Referencia del paquete interfaz

Archivo que incluye las funciones y definiciones necesarias para manejar la interfaz de usuario de la herramienta.

Funciones

def **guardarDefaultArgs** ()

Función que guarda los parámetros utilizados en la ejecución como parámetros por defecto.

def **leerImagenes** (path, path_out)

Función que se encarga de leer del disco las imágenes de entrada a la herramienta.

def **guardarImagenes** (imgs, mascaras, path_out)

Función que se encarga de almacenar las imágenes de salida que genera la herramienta.

Variables

parser = argparse.ArgumentParser()
default_args = json.load(json_data)
args = parser.parse_args()
sep = **separacion.separacion**(args.path_model, model, gpu= False)

Descripción detallada

Paquete que incluye las funciones y definiciones necesarias para manejar la interfaz de usuario de la herramienta.

Los parámetros que acepta la interfaz se definen mediante la librería argparse y se utiliza JSON para almacenar los parámetros por defecto. Luego, se definen funciones que se encargan del cargado y almacenamiento de las imágenes de entrada y salida.

Documentación de las funciones

def interfaz.guardarDefaultArgs ()

Función que guarda los parámetros utilizados en la ejecución como parámetros por defecto.

Éstos son almacenados con formato JSON en el archivo 'default_args.json' ubicado en la carpeta 'config' de la herramienta.

def interfaz.guardarImagenes (*imgs*, *mascaras*, *path_out*)

Función que se encarga de almacenar las imágenes de salida que genera la herramienta.

Para cada imagen, se busca en su correspondiente máscara las clases que ésta contiene. Para cada clase, se genera una máscara y se aplica a la imagen para obtener sólo el cromosoma de esa clase. Esta última imagen es almacenada en la carpeta de salida en formato '.png' con un nombre del tipo 'nn_cc.png', donde nn es el número de cluster y cc el número de clase.

Parámetros:

imgs	Lista con las imágenes a guardar en formato de arreglo de NumPy.
mascaras	Lista con las máscaras correspondiente a cada imagen de imgs en formato de arreglo de NumPy.

path_out	String que es el directorio de salida de las imágenes de salida.

def interfaz.leerImagenes (*path*, *path_out*)

Función que se encarga de leer del disco las imágenes de entrada a la herramienta.

Acepta imágenes del tipo [".jpg", ".bmp", ".png", ".tiff", ".jpeg", ".tif"] o un archivo ".txt" en el que haya en cada línea un path a las imágenes con el formato mencionado anteriormente. Esto último es para permitir procesar muchas imágenes a la vez.

Parámetros:

path	Directorio del archivo a procesar. Se verifica que al menos tenga 7 caracteres.
path_out	Directorio de salida, utilizado para generar un path de salida para cada imagen de entrada ya que se crea una carpeta para cada una de ellas.

Devuelve:

Tupla de dos listas. La primera con las imágenes cargadas y la segunda con el path de salida para cada una de ellas.

Referencia del módulo postprocesamiento

Paquete que incluye las funciones necesarias para el post-procesamiento de la predicción obtenida por la red convolucional con el objetivo de mejorar el desempeño de la misma.

Funciones

def **lineal2prob** (img)

Transforma la salida lineal de la red convolucional en probabilidades mediante la función softmax de PyTorch.

def **mayorVecino** (img, maskParcial)

Dada una zona de una imagen, devuelve la clase con más ocurrencias en los vecinos inmediatos.

def **EPI** (imgFinal, tamDespreciable=100)

Función que descarta las zonas menores al tamaño indicado directamente de la imagen y le asigna la clase que predomine en su vecindad inmediata.

def **knn** (img, umbralKNN=.9)

Postprocesamiento usando el algoritmo de k-NN provisto por la librería scikit-learn.

def **CC** (dataNP, umbralCC=0.9)

Funcion utilizada para hacer 0 los canales de las clases que tengan menor probabilidad al umbral y posteriormente corregir las probabilidades.

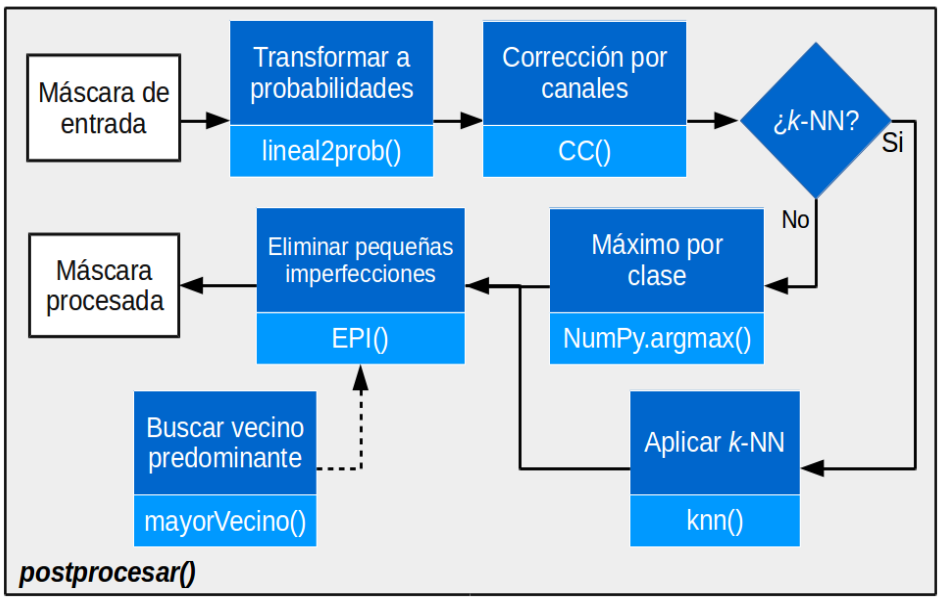
def **postprocesar** (data, umbralCC=0.9, umbralKNN=0, umbralEPI=100)

Función que integra a las demás para cumplir el objetivo del paquete.

Descripción detallada

Paquete que incluye las funciones necesarias para el post-procesamiento de la predicción obtenida por la red convolucional con el objetivo de mejorar el desempeño de la misma.

En la siguiente figura se ve como se interrelacionan las funciones definidas.



Primero se aplica la corrección por canales ‘CC()’ si el umbral ‘umbralCC’ fuera mayor a 0.

Luego, se aplica el algoritmo k-NN ‘knn()’ si ‘umbralKNN’ fuera mayor a 0, sino se elige la clase definitiva para cada píxel simplemente como la que tenga mayor probabilidad.

Por último, se aplica la eliminación de pequeñas imperfecciones ‘EPI()’ si ‘umbralEPI’ fuese mayor a 0.

Documentación de las funciones

def postprocesamiento.CC (*dataNP*, *umbralCC* = `0.9`)

Funcion utilizada para hacer 0 los canales de las clases que tengan menor probabilidad al umbral y posteriormente corregir las probabilidades.

Se determinan las clases confiables como aquellas que tengan al menos un píxel mayor a 'umbralCC'. Luego se hacen 0 los canales de las restantes clases y se recalculan las probabilidades para cada píxel de forma que sumen 1.

Parámetros:

img	Arreglo de NumPy de cuatro dimensiones correspondiente a la salida de la red convolucional.
umbralCC	Umbral utilizado para determinar las "clases confiables".

Devuelve:

Arreglo de NumPy de cuatro dimensiones con los canales de las "clases no confiables" llevados a cero.

def postprocesamiento.EPI (*imgFinal*, *tamDespreciable* = `100`)

Función que descarta las zonas menores al tamaño indicado directamente de la imagen y le asigna la clase que predomine en su vecindad inmediata.

Para ello, genera una máscara para cada clase y busca componentes conexas con el algoritmo 'findContours()' de OpenCV. Luego, analiza cada componente y si es menor al tamaño dado 'tamDespreciable', le asigna una clase indefinida número '24'. Por último, se detectan las componentes conexas que correspondan a dicha clase indefinida '24' y se le asigna la clase mayoritaria en su vecindad mediante '**mayorVecino()**'.

Parámetros:

imgFinal	Arreglo de NumPy de dos dimensiones que contiene en cada píxel la clase a la que éste pertenece.
tamDespreciable	Umbral de tamaño. Se descartan todas las zonas menores a él.

Devuelve:

Arreglo de NumPy de dos dimensiones con las zonas menores al umbral de tamaño reemplazadas por su vecino más concurrente.

def postprocesamiento.knn (*img*, *umbralKNN* = `.9`)

Postprocesamiento usando el algoritmo de k-NN provisto por la librería scikit-learn.

Se entrena k-NN con los píxeles que tengan una mayor probabilidad a 'umbralKNN' y luego se predice la clase de los restantes. Cada píxel tiene 24 valores puesto que posee una probabilidad para cada clase.

Parámetros:

img	Arreglo NumPy de 24 canales con probabilidades por clase.

umbralKNN	Umbral utilizado para determinar los "píxeles confiables".

Devuelve:

Arreglo de NumPy de dos dimensiones que contiene en cada píxel la clase a la que éste pertenece.

def postprocesamiento.lineal2prob (*img*)

Transforma la salida lineal de la red convolucional en probabilidades mediante la función softmax de PyTorch.

Para ello primero convierte la imagen en Tensor de PyTorch y luego reconvierte nuevamente a arreglo de NumPy.

Parámetros:

img	Arreglo 2D de NumPy con la salida de la red convolucional.

Devuelve:

Arreglo de NumPy con probabilidades de pertenencia a cada clase.

def postprocesamiento.mayorVecino (*img*, *maskParcial*)

Dada una zona de una imagen, devuelve la clase con más ocurrencias en los vecinos inmediatos.

Para ello, se aplica la operación morfológica de dilatación para obtener los vecinos de la zona indicada y luego se cuentan las clases mediante la función 'unique' de NumPy.

Parámetros:

img	Arreglo de NumPy de dos dimensiones que contiene en cada píxel la clase a la que éste pertenece.
maskParcial	Arreglo de NumPy de dos dimensiones que indica la zona en la que se quiere evaluar los vecinos. Se toman los píxeles mayores a cero.

Devuelve:

La clase con más ocurrencias en las inmediaciones de la zona indicada.

def postprocesamiento.postprocesar (*data*, *umbralCC* = `0.9`, *umbralKNN* = `0`, *umbralEPI* = `100`)

Función que integra a las demás para cumplir el objetivo del paquete.

Parámetros:

data	Arreglo 4D de NumPy correspondiente a la salida de la red convolucional.
umbralCC	Umbral utilizado para la determinación de "clases confiables" para la corrección por canales. Si es 0 no se hace.
umbralKNN	Umbral utilizado para la determinación de "píxeles confiables" para la corrección mediante el algoritmo de k-nn. Por defecto es 0 y no se hace.
umbralEPI	Umbral que se utiliza en la corrección de pequeñas imperfecciones. Por defecto es 100. Si es 0, no se aplica EPI.

Devuelve:

Arreglo de NumPy de dos dimensiones que contiene en cada píxel la clase a la que éste pertenece.

Referencia del módulo preprocesamiento

Paquete que incluye las funciones necesarias para el pre-procesamiento de la imagen de entrada y la extracción de los cromosomas del fondo.

Funciones

def **preprocesar** (img, tamCuadFondo=10, tamCuadUmbral=[0,100], maxTamAgujero=10, eeSize=7, umbralSegm=30, umbralArea=4000, umbralCH=0.8, TilesGridSize=8, ClipLimit=40)

Función que integra a las demás para cumplir el objetivo del paquete.

def **esNegroFondo** (img, tamCuadFondo=10)

Verifica si el fondo de la imagen es negro o no tomando ROIs de las esquinas de la imagen.

def **realceImagen** (img, TilesGridSize=8, ClipLimit=40)

Función que realza una imagen en escala de grises aplicando CLAHE y luego normalizando en el rango [0-255].

def **umbralAdaptado** (img, tamCuadUmbral=[0,100])

Función que segmenta mediante la combinación de umbrales adaptados de Otsu de distintos tamaños de ventana.

def **eliminarResiduos** (img, eeSize=7)

Función que elimina los pequeños residuos mediante operaciones morfológicas.

def **rellenoAgujeros** (img, menoresA=0)

Función que rellena los agujeros de una imagen binaria utilizando reconstrucción morfológica por dilatación geodésica.

def **compConexas** (img, umbralSegm=30, umbralArea=4000, umbralCH=0.8)

Función analiza cada componente conexa para determinar si corresponde a uno o más cromosomas, o a objetos no deseados.

def **dividirROIs** (data, mask, contours)

Función que se encarga de separar los objetos de una imagen en un vector de subimágenes.

def **eliminarObjBorde** (img, umbralSegm=0)

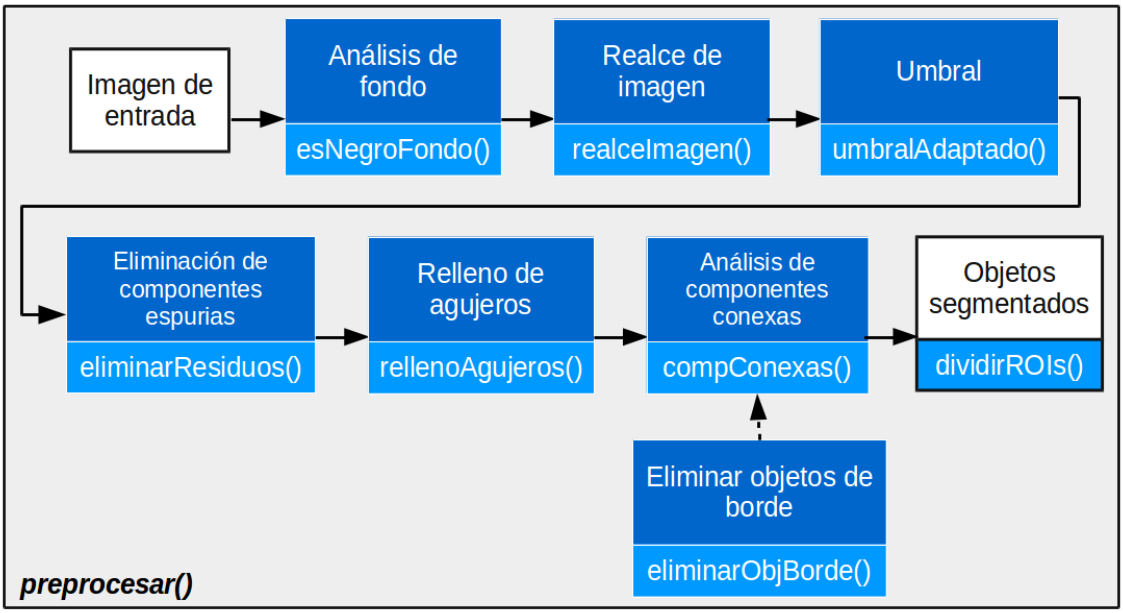
Función que elimina los objetos que están en contacto con el borde de una imagen verificando el tamaño del segmento que pertenece tanto al borde como al objeto.

Descripción detallada

Paquete que incluye las funciones necesarias para el pre-procesamiento de la imagen de entrada y la extracción de los cromosomas del fondo.

Primero se verifica que el fondo sea negro, de lo contrario calcula la inversa de la imagen de entrada 'img', si 'tamCuadFondo' es mayor a 0. Luego, en este orden, se van aplicando las funciones del paquete para realzar la imagen 'realceImagen()', binarizarla con 'umbralAdaptado', eliminar residuos mediante 'eliminarResiduos()' rellenarle los agujeros con 'rellenoAgujeros'. Al resultado, se le analiza las componentes conexas para la eliminación de residuos mediante 'compConexas'. Por último, se devuelven los objetos segmentados en forma de lista con su correspondiente máscara.

En la siguiente imagen se ve como se interrelacionan las funciones definidas.



Documentación de las funciones

def preprocesamiento.compConexas (*img*, *umbralSegm* = `30`, *umbralArea* = `4000`, *umbralCH* = `0.8`)

Función analiza cada componente conexa para determinar si corresponde a uno o más cromosomas, o a objetos no deseados.

Primero se eliminan los objetos del borde con la función **eliminarObjBorde()**. Luego, utilizando la función de detección de bordes de cada componente conexa provista por OpenCV, se analiza el área de cada una de ellas. De ser necesario, se obtiene su convex hull con una función de OpenCV y se calcula la proporción entre ambas áreas para comparar si el objeto debe descartarse o no.

Parámetros:

img	Imagen binaria en formato de arreglo de NumPy de dos dimensiones.
umbralSegm	Umbral que determina el tamaño máximo que puede tener el segmento en contacto con el borde de la imagen de un elemento. Si es mayor, se elimina.
umbralArea	Umbral que determina el tamaño máximo que puede tener un elemento para no ser considerado un posible residuo. Si es mayor, se utiliza el criterio del convex hull.
umbralCH	Umbral que determina la máxima proporción entre el área de un objeto y el área de su convex hull para no ser considerado un residuo. Si es mayor, se elimina.

Devuelve:

Lista que contiene los contornos de las componentes que no son residuos.

def preprocesamiento.dividirROIs (*data*, *mask*, *contours*)

Función que se encarga de separar los objetos de una imagen en un vector de subimágenes.

Para cada componente conexa de contours se calcula el mínimo rectángulo que la contiene con una función de OpenCV, se le aplica la máscara para la eliminación de otros objetos que pueda haber en el rectángulo, se le agrega un margen de dos píxeles a cada lado y luego se devuelven juntas en un vector de imágenes.

Parámetros:

data	Imagen en escala de grises en formato de arreglo de NumPy de dos dimensiones.
mask	Máscara que indica los píxeles que pertecen al fondo y a los objetos.
contours	Lista en la que cada elemento es un vector que contiene los contornos de un objeto.

Devuelve:

Tupla de listas. En la primera, cada elemento es una imagen de un objeto segmentado, mientras que en la segunta está su correspondiente máscara que indica con 255 los píxeles donde está el cromosoma y con 0 donde es fondo.

def preprocesamiento.eliminarObjBorde (*img*, *umbralSegm* = 0)

Función que elimina los objetos que están en contacto con el borde de una imagen verificando el tamaño del segmento que pertenece tanto al borde como al objeto.

Se inicializa la imagen semilla que serán utilizadas para la reconstrucción morfológica por dilatación geodésica mediante la función provista por skimage. Antes de dicha reconstrucción, mediante erosiones con dos elementos estructurantes horizontal y vertical se eliminan de la semilla los elementos que tengan un segmento en contacto con el borde de la imagen menor al parámetro dado. Así, con la reconstrucción mencionada se obtienen los objetos a eliminar de la imagen binaria.

Parámetros:

img	Imagen binaria.
umbralSegm	Umbral que determina el tamaño máximo que puede tener el segmento en contacto con el borde de la imagen de un elemento. Si es mayor, se elimina. Si es 0 elimina todos los objetos que están en el borde.

Devuelve:

Imagen binaria sin los objetos del borde que no cumplen el criterio mencionado.

def preprocesamiento.eliminarResiduos (*img*, *eeSize* = 7)

Función que elimina los pequeños residuos mediante operaciones morfológicas.

Primero se realiza una erosión con un elemento estructurante cuadrado de tamaño eeSize con todos los componentes iguales a 255 mediante una función de OpenCV provista para tal fin. Luego, se lleva a cabo una reconstrucción morfológica por dilatación geodésica mediante una función de skimage.

Parámetros:

img	Imagen binaria en formato de arreglo de NumPy de dos dimensiones
eeSize	Tamaño del elemento estructurante que es utilizado para la eliminación de los residuos pequeños.

Devuelve:

Imagen binaria sin los residuos pequeños.

def preprocesamiento.esNegroFondo (*img*, *tamCuadFondo* = `10`)

Verifica si el fondo de la imagen es negro o no tomando ROIs de las esquinas de la imagen.

Parámetros:

img	Imagen en escala de grises en formato de arreglo de NumPy de dos dimensiones.
tamCuadFondo	Tamaño del cuadrado del fondo que se toma de las esquinas.

Devuelve:

Verdadero si el promedio de los valores de las ROIs es más cercano a 0, de lo contrario Falso.

def preprocesamiento.preprocesar (*img*, *tamCuadFondo* = `10`, *tamCuadUmbral* = `[0, 100]`, *maxTamAgujero* = `10`, *eeSize* = `7`, *umbralSegm* = `30`, *umbralArea* = `4000`, *umbralCH* = `0.8`, *TilesGridSize* = `8`, *ClipLimit* = `40`)

Función que integra a las demás para cumplir el objetivo del paquete.

Primero se verifica que el fondo sea negro, de lo contrario calcula la inversa de la imagen de entrada 'img', si 'tamCuadFondo' es mayor a 0. Luego, en este orden, se van aplicando las funciones del paquete para realzar la imagen 'realceImagen()', binarizarla con 'umbralAdaptado', eliminar residuos mediante 'eliminarResiduos()' rellenarle los agujeros con 'rellenoAgujeros'. Al resultado, se le analiza las componentes conexas para la eliminación de residuos mediante 'compConexas'. Por último, se devuelven los objetos segmentados en forma de lista con su correspondiente máscara.

Parámetros:

img	Imagen en escala de grises.
tamCuadFondo	Tamaño del cuadrado del fondo que se toma de las esquinas. Si es 0, no se verifica el fondo.
tamCuadUmbral	Lista que contiene los tamaños de la ventana cuadrada que se utiliza en el umbral adaptado. Si es 0, calcula el umbral de Otsu sobre toda la imagen.
maxTamAgujero	Tamaño máximo que puede tener un agujero interno a un cromosoma para que se rellene. Cuando es mayor, no se rellena.
eeSize	Tamaño del elemento estructurante que es utilizado para la eliminación de los residuos pequeños.
umbralSegm	Umbral que determina el tamaño máximo que puede tener el segmento en contacto con el borde de la imagen de un elemento. Si es mayor, se elimina. Si es 0 elimina todos los objetos que están en el borde.
umbralArea	Umbral que determina el tamaño máximo que puede tener un elemento para no ser considerado un posible residuo. Si es mayor, se utiliza el criterio del convex hull.
umbralCH	Umbral que determina la máxima proporción entre el área de un objeto y el área de su convex hull para no ser considerado un residuo. Si es mayor, se elimina.
TilesGridSize	Tamaño de las ventanas cuadradas que se aplican para CLAHE.
ClipLimit	Límite para el contraste utilizado en CLAHE.

Devuelve:

Tupla de listas. En la primera, cada elemento es una imagen de un objeto segmentado, mientras que en la segunta está su correspondiente máscara que indica con 255 los píxeles donde está el cromosoma y con 0 donde es fondo.

def preprocesamiento.realceImagen (*img*, *TilesGridSize* = `8`, *ClipLimit* = `40`)

Función que realza una imagen en escala de grises aplicando CLAHE y luego normalizando en el rango [0-255].

Primero se aplica Constrained Limited Adaptive Histogram Equalization (CLAHE), que aplica la ecualización por ROIs limitando la amplificación del contraste. Luego, se normaliza en el rango [0-255] mediante transformaciones afines. Ambas operaciones se realizan con las funciones provistas por OpenCV.

Parámetros:

img	Imagen en escala de grises.
TilesGridSize	Tamaño de las ventanas cuadradas que se aplican para CLAHE.
ClipLimit	Límite para el contraste utilizado en CLAHE.

Devuelve:

Imagen en escala de grises realzada.

def preprocesamiento.rellenoAgujeros (*img*, *menoresA* = `0`)

Función que rellena los agujeros de una imagen binaria utilizando reconstrucción morfológica por dilatación geodésica.

Se calcula el complemento de la imagen y se inicializa la semilla que serán utilizadas para la reconstrucción morfológica por dilatación geodésica mediante la función provista por skimage. Luego, se opera para obtener sólo los agujeros rellenos y se analizan los mismos utilizando la función de detección de bordes de cada componente conexa provista por OpenCV para saber si corresponde llenarlos o no.

Parámetros:

img	Imagen binaria en formato de arreglo de NumPy de dos dimensiones.
menoresA	Tamaño máximo que puede tener un agujero interno a un cromosoma para que se rellene. Cuando es mayor, no se rellena. Si es 0, rellena todos los agujeros.

Devuelve:

Imagen binaria con los agujeros rellenos en formato de arreglo de NumPy de dos dimensiones.

def preprocesamiento.umbralAdaptado (*img*, *tamCuadUmbral* = `[0,100]`)

Función que segmenta mediante la combinación de umbrales adaptados de Otsu de distintos tamaños de ventana.

La segmentación de la imagen se realiza calculando el umbral de Otsu por ventanas usando la función provista por OpenCV para tal fin y luego se interpola para llevar los umbrales al tamaño de la imagen de entrada. Luego, aplica el umbral a cada píxel. Por último, combina los resultados de cada tamaño de ventana utilizado mediante operaciones OR bit a bit entre ellas.

Parámetros:

img	Imagen en escala de grises a segmentar en formato de arreglo de NumPy de dos dimensiones.
tamCuadUmbral	Lista que contiene los tamaños de la ventana cuadrada que se utiliza en el umbral adaptado. Si es 0, calcula el umbral de Otsu sobre toda la imagen.

Devuelve:

Imagen binaria resultante de aplicar los umbrales correspondiente a cada tamaño. El resultado es un OR bit a bit entre cada máscara obtenida.

Referencia del paquete RedW

Paquete que incluye la clase 'RedW' que define la arquitectura de la red W, similar a dos red U conectadas.

Clases

class **RedW**

Descripción detallada

Paquete que incluye la clase 'RedW' que define la arquitectura de la red W, similar a dos red U conectadas.

Dicha red U se determina en 'OverlapSegmentationNet.py'. Se hereda de la clase 'nn.Module' de PyTorch y se sobrecarga la función que se encarga de hacer el pasaje hacia adelante de la imagen 'forward()'.

Referencia del paquete segmentador

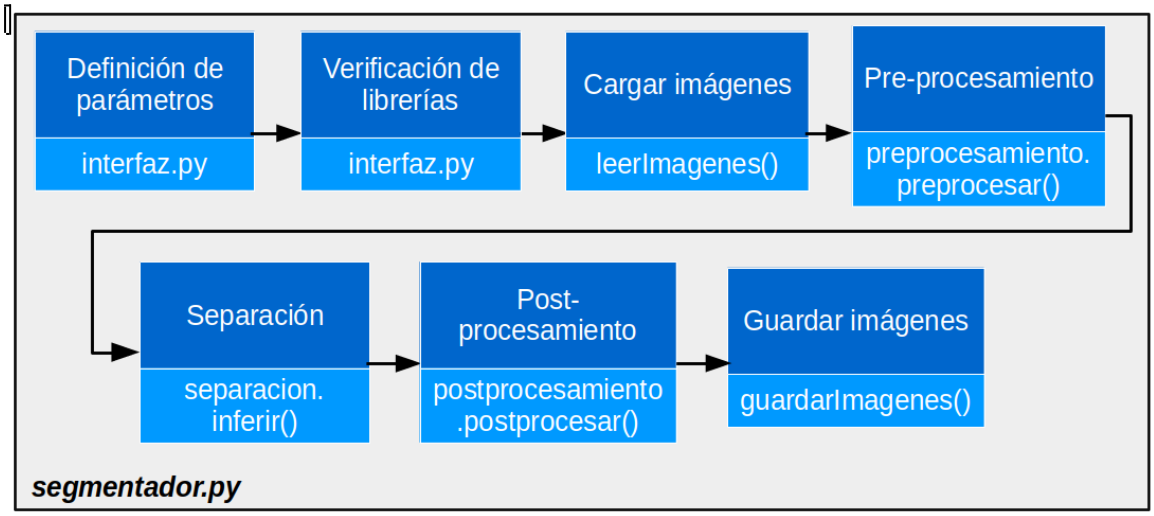
Archivo que incluye a los demás módulos para la integración y ejecución de la herramienta.

Descripción detallada

Archivo que incluye a los demás módulos para la integración y ejecución de la herramienta.

Primero define los parámetros mediante 'interfaz.py' para luego carga las imágenes de entrada con '**interfaz.leerImagenes()**'. Para cada imagen, conecta los módulos 'preprocesamiento', 'separacion' y 'postprocesamiento' pasando la salida de uno a la entrada del siguiente. A continuación, guarda las imágenes de salida utilizando la funcion '**interfaz.guardarImagenes()**'. El proceso se repite si hubiese más imágenes para finalmente guardar los parámetros por defecto mediante 'interfaz.saveDefaultArgs()' si así se pidiese.

En la siiguiente imagen se ve cómo se relacionan los bloques que lo componen.



Referencia del módulo separacion

Archivo que incluye la clase separacion, necesaria para separar un solapamiento de cromosomas mediante la aplicación de una red convolucional.

Clases

class **separacion**

Descripción detallada

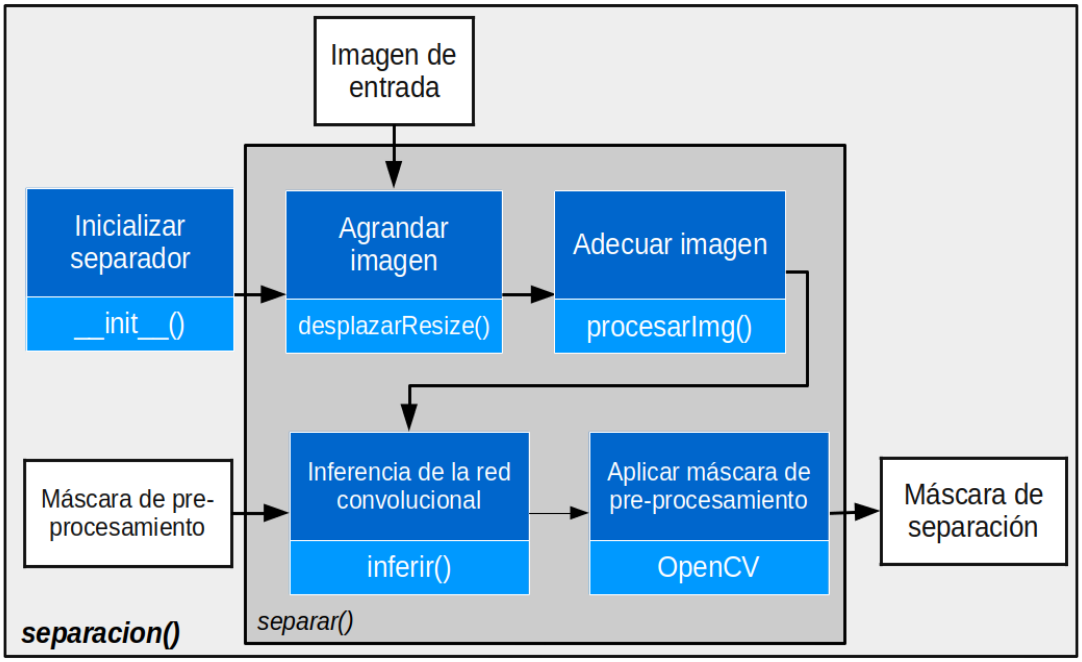
Paquete que incluye la clase separacion, necesaria para separar un solapamiento de cromosomas mediante la aplicación de una red convolucional.

Se inicializa la arquitectura de la red, se cargan sus parámetros entrenados y se manda a la GPU si se pidiera y pudiera en '__init__()'.

Luego se incluyen métodos para asemejar la imagen de entrada a los datos sintéticos generados 'procesarImg()' si así se quisiese, para redimensionar la imagen 'desplazarResize()' y para pasar

la imagen por la red convolucional ‘inferir()’. Por último, el método 'separar()' combina todas las funciones anteriores.

En la siguiente figura se ve cómo diseñó el mismo.



Referencia de la Clase OverlapSegmentationNet.OverlapSegmentationNet

Métodos públicos

def **__init__** (self, canalesEntrada=1)

Constructor que define la estructura de la red convolucional.

def **forward** (self, x)

Sobrecarga de la función homóloga de PyTorch que realiza la pasada hacia adelante por la red convolucional.

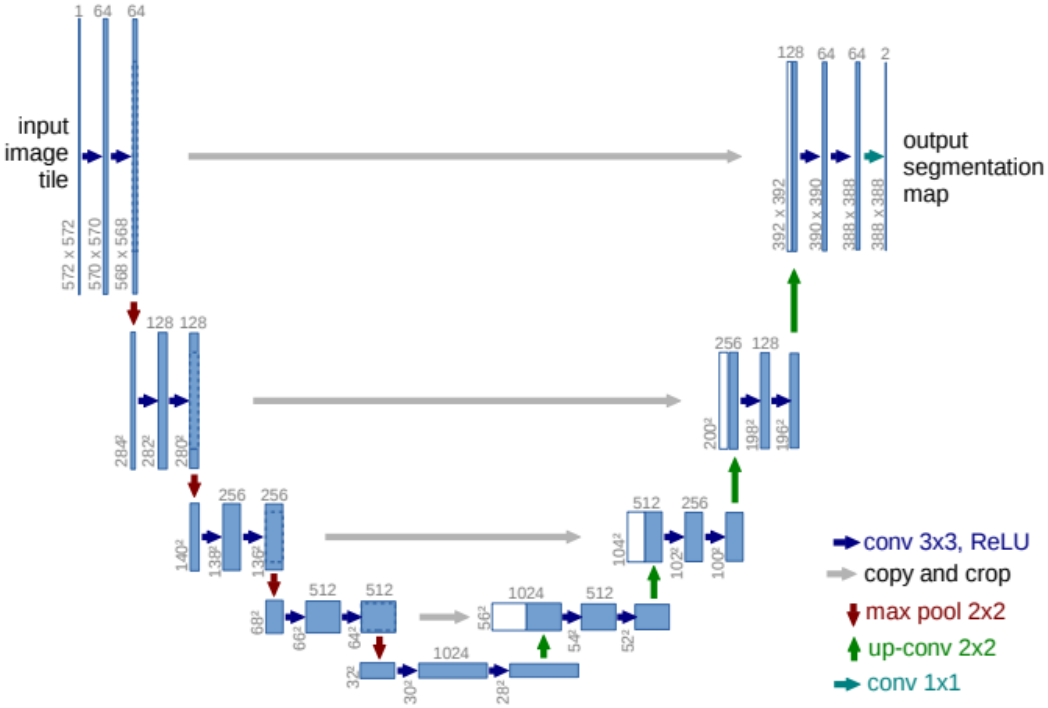
Atributos públicos

canalesEntrada: determina cuantos canales de entrada se definirán en la primera capa del modelo.

Documentación del constructor y destructor

def OverlapSegmentationNet.OverlapSegmentationNet.__init__ (*self*, *canalesEntrada* = 1)

Constructor que define la estructura de la red convolucional. Ésta es similar a la red U de la siguiente imagen.



Documentación de las funciones miembro

def OverlapSegmentationNet.OverlapSegmentationNet.forward (*self*, *x*)

Sobrecarga de la función homóloga de PyTorch que realiza la pasada hacia adelante por la red convolucional.

Se encarga de determinar cómo se interrelacionan entre sí las capas definidas anteriormente en el constructor.

Parámetros:

x	Entradas en formato Tensor de 4 dimensiones de PyTorch. La primera corresponde a la cantidad de datos, la segunda a la cantidad de canales y las últimas dos al tamaño de la imagen.

Devuelve:

Salida de la red convolucional en formato Tensor de 4 dimensiones de PyTorch. La primera corresponde a la cantidad de datos, la segunda a la cantidad de canales (24 por la cantidad de clases) y las últimas dos al tamaño de la imagen.

La documentación para esta clase fue generada a partir del siguiente fichero:

0 OverlapSegmentationNet.py

Referencia de la Clase RedW.RedW

Métodos públicos

def **__init__** (self)

Constructor que define la estructura de la red convolucional.

Documentación del constructor y destructor

def RedW.RedW.__init__ (*self*)

Constructor que define la estructura de la red convolucional. La misma consta de dos redes U definidas en 'OverlapSegmentationNet.py', en la que se conecta la salida de una a la entrada de la otra.

Documentación de las funciones miembro

def RedW.RedW.forward (*self*, *x*)

Sobrecarga de la función homóloga de PyTorch que realiza la pasada hacia adelante por la red convolucional.

Se encarga de determinar cómo se interrelacionan entre sí las capas definidas anteriormente en el constructor y de normalizar los datos con una media de 0.5 y un desvío de 0.5, mediante la función Normalize() de la librería torchvision.

Parámetros:

x	Entradas en formato Tensor de 4 dimensiones de PyTorch. La primera corresponde a la cantidad de datos, la segunda a la cantidad de canales y las últimas dos al tamaño de la imagen.

Devuelve:

Salida de la red convolucional en formato Tensor de 4 dimensiones de PyTorch. La primera corresponde a la cantidad de datos, la segunda a la cantidad de canales (24 por la cantidad de clases) y las últimas dos al tamaño de la imagen.

La documentación para esta clase fue generada a partir del siguiente fichero:

1 RedW.py

Referencia de la Clase separacion.separacion

Métodos públicos

def **__init__** (self, path, model, gpu=True)

Constructor de la clase, encargado de cargar los parámetros entrenados de un modelo dado.

def **histGauss** (self, source, sigma=60)

Función que aplica histogram matching con una gaussiana de media 128 y desvío indicado.

def **desplazarResize** (self, img, tamImagenSalida, fondo=0)

Función que agranda la imagen que recibe al tamaño especificado, dejando en el centro la primera.

def **procesarImg** (self, img, std=60, blur=0.7, invertirImg=True, fondo=0)

Procesa una imagen de un cromosoma calculando su inversa, aplicando histogram matching y un filtro gaussiano de 3x3.

def **inferir** (self, img)

Dada una imagen, devuelve su segmentación mediante el pasaje de la misma por la red convolucional.

def **separar** (self, imgs, imgs_mask=[], tamImagen=(256, 256), std=60, blur=0.7, invertirImg=True)

Función integradora de las demás del paquete.

Atributos públicos

model: red convolucional utilizada para la separación.

Documentación del constructor y destructor

def separacion.separacion.__init__ (*self*, *path*, *model*, *gpu* = `True`)

Constructor de la clase, encargado de cargar los parámetros entrenados de un modelo dado.

Tener en cuenta que el modelo debe estar guardado en el formato que se indica en el parámetro. Esto es por compatibilidad con el proyecto utilizado para el entrenamiento de la red convolucional.

Parámetros:

path	Path del archivo que contiene los parámetros del modelo. Debe estar en un diccionario con la clave "state_dict" y guardado serialmente (idealmente mediante la función de pytorch).
model	Modelo de pytorch al cual se le cargarán los parámetros entrenados.

Devuelve:

Modelo de pytorch con los parámetros cargados.

Documentación de las funciones miembro

def separacion.separacion.desplazarResize (*self*, *img*, *tamImagenSalida*, *fondo* = `0`)

Función que agranda la imagen que recibe al tamaño especificado, dejando en el centro la primera.

Si la imagen fuera mayor al tamaño deseado, éste se duplica y se intenta nuevamente.

Parámetros:

img	Imagen a agrandar en formato de arreglo numpy de dos dimensiones.
tamImagenSalida	Tupla que indica el tamaño de salida deseado.
fondo	Indica el color del fondo para rellenar la imagen agrandada.

Devuelve:

Imagen agrandada en formato de arreglo numpy de dos dimensiones.

def separacion.separacion.histGauss (*self*, *source*, *sigma* = `60`)

Función que aplica histogram matching con una gaussiana de media 128 y desvío indicado.

Parámetros:

source	Imagen o píxeles a los que se aplicará.
sigma	Desvío de la gaussiana que se utilizará.

Devuelve:

Imagen o píxeles corregidos según la gaussiana.

def separacion.separacion.inferir (*self*, *img*)

Dada una imagen, devuelve su segmentación mediante el pasaje de la misma por la red convolucional.

También se encarga de llevar la imagen del rango [0,255] al rango [0,1], del pasaje de numpy a tensor de pytorch para la inferencia y del pasaje inverso para la devolución del resultado.

Parámetros:

img	Imagen a segmentar en formato de arreglo numpy de dos dimensiones.

Devuelve:

Imagen de dos dimensiones indicando en cada píxel el número de clase.

def separacion.separacion.procesarImg (*self*, *img*, *std* = `60`, *blur* = `0.7`, *invertirImg* = `True`, *fondo* = `0`)

Procesa una imagen de un cromosoma calculando su inversa, aplicando histogram matching y un filtro gaussiano de 3x3.

Parámetros:

img	Imagen a segmentar en formato de arreglo numpy de dos dimensiones.
std	Desvío utilizado para aplicar histogram matching con una gaussiana.
blur	Desvío utilizado en el filtro gaussiano.
invertirImg	Booleano que indica si se calcula la inversa de la imagen o no.
fondo	Indica el color del fondo para excluirlo del calculo del histogram matching.
Imagen	de dos dimensiones indicando en cada píxel el número de clase.

def separacion.separacion.separar (*self*, *imgs*, *imgs_mask* = `[]`, *tamImagen* = `(256,256)`, *std* = `60`, *blur* = `0.7`, *invertirImg* = `True`)

Función integradora de las demás del paquete.

No solo conecta las demás funciones, sino que también se encarga de manejar múltiples inferencias sin tener que recargar el modelo.

Parámetros:

imgs	Lista de imágenes a segmentar.
imgs_mask	Lista de máscaras de las imágenes a segmentar. Si la lista tiene longitud distinta a imgs, no se aplica.
tamImagen	Tupla que indica el tamaño al que se agrandará la imagen antes de pasar por la red convolucional.
std	Desvío utilizado para histogram matching. Si es 0 no se aplica.
blur	Desvío utilizado para aplicar filtro gaussiano de 3x3. Si es 0 no se aplica.

Devuelve:

Tupla con dos listas. Una de la imagen original ampliada y otra de imágenes de dos dimensiones indicando en cada píxel el número de clase.

La documentación para esta clase fue generada a partir del siguiente fichero:

2 separacion.py