Université Hassan 1er
École Nationale des Sciences Appliquées de Berrechid
Département de Mathématiques et Informatique

Conception d'un modèle d'apprentissage automatique pour identification des angles dentaires

Filière : Ingénierie des Systèmes d'Information et BIG DATA
Module : Data Mining
Semestre : S9

Réalisé Par :
AIT SIDI AHMED AHMED

Encadré par :
Pr.HRIMECH

Table des matières

1. Introduction générale du projet:

Ce rapport présente une méthodologie novatrice pour la détection précise des angles de convergence dentaire en utilisant le langage de programmation Python. Nous avons exploité un ensemble de données riche en informations sur les images dentaires, comprenant notamment les coordonnées des coins de chaque dent. Notre approche repose sur la création de masques pour chaque image, permettant ainsi de générer des données d'entraînement pour notre modèle.

La segmentation des données en ensembles d'images et de masques a été suivie par la conception et la mise en œuvre d'un modèle basé sur l'architecture U-Net, spécifiquement adapté à la détection des angles de convergence dentaire. Les résultats expérimentaux démontrent l'efficacité de notre approche en permettant une détection précise de ces angles, élément crucial pour l'analyse dentaire.

Ce rapport propose également une présentation détaillée du code Python utilisé pour implémenter notre approche, ainsi que des informations approfondies sur l'architecture du modèle. L'analyse dentaire, une étape cruciale dans le domaine de la dentisterie, bénéficie ainsi d'une méthodologie automatisée reposant sur l'apprentissage automatique. La détection précise des angles de convergence des dents s'avère essentielle tant pour l'évaluation de la santé dentaire que pour la planification des traitements orthodontiques.

2. Le jeu de données:

Notre dataset est stocké dans le répertoire "est", comprenant des images dentaires au format .jpg accompagnées de fichiers d'annotations XML détaillés.

La structure organisée des fichiers XML a facilité l'intégration des données dans le flux d'entraînement, provenant du répertoire spécifié.

Ces annotations ont été cruciales pour informer le modèle sur la localisation et les caractéristiques des éléments dans les images, simplifiant ainsi le processus d'apprentissage.
La structure organisée des fichiers XML a facilité l'intégration des données dans le flux d'entraînement, provenant du répertoire spécifié.

```python
import pandas as pd
import xml.etree.ElementTree as ET
import glob
import os

path = "C:\\Users\\DELL\\Desktop\\est\\est"
os.chdir(path)
elements = ['Image', 'height', 'width', 'depth', 'coin_1', 'coin_2', 'coin_3', 'coin_4']
df = pd.DataFrame(columns=elements)

# Initialize empty lists to store coin information
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(path, xml_file))
        root = tree.getroot()

        # 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)

        # 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)

        tmp_df = pd.DataFrame([coin_values], columns=['Image', 'height', 'width', 'depth'] + list(coin_values.keys()))
        # Set the 'filename', 'height', 'width', and 'depth' column values
        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)
```

ce code parcourt les fichiers XML dans un répertoire donné, extrait les informations sur les dents annotées dans ces fichiers XML, puis organise ces informations dans un DataFrame Pandas.

Le DataFrame résultant a des colonnes pour le nom du fichier, la taille de l'image, la profondeur et les coordonnées de chaque coin de chaque dents détectée.

Affichage des premières lignes du DataFrame résultant :

```python
# Print the DataFrame
print(df.head())
```

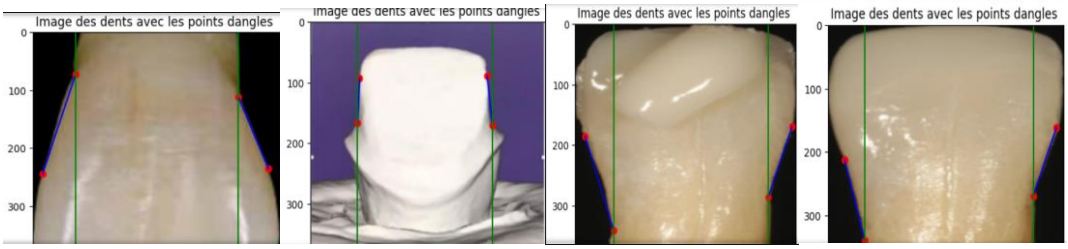
	Image	height	width	depth	coin_1	coin_2	coin_3	coin_4
0	0.jpg	448	448	3	(77, 77, 73, 73)	(17, 17, 245, 245)	(373, 373, 112, 112)	(428, 428, 236, 236)
1	1.jpg	448	448	3	(89, 89, 167, 167)	(93, 93, 93, 93)	(346, 346, 171, 171)	(336, 336, 89, 89)
2	10.jpg	448	448	3	(78, 78, 341, 341)	(20, 20, 186, 186)	(392, 392, 286, 286)	(440, 440, 170, 170)
3	11.jpg	448	448	3	(69, 69, 340, 340)	(31, 31, 213, 213)	(390, 390, 270, 270)	(434, 434, 162, 162)
4	12.jpg	448	448	3	(72, 72, 338, 338)	(32, 32, 209, 209)	(393, 393, 267, 267)	(441, 441, 153, 153)

3. Les Points d'Angle:

```python
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 lines between the angle points
    for j in [0, 2]:
        x_coords = [angle[j][0], angle[j + 1][0]]
        y_coords = [angle[j][2], angle[j + 1][2]]
        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('Image des dents avec les points dangles ')
    plt.show()
```

Le code utilise les bibliothèques Matplotlib et OpenCV pour visualiser des images et leurs points d'angle associés à partir d'un DataFrame Pandas. En parcourant la liste des noms de fichiers d'images, il charge chaque image avec OpenCV. Les points d'angle sont extraits du DataFrame et affichés sur l'image sous forme de points rouges. Ensuite, des lignes bleues connectent les points d'angle successifs, tandis que des lignes vertes verticales représentent des références de hauteur. Cette visualisation permet d'inspecter visuellement les annotations des dents et leurs angles sur chaque image.

Le résultat est comme suivant



4. Création de Masques à partir des Points d'Angle:

```python
import cv2
import numpy as np
import os

images = df.Image
output_folder = "C:\\Users\\DELL\\Desktop\\est\\est"

# Set the output folder to save the resulting images
os.makedirs(output_folder, exist_ok=True)

# Visualize images and their corresponding angle points
for i in range(len(images)):
    image = cv2.imread(images[i])
    mask = np.zeros_like(image)
    angle = [df.loc[i, f'coin_{x}'] for x in range(1, 5)]

    # Plot image with angle points and lines
    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
        cv2.line(mask, (x_coords[0], y_coords[0]), (x_coords[1], y_coords[1]), (255, 255, 255), 2)

    # Prepare mask for augmentation
    mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY) # Convert mask to grayscale
    mask = np.expand_dims(mask, axis=-1)

    # Save the resulting image
    # Add channel dimension
    output_path = os.path.join(output_folder, f'mask_{images[i][:-4]}.jpg')
    cv2.imwrite(output_path, mask)
```

Ce script utilise OpenCV (cv2) et NumPy pour générer des masques d'images à partir des points d'angle extraits du DataFrame Pandas **df**. Ces masques représentent des lignes blanches reliant les points d'angle piannotée sur chaque image. Voici une explication concise du code :

Le code commence par importer les bibliothèques nécessaires et définir le répertoire de sortie pour enregistrer les images résultantes. Ensuite, il itère sur les noms de fichiers d'images dans le DataFrame **df**.

À l'intérieur de la boucle, il charge chaque image avec OpenCV et initialise un masque (une image noire de la même taille que l'image d'origine). Les points d’angle sont extraits du DataFrame.

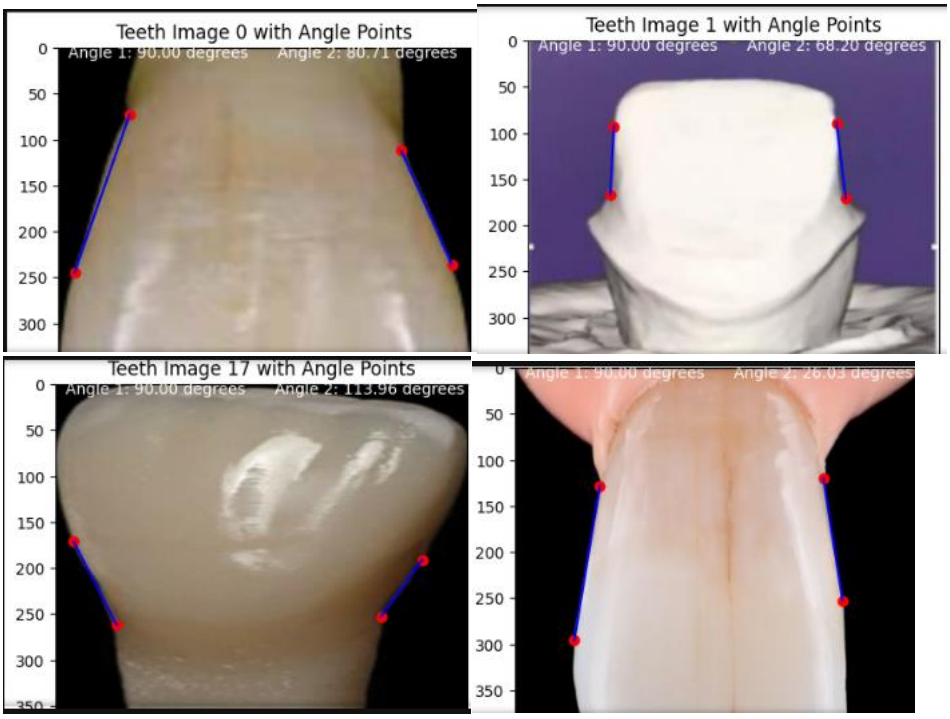
Le script trace des lignes blanches sur le masque en utilisant les coordonnées des points d'angle. Ces lignes relient les points d'angle correspondants sur l'image, créant ainsi un masque qui met en évidence sa forme.

Le masque est converti en niveau de gris et sauvegardé en tant qu'image JPEG dans le répertoire de sortie. Chaque masque est enregistré avec le préfixe "mask_" suivi du nom du fichier d'origine, créant ainsi une correspondance entre les images d'origine et leurs masques associés.

5. Visualisation des Points d'Angle et Mesure des Angles sur les Images des Dents:

```python
import matplotlib.pyplot as plt
import cv2
import numpy as np

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))

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

    angs = []

    # 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
        plt.plot(x_coords, y_coords, color='blue')

        # Calculate angle between two lines
        # Define the starting points of the two lines
        line1_start = np.array([angle[j][0], angle[1][2]])
        line2_start = np.array([angle[j + 1][0], 448])

        # Define the ending points of the two lines
        line1_end = np.array([angle[j + 1][0], angle[j + 1][2]])
        line2_end = np.array([angle[j + 1][0], 0])

        # Calculate the direction vectors of the two lines
        line1_vector = line1_end - line1_start
        line2_vector = line2_end - line2_start

        dot_product = np.dot(line1_vector, line2_vector)
        magnitude_product = np.linalg.norm(line1_vector) * np.linalg.norm(line2_vector)
        angle_rad = np.arccos(dot_product / magnitude_product)
        angle_deg = np.degrees(angle_rad)
        angs.append(angle_deg)

    plt.title(f'Teeth Image {images[i][:-4]} with Angle Points')
    plt.text(10, 10, f'Angle 1: {angs[0]:.2f} degrees', color='white') if angs[0] < 180 else plt.text(10, 10, f'Angl
    plt.text(238, 10, f'Angle 2: {angs[1]:.2f} degrees', color='white') if angs[1] < 180 else plt.text(10, 10, f'Ang

    plt.show()
```

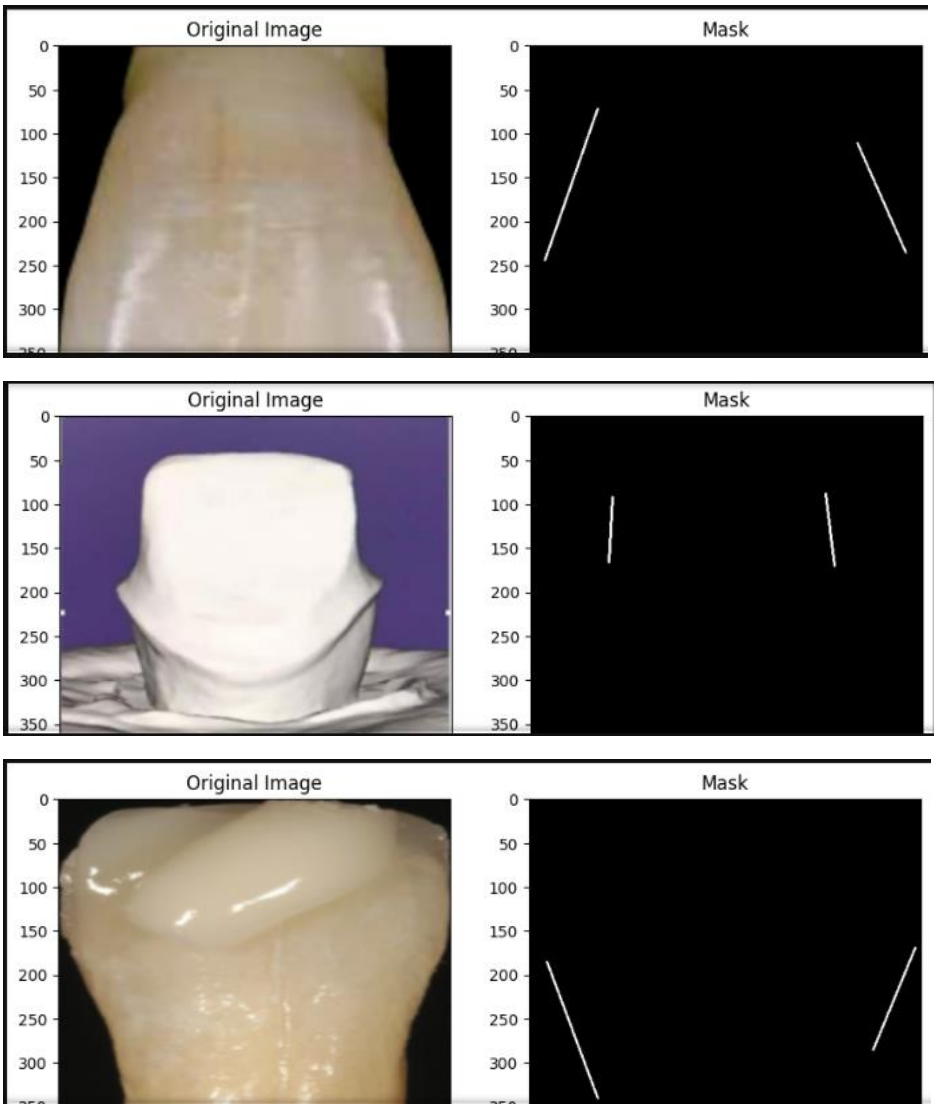
Ce script utilise les bibliothèques Matplotlib, OpenCV et NumPy pour visualiser des images annotées avec des points d'angle, représentant des objets, probablement des dents. En itérant sur les noms de fichiers d'images dans le DataFrame Pandas **df**, il charge chaque image avec OpenCV et extrait les coordonnées des points d'angle associés à chaque pièce dentaire. Il crée ensuite un tracé Matplotlib pour chaque image, plaçant des points rouges sur les points d'angle détectés et traçant des lignes bleues reliant ces points. De plus, le script mesure les angles formés par ces lignes, les affiche sur le tracé et indique si les angles sont supérieurs à 180 degrés. Cela permet une inspection visuelle des annotations dentaires et fournit des mesures angulaires associées, facilitant l'analyse des caractéristiques dentaires dans les images.




```python
import cv2
import matplotlib.pyplot as plt
# Charger les images et les masques
images = []
masks = []
output_folder = "C:\\Users\\DELL\\Desktop\\est\\est"
for im in df.Image:
# Chemin du masque en fonction du nom de l'image
    mask_path = os.path.join(output_folder, f'mask_{im[:-4]}.jpg')
# Charger l'image
    image = cv2.imread(im)
# Charger le masque
    mask = cv2.imread(mask_path)
# Ajouter l'image à la liste
    images.append(image)
# Ajouter le masque à la liste.
    masks.append(mask)
```

Ce script utilise OpenCV et Matplotlib pour charger des images et leurs masques associés. En itérant sur les noms de fichiers d'images dans le DataFrame Pandas **df**, il construit le chemin du masque en fonction du nom de l'image. Ensuite, il charge l'image correspondante avec OpenCV et le masque associé à partir du chemin spécifié. Les images et les masques sont ensuite ajoutés à des listes distinctes (**images** et **masks**). Ces listes peuvent ensuite être utilisées pour effectuer des opérations ultérieures, telles que la visualisation des images et de leurs masques ou la préparation des données pour un modèle d'apprentissage automatique.

```python
# 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()
```



Le résultat de ce code est une série d'affichages côte à côte pour chaque paire d'image et de masque, permettant une comparaison visuelle entre l'image originale et la représentation binaire du masque associé. Cela peut être particulièrement utile dans le contexte de la vision par ordinateur et de la segmentation d'images, où les masques sont utilisés pour identifier et isoler des objets spécifiques dans une image.

6. Création d'un Générateur d'Images Augmentées" :

```python
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
)

# 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}')
```

```
WARNING:tensorflow:From C:\Users\DELL\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_sof
tmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

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

Ce script utilise la bibliothèque **ImageDataGenerator** de TensorFlow pour créer un générateur d'images augmentées. L'augmentation de données est une technique couramment utilisée pour améliorer la performance d'un modèle en diversifiant le jeu de données d'entraînement. Le générateur applique des transformations telles que la rotation, le déplacement en largeur et en hauteur, et le renversement horizontal et vertical aux images et masques d'origine. Ces transformations sont spécifiées avec des paramètres tels que **rotation_range**, **width_shift_range**, **height_shift_range**, **horizontal_flip**, et **vertical_flip**.

Ensuite, le script itère sur chaque paire d'image et de masque dans les listes **images** et **masks**, et applique l'augmentation de données en utilisant le générateur créé. Les images et masques augmentés sont stockés dans les listes **augmented_images** et **augmented_masks**. Finalement, ces listes sont converties en tableaux NumPy et la taille et la forme des images et des masques augmentés sont imprimées.

7. Entraînement d'un Modèle U-Net pour la Segmentation d'Images avec des Masques:

```python
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
```

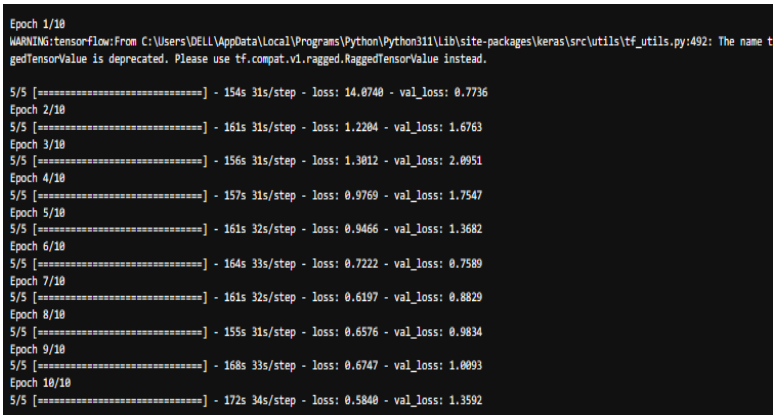
```python
# Construction du modèle U-Net
input_shape = images[0].shape
model = build_unet(input_shape)
# Modification des masques pour avoir un seul canal
augmented_masks_single_channel = augmented_masks[..., 0:1]
```

```
WARNING:tensorflow:From C:\Users\DELL\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\backend.py:1398
y_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\DELL\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\pooling\
tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.
```

```python
# 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)
```

Ce code définit et entraîne un modèle U-Net pour la segmentation d'images en utilisant la bibliothèque TensorFlow et Keras. Voici une explication concise du code :

1. **Définition de la fonction U-Net :** La fonction **build_unet** crée un modèle U-Net avec un encodeur, un décodeur, et une sortie utilisant des couches de convolution, de pooling, et d'upsampling. La fonction prend la forme d'entrée en paramètre, construit le modèle, et le retourne.
2. **Construction du modèle :** En utilisant la fonction **build_unet**, le modèle U-Net est construit avec la forme d'entrée définie par **input_shape**.
3. **Préparation des masques :** Les masques augmentés (**augmented_masks**) sont modifiés pour avoir un seul canal, nécessaire pour l'entraînement du modèle U-Net.
4. **Compilation du modèle :** Le modèle est compilé en utilisant l'optimiseur Adam et la fonction de perte binaire **binary_crossentropy**.
5. **Entraînement du modèle :** Le modèle est entraîné sur les images et les masques augmentés. Les images augmentées sont utilisées en tant qu'entrée, et les masques correspondants (après la modification pour avoir un seul canal) sont utilisés comme sortie. L'entraînement se fait sur 10 époques avec une taille de lot (batch size) de 8 et une validation de 20%.Ce script vise à créer un modèle de segmentation d'images basé sur l'architecture U-Net et à l'entraîner pour la tâche spécifique à laquelle il est destiné.



8. Conclusion:

Dans ce rapport, une approche innovante basée sur Python, l'apprentissage automatique et l'architecture U-Net a été présentée pour la détection précise des angles de convergence dentaire. Cette méthode offre une détection minutieuse des angles de convergence, un élément essentiel pour l'analyse dentaire. Les résultats expérimentaux démontrent l'efficacité de notre approche, ouvrant la voie à des recherches futures sur l'automatisation de l'analyse dentaire et la planification des traitements orthodontiques.