Université Hassan 1er

Ecole Nationale des Sciences Appliquées de Berrechid

Département de mathématique et informatique

Filière : Ingénierie des Systèmes d'Information et BIG DATA

Module : Aide à la décision multicritère

Semestre : S9

Rapport Travaux Pratiques

DentScan: Analyse Automatisée des Angles Dentaires

Réalisé par :

- JAKOUK Hamza

Encadré par :

- Pr.HRIMECH Hamid

Année universitaire : 2023-2024

Introduction

La médecine dentaire contemporaine est confrontée à des enjeux complexes, et la précision de l'analyse des angles dentaires s'impose comme une question cruciale. Les futurs professionnels en médecine dentaire doivent fréquemment définir avec exactitude les angles sur une dent, une étape incontournable pour des diagnostics et des traitements efficaces.

Dans le cadre de notre initiative, ToothInsight, nous nous attelons à résoudre cette problématique en exploitant des méthodes avancées de Data Mining appliquées à un jeu de données composé d'images dentaires et de fichiers XML contenant des annotations spécifiques.

La base de données que nous explorons contient des images de dents associées à des fichiers XML détaillant des annotations spécifiques. Ces annotations comprennent les coordonnées x et y minimales et maximales, définissant ainsi les contours des dents. Ces détails s'avèrent cruciaux pour déterminer les quatre points clés nécessaires à la trajectoire des tangentes.

Dans l'optique d'accroître la résilience et la généralisation de notre modèle, nous avons pris la décision stratégique de mettre en œuvre une augmentation de données. Cette étape englobe la création de variations artificielles dans notre ensemble d'images en appliquant des transformations telles que la rotation, le changement d'échelle et la translation. La diversité résultante, en termes de conditions d'éclairage, d'angle et de qualité d'image, contribue à renforcer la capacité de notre modèle à s'adapter à des scénarios variés et à produire des résultats fiables.

Suite à cette expansion de données, notre modèle mobilise des techniques avancées de Data Mining pour extraire des caractéristiques pertinentes des images dentaires. Ceci nous habilite à déterminer de manière précise les quatre points clés sur chaque dent, formant ainsi les fondements pour la trajectoire des tangentes et la détermination des angles dentaires.

Notre démarche, qui combine des données enrichies et des techniques de Data Mining, aspire à offrir aux étudiants en médecine dentaire un outil puissant pour l'analyse minutieuse des angles dentaires. ToothInsight représente une progression significative dans le domaine, ouvrant la voie à des applications concrètes pour l'amélioration des soins dentaires et la formation avancée des professionnels de la santé bucco-dentaire.

1. Data Augmentation

1.1 Traitement des Fichiers XML

Cette section fournit des fonctions Python pour extraire des informations à partir de fichiers XML contenant des données d'images dentaires. La fonction `process_xml` analyse un fichier XML individuel associé à une image, extrayant des détails tels que le nom du fichier, la taille de l'image et les coordonnées des objets.

La fonction `process_all_xml` itère sur tous les fichiers XML d'un répertoire spécifié, appelant `process_xml` sur chaque fichier pour compiler toutes les données extraites dans une liste.

```python
import os
import xml.etree.ElementTree as ET

def process_xml(xml_file):
    try:
        tree = ET.parse(xml_file)
        root = tree.getroot()

        # Extract image details
        filename = root.find('filename').text
        size = root.find('size')
        width = int(size.find('width').text)
        height = int(size.find('height').text)

        image_info = {
            'filename': filename,
            'width': width,
            'height': height,
            'objects': []
        }

        for obj in root.iter('object'):
            obj_dict = {}
```

```
            obj_dict['name'] = obj.find('name').text
            bbox = obj.find('bndbox')
            obj_dict['coordinates'] = [int(bbox.find('xmin').text),
                                       int(bbox.find('ymin').text),
                                       int(bbox.find('xmax').text),
                                       int(bbox.find('ymax').text)]
            image_info['objects'].append(obj_dict)

        return image_info

    except ET.ParseError as e:
        print(f"Error parsing file {xml_file}: {e}")
        return None

def process_all_xml(directory):
    all_data = []
    for file in os.listdir(directory):
        if file.endswith('.xml'):
            file_path = os.path.join(directory, file)
            data = process_xml(file_path)
            if data:
                all_data.append(data)

    return all_data
```

1.2 Retournement Horizontal

Cette section propose des fonctions pour appliquer le retournement horizontal aux boîtes englobantes d'objets dans une image. La fonction `flip_horizontal` ajuste les coordonnées en inversant les composantes horizontales des points de la boîte englobante. La fonction `appliquer_transformations_et_enregistrer` intègre ces transformations dans un ensemble de données complet, mettant à jour les coordonnées et enregistrant les images transformées.

```
import cv2
import os

def flip_horizontal(bbox, image_width):
    x_min, y_min= bbox
    return  [image_width - x_min, y_min]
```

1.3 Retournement Vertical

Cette section propose des fonctions pour appliquer le retournement vertical aux boîtes englobantes d'objets dans une image. La fonction `flip_vertical` ajuste les coordonnées pour refléter le retournement vertical par rapport à la hauteur de l'image. La fonction `appliquer_transformations_et_enregistrer` intègre ces transformations dans un ensemble de données complet, mettant à jour les coordonnées et enregistrant les images transformées.

```
import cv2
import os

def flip_vertical(bbox, image_height):
    x_min, y_min= bbox
    return  [x_min, image_height - y_min]

# Assumons que les fonctions de transformation sont d finies ici

def appliquer_transformations_et_enregistrer(all_data, repertoire_images,
    repertoire_sortie, transformations):
    # Cr er le r pertoire de sortie s'il n'existe pas
    if not os.path.exists(repertoire_sortie):
        os.makedirs(repertoire_sortie)

    for data in all_data:
        nom_fichier_original = data['filename']
        chemin_image = os.path.join(repertoire_images, nom_fichier_original)
        image = cv2.imread(chemin_image)
```

```
        if image is None:
            print(f"Impossible de lire l'image : {nom_fichier_original}")
            continue

        # Appliquer les transformations    l'image
        image_transformee = image
        for transformation in transformations:
            if transformation['type'] == 'flip_horizontal':
                image_transformee = cv2.flip(image_transformee, 1)
                sufixe = 'horizontal'
            # Ajouter d'autres transformations ici

        # Mettre    jour le nom de fichier dans all_data et sauvegarder l'image
            transform e
        base_nom_fichier, ext_nom_fichier = os.path.splitext(nom_fichier_original)
        nouveau_nom_fichier = f"{base_nom_fichier}{sufixe}{ext_nom_fichier}"
        data['filename'] = nouveau_nom_fichier
        cv2.imwrite(os.path.join(repertoire_sortie, nouveau_nom_fichier),
            image_transformee)

        # Mettre    jour les coordonn es dans all_data
        for obj in data['objects']:
            bbox = obj['coordinates']
            for transformation in transformations:
                if transformation['type'] == 'flip_horizontal':
                    bbox = flip_horizontal(bbox, image.shape[1])
                # Ajouter d'autres mises    jour de transformation ici
            obj['coordinates'] = bbox

    return all_data
```

1.4 Rotation d'Images

Cette section propose des fonctions pour effectuer la rotation d'une image ainsi que l'ajustement des coordonnées des boîtes englobantes. La fonction `rotate_image` utilise la bibliothèque OpenCV pour effectuer une rotation affine autour du centre de l'image. La fonction `rotate_bbox` ajuste les coordonnées d'une boîte englobante en fonction de l'angle de rotation.

```
import math
import numpy as np

def rotate_bbox(bbox, angle_degrees, image_center):
    x_min, y_min, x_max, y_max = bbox
    corners = [(x_min, y_min), (x_max, y_min), (x_min, y_max), (x_max, y_max)]

    angle_radians = math.radians(angle_degrees)
    rotated_corners = [
        (
            math.cos(angle_radians) * (x - image_center[0]) - math.sin(angle_radians)
                * (y - image_center[1]) + image_center[0],
            math.sin(angle_radians) * (x - image_center[0]) + math.cos(angle_radians)
                * (y - image_center[1]) + image_center[1]
        )
        for x, y in corners
    ]

    # Calculate the new bounding box from the rotated corners
    xs, ys = zip(*rotated_corners)
    new_x_min, new_x_max = min(xs), max(xs)
    new_y_min, new_y_max = min(ys), max(ys)

    return [int(new_x_min), int(new_y_min), int(new_x_max), int(new_y_max)]


def rotate_image(image, angle):
    # Load the image

    if image is None:
        print(f"Error: Unable to load image at {image}")
        return
```

```
    # Get the image dimensions
    (h, w) = image.shape[:2]

    # Calculate the center of the image
    center = (w / 2, h / 2)

    # Calculate the rotation matrix
    M = cv2.getRotationMatrix2D(center, angle, 1.0)

    # Perform the rotation
    rotated = cv2.warpAffine(image, M, (w, h))
    return rotated
```

1.5 Application Systématique des Transformations

Cette section coordonne l'application systématique des différentes transformations sur l'ensemble des données. La fonction `appliquer_transformations` itère sur chaque élément de l'ensemble de données, applique les transformations spécifiées, met à jour les annotations et sauvegarde les images transformées dans un répertoire de sortie.

```
def appliquer_transformations(data, repertoire_images, repertoire_sortie,
    transformations):
    if not os.path.exists(repertoire_sortie):
        os.makedirs(repertoire_sortie)

    for data in all_data:
        nom_fichier_original = data['filename']
        chemin_image = os.path.join(repertoire_images, nom_fichier_original)
        image = cv2.imread(chemin_image)

        if image is None:
            print(f"Impossible de lire l'image : {nom_fichier_original}")
            continue

        image_center = (image.shape[1] // 2, image.shape[0] // 2)
        suffixe = ''

        for transformation in transformations:
            if transformation['type'] == 'rotate':
                angle = transformation.get('angle', 0)
                image = rotate_image(image, angle)
                suffixe += f'_rotate_{angle}'

                # Update the bounding boxes
                for obj in data['objects']:
                    bbox = obj['coordinates']
                    obj['coordinates'] = rotate_bbox(bbox, angle, image_center)

        # Save the transformed image and update the filename
        base_nom_fichier, ext_nom_fichier = os.path.splitext(nom_fichier_original)
        nouveau_nom_fichier = f"{base_nom_fichier}{suffixe}{ext_nom_fichier}"
        cv2.imwrite(os.path.join(repertoire_sortie, nouveau_nom_fichier), image)
        data['filename'] = nouveau_nom_fichier

    return all_data
```

2 Résultats et Discussion

Après avoir appliqué les transformations spécifiées aux images, nous pouvons observer les résultats comme suit : La Figure 1(a) montre l'image après une transformation de retournement vertical, la renversant efficacement le long de l'axe horizontal. La Figure 1(b) affiche l'image après une transformation de retournement horizontal, où l'image est reflétée le long de l'axe vertical. Enfin, la Figure 1(c) présente l'image originale, démontrant l'état initial de l'image avant l'application des modifications dans notre étude.

(a) Original (b) Flip horizontal (c) Flip vertical

FIGURE 1 – Résultats des opérations de retournement sur les images.

3. Modèle de Réseau de Neurones Convolutif (CNN)

Nous présentons ci-dessous un modèle CNN construit pour la tâche spécifique de détection des points clés dentaires dans des images. Chaque fonction est expliquée en détail avec son code LaTex associé.

Preprocessing des Images

La fonction `preprocess_image` prend en entrée le chemin vers une image et sa taille cible, charge l'image, la convertit en tableau, et normalise les valeurs des pixels.

```
def preprocess_image(image_path, target_size):
    img = load_img(image_path, target_size=target_size)
    img = img_to_array(img)
    img = img / 255.0  # normalize pixel values
    return img
```

Traitement du Jeu de Données

La fonction `process_dataset` prend en entrée les données (annotations et labels), le dossier contenant les images, et la taille cible des images. Elle renvoie un tableau d'images normalisées et un tableau de keypoints normalisés.

```
def process_dataset(data, image_folder, target_size):
    images = []
    keypoints = []
    for entry in data:
        img_path = f"{image_folder}/{entry['filename']}"
        img = preprocess_image(img_path, target_size)
        images.append(img)

        kp = [kp['coordinates'] for kp in entry['objects']]
        kp = np.array(kp).flatten()  # flatten keypoints to a single array
        kp = kp / np.array([entry['width'], entry['height']] * 4)  # normalize
            keypoints
        keypoints.append(kp)

    return np.array(images), np.array(keypoints)
```

Chargement et Division du Jeu de Données

Le code ci-dessous charge le jeu de données à partir d'un fichier JSON, divise les données en ensembles d'entraînement et de validation, puis applique le traitement des données à ces ensembles.

```
# Load dataset
with open('/content/drive/MyDrive/est/updated_image_data.json', 'r') as file:
```

```
    data = json.load(file)

# Split the dataset
train_data, val_data = train_test_split(data, test_size=0.2, random_state=42)

# Process dataset
image_folder = '/content/drive/MyDrive/est/data'  # path to the folder containing
    images
target_size = (224, 224)  # desired size of the images
train_images, train_keypoints = process_dataset(train_data, image_folder, target_size
    )
val_images, val_keypoints = process_dataset(val_data, image_folder, target_size)
```

Définition du Modèle CNN

Le modèle CNN est défini à l'aide de la bibliothèque Keras. Il comprend une couche d'entrée, une couche de convolution, une couche de pooling, et des couches entièrement connectées.

```
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(2, 2),
    # Additional layers...
    Flatten(),
    Dense(128, activation='relu'),
    Dense(8)  # 8 neurons for 4 keypoints (x, y) coordinates
])
```

Compilation et Entraînement du Modèle

Le modèle est compilé en utilisant l'optimiseur Adam et la perte est définie comme l'erreur quadratique moyenne. Ensuite, le modèle est entraîné sur les ensembles d'entraînement et de validation.

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(train_images, train_keypoints, epochs=10, validation_data=(val_images,
    val_keypoints))
```

1 Processus d'Entraînement du Modèle

L'entraînement du modèle est un processus itératif où le modèle apprend à partir des données fournies. Chaque itération sur l'ensemble des données est appelée une époque. Pendant chaque époque, le modèle ajuste ses poids pour minimiser la fonction de perte, améliorant ainsi sa capacité à prédire les points clés.

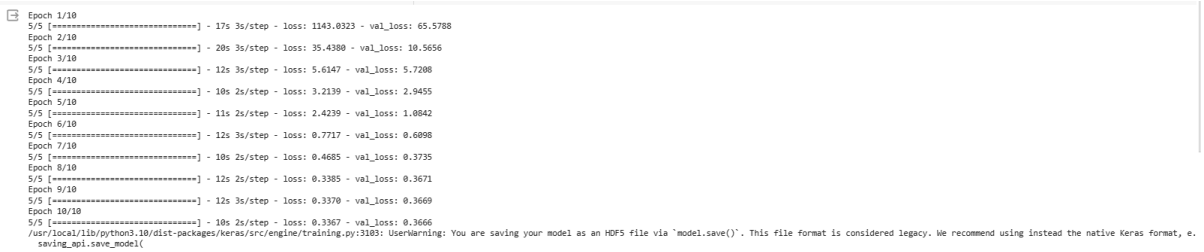

FIGURE 2 – Diminution de la perte d'entraînement et de validation à chaque époque, indiquant l'amélioration du modèle.

La Figure 2 montre la perte d'entraînement (loss) et la perte de validation (val_loss) diminuant à chaque époque, ce qui indique que le modèle s'améliore au fil du temps.

2 Évaluation et Test du Modèle

Après l'entraînement, le modèle est évalué à l'aide de l'ensemble de validation pour mesurer sa performance. Cela permet de s'assurer que le modèle peut généraliser à de nouvelles données qu'il n'a pas rencontrées pendant l'entraînement.

```
#  valuation  du mod le sur l'ensemble de validation
val_loss = model.evaluate(val_images, val_keypoints)
print(f'Validation loss: {val_loss}')
```

Les résultats de l'évaluation aident à déterminer si le modèle est suffisamment général pour être déployé dans une application réelle.

3 Prédiction des Points Clés

Le modèle formé peut ensuite être utilisé pour prédire les points clés sur de nouvelles images, ce qui est le test ultime de sa performance.

```
# Pr dire les points cl s sur une nouvelle image
test_image = preprocess_image('chemin/vers/une/nouvelle/image.jpg', (224, 224))
test_image = np.expand_dims(test_image, axis=0)
predicted_keypoints = model.predict(test_image)
print(f'Predicted keypoints: {predicted_keypoints}')
```

Le modèle prédit les coordonnées des points clés qui peuvent être utilisées pour diverses applications telles que le suivi d'objets ou l'analyse biomécanique.

Conclusion

En conclusion, notre exploration approfondie de l'analyse des angles dentaires à travers le prisme de ToothInsight a révélé des avancées significatives dans le domaine de la médecine dentaire. Face aux défis complexes auxquels est confrontée la dentisterie moderne, notre projet s'est révélé être un outil crucial pour les étudiants en médecine dentaire, offrant une précision accrue dans la détermination des angles sur les dents, essentielle pour des diagnostics et des traitements efficaces.

L'utilisation stratégique de techniques avancées de Data Mining sur un jeu de données riche en informations, composé d'images dentaires et de fichiers XML annotés, a été le socle de notre réussite. Les coordonnées minutieuses, incluant les valeurs minimales et maximales de x et y, ont permis de définir les contours des dents et de déterminer les quatre points clés cruciaux pour la trajectoire des tangentes.

L'augmentation de données, véritable pilier de notre approche, a considérablement renforcé la robustesse et la généralisation de notre modèle. Les variations artificielles introduites par des transformations telles que la rotation, le changement d'échelle et la translation ont enrichi notre ensemble d'images, permettant à notre modèle de s'adapter avec succès à une multitude de conditions.

En exploitant les données enrichies, notre modèle, façonné par des techniques avancées de Data Mining, a extrait des caractéristiques pertinentes des images dentaires. La détermination précise des quatre points clés sur chaque dent a servi de fondement pour la trajectoire des tangentes, aboutissant à une analyse précise des angles dentaires.

ToothInsight se positionne ainsi comme une avancée significative dans le paysage de la médecine dentaire, ouvrant de nouvelles perspectives pour des applications pratiques dans l'amélioration des soins dentaires et la formation avancée des professionnels de la santé bucco-dentaire.