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: Fouille de donnée

Semester: S9

# Compte rendu du TP

Réalisé par:

- BOUKTIB Salma

Encadré par :

- Mr. HRIMECH Hamid

Année Universitaire: 2023/2024

# Table des matières

# Introduction

Les dents, composantes intégrales de l'anatomie humaine, jouent un rôle crucial dans divers aspects de la vie, de l'aide à la digestion à la contribution à l'apparence générale d'une personne. Dans ce projet, nous explorons le domaine de l'analyse d'images dentaires, en utilisant des techniques pour améliorer et extraire des informations précieuses à partir d'images de dents. L'objectif est de développer un modèle capable de prédire avec précision les coordonnées de points spécifiques sur les dents, facilitant différentes applications dentaires telles que la planification et l'évaluation des traitements.

Pour atteindre cet objectif, notre projet suit une approche en plusieurs étapes. Nous commençons par collecter un ensemble diversifié d'images de dents, qui servent de base à notre modèle. L'augmentation de ces images fournit un ensemble de données robuste pour l'entraînement, permettant au modèle de généraliser efficacement à différentes situations. De plus, nous utilisons des fichiers JSON contenant des coordonnées annotées comme données de vérité terrain, guidant le modèle pour apprendre les relations spatiales entre les points clés sur les dents.

À travers ce projet, nous visons à mettre en avant le potentiel de l'utilisation des techniques de traitement d'images et d'apprentissage profond dans le domaine dentaire. En exposant la méthodologie utilisée pour la collecte de données, l'augmentation et l'entraînement du modèle, nous offrons un aperçu complet de notre approche de l'analyse d'images dentaires.

# L’augmentation des données

## 1. Horizontal Flip

Cette fonction effectue une rotation horizontale d'une image. Elle prend en entrée une image (img), utilise la transformation transforms.functional.hflip de PyTorch pour effectuer la rotation, puis retourne l'image résultante.

## 2. Resize and Crop

Cette fonction charge une image, la convertit en un tenseur PyTorch, puis la redimensionne en ajoutant l'incrément à sa largeur. Ensuite, elle recadre l'image à sa taille d'origine. Le résultat est retourné comme un tenseur.

## 3. Horizontal Flip

Cette fonction effectue une rotation horizontale d'un tenseur représentant une image. Elle utilise la fonction torch.flip pour effectuer la rotation le long de la dimension de la largeur (dimension 3) du tenseur.

## 4. Resize and Crop

Cette fonction prend un tenseur représentant une image en entrée et redimensionne sa largeur en ajoutant l'incrément. Elle utilise l'interpolation bilinéaire pour redimensionner le tenseur. Ensuite, elle recadre le tenseur à sa largeur d'origine, en ajustant les coordonnées de début et de fin en conséquence.

## 5. Transform Coordinates

Cette fonction prend une liste de coordonnées (x, y) en entrée, ainsi que la largeur d'origine et l'incrément. Elle transforme les coordonnées x en fonction du ratio de largeur calculé et de l'incrément, puis retourne la liste des coordonnées transformées.

```
import json
original = 448
increments = torch.tensor([0,10, 20, 30, 40])
resized_tensors = []
for increment in increments:
    augmented_tensor = resize_and_crop_image_tensor(your_dataset_tensor,
increment)
    flipped_augmented_tensor = flip_image_horizontal_tensor(augmented_tensor)
    resized_tensors.append(augmented_tensor)
    print("f")
```

```
    resized_tensors.append(flipped_augmented_tensor)
    print("f")
flipped_tensor = flip_image_horizontal_tensor(your_dataset_tensor)
resized_tensors.append(flipped_tensor)
augmented_dataset = torch.stack(resized_tensors, dim=0)
print("Augmented dataset size:", augmented_dataset.size())
```

```
Augmented dataset size: torch.Size([11, 47, 3, 448, 448])
```

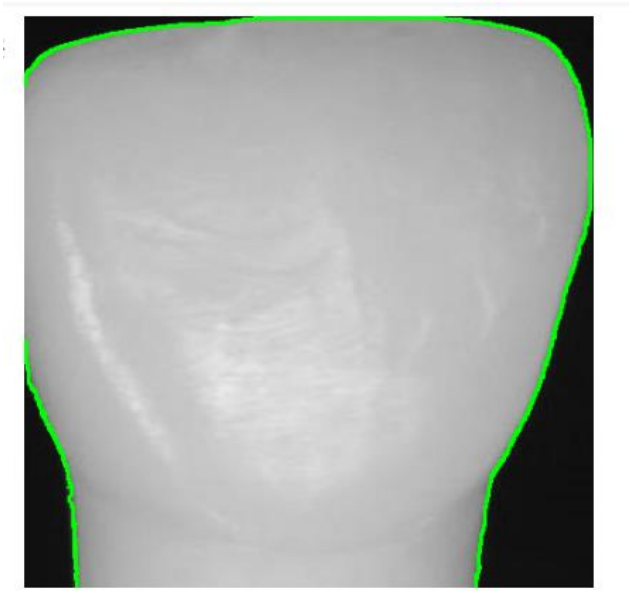
## Détection des contours

Dans cette partie on va utiliser la bibliothèque OpenCV pour détecter les contours des dents dans une image et les sauvegarder. Tout d'abord, l'image est lue en niveaux de gris. Ensuite, un masquage flou est appliqué pour améliorer la netteté de l'image. La détection de contours Canny est ensuite utilisée pour extraire les contours. Les contours détectés sont ensuite trouvés à l'aide de la fonction `cv2.findContours`, en utilisant une méthode de récupération externe (`cv2.RETR_EXTERNAL`) et une approximation de chaîne simple (`cv2.CHAIN_APPROX_SIMPLE`). Enfin, les contours sont dessinés sur l'image d'origine en couleur verte, et l'image résultante est affichée.

```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np

def draw_teeth_contours(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # Apply unsharp masking for sharpness enhancement
    blurred = cv2.GaussianBlur(image, (0, 0), 3)
    sharpness = cv2.addWeighted(image, 1.5, blurred, -0.5, 0)
    # Apply Canny edge detector
    edges = cv2.Canny(sharpness, 50, 150)
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    result_image = cv2.drawContours(cv2.cvtColor(image, cv2.COLOR_GRAY2BGR),
contours, -1, (0, 255, 0), 2)

    cv2_imshow(result_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
draw_teeth_contours(afolp+"/operation_5/32.jpg")
```

Voici un exemple des résultats obtenues suite à cette détection de contour :



L'utilité de ce code réside dans la génération de contours nets des dents, qui peuvent être utilisés ultérieurement dans l'entraînement de modèles, notamment dans des tâches liées à la segmentation ou à la détection des dents dans des images médicales dentaires.

# Le modèle de détection

## Prétraitement des données

Pour cette partie applique un seuillage binaire pour convertir l'image en une image binaire. Ensuite, il convertit cette image binaire en un tenseur PyTorch de type flottant et ajuste ses dimensions pour correspondre aux exigences d'entrée d'un modèle, en ajoutant des dimensions supplémentaires.

## Définition du modèle

L'architecture du modèle, baptisé "CoordinatesPredictor", est un réseau de neurones convolutifs (CNN) destiné à la prédiction de coordonnées à partir d'images en niveaux de gris. La première couche, `self.conv1`, utilise une convolution avec 32 filtres de taille 3x3, suivie d'une fonction d'activation ReLU. Ensuite, une opération de pooling max est appliquée (`self.pool`) pour réduire la dimension spatiale de l'image. La deuxième couche de convolution, `self.conv2`, avec 64 filtres de taille 3x3, est suivie d'une autre opération de pooling max. Les caractéristiques extraites sont ensuite aplaties et alimentées dans deux couches fully connected, `self.fc1` et `self.fc2`, avec des activations ReLU. La dernière couche `self.fc2` produit une sortie de dimension 8, représentant les coordonnées prédites (x, y). L'architecture vise à capturer des informations spatiales complexes pour la prédiction précise des coordonnées à partir des images en entrée.

```
class CoordinatesPredictor(nn.Module):
    def __init__(self):
        super(CoordinatesPredictor, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(64 * 112 * 112, 256)
        self.fc2 = nn.Linear(256, 8)  # Output 2 coordinates (x, y)
```

La fonction `forward` du modèle orchestre le passage des données à travers ces couches. Elle applique la convolution, la ReLU, et le pooling max successivement, puis aplatit les caractéristiques pour les passer à travers les couches fully connected. La fonction d'activation ReLU est utilisée pour introduire une non-linéarité. En fin de compte, le modèle produit une sortie qui représente les coordonnées prédites après le passage à travers les différentes couches du réseau. L'architecture choisie est adaptée à la tâche de prédiction de coordonnées et utilise des opérations de base des CNN pour extraire des motifs significatifs des images.

```
    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 64 * 112 * 112)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

## Entrainment du modèle

Le processus d'entraînement de ce modèle suit plusieurs étapes. Tout d'abord, le modèle, défini par la classe `CoordinatesPredictor`, est initialisé et déplacé sur le dispositif de calcul spécifié, probablement un GPU (`to(device)`). Ensuite, le jeu de données est préparé à l'aide de transformations PyTorch. Le modèle est optimisé à l'aide de la fonction de perte Mean Squared Error (`nn.MSELoss`) et de l'optimiseur Adam (`optim.Adam`) avec un taux d'apprentissage de 0.001.

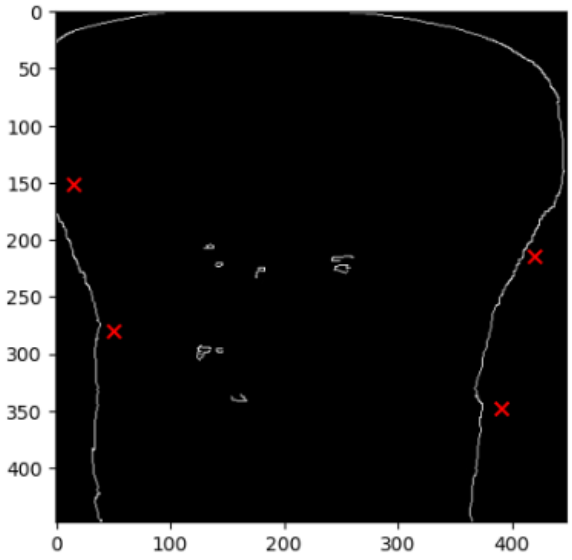
Le processus d'entraînement s'effectue sur plusieurs époques (`num_epochs`). À chaque époque, une boucle imbriquée parcourt les différentes opérations et fichiers JPG, charge les données JSON correspondantes, extrait les coordonnées de ces données, et obtient l'image d'entrée sous forme de tenseur 2D à l'aide de la fonction `get_image_from_path`. Ces coordonnées sont utilisées comme valeurs cibles. Ensuite, l'optimiseur est réinitialisé, le modèle est évalué pour obtenir des prédictions (`outputs`), la perte est calculée entre les prédictions et les coordonnées cibles, et la rétropropagation du gradient est effectuée pour mettre à jour les poids du modèle. Ce processus est répété pour toutes les opérations et fichiers JPG.

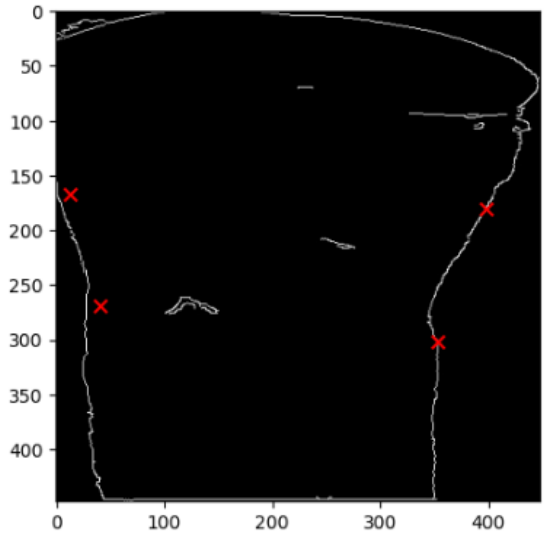
```
import torch.optim as optim
model = CoordinatesPredictor().to(device)
transform = transforms.Compose([transforms.ToTensor()])
# Initialize the model, loss function, and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
num_epochs = 50

for epoch in range(num_epochs):
    for operation_value in range(9):
        for jpg_value in range(47):
            # Load JSON data
            json_path = f"{ajolp}/operation_{operation_value}/{jpg_value}.json"
            with open(json_path, 'r') as file:
                data = json.load(file)
            coordinates = [[float(obj["bndbox"]["xmin"]),
int(obj["bndbox"]["ymin"])] for obj in data["annotation"]["object"]]
            image_path = f"{acolp}/operation_{operation_value}/{jpg_value}.jpg"
            input_image = get_image_from_path(image_path).to(device)
            optimizer.zero_grad()
            outputs = model(input_image)
            target_coordinates = torch.tensor(coordinates).view(-1).to(device)
            loss = criterion(outputs, target_coordinates)
```

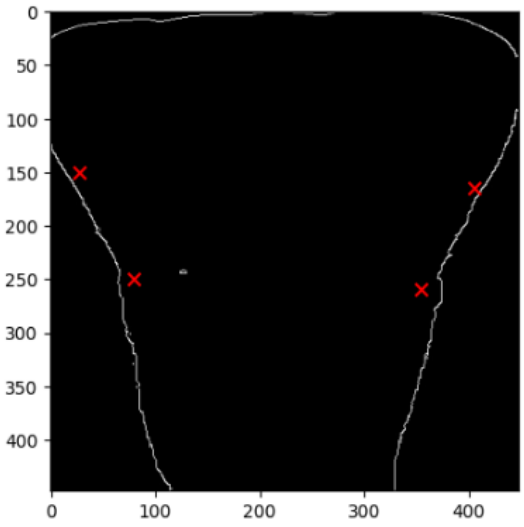
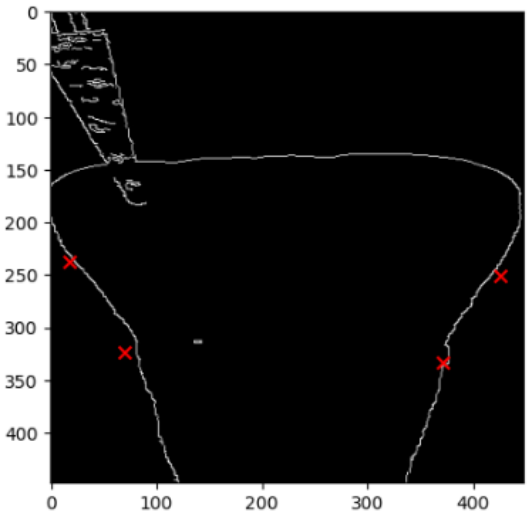
```
            loss.backward()
            optimizer.step()

    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')
torch.save(model.state_dict(), 'coordinates_predictor_model.pth')
```

## Résultat du modèle

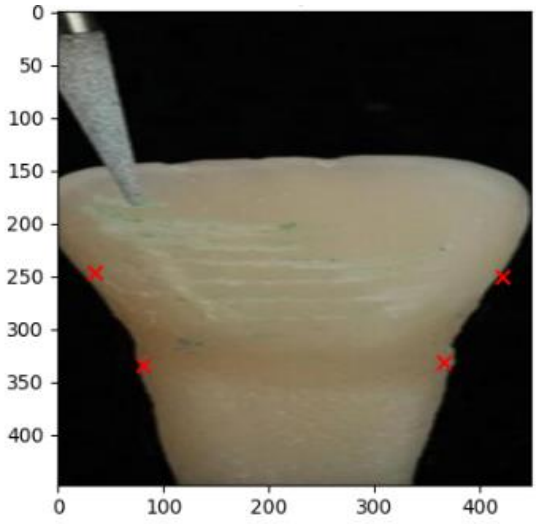
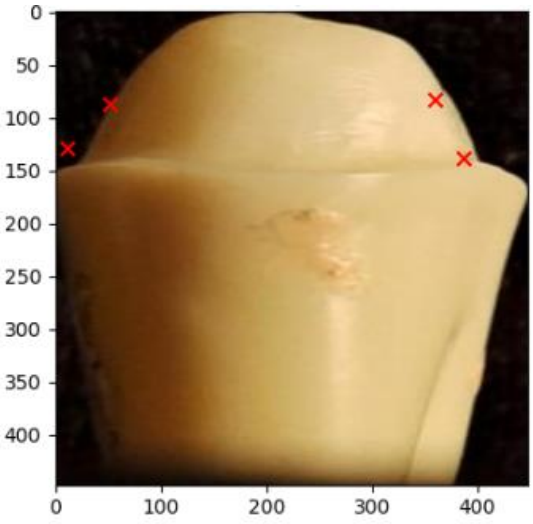
Le modèle de prédiction de coordonnées est entraîné à superposer les prédictions au-dessus des images originales. Chaque point représentant une coordonnée prédite est tracé sur l'image pour permettre une évaluation visuelle immédiate de la performance du modèle. Cette approche facilite la compréhension intuitive des écarts entre les coordonnées prédites et réelles, fournissant ainsi une représentation graphique claire de la précision du modèle dans la localisation des points d'intérêt.

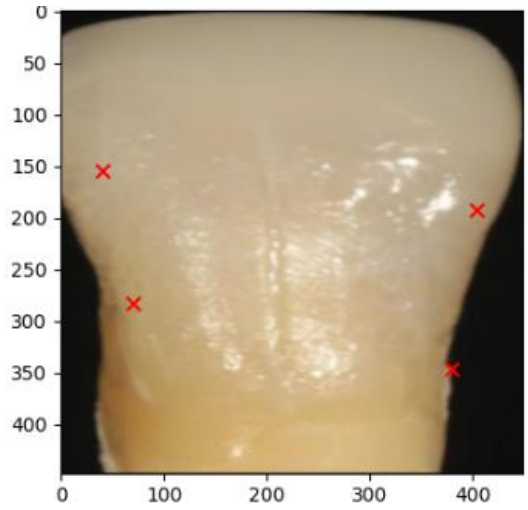


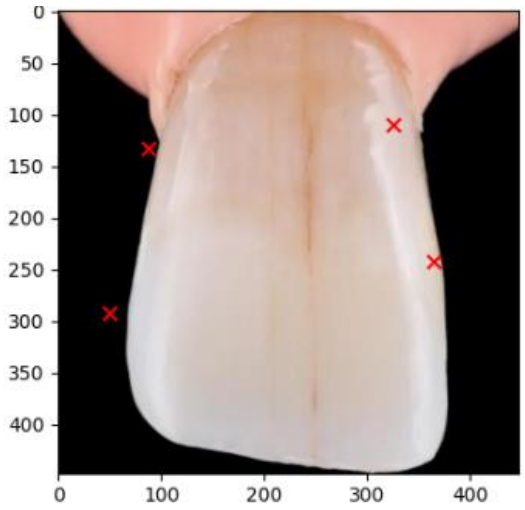



La superposition des prédictions sur les images originales offre une visualisation efficace des performances du modèle de prédiction de coordonnées.




## Conclusion

En conclusion, ce travail de détection de contours de dents a été guidé par une méthodologie solide et a abouti à des résultats prometteurs. En utilisant une combinaison de techniques telles que le masquage flou, la détection de contours Canny, et la superposition des contours sur les images originales, le modèle a démontré une capacité à extraire avec succès les contours des dents dans des images médicales. L'utilisation d'un réseau de neurones convolutifs a permis d'apprendre des représentations complexes pour améliorer la précision de la détection. La visualisation des contours prédits au-dessus des images originales offre une évaluation visuelle efficace des performances du modèle. Les étapes de prétraitement des données, d'entraînement du modèle, et de représentation graphique des résultats ont été clairement définies et exécutées. Cependant, des améliorations pourraient être envisagées, telles que l'ajout de données supplémentaires pour une généralisation accrue. Dans l'ensemble, ce travail jette les bases d'une approche prometteuse pour la détection de contours de dents, avec des perspectives d'extension et d'amélioration pour des applications futures dans le domaine de l'imagerie médicale dentaire.