiento, ya que, en las situaciones recién descritas con *KMeans* obtenemos dos clases distintas de píxeles distribuidos en todo el espacio de la imagen, pero en realidad, la imagen tiene una sola clase de textura.

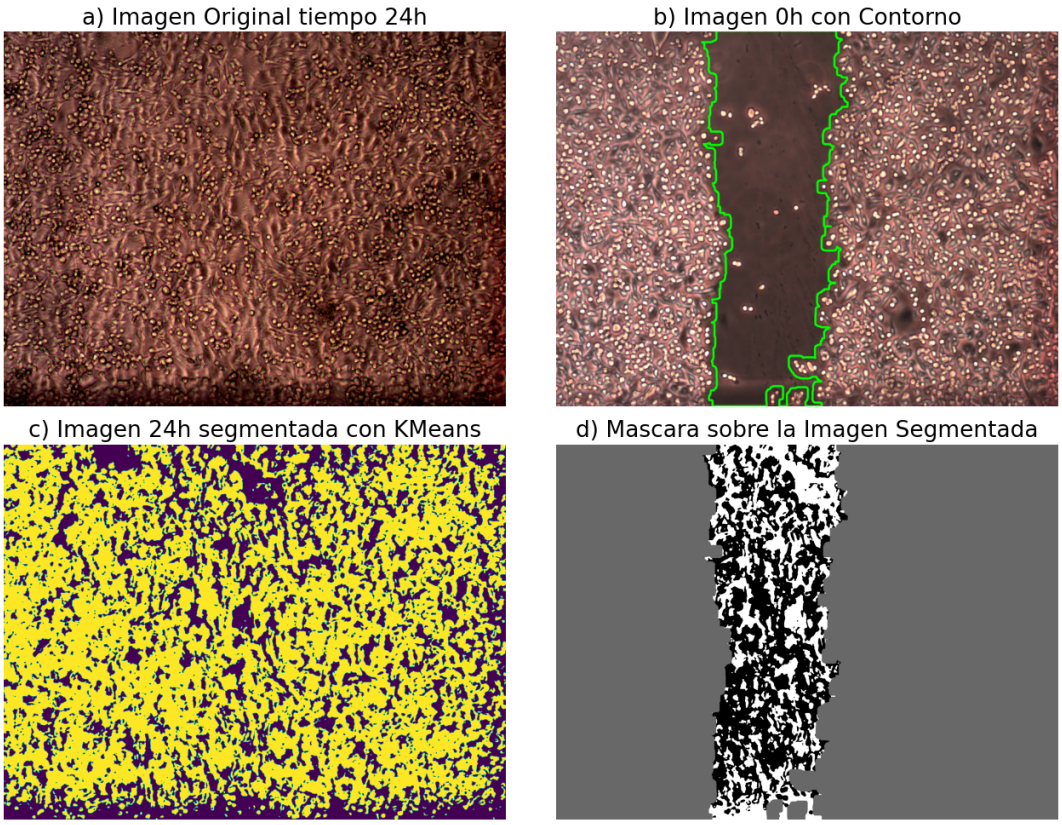


Figura 7: "Ejemplo de los problemas del Método de Detección de Contornos." a) Imagen del tiempo 24h en donde la herida cerró por completo. b) Imagen del tiempo 0h desde donde obtenemos la máscara para procesar la imagen con la herida cerrada. c) Segmentación en dos colores a través de *KMeans* en la imagen del tiempo 24h. d) Imagen segmentada a través de *KMeans* con la máscara de la herida.

Además, el usar la máscara para medir solo el área dentro de esta nos aporta con ruido en las imágenes con puntos de vistas muy distintos, ya que no coinciden las locaciones de las heridas entre las imágenes. Por estas razones decidimos enfocarnos en trabajar sobre el segundo método en donde la segmentación se hace a través de la detección de texturas en las imágenes. En este algoritmo, a partir de una imagen de referencia, definimos las dos clases de texturas a segmentar para así entrenar un modelo de *Random Forest* con el que luego se procede a segmentar el resto de las imágenes.

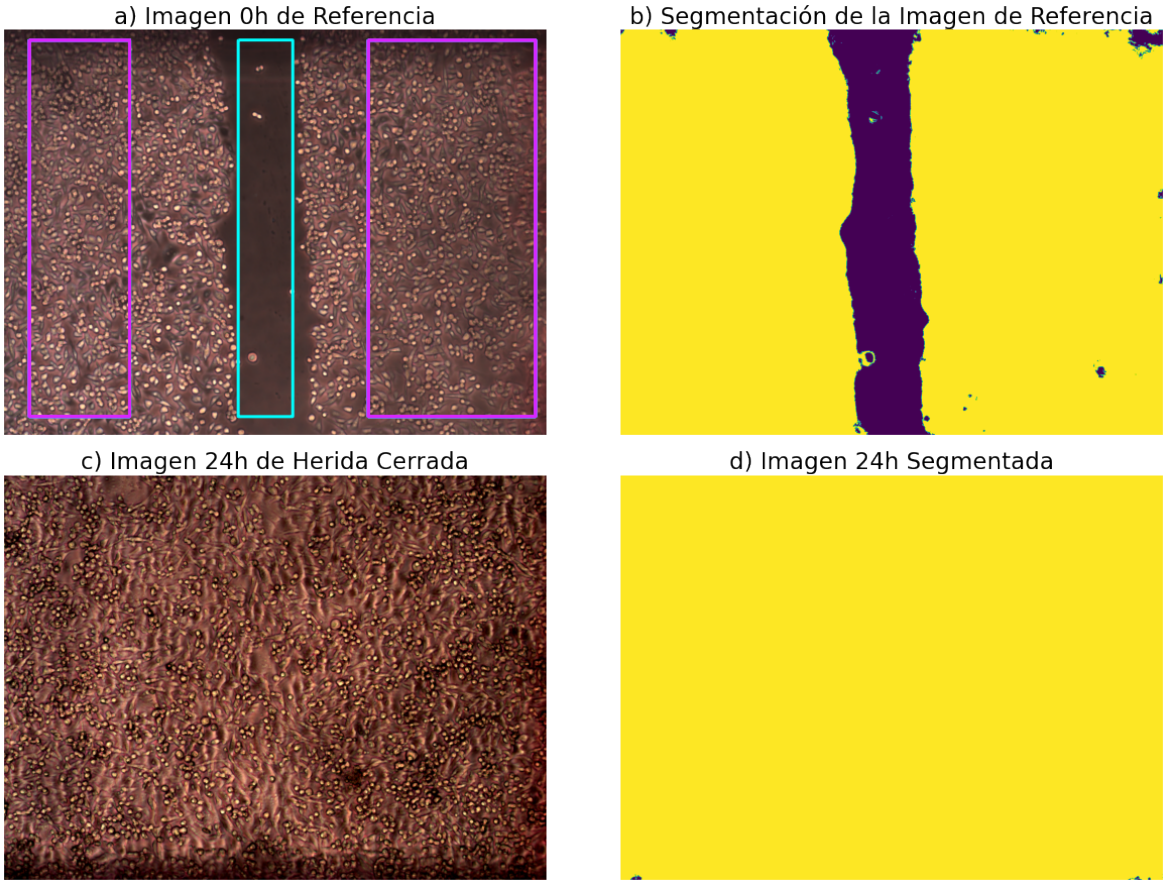


Figura 8: "Segmentación a través de Detección de Texturas." a) Imagen de referencia del tiempo 0h con la especificación de las regiones con las texturas a clasificar. b) Imagen de referencia segmentada. c) Imagen del tiempo 24h para probar la segmentación de este método en un caso en donde la herido cerró por completo. d) Segmentación de la imagen del tiempo 24h. Para b) y d) los píxeles amarillos representan las regiones con células mientras que los píxeles morados representas las regiones sin células.

En la figura 8 se aprecia como es que al usar este método se logran detectar ambos tipos de regiones en las fotos que tienen una herida abierta, por otro lado, en las fotos en donde la herida cerró se detecta una sola clase de textura. En particular para este ejemplo la imagen de referencia fue la misma imagen del tiempo 0h que mostramos en la figura 5 y para la imagen a segmentar estamos usando la misma imagen del tiempo 24h que se uso en la figura 7.

Con el algoritmo de detección de texturas segmentamos todas las imágenes del dataset. En este proceso pudimos encontrar características en la imágenes que dificultan la segmentación. En la figura 9 mostramos dos ejemplos de esto.

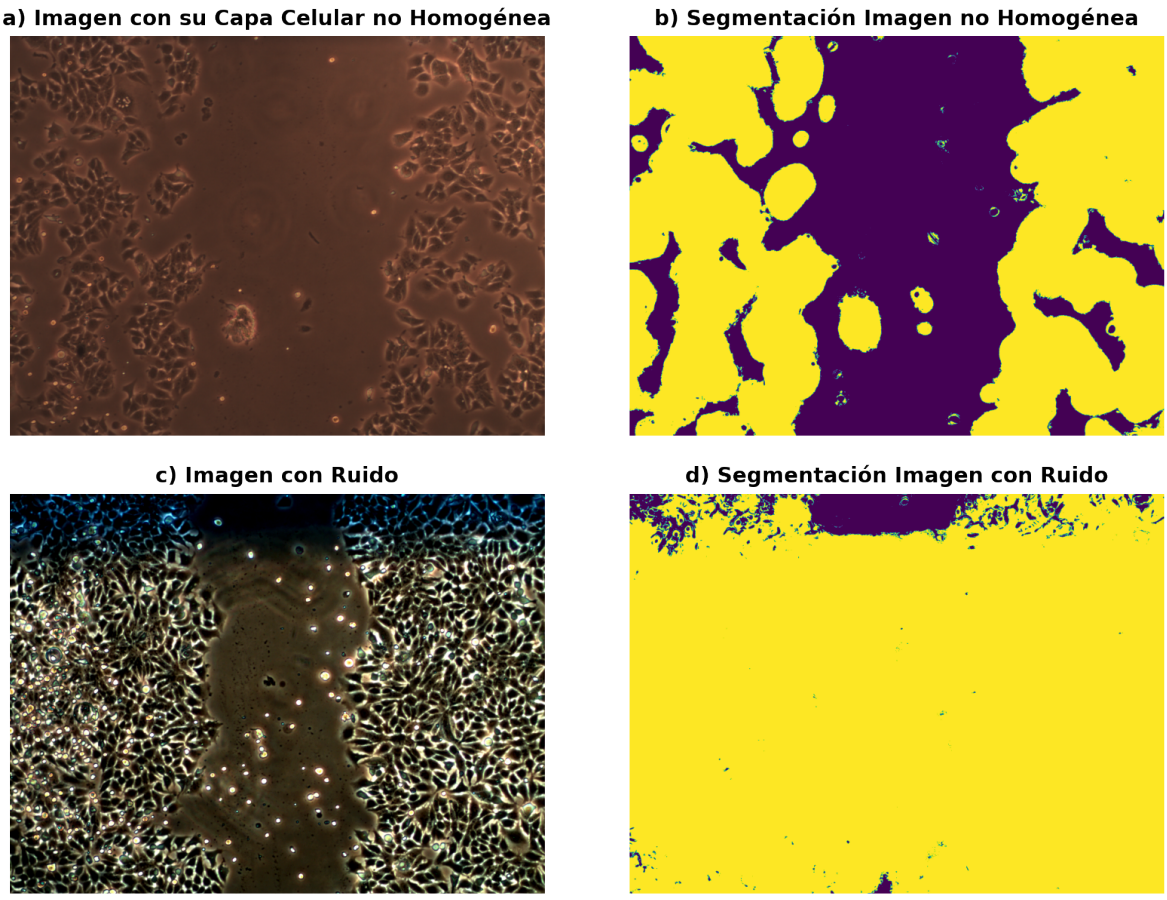


Figura 9: "Segmentación con Detección de Texturas". a) Imagen en donde su capa de células no era totalmente homogénea. b) Imagen no homogénea segmentada. c) Imagen con ruido. d) Segmentación de la imagen con ruido.

En la figura 9 se aprecian los resultados de la segmentación en una imagen en donde la capa celular no era homogénea y en una imagen que tiene alta varianza en su espacio de colores en las distintas regiones locales de la imagen. Como se ve en la imagen en "b)" de la figura, en las regiones de células (píxeles amarillos) se aprecian importante regiones de píxeles que corresponden a la textura de la herida (píxeles morados), así, al contar la cantidad de píxeles morados de la imagen vamos a contar también las regiones abiertas que están dentro de la región de células aportando con ruido a la medida del área de la herida. Por otro lado, en la imagen "d)" de la figura vemos que la mayoría de los píxeles de la imagen fueron clasificados como píxeles de la región de células mientras que en la imagen "c)" se puede apreciar claramente la herida.

Para disminuir el ruido que aportan las regiones de células no homogéneas y además, el ruido que se aprecia en las regiones de herida debido "artefactos" (como marcas en las placas o células individuales que quedan dentro de las heridas), aplicamos técnicas de dilatación y erosión en las imágenes segmentadas. El objetivo de la dilatación es llenar los espacios clasificados como heridas

dentro de las regiones de células, mientras que la erosión fue aplicada para eliminar las regiones clasificadas como células dentro de las regiones de herida. Este procedimiento trajo tanto resultados positivos como negativos, dependiendo de las condiciones de las imágenes. A continuación en la figura 10 mostramos resultados positivos de estas técnicas aplicadas en imágenes segmentadas.

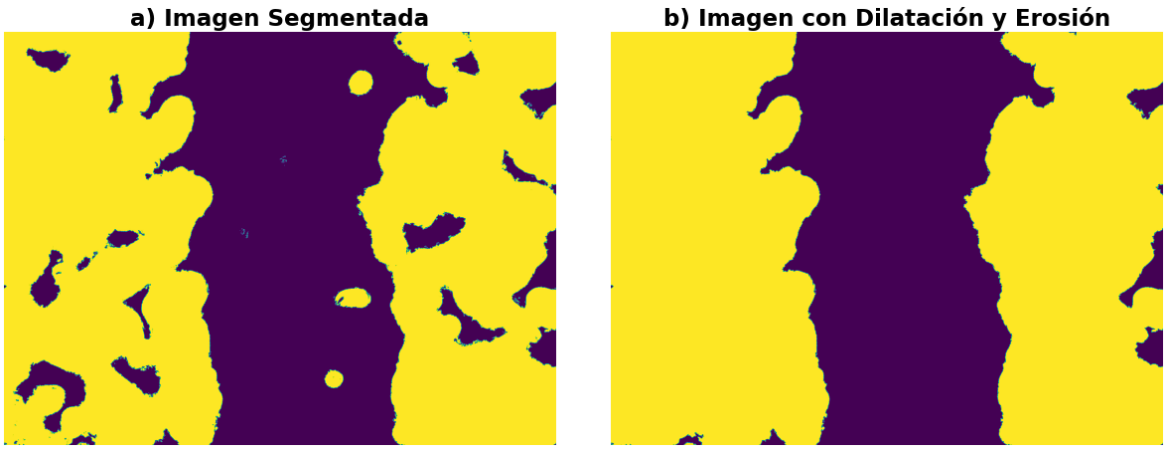


Figura 10: "Ejemplo Positiva de Dilatación y Erosión ". a) Imagen segmentada en donde su capa de células no era totalmente homogénea. b) Imagen limpia tras aplicar dilatación y erosión.

En la figura 10 se aprecia como es que las técnicas de dilatación y erosión en simultaneo logran remover regiones clasificadas como herida en las regiones de células y viceversa. En particular elegimos esta imagen como ejemplo para mostrar que aunque las técnicas funcionen, no es perfecta la limpieza del ruido, ya que en el lado derecho de ambas imágenes, podemos ver regiones clasificadas como herida (píxeles morados) dentro de las regiones de células (píxeles amarillos) pero en general, este proceso de limpieza fue un aporte para disminuir el ruido en las mediciones finales.

Por otro lado en la figura 11 ejemplificamos caso en donde estas técnicas no fueron útiles para disminuir el ruido. En este ejemplo la región de la herida en la imagen segmentada tiene altos niveles de ruido, en donde incluso se aprecia que las regiones de células de los lados opuestos de la imagen están conectados cuando en realidad, en dichas localidades de la imagen original uno encuentra células individuales que quedaron en medio de la herida. Al ejecutar la dilatación y la erosión con esta imagen segmentada, vemos como es que la región de la herida fue casi totalmente re-segmentada como píxeles de células.

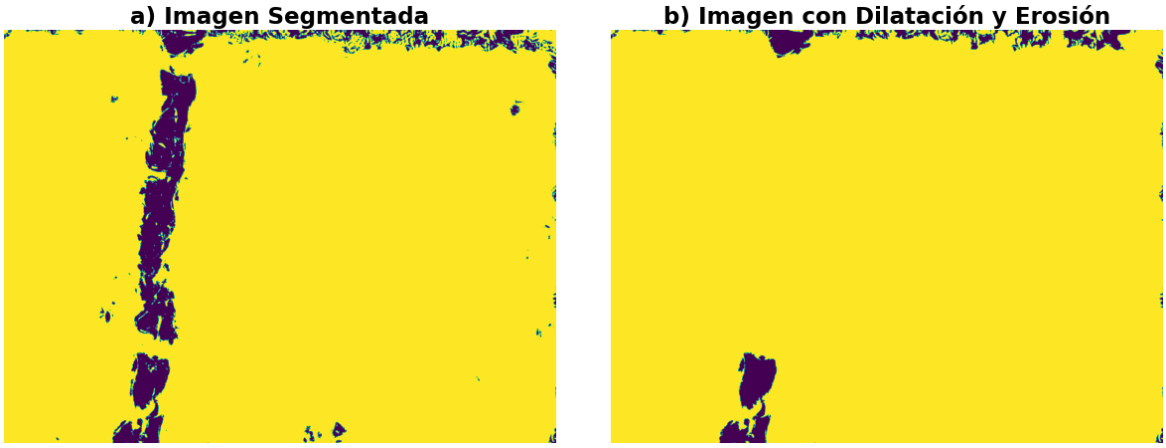


Figura 11: "Ejemplo de Dilución y Erosión Errónea". a) Imagen segmentada con importante ruido en su región de herida. b) Imagen prácticamente sin su herida tras aplicar dilatación y erosión.

Hasta este momento hemos visto que ambos métodos tienen sus ventajas y desventajas, por lo que decidimos priorizar el trabajo sobre uno de estos algoritmos. Como con el método de detección de texturas hemos obtenido mejores y más rápidos resultados en la segmentación de las imágenes, las mediciones de las áreas de las heridas las obtuvimos a partir de este algoritmo, así, en la siguiente sección se presentaran los resultados de las mediciones de las áreas de las heridas generadas a partir de nuestro método de segmentación de texturas y también las mediciones hechas de forma manual a través de *ImageJ* por uno de los integrantes del laboratorio del *CMR*.

4.2. Mediciones del Área de las Heridas

Una vez que le aplicamos los operadores de erosión y dilatación a las imágenes segmentadas, podemos contar la cantidad de píxeles de cada color. En todos los ejemplos de imágenes segmentadas que hemos mostrado hasta el momento, las regiones con células y las regiones de herida han sido representadas con píxeles amarillos y morados respectivamente. Así, para cuantificar el cierre de las heridas, extraemos la cantidad de píxeles morados (de la herida) que tiene cada imagen.

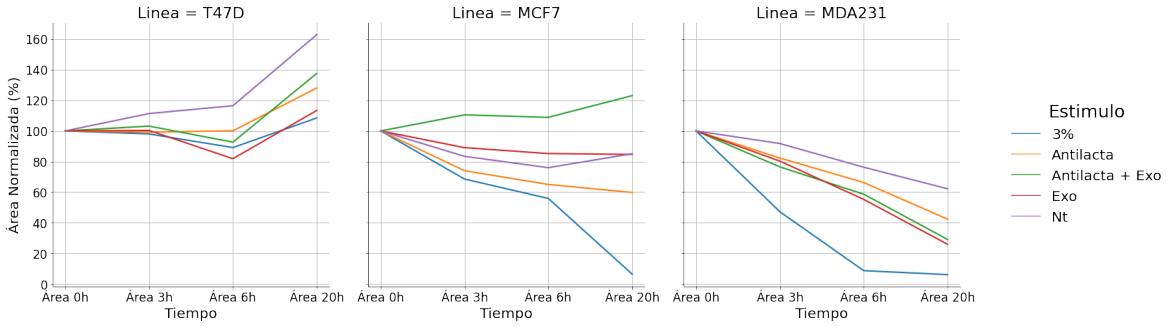
Recordemos que en la obtención de las imágenes, en cada placa donde se cultivan células primero se hace la herida y luego se documenta el estado de la placa en múltiples imágenes. Entonces para cada condición experimental tendremos múltiples imágenes en tiempos consecutivos, lo que nos lleva a múltiples mediciones de las áreas de la herida de los tiempos consecutivos. Como esto ocurre con todas las condiciones que se están estudiando, el procedimiento es normalizar los valores de las áreas que se midieron al tamaño del área de la herida de la primera imagen que se obtuvo para la condición. Volviendo a usar el ejemplo de la tabla 1, en este caso, tendríamos que normalizar todos los valores de las áreas según el valor de Área_1.

Entonces para cada condición (cada combinación de linea celular y estimulo) ajustamos los datos de los conteos de píxeles que obtuvimos en cada imagen con la ecuación 2.

$$\text{Área Normalizada} = \frac{\text{Área(tiempo} > \text{0h)}}{\text{Área(tiempo} = \text{0h)}} * 100 \tag{2}$$

Así las medidas de las áreas de cada condiciones estarán bajo la misma escala de valores, lo que facilita la comparación del comportamiento migratorio de las células entre las distintas condiciones.

(a) **"Mediciones Automáticas del Área. Método de Detección de Texturas"**



(b) **"Mediciones del Área a través de *ImageJ*"**

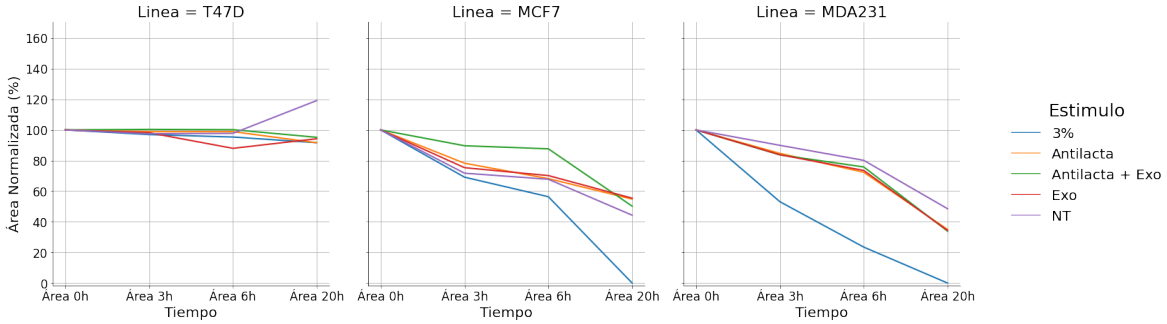


Figura 12: Para ambas figuras en cada gráfico el eje X representa el tiempo en el que se sacaron las fotografías y el eje Y corresponde al área normalizada, en donde un 100 % representa el tamaño de la herida en el tiempo 0h y un 0 % representaría una imagen con la herida totalmente cerrada.

En las figuras 12a y 12b tenemos respectivamente los resultados de la medición de las áreas a través del método de detección de texturas y los resultados de la medición manual que hizo una de las personas del *CMR* en el tercer set de imágenes. En estos experimentos se trabajó con las lineas celulares MCF7, MDA231 y la T47D. Todas estas lineas celulares son de distintos tipos de células tumorales de cáncer de mama. Cada uno de los tres gráficos de ambas figuras corresponde a una de estas lineas celulares. Además en cada linea celular se experimento con los mismos cinco escenarios de estímulos, cada uno representado con una curva en los gráficos.

A partir de los datos de las áreas normalizadas de las mediciones manuales y las automáticas calculamos el error absoluto medio (MAE), el error cuadrático medio (MSE), la raíz del error cuadrático medio (RMSE), el coeficiente de determinación (R^2) y también la versión corregida de esta ultima métrica de error (R^2_{adj}).

	T47D	MCF7	MDA231	Total
MAE	0.1092	0.1267	0.0607	0.0989
MSE	0.0319	0.0471	0.0069	0.0286
RMSE	0.1785	0.2170	0.0830	0.1692
R^2	-8.2009	0.2025	0.9199	0.5730
R^2_{adj}	-11.3375	-0.0694	0.8926	0.4275

Tabla 4: "Métricas de Error." Se presentan las medidas de los errores de los datos del área normalizada para cada linea celular y para el modelo completo, considerando los datos provenientes de las mediciones manuales como los valores verdades y los datos automáticos como los valores predichos.

4.3. Implementación del Código y Tiempos de computo

La implementación del modelo de detección de texturas ha sido desde la plataforma de *Google - Colab* en donde generamos dos *notebooks* y un código de python con funciones auxiliares. Con el primer *notebook* se ejecuta la segmentación de imágenes y se obtiene un archivo CSV con las medidas de las áreas, mientras que con el segundo *notebook* a partir de los archivos CSV's se obtienen gráficos como los que presentamos en las figuras 12a y 12b. Se puede acceder libremente a ambos *notebooks* y al código de Python desde un repositorio publico de nuestro *github* en el siguiente enlace `https://github.com/seburrejola/Proyecto-de-Grado-MDS-2021`.

Para la segmentación el usuario debe especificar primero la ruta al directorio de *Google - Drive* en donde se tienen las imágenes, luego generamos una tabla con la información de las rutas a las imágenes, las lineas celulares, los tiempos y los estímulos de las fotos basándonos en la estructura de directorios en que se han almacenado las imágenes. Esta tabla de metadata es sumamente importante porque sobre ella iteramos para procesar cada una de las imágenes. Los usuarios del código pueden especificar en que orden se encuentran estos directorios, pero por defecto consideramos que primero tendremos carpetas para las lineas celulares, luego carpetas para los tiempos y finalmente dentro de cada carpeta de los tiempos, una carpeta para cada estimulo que contenga finalmente a las imágenes, tal como se muestra a continuación.

Google - Drive//...//Data//Lineas Celulares//Tiempos//Estimulos//Fotos.tif

Además el usuario debe especificar en que formato estan guardadas las imágenes, por defecto consideramos que es en formato *.tif*. Una vez que se ha logrado obtener la tabla de metadata correctamente, se procede a seleccionar la imagen de referencia en donde se definen las texturas a clasificar demarcando las regiones de interés en la imagen.

En este punto el usuario puede modificar los parámetros de operación del método, estos parámetros son el radio de observación del filtro entrópico, el rango de las desviaciones estándar de los filtros

de desenfoque gaussiano en la extracción de características y los parámetros de la cantidad de arboles, su profundidad y la cantidad de muestras por árbol del modelo de *Random Forest*. No es necesario alterar estos valores para ejecutar la segmentación, pero en el caso de querer hacerlo es necesario que los cambios se hagan antes del entrenamiento. Una vez establecidas las regiones de la imagen de referencia y los parámetros de operación se procede a ejecutar el entrenamiento del modelo. Después del entrenamiento se despliega el resultado de la segmentación en la imagen de referencia para que así el usuario pueda cerciorarse de que haya sido correcta, en caso contrario se recomienda volver a definir las regiones evitando demarcar regiones de células dentro de las regiones de herida o viceversa. Finalmente cuando el modelo ya haya sido entrenado exitosamente se procede a segmentar todas las imágenes. Al terminar el procesamiento se guarda un archivo CSV con las medidas de las áreas en el directorio que se especificó al inicio.

Una vez que ya se tiene el archivo CSV de las áreas con el segundo *notebook* se obtienen los gráficos del cambio del área de las heridas en el tiempo. Para esto el usuario debe especificar la ruta en donde se encuentra el archivo CSV y el titulo que se le quiere dar a la figura. Además el usuario puede modificar el tamaño de fuente de los distintos textos del gráfico como el titulo, las leyendas o las etiquetas de los ejes.

Actualmente desde *Google - Colab*, procesar una imagen con el método de segmentación a través de detección de texturas toma aproximadamente 40 segundos, mientras que la persona del *CMR* que procesó las imágenes de forma manual tardó aproximadamente 50 segundos por foto. Así nuestro método esta tardando un 80 % del tiempo que le toma a una persona calificada para hacer este procesamiento.

5. Discusión y Limitaciones

En relación al método de detección de contornos, pensamos que se podría mejorar la calidad de sus resultados implementando un filtro para las imágenes donde la herida cerró por completo para así no forzar dos clases de píxeles con la segmentación a través de *KMeans*, además removiendo las máscaras disminuiríamos el ruido que aportan en la medida de las áreas de las heridas debido al cambio de punto de vista entre las distintas imágenes de la misma condición, pero este procedimiento ya lo hemos visto en la literatura, como el caso de [7].

Por otro lado, con el método de segmentación de imágenes a través de detección de texturas se segmentaron las 1696 imágenes que generaron en el *CMR* usando una sola imagen de referencia para el entrenamiento. Dicha imagen de referencia pertenecía al tercer set de imágenes (set de mejor calidad) y es precisamente en ese grupo de imágenes en donde se obtuvieron los mejores resultados (figuras 12a y 12b). Tras comparar cualitativamente los resultados de la cuantificación de las áreas, a excepción de la linea celular T47D, los resultados de los conteos de áreas de nuestra solución automatizada se asemejan a los resultados obtenidos de forma manual. Como por ejemplo en la linea celular MDA231 se mantuvieron las relaciones entre los distintos estímulos, en donde el estimulo “3 %” se mantuvo en un porcentaje más bajo que el resto de los estímulos, mientras que el estimulo “NT” siempre se mantuvo en un mayor porcentaje de área. En el caso particular de la linea celular T47D aunque con nuestros resultados se podría inferir que las heridas quedaron abiertas (ya que no llegan al 0 %) igual que en las mediciones manuales, los valores de áreas que medimos son claramente incorrectos y esto se debe a que las imágenes tenían artefactos que no nos permitieron lograr una buena segmentación

a nivel general en esta linea celular.

Tras hacer la segmentación de todas las imágenes con el método de detección de texturas y revisar los resultados, llegamos a los criterios de inclusión que permiten que el modelo funcione de forma adecuada con las imágenes seleccionadas. En general, mientras más similares sean las características del brillo y tonalidades de colores de las imágenes a segmentar en relación a la imagen de referencia que se uso para entrenar, la segmentación será eficaz. En sí, se podría segmentar cada imagen entrenando el modelo en dicha imagen, pero esto tomaría más tiempo que analizarlas de forma manual, por lo que mientras se puedan hacer grupos "naturales" de las imágenes similares, se podrá hacer la segmentación de forma efectiva. Con grupos "naturales" nos referimos a que se puedan agrupar las imágenes según sus atributos propios, como los tiempos en que se obtuvieron, las lineas celulares y/o sus estímulos. Por ejemplo, en el set 3 tuvimos que volver a segmentar todas las imágenes de los tiempo 20h usando una imagen de referencia distinta a la que se uso en el resto de las segmentaciones debido a que este grupo de imágenes (set 3 - tiempos 20h) era más brillante que el resto y en la primera segmentación que habíamos hecho, los resultados que tuvimos representaban imágenes en donde las heridas habían cerrado por completo pero esto no era acertado.

Respecto al modelo de aprendizaje automático, en un inicio buscamos distintos modelos de clasificación de *Scikit - Learn* que tuviesen el parámetro *warm start* con el objetivo de generar un método que pudiésemos entrenar múltiples veces sin perder la información obtenida en entrenamientos previos. Este fue nuestro enfoque debido a que las imágenes que eran muy distintas a la imagen de referencia no quedaban bien segmentadas, por lo que pensábamos entrenar nuevamente el modelo con dichas imágenes problemáticas para así obtener un modelo más robusto. Por otro lado, no fue posible medir el desempeño del modelo basándonos en métricas comunes de clasificación o segmentación puesto que no teníamos datos etiquetados previamente, es decir, no teníamos datos *ground truth* con los que pudiésemos comparar nuestros resultados con el fin de evaluar la calidad del modelo, más sobre esto más adelante. Por esta razón, para saber si los modelos estaban funcionando de forma adecuada tuvimos que comparar empíricamente los resultados de las imágenes segmentadas con sus respectivas imágenes originales. Finalmente desechamos la idea de un modelo que pudiésemos entrenar múltiples veces sin perder lo obtenido en los entrenamientos previos y nos quedamos con el modelo *Random Forest* que fue el que empíricamente tuvo los mejores resultados en la predicción de las texturas.

Para llegar a las configuraciones que usamos en el *Random Forest* y el filtro entrópico, en primer lugar, debido a que no disminuimos las dimensiones de las imágenes, normalmente en la etapa de entrenamiento obteníamos más de dos millones de vectores de características (cada imagen tiene poco mas de 5 millones de píxeles) por lo que decidimos usar un valor relativamente bajo para la cantidad máxima de muestras con las que cada árbol se entrenaba, fijando este parámetro en un 5 % del total de muestras disponibles. Al fijar este valor, cada árbol tenía un aproximado de 100 mil muestras para entrenar, entonces para que el tiempo de inferencia y principalmente el de entrenamiento no fuesen muy altos, decidimos usar un total de 50 arboles de decisiones con un máximo de 10 nodos. En el caso del filtro entrópico se usa una ventana de observación para calcular la entropía local. El parámetro del tamaño de la ventana tiene una gran influencia sobre el tiempo que tarda el modelo en procesar cada imagen (ya sea en el entrenamiento o la segmentación de todas las imágenes) debido a que calculo de la entropía se hace píxel a píxel y cada imagen tiene poco más de 5 millones de píxeles. Nosotros decidimos usar una ventana circular con un radio de 5 píxeles. Usamos un radio de observación relativamente pequeño (considerando las dimensiones de la imagen) para que el calculo de la entropía no agregue mucho tiempo de computo. Con esta configuración el entrenamiento del modelo y la segmentación de las imágenes tardaba en general menos de un minuto, así en el caso de que no se

logren buenos resultados tras hacer el entrenamiento, el usuario no tendría que esperar mucho tiempo para volver a entrenar con otra imagen o alterando las regiones con las texturas de interés.

En relación a las técnicas de dilatación y la erosión, pensamos que implementarlas fue una buena forma de refinar las medidas de las áreas y así lograr un algoritmo más robusto ante imágenes de calidades variadas. Si bien en algunas ocasiones donde imágenes eran muy ruidosas estas técnicas aumentaron el ruido, dichas imágenes se podrían volver a segmentar entrenando nuevamente el algoritmo.

Como mencionamos anteriormente, en nuestro dataset no teníamos imágenes segmentadas por el laboratorio para poder comparar nuestros resultados y así medir el desempeño del modelo. Para poder obtener una medida de la calidad de nuestra solución decidimos enfrentarnos a esto como un problema de regresión, usando el área normalizada como variable de respuesta, tal como se muestra en la tabla 4. Si bien, lo ideal sería poder comparar resultados entre modelos distintos, nosotros solo tenemos los resultados obtenidos a partir del modelo de detección de texturas ya que durante este trabajo decidimos dejar de usar le método de detección de contornos. Debido a esto, en la tabla 4 se muestran las medidas de los errores que obtuvimos en cada linea celular y también a modo global en nuestros resultados, para poder inspeccionar de forma objetiva en que momento el modelo estaba funcionando bien y cuando no. En particular nos enfocaremos en el coeficiente de determinación que nos da una medida de la proporción de la varianza de los datos reales que el modelo explica. Viendo la tabla notamos el R^2 y el R^2_{adj} de la linea celular MDA231 son los más altos de la tabla, en ambos casos siendo valores relativamente cercanos a 1. Mientras que por el otro lado, en las otras dos lineas celulares estas medidas son mucho mas bajas, llegando a valores negativos para el R^2 y el R^2_{adj} en la linea celular T47D, esto ultimo lo atribuimos a que en las imágenes de esta linea celular habían artefactos que no nos permitieron hacer segmentaciones correctas.

Al hacer la implementación del código desde *Google - Colab* evitamos tener que manejar la instalación y compatibilidad de las librerías necesarias para el modelo. También con este enfrentamiento el computo se ejecuta en maquinas virtuales de la nube de *Google* y podemos montar el *Google - Drive*, lo que nos permite trabajar sin tener las imágenes almacenadas de forma local. La mayor desventaja de la implementación desde esta plataforma yace en que aunque ejecutar código en un *notebook* de *Google - Colab* sea sencillo para quienes tengan experiencia programando, en el caso de personas que nunca han programado puede llegar a ser complejo puesto a que se estarían enfrentando a una interfaz totalmente nueva para dichas personas, además los términos y condiciones de uso de la plataforma pueden cambiar lo que no nos asegura que se pueda seguir usando el modelo el futuro. Aun así pensamos que la implementación desde *Google - Colab* ha sido un acierto, en particular por que al ejecutar el computo en la nube y no de forma local, no nos tenemos que preocupar del hardware al que se tenga acceso en el laboratorio, aspecto sumamente importante a la hora de considerar el tiempo que tarda el algoritmo en procesar las imágenes.

6. Conclusiones

Contrastando los resultados obtenidos con los objetivos propuestos, logramos generar el método de detección de texturas que se puede ejecutar desde un *notebook* en donde los usuarios pueden modificar un total de 5 parámetros de operación para así segmentar las imágenes y conseguir las medidas de las áreas de las heridas. Además generamos un segundo *notebook* desde el que se pueden hacer gráficos de los datos de las áreas que se obtengan a partir del primer *notebook*. También se logró el objetivo especifico de disminuir el tiempo que toma hacer el análisis usando *ImageJ*, ya que la segmentación individual de cada imagen en promedio tomo 40 segundos mientras que el investigador científico que analizó manualmente las imágenes tardaba 50 segundos por imagen.

Por último, siempre que los grupos de imágenes que se quieran segmentar cumplan los criterios de inclusión descritos en la sección anterior, el algoritmo de detección de texturas que generamos, analiza las imágenes de ensayos de herida de manera más rápida e independiza el resultado del análisis del operador científico, entregando resultados replicables y consistentes.

7. Trabajos Futuros

En relación a la medición de la calidad del modelo, tenemos que evaluar nuestra solución con un conjuto de imágenes que tenga segmentaciones ya hechas para poder comparar nuestros resultados. Para esto planeamos segmentar de forma manual las imágenes que se obtengan en el laboratorio la siguiente vez que hagan ensayos de herida. También pensamos evaluar nuestro modelo con conjuntos de imágenes de referencia que se puedan hallar en linea para así comparar nuestro modelo con modelos del estado del arte, como los descritos en la revisión bibliográfica.

En vista de los resultados obtenidos hasta el momento, para disminuir el tiempo que le toma a nuestro modelo segmentar cada imagen, proponemos disminuir los tamaños de las imágenes desde el mismo código, así, los procesos de calcular la entropía local, la obtención de características y el entrenamiento junto con la segmentación tomarían considerablemente menos tiempo.

Pensando en las limitaciones que tiene el método de detección de texturas, en general se deben a las características de las imágenes y a que la clasificación de píxeles se hace píxel a píxel, cuando el contexto de cada píxel es importante a la hora de clasificarlo como perteneciente a regiones de heridas o células. Por lo tanto, una implementación de un modelo en donde el contexto de cada píxel sea más importante podría tener mejores resultados, así, pensamos que una red neuronal convolucional podría hacer mejor el trabajo de segmentar las imágenes. En este escenario, necesitaríamos imágenes previamente clasificadas e imágenes más pequeñas para poder generar modelos con cantidades manejables de parámetros. Así, ejecutar lo propuesto en el párrafo anterior nos permitiría obtener imágenes segmentadas para usar en el entrenamiento de esta nueva red y además también tendríamos la disminución de las dimensiones de las imágenes necesaria para mantener el numero de parámetros de la red en valores manejables. Este enfrentamiento nos permitiría también trabajar desde la GPU lo que podría aumentar la velocidad con la que logramos hacer la segmentación de las imágenes.

En relación a la implementación de esta solución automática, pensamos que se podría implementar el modelo como un software con su propia interfaz gráfica para que los usuarios del código no tengan que ejecutarlo desde un *notebook* de *Google - Colab*, pero al hacer esto pasamos de manejar el computo

en la nube a manejarlo de forma local ya que desde *Colab* no es posible implementar la interfaz gráfica. Por lo que sería necesario que el laboratorio del *CMR* (o usuarios de otras instituciones) inviertan en un computador con una CPU que pueda procesar las imágenes en menos tiempo del que le toma a una persona. A modo de referencia, desde *Colab* hemos tenido acceso a CPU's de 2.2GHz lo que nos ha permitido procesar cada imagen en aproximadamente un 80 % del tiempo del que le toma una persona y de forma local hemos trabajado con una CPU de 3.2GHz en donde la segmentación de cada imagen tomo aproximadamente un 20 % del tiempo del que le toma una persona, por lo que hacer esta inversión podría ser bastante fructífera. Además al generar esta aplicación *standalone* se podrían implementar más modelos para la automatización del análisis y procesamiento de los resultados de otros experimentos que se hagan en el laboratorio.

Referencias

[1] Bruce Alberts, Alexander Johnson, Julian Lewis, David Morgan, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell*. W.W. Norton & Co, 6 edition, 1983.

[2] Ignacio Arganda-Carreras, Verena Kaynig, Curtis Rueden, Kevin W Eliceiri, Johannes Schindelin, Albert Cardona, and H Sebastian Seung. Trainable weka segmentation: a machine learning tool for microscopy pixel classification. *Bioinformatics*, 33(15):2424 – 2426, March 2017.

[3] T.F. Chan and L.A. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10:266 – 277, February 2001.

[4] dr. Guy Regnard. Wound healing assay - what, why and how.

[5] Fernanda Garcia-Fossa, Vladimir Gaal, and Marcelo Bispo de Jesus. Pyscratch: An ease of use tool for analysis of scratch assays. *ELSEVIER Computer Methods and Programs in Biomedicine*, 193, March 2020.

[6] Tobias Gebäck, Martin Michael Peter Schulz, Petros Koumoutsakos, and Michael Detmar. Tscratch: a novel and simple software tool for automated analysis of monolayer wound healing assays. *BioTechniques Short Technical Reports*, 46:265–274, April 2009.

[7] Markus Glaß, Birgit M.^{o}ller, Anne Zirkel, Kristin W.^{a}chter, Stefan Hüttelmaier, and Stefan Posch. Cell migration analysis: Segmenting scratch assay images with level sets and support vector machines. *ELSEVIER Pattern Recognition*, 45:3154–3165, March 2012.

[8] Ayman Grada1, Marta Otero-Vinas, Francisco Prieto-Castrillo, Zaidal Obagi, and Vincent Falanga. Research techniques made simple: Analysis of collective cell migration using the wound healing assay. *Journal of Investigative Dermatology*, 137:e11–e16, Nov 2016.

[9] Michael R. Lamprecht, David M. Sabatini, and Anne E. Carpenter. Cellprofiler™: free, versatile software for automated biological image analysis. *BioTechniques Short Technical Reports*, 42:71–75, January 2007.

[10] Xingyu Li and Konstantinos N. Plataniotis. Computational scratch assay - a new frontier for image analysis: Preliminary study of multi-cellular segmentation. *IEEE GlobalSIP 2017*, November 2017.

[11] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Gray scale and rotation invariant texture classification with local binary patterns. *Lecture Notes in Computer Science*, June 2000.

[12] Anne Stamm, Kerstin Reimers, Sarah Strauß, Peter Vogt, Thomas Scheper, and Iliyana Pepelanova. In vitro wound healing assays – state of the art. *BioNanoMat*, 17(1-2):79–87, Apr 2016.

[13] Alejandra Suarez-Arnedo, Felipe Torres Figueroa, Camila Clavijo, Pablo Arbelaéz, Juan C. Cruz, and Carolina Muñoz-Camargo. An image j plugin for the high throughput image analysis of in vitro scratch wound healing assays. *PLoS ONE*, 15, July 2020.

[14] Assaf Zaritsky, Sari Natan, Judith Horev, Inbal Hecht, Lior Wolf, Eshel Ben-Jacob, and Ilan Tsarfaty. Cell motility dynamics: A novel segmentation algorithm to quantify multi-cellular bright field microscopy images. *PLoS ONE*, 6, November 2011.