

Université Hassan 1er
Ecole Nationale des Sciences Appliquées
- BERRECHID -



Rapport de TP Détection des angles de convergences pour la création d'une couronne en céramique dentaire

Réalisé par | Chaimaa OUKNIDER

Encadré par | Pr. Hamid HRIMECH

5eme année en Ingénierie des systèmes d'information et Big Data
Année Universitaire : 2023-2024

Introduction et problématique :

L'angle de convergence est traditionnellement défini comme l'angle formé par les surfaces des parois d'une cavité préparée dans une dent. Cette mesure revêt une importance fondamentale dans la planification et l'exécution de restaurations dentaires, influençant directement la qualité et la durabilité des interventions.

Dans ce rapport, nous proposons une approche basée sur l'apprentissage automatique pour la détection des angles de convergences des dents en utilisant Python.

Au cours de ce travail pratique, nous explorerons le processus d'extraction de données pertinentes à partir d'images annotées, en mettant particulièrement l'accent sur la détermination précise de l'angle de convergence sur le bord des images dentaires. La méthodologie implique également la création et l'entraînement d'un modèle U-Net personnalisé, spécifiquement conçu pour cette tâche.

Le DataSet :

Nous utilisons un jeu de données contenant des informations sur les images dentaires, y compris les coordonnées des coins de chaque dent. Pour intégrer ces informations provenant des fichiers XML dans un DataFrame, vous pouvez analyser chaque fichier XML et extraire les dimensions, les coordonnées des coins et le nom de l'image. Ensuite, vous pouvez créer un DataFrame vide avec les colonnes appropriées et ajouter les informations extraites au DataFrame. Cela vous permettra de stocker et d'organiser les données des images dentaires dans une structure de tableau facile à manipuler. En utilisant ce DataFrame, vous pourrez effectuer des analyses et des traitements sur les données, ce qui peut être utile pour la recherche dentaire et l'élaboration de stratégies de traitement

Itération sur les Fichiers XML

```
# Initialisation de listes vides pour stocker les informations sur les pièces
coin_names = []
coin_values = {}
# Itération sur les fichiers XML dans le répertoire
for xml_file in glob.glob('*.xml'):
    if xml_file.endswith('.xml'):
        # Analyse du fichier XML
        tree = ET.parse(os.path.join(_dir, xml_file))
        root = tree.getroot()
        # Extraction du nom du fichier
        file_name = root.find('filename').text
        # Extraction des informations de taille
        width = int(root.find('size/width').text)
        height = int(root.find('size/height').text)
        depth = int(root.find('size/depth').text)
```

```
# Itération sur les éléments 'object'
for obj in root.iter('object'):
    # Extraction du nom de la pièce
    coin_name = obj.find('name').text
    # Extraction de la valeur de la pièce (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)
    # Stockage de la valeur de la pièce dans le dictionnaire
    coin_values[coin_name] = (xmin, xmax, ymin, ymax)
```

Output :

```
.. <bound method NDFrame.describe of
```

	Image	height	width	depth	coin_
0	0.jpg	448	448	3	(77, 77, 73, 73) (17, 17, 245, 245)
1	1.jpg	448	448	3	(89, 89, 167, 167) (93, 93, 93, 93)
2	10.jpg	448	448	3	(78, 78, 341, 341) (20, 20, 186, 186)
3	11.jpg	448	448	3	(69, 69, 340, 340) (31, 31, 213, 213)
4	12.jpg	448	448	3	(72, 72, 338, 338) (32, 32, 209, 209)
5	13.jpg	448	448	3	(74, 74, 277, 277) (26, 26, 163, 163)
6	14.jpg	448	448	3	(100, 100, 284, 284) (42, 42, 169, 169)
7	15.jpg	448	448	3	(87, 87, 256, 256) (35, 35, 158, 158)
8	16.jpg	448	448	3	(101, 101, 249, 249) (39, 39, 147, 147)
9	17.jpg	448	448	3	(66, 66, 262, 262) (19, 19, 171, 171)
10	18.jpg	448	448	3	(81, 81, 287, 287) (28, 28, 196, 196)
11	19.jpg	448	448	3	(72, 72, 314, 314) (17, 17, 233, 233)
12	2.jpg	448	448	3	(56, 56, 131, 131) (61, 61, 41, 41)
13	20.jpg	448	448	3	(72, 72, 243, 243) (23, 23, 149, 149)
14	21.jpg	448	448	3	(92, 92, 211, 211) (44, 44, 247, 247)

Vérification des Coordonnées des Pièces et Filtrage par Hauteur

```
# Vérification des coordonnées des pièces pour chaque pièce de 1 à 4
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)])
# Filtrage du DataFrame par hauteur inférieure à 448 pixels
print(df[df.height<448])

[Empty DataFrame
Columns: [Image, height, width, depth, coin_1, coin_2, coin_3, coin_4]
Index: [], Empty DataFrame
Columns: [Image, height, width, depth, coin_1, coin_2, coin_3, coin_4]
Index: [], Empty DataFrame
Columns: [Image, height, width, depth, coin_1, coin_2, coin_3, coin_4]
Index: [], Empty DataFrame
Columns: [Image, height, width, depth, coin_1, coin_2, coin_3, coin_4]
Index: []]
Empty DataFrame
Columns: [Image, height, width, depth, coin_1, coin_2, coin_3, coin_4]
Index: []
```

Visualisation des Images et Points d'Angle

```

import matplotlib.pyplot as plt
import cv2

# Sélection des noms de fichier des images depuis le DataFrame
images = df.Image

# Visualisation des images et de leurs points d'angle correspondants
for i in range(len(images)):
    # Lecture de l'image à partir du chemin du fichier
    image = cv2.imread(images[i])

    # Extraction des coordonnées des angles pour l'image actuelle
    angle = [df.loc[i, f'coin_{x}'] for x in range(1, 5)]

    # Tracé de l'image avec les points d'angle
    plt.figure()
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

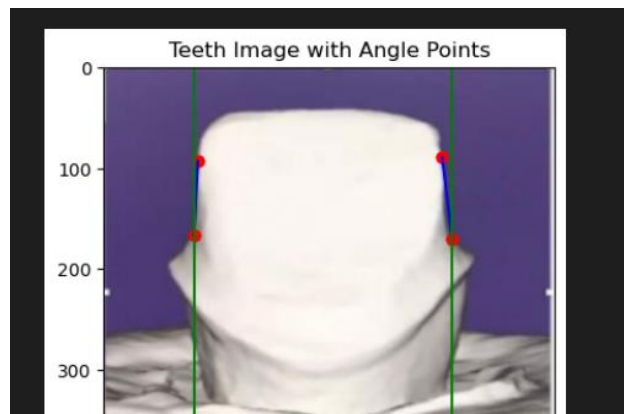
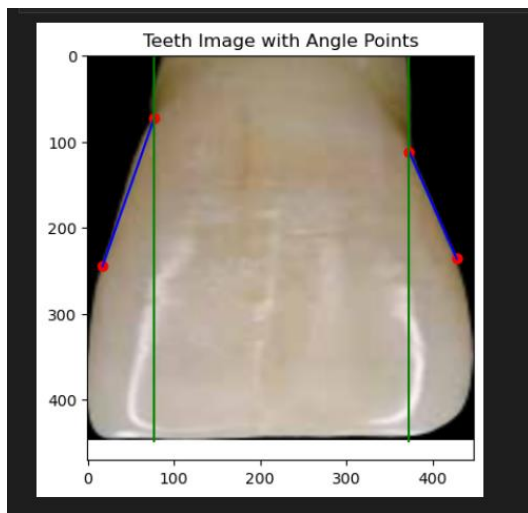
    # Tracé des points d'angle
    for p in angle:
        plt.scatter([p[0], p[1]], [p[2], p[3]], color='red') # Tracé des coordonnées x et y séparément
        plt.plot() # Tracé des points sans liaison

    # Tracé des lignes entre les points d'angle
    for j in [0, 2]:
        x_coords = [angle[j][0], angle[j + 1][0]] # Coordonnées x
        y_coords = [angle[j][2], angle[j + 1][2]] # Coordonnées y
        x_c00rds = [angle[j][0], angle[j][0]] # Coordonnées x pour la ligne verticale
        y_c00rds = [448, 0] # Coordonnées y pour la ligne verticale
        plt.plot(x_coords, y_coords, color='blue') # Tracé de la ligne entre les points d'angle
        plt.plot(x_c00rds, y_c00rds, color='green') # Tracé de la ligne verticale

    # Titre du graphique
    plt.title('Teeth IMge with Angle Points')
    plt.show()

```

Output :



Nous créons des masques pour chaque image en définissant tous les pixels à zéro, sauf ceux des deux lignes en blanc qui représentent les angles de convergences. Cela nous permet de générer des données d'entraînement pour notre modèle.

```

import cv2
import numpy as np

# Sélection des noms de fichier des images depuis le DataFrame
images = df.Image

# Définition du dossier de sortie pour sauvegarder les images résultantes
output_folder = 'D:/output_folder'

# Création du dossier de sortie s'il n'existe pas
os.makedirs(output_folder, exist_ok=True)

# Visualisation des images et de leurs points d'angle correspondants
for i in range(len(images)):
    # Lecture de l'image à partir du chemin du fichier
    image = cv2.imread(images[i])

    # Création d'un masque noir pour les points d'angle
    mask = np.zeros_like(image)

    # Extraction des coordonnées des angles pour l'image actuelle
    angle = [df.loc[i, f'coin_{x}']] for x in range(1, 5)]

    # Tracé des lignes entre les points d'angle sur le masque
    for j in [0, 2]:
        x_coords = [angle[j][0], angle[j + 1][0]] # Coordonnées x
        y_coords = [angle[j][2], angle[j + 1][2]] # Coordonnées y
        # Tracé des lignes bleues
        cv2.line(mask, (x_coords[0], y_coords[0]), (x_coords[1], y_coords[1]), (255, 255, 255), 2)

    # Préparation du masque pour l'augmentation
    mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY) # Conversion du masque en niveaux de gris
    mask = np.expand_dims(mask, axis=-1) # Ajout de la dimension de canal

    # Sauvegarde de l'image résultante
    output_path = os.path.join(output_folder, f'mask_{images[i][:4]}.jpg')
    cv2.imwrite(output_path, mask)

```

Tracé des Lignes entre les Points d'Angle et Calcul des Angles

```

# Extraction des coordonnées des angles pour l'image actuelle
angle = [df.loc[i, f'coin_{x}'] for x in range(1, 5)]

# Tracé de l'image avec les points d'angle
plt.figure()
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

# Tracé des points d'angle
for p in angle:
    plt.scatter([p[0], p[1]], [p[2], p[3]], color='red')

angs = []

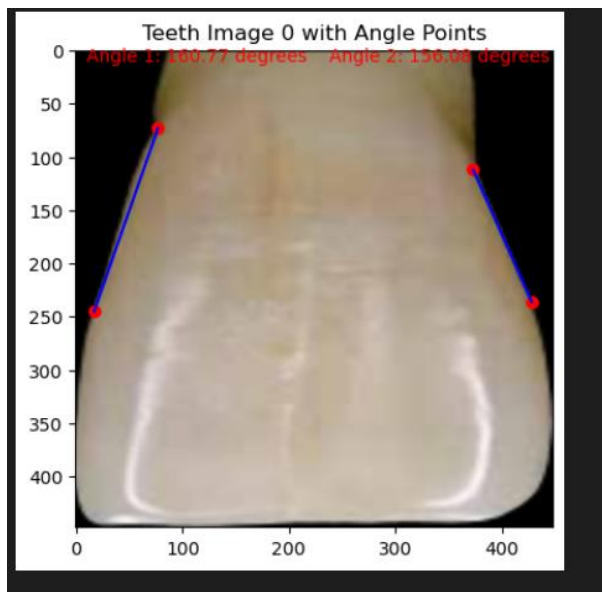
# Tracé des lignes entre les points d'angle
for j in [0, 2]:
    x_coords = [angle[j][0], angle[j + 1][0]] # Coordonnées x
    y_coords = [angle[j][2], angle[j + 1][2]] # Coordonnées y
    plt.plot(x_coords, y_coords, color='blue')

    # Calcul de l'angle entre deux lignes
    # Définition des points de départ des deux lignes
    line1_start = np.array([angle[j][0], angle[j][2]])
    line2_start = np.array([angle[j+1][0], 448])

    # Définition des points d'arrêt des deux lignes
    line1_end = np.array([angle[j+1][0], angle[j+1][2]])
    line2_end = np.array([angle[j+1][0], 0])

```

Output :



Affichage d'Images et de Masques côte à côte

```
# Afficher les images et les masques côte à côte
for image, mask in zip(images, masks):
    # Création de la figure avec deux sous-graphiques
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

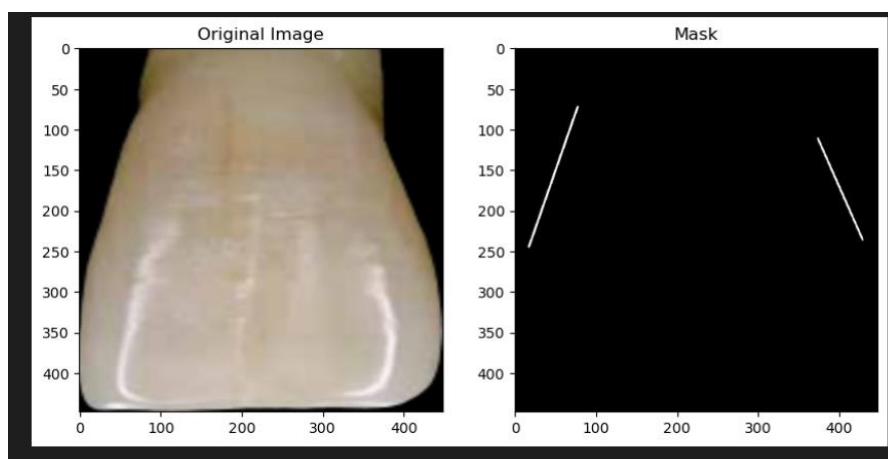
    # Affichage de l'image originale
    axes[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    axes[0].set_title('Original Image')

    # Affichage du masque
    axes[1].imshow(mask, cmap='gray')
    axes[1].set_title('Mask')

    # Affichage de la figure
    plt.show()
```

Cette section utilise une boucle pour itérer à travers chaque paire d'image et de masque. Pour chaque itération, une figure avec deux sous-graphiques est créée.

Output :



Création d'un Générateur de Données d'Image avec des Augmentations Désirées

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Création d'un générateur de données d'image avec des augmentations désirées
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
)
```

Créer un objet ImageDataGenerator avec des paramètres spécifiés, tels que la plage de rotation, le décalage horizontal et vertical, et le retournement horizontal et vertical des images.

Génération d'Images et de Masques Augmentés

```
# Génération d'images et de masques augmentés
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)

    # Application de l'augmentation de données aux images
    augmented_image_batch = data_generator.flow(image_batch, batch_size=1, shuffle=False)

    # Application de l'augmentation de données aux masques (utilisation de la même graine pour la cohérence)
    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)

# Conversion des listes en tableaux NumPy
augmented_images = np.array(augmented_images)
augmented_masks = np.array(augmented_masks)

# Vérification des formes des images et des masques augmentés
print(f'Augmented Images Size : {augmented_images.size}, Shape : {augmented_images.shape}')
print(f'Augmented Masks Size : {augmented_masks.size}, Shape : {augmented_masks.shape}')
```

Python

Output :

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

Définition de la Fonction de Construction du Modèle U-Net

```
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
```

Construction du Modèle U-Net et Modification des Masques pour Avoir un Seul Canal

```
# Construction du modèle U-Net
input_shape = images[0].shape
model = build_unet(input_shape)

# Modification des masques pour avoir un seul canal
augmented_masks_single_channel = augmented_masks[..., 0:1]
```

Compilation et Entraînement du Modèle

```
# Compilation du modèle
model.compile(optimizer='adam', loss='binary_crossentropy')

# Entraînement du modèle
model.fit(augmented_images, augmented_masks_single_channel, batch_size=8, epochs=10, validation_split=0.2)
```

Entraînement :

```
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
```

Évaluation du Modèle

```
#Évaluation du Modèle
eval_loss = model.evaluate(val_images, val_masks)
print("Evaluation Loss:", eval_loss)

1/1 [=====] - 18s 18s/step - loss: 0.6981
Evaluation Loss: 0.6981333494186401
```

Inférence sur de Nouvelles Images avec un Modèle U-Net

```

# Chargement de la nouvelle image
new_image_path = '23.jpg'
new_image = cv2.imread(new_image_path)

# Prétraitement de la nouvelle image
new_image = np.expand_dims(new_image, axis=0)

# Prédiction du masque de lignes
line_mask = model.predict(new_image)

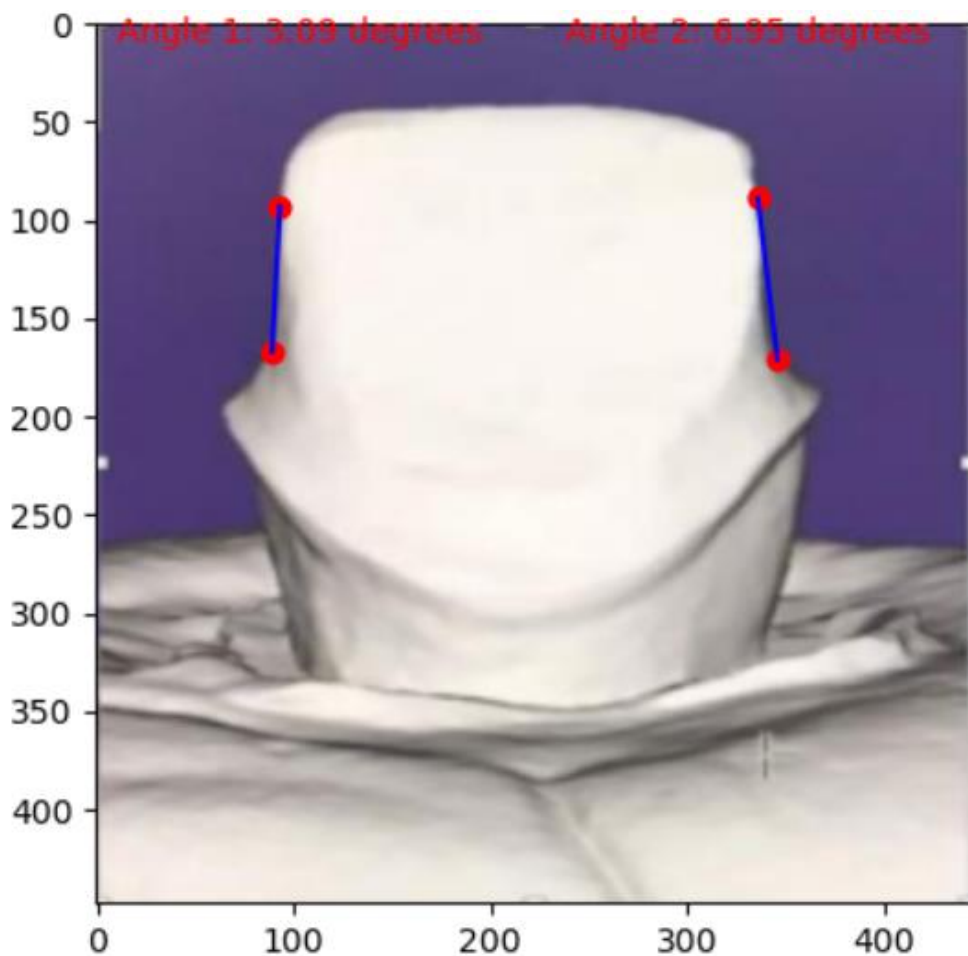
# Seuillage du masque de lignes
threshold = 0.5
thresholded_mask = (line_mask > threshold).astype(np.uint8)

# Visualisation du masque de lignes
cv2.imshow("Line Mask", thresholded_mask[0] * 255)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

1/1 [=====] - 1s 794ms/step

Résultat Expérimentale :



Conclusion :

Dans ce rapport, nous avons présenté une méthode pour détecter les angles de convergence des dents en utilisant Python. Notre approche, basée sur l'apprentissage automatique et l'architecture U-Net, permet une détection précise des angles de convergence, ce qui est essentiel pour l'analyse dentaire. Les résultats expérimentaux démontrent l'efficacité de notre approche. Notre travail ouvre la voie à des recherches futures sur l'automatisation de l'analyse dentaire et la planification des traitements orthodontiques.