جامعة الحسن الأول
UNIVERSITÉ HASSAN 1er

Université Hassan 1 er

Ecole national des sciences applique Berrechid

ENSA Ecole Nationale des Sciences Appliquées BERRECHID

Compte rendu du TP : Analyse d'annotations d'images de dents et mise en place d'un modèle pour la prédiction des angles de dépouille

Module : Fouille de données

Filière & Semestre : CI-ISIBD/S9

Année Universitaire : 2023-2024

Encadrant Pédagogique :

Pr. HRIMECH HAMID

Réalisé par :

FEKKAK HIND

Table des matières

I. Analyse des données du DataFrame :

A. Définir le Répertoire et les Éléments :

Changer le répertoire courant vers celui spécifié ('est').

Définir une liste d'éléments XML à extraire, comprenant les détails de l'image et les positions des pièces.

```
import pandas as pd
import xml.etree.ElementTree as ET
import glob
import os
_dir = 'est'
os.chdir(_dir)
elements = ['Image', 'height', 'width', 'depth', 'coin_1', 'coin_2', 'coin_3', 'coin_4']
df = pd.DataFrame(columns=elements)
```

B. Initialiser les Listes et Iterer sur les Fichiers XML :

Initialiser des listes vides pour stocker les informations sur les pièces (coin_names et coin_values).

Itérer sur tous les fichiers XML dans le répertoire.

```
# Initialize empty lists to store coin information
coin_names = []
coin_values = {}
# Iterate over XML files in the directory
for xml_file in glob.glob('*.xml'):
    if xml_file.endswith('.xml'):
        # Parse the XML file
        tree = ET.parse(os.path.join(_dir, xml_file))
        root = tree.getroot()
```

C. Extraire des Informations Basiques de Chaque Fichier XML :

Extraire le nom du fichier image, la largeur, la hauteur et la profondeur du fichier XML.

```
# Extract the annotation file name
    file_name = root.find('filename').text

    # Extract size information
    width = int(root.find('size/width').text)
    height = int(root.find('size/height').text)
    depth = int(root.find('size/depth').text)
```

D. Itérer sur les Éléments 'Object' et Extraire les Informations sur les Pièces :

Itérer sur les éléments 'object' dans le fichier XML.

Extraire le nom de la pièce, xmin, ymin, xmax et ymax pour chaque pièce.

Stocker ces informations dans le dictionnaire coin_values.

```
    # Iterate over the 'object' elements
    for obj in root.iter('object'):
        # Extract the coin name
        coin_name = obj.find('name').text

        # Extract the coin value (xmin, ymin, xmax, ymax)
        bbox = obj.find('bndbox')
        xmin = int(bbox.find('xmin').text)
        ymin = int(bbox.find('ymin').text)
        xmax = int(bbox.find('xmax').text)
        ymax = int(bbox.find('ymax').text)

        # Store the coin value in the dictionary
        coin_values[coin_name] = (xmin, xmax, ymin, ymax)
```

E. Créer un DataFrame et Ajouter les Données :

Créer un DataFrame temporaire (tmp_df) à partir des informations extraites.

Définir les colonnes 'Image', 'height', 'width' et 'depth'.

Ajouter les données au DataFrame principal (df).

```
    tmp_df = pd.DataFrame([coin_values], columns=['Image', 'height', 'width', 'depth'] + list(coin_values.keys()))

    # Set the 'filename' column value

    tmp_df['Image'] = file_name

    tmp_df['height'] = height

    tmp_df['width'] = width

    tmp_df['depth'] = depth

    # Append the data to the DataFrame

    df = pd.concat([df, tmp_df], ignore_index=True)
```

F. Afficher le Résumé du DataFrame :

Enfin, afficher les statistiques sommaires du DataFrame.

```
# Print the DataFrame

print(df.describe)
```

→Ce script suppose une structure spécifique dans les fichiers XML, où chaque fichier représente une image avec des pièces associées, et chaque pièce a un nom et des coordonnées de boîte englobante. Le DataFrame résultant aura des colonnes pour les détails de l'image et la position de chaque pièce. Le script imprime les statistiques sommaires du DataFrame, fournissant des informations sur la distribution des données.

II. Vérification des erreurs dans les coordonnées des coins

Le code vérifie s'il y a des erreurs dans les coordonnées des coins en comparant les valeurs y_min et y_max. Aucune erreur n'est détectée, comme indiqué par les DataFrames vides renvoyés pour chaque coin.

```
print([df[df[f'coin_{i}'].apply(lambda x: x[2]) != df[f'coin_{i}'].apply(lambda x: x[3])] for i in range(1, 5)])

print(df[df.height < 448])
```

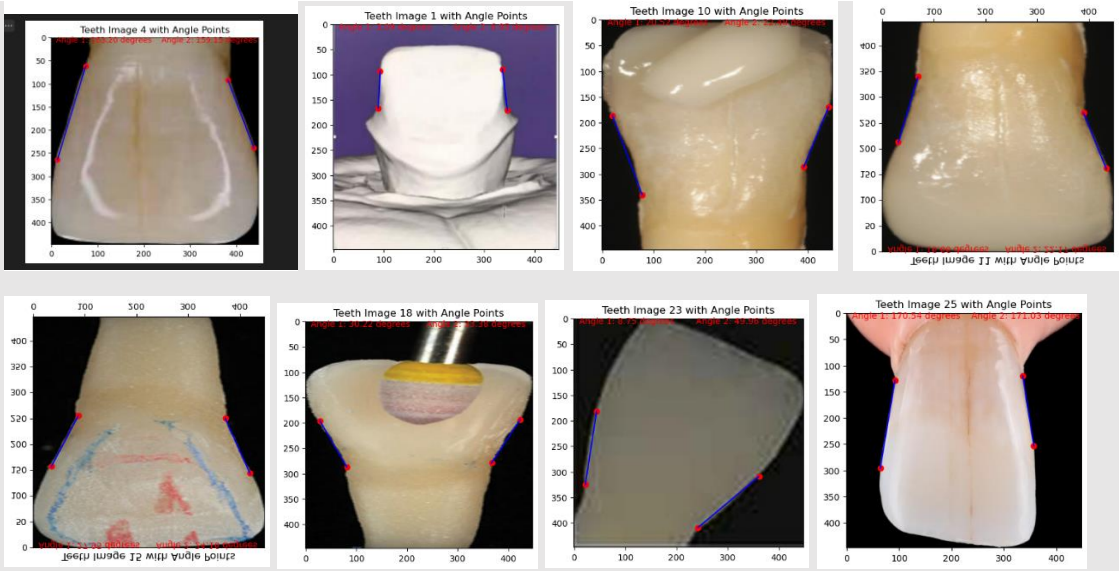
III. Visualisation des images avec les points d'angle

Les images sont visualisées avec les points d'angle représentant les coins de chaque dent.

Des lignes sont tracées entre les points pour visualiser les angles de dépouille.

Les points d'angle sont affichés en rouge, les lignes entre les angles en bleu et une ligne verticale verte est tracée à partir du premier point d'angle vers le bas de l'image.

```
import matplotlib.pyplot as plt
import cv2
images = df.Image
# Visualize images and their corresponding angle points
for i in range(len(images)):
    image = cv2.imread(images[i])
    angle = [df.loc[i, f'coin_{x}'] for x in range(1, 5)]
    # Plot image with angle points
    plt.figure()
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    for p in angle:
        plt.scatter([p[0], p[1]], [p[2], p[3]], color='red')  # Plot the x and y coordinates separately
        # Draw a line between the two points
        plt.plot()
    # Draw lines between the angle points
    for j in [0, 2]:
        x_coords = [angle[j][0], angle[j + 1][0]]  # x-coordinates
        y_coords = [angle[j][2], angle[j + 1][2]]  # y-coordinates
        x_c00rds = [angle[j][0], angle[j][0]]
        y_c00rds = [448, 0]
        plt.plot(x_coords, y_coords, color='blue')
        plt.plot(x_c00rds, y_c00rds, color='green')
    plt.title('Teeth Image with Angle Points')
    plt.show()
```



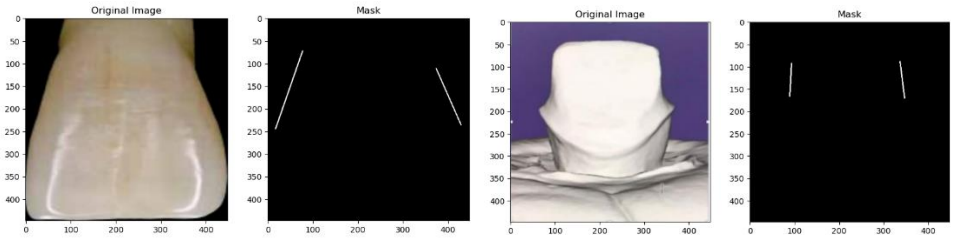

```
import cv2
import matplotlib.pyplot as plt
# Load images and masks
images = []
masks = []
for im in df.Image:
    mask_path = f'masks/mask_{im[:-4]}.jpg'

    image = cv2.imread(im)
    mask = cv2.imread(mask_path)
    images.append(image)
    masks.append(mask)
# Display images and masks side by side
for image, mask in zip(images, masks):
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

    # Display the original image
    axes[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    axes[0].set_title('Original Image')

    # Display the mask
    axes[1].imshow(mask, cmap='gray')
    axes[1].set_title('Mask')

    plt.show()
```



IV. Augmentation des données :

Le code utilise TensorFlow pour créer un générateur d'images avec des augmentations telles que la rotation, le décalage horizontal et vertical, et la bascule horizontale et verticale. Ces augmentations sont appliquées aux images et aux masques correspondants.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Create an image data generator with desired augmentations
data_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='constant',
    cval=0
)
# Generate augmented images and masks
augmented_images = []
augmented_masks = []
for image, mask in zip(images, masks):
    image_batch = np.expand_dims(image, axis=0)
    mask_batch = np.expand_dims(mask, axis=0)
    # Apply data augmentation to images
    augmented_image_batch = data_generator.flow(image_batch, batch_size=1, shuffle=False)
    # Apply data augmentation to masks (use the same seed for consistency)
    augmented_mask_batch = data_generator.flow(mask_batch, batch_size=1, shuffle=False)
    augmented_image = next(augmented_image_batch)[0]
    augmented_mask = next(augmented_mask_batch)[0]
```

```
    augmented_images.append(augmented_image)
    augmented_masks.append(augmented_mask)
augmented_images = np.array(augmented_images)
augmented_masks = np.array(augmented_masks)
# Check the shapes of augmented images and masks
print(f'Augmented Images Size : {augmented_images.size}, Shape : {augmented_images.shape}')
print(f'Augmented Masks Size : {augmented_masks.size}, Shape : {augmented_masks.shape}')
```

```
Augmented Images Size : 28901376, Shape : (48, 448, 448, 3)
Augmented Masks Size : 28901376, Shape : (48, 448, 448, 3)
```

V. Construction du modèle :

Un modèle U-Net est construit pour prédire les angles de dépouille des images de dents. Le modèle comprend une architecture d'encodeur-décodeur, et il est compilé avec la fonction de perte binaire 'binary_crossentropy'. Le modèle est ensuite entraîné sur les images augmentées.

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate
def build_unet(input_shape):
    inputs = Input(input_shape)
    # Encoder
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    # Decoder
    up2 = UpSampling2D(size=(2, 2))(pool1)
    up2 = Conv2D(64, 2, activation='relu', padding='same')(up2)
    merge2 = concatenate([conv1, up2], axis=3)
    conv2 = Conv2D(64, 3, activation='relu', padding='same')(merge2)
```

```
    conv2 = Conv2D(64, 3, activation='relu', padding='same')(conv2)
    outputs = Conv2D(1, 1, activation='sigmoid')(conv2)
    model = Model(inputs=inputs, outputs=outputs)
    return model
# Build the U-Net model
input_shape = images[0].shape
model = build_unet(input_shape)
# Modify the masks to have a single channel
augmented_masks_single_channel = augmented_masks[..., 0:1]
# Compile the model
model.compile(optimizer='adam', loss=')
```

VI. Résultats de l'entraînement du modèle

Les résultats d'entraînement du modèle U-Net sont affichés, montrant les valeurs de perte pour chaque époque. La perte de validation est également surveillée pour évaluer la performance du modèle sur des données non vues.

```
Epoch 1/10
5/5 [==============================] - 434s 83s/step - loss: 8.1781 - val_loss: 4.4443
Epoch 2/10
5/5 [==============================] - 381s 76s/step - loss: 2.4653 - val_loss: 3.3567
Epoch 3/10
5/5 [==============================] - 379s 77s/step - loss: 1.5102 - val_loss: 1.1518
Epoch 4/10
5/5 [==============================] - 390s 79s/step - loss: 0.7155 - val_loss: 0.9183
Epoch 5/10
5/5 [==============================] - 384s 75s/step - loss: 0.6190 - val_loss: 0.7720
Epoch 6/10
5/5 [==============================] - 372s 74s/step - loss: 0.4876 - val_loss: 1.1563
Epoch 7/10
5/5 [==============================] - 379s 76s/step - loss: 0.2955 - val_loss: 1.4839
Epoch 8/10
5/5 [==============================] - 376s 75s/step - loss: -0.0531 - val_loss: 2.0707
Epoch 9/10
5/5 [==============================] - 378s 75s/step - loss: -1.3355 - val_loss: 4.5477
Epoch 10/10
5/5 [==============================] - 367s 74s/step - loss: -4.3945 - val_loss: 8.2508
<keras.src.callbacks.History at 0x1afd5811210>
```