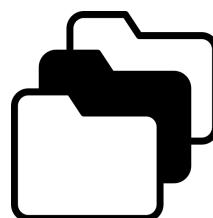


Entendimiento de la data

Fully automatic wound segmentation with deep convolutional neural networks

Grupo 4

2024-2



Descripción

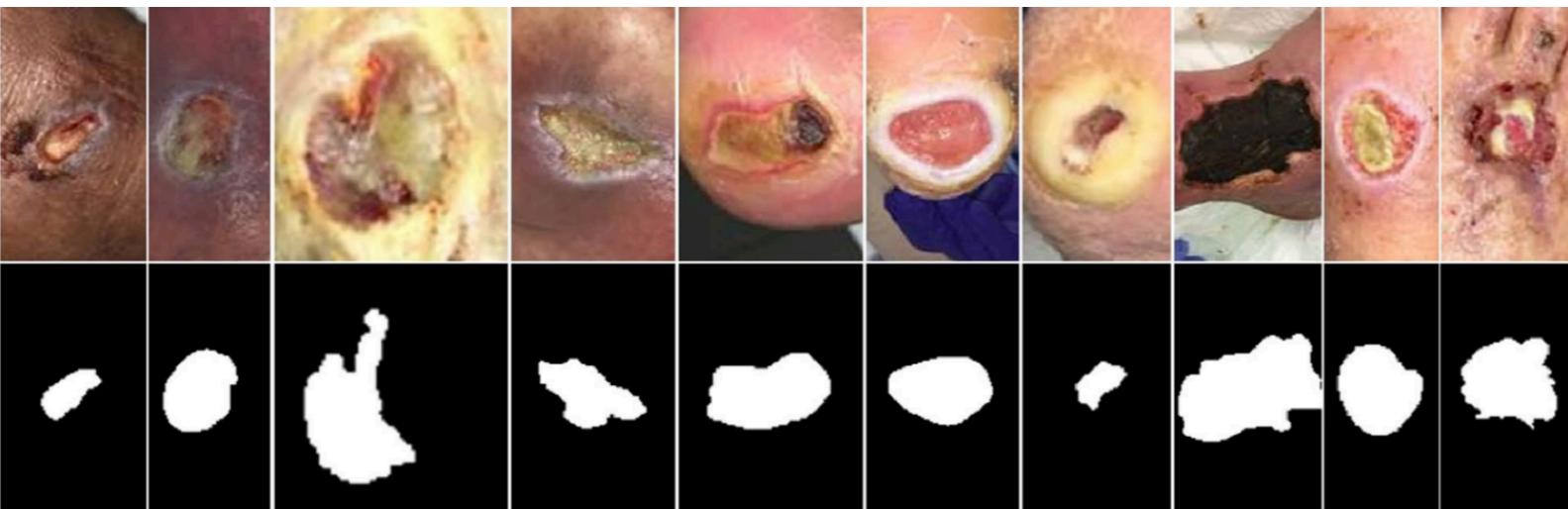
El dataset sobre heridas crónicas a utilizar incluye 1109 imágenes de úlceras en el pie tomadas de 889 pacientes, en colaboración con el Centro de Heridas y Vascular para Promover el Cenit de la Atención Sanitaria (AZH).

Construcción

Unificación de Tamaño	Las imágenes fueron unificadas en tamaño utilizando un modelo de localización de objetos YOLOv3 para colocar cuadros delimitadores alrededor de las heridas.
Ampliación del Dataset	Se aplicaron transformaciones a las imágenes para generar variaciones, creando un conjunto de entrenamiento de 3645 imágenes y un conjunto de prueba de 405 imágenes.
Etiquetado	Se utilizaron herramientas como LabelImg para etiquetar manualmente todas las imágenes, siguiendo el formato de YOLO.
Anotación Manual	Las imágenes fueron anotadas manualmente con máscaras de segmentación, revisadas y validadas por especialistas en cuidado de heridas.

Preprocesamiento

Recorte y Relleno	Las imágenes son recortadas alrededor de las heridas y luego ajustadas a un tamaño estándar de 224x224 píxeles mediante el relleno con ceros
Aumento de Datos	Mejorar la generalización para ser mas preciso con sus predicciones, reducción del overfitting, puede aprender demasiado bien las características específicas del conjunto de entrenamiento pero fallar al predecir con precisión en nuevas imágenes, al aumentar el conjunto de datos, introduce variabilidad que ayuda a mitigar este problema. y simulación de condiciones reales. Rotaciones aleatorias, volteos horizontales y verticales, y zoom aleatorio.
Transformaciones No Rígidas	Se utilizó el zoom aleatorio como la única transformación no rígida, ya que se sospecha que otras transformaciones, como los cizallamientos, no representan variaciones comunes en la forma de las heridas.
Generación de Dataset de Entrenamiento	El dataset de entrenamiento fue aumentado hasta alcanzar aproximadamente 5000 imágenes, mientras que el conjunto de



linear_intensity_normalization

standardization_intensity_normalization

intensityNormalisationFeatureScaling

intensityMaxClipping

intensityProjection

```
Code Blame 34 lines (28 loc) · 1.07 KB ⚙ Code 55% faster with GitHub Copilot
Para salir de la pantalla completa presiona Esc
1 # -----
2 #
3 # file : preprocessing/normalisation.py
4 # author : CM
5 #
6 # -----
7 import numpy as np
8
9 # Rescaling (min-max normalization)
10 def linear_intensity_normalization(loader_dataset):
11     loader_dataset = (loader_dataset / loader_dataset.max())
12     return loader_dataset
13
14 # Preprocess dataset with intensity normalisation
15 # (zero mean and unit variance)
16 def standardization_intensity_normalization(dataset, dtype):
17     mean = dataset.mean()
18     std = dataset.std()
19     return ((dataset - mean) / std).astype(dtype)
20
21 # Intensities normalized to the range [0, 1]
22 def intensityNormalisationFeatureScaling(dataset, dtype):
23     max = dataset.max()
24     min = dataset.min()
25
26     return ((dataset - min) / (max - min)).astype(dtype)
27
28 # Intensity max clipping with c "max value"
29 def intensityMaxClipping(dataset, c, dtype):
30     return np.clip(a=dataset, a_min=0, a_max=c).astype(dtype)
31
32 # Intensity projection
33 def intensityProjection(dataset, p, dtype):
34     return (dataset ** p).astype(dtype)
```

```
getThreshold(dataset_mra, dataset_gd)
```

```
thresholding(data, threshold)
```

Carga de las imágenes y cálculo del umbral

Aplicación del umbral a las imágenes de prueba y entrenamiento

```
Code Blame 109 lines (88 loc) · 3.21 KB Code 55% faster with GitHub Copilot
```

```
8     import os
9     import sys
10    import numpy as np
11    import nibabel as nib
12    from utils.io.write import npToNii
13    from utils.config.read import readConfig
14
15    # Get the threshold for preprocessing
16    def getThreshold(dataset_mra, dataset_gd):
17        threshold = dataset_mra.max()
18
19        for i in range(0, len(dataset_mra)):
20            mra = dataset_mra[i]
21            gd = dataset_gd[i]
22
23            for x in range(0, mra.shape[0]):
24                for y in range(0, mra.shape[1]):
25                    for z in range(0, mra.shape[2]):
26                        if(gd[x,y,z] == 1 and mra[x,y,z] < threshold):
27                            threshold = mra[x,y,z]
28
29        return threshold
30
31    # Apply threshold to an image
32    def thresholding(data, threshold):
33        output = np.copy(data)
34        for x in range(0, data.shape[0]):
35            for y in range(0, data.shape[1]):
36                for z in range(0, data.shape[2]):
37                    if data[x,y,z] > threshold:
38                        output[x,y,z] = data[x,y,z]
39                    else:
40                        output[x,y,z] = 0
41
42        return output
43
44    config_filename = sys.argv[1]
45    if(not os.path.isfile(config_filename)):
46        sys.exit(1)
47
48    config = readConfig(config_filename)
```