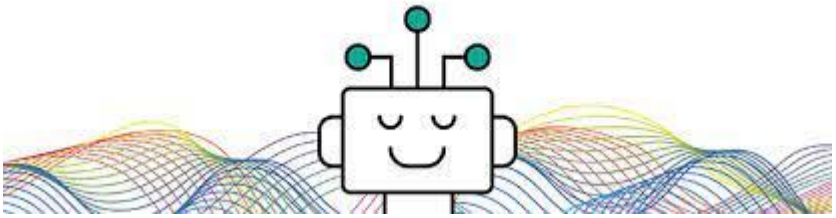
Université Hassan 1er
Ecole Nationale des Sciences Appliquées de Berrechid
Département de mathématique et informatique
Filière : Ingénierie des Systèmes d'Information et BIG DATA
Module : fouille de donnée
Semestre : S9

COMPTE RENDU DU TP

Détection des angles de Taper dans les dents artificielles par le biais de l'apprentissage automatique



Réalisé Par : SAGHIR Aya

Encadré par : Mr Hrimech

Année Universitaire : 2023/2024

Contents

Introduction

Les dents, éléments intégraux de l'anatomie humaine, jouent un rôle essentiel dans divers aspects de la vie, allant de la digestion à l'impact sur l'apparence générale. Ce projet explore le domaine de l'analyse d'images dentaires en utilisant des techniques visant à améliorer et à extraire des informations précieuses à partir d'images de dents. L'objectif est de mettre au point un modèle capable de prédire avec précision les coordonnées de points spécifiques sur les dents, ce qui serait bénéfique pour diverses applications dentaires telles que la planification et l'évaluation des traitements.

Pour atteindre cet objectif, notre projet suit une approche en plusieurs étapes. Nous débutons par la collecte d'un ensemble diversifié d'images de dents, qui constitue la base de notre modèle. L'augmentation de ces images crée un ensemble de données robuste pour l'entraînement, permettant ainsi au modèle de s'adapter efficacement à différentes situations. De plus, nous utilisons des fichiers JSON contenant des coordonnées annotées en tant que données de référence, guidant ainsi le modèle dans l'apprentissage des relations spatiales entre les points clés sur les dents.

À travers ce projet, notre objectif est de mettre en lumière le potentiel de l'utilisation des techniques de traitement d'images et d'apprentissage profond dans le domaine dentaire. En expliquant la méthodologie employée pour la collecte de données, l'augmentation et l'entraînement du modèle, nous offrons une vue d'ensemble complète de notre approche de l'analyse d'images dentaires.

Data Augmentation

Ce code concerne l'augmentation de données dans le contexte du traitement d'images, notamment pour des tâches liées à des coordonnées spécifiques.

1.1 Inversion Horizontale d'Image

La fonction `flip_image_horizontal(img)` inverse horizontalement une image d'entrée en utilisant la transformation `transforms.functional.hflip`.

```
def flip_image_horizontal(img):
    transform = transforms.functional.hflip
    flipped_img = transform(img)
    return flipped_img
```

1.2 Redimensionnement et Recadrage d'Image

La fonction `resize_and_crop_image(image_path, increment)` ouvre une image à partir d'un chemin donné, la transforme en tenseur, puis la redimensionne et la recadre selon des paramètres spécifiques.

```python
def resize_and_crop_image(image_path, increment):
    img = Image.open(image_path)
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Resize((img.size[1], img.size[0] + increment)),  # Resize width only
        transforms.CenterCrop((img.size[1], img.size[0])),  # Crop to original size
    ])
    resized_and_cropped_img = transform(img)
    return resized_and_cropped_img
```

1.3 Inversion Horizontale de Tenseur

La fonction `flip_image_horizontal_tensor(img_tensor)` inverse horizontalement un tenseur d'image en utilisant la fonction `torch.flip`.

```python
def flip_image_horizontal_tensor(img_tensor):
    return torch.flip(img_tensor, dims=[3])
```

1.3.1 Redimensionnement et Recadrage de Tenseur

La fonction `resize_and_crop_image_tensor(img_tensor, increment)` redimensionne et recadre un tenseur d'image en utilisant l'interpolation bilinéaire.

```python
def resize_and_crop_image_tensor(img_tensor, increment):
    print(img_tensor.size())
    batch_size ,num_channels, height, width  = img_tensor.size()
    new_width = width + increment

    # Resize along the width dimension
     resized_tensor = torch.nn.functional.interpolate(img_tensor, size=(height, new_width),
mode='bilinear')

    # Calculate cropping range for original width
    crop_start = 0
    crop_end = width

    if increment > 0:
        crop_end = min(crop_end, new_width - increment)

    cropped_tensor = resized_tensor[:, :, :, crop_start:crop_end]  # Crop to original width

    return cropped_tensor
```

2. Transformation de Coordonnées

La fonction `transform_coordinates(points, original_width, increment)` transforme les coordonnées d'un point en fonction de la largeur originale et de l'incrément.

```python
def transform_coordinates(points, original_width, increment):
    width_ratio =( original_width + increment )/ original_width

    transformed_points = []
    for x, y in points:
        transformed_x = x * width_ratio - increment/2
```

```
        transformed_points.append((transformed_x, y))
    print(transformed_points)
    return transformed_points
```

2.1 Transformation de Coordonnées en JSON

La fonction `transform_coordinates_in_json(jsondata, original_width, increment)` applique la transformation de coordonnées à un fichier JSON contenant des annotations.

```
def transform_coordinates_in_json(jsondata, original_width, increment):
    json_data = copy.deepcopy(jsondata)
    width_ratio = (original_width + increment) / original_width

    for obj in json_data['annotation']['object']:
        xmin = int(obj['bndbox']['xmin'])
        xmax = int(obj['bndbox']['xmax'])

        transformed_xmin = xmin * width_ratio - increment / 2
        transformed_xmax = xmax * width_ratio - increment / 2

        obj['bndbox']['xmin'] = str(int(transformed_xmin))
        obj['bndbox']['xmax'] = str(int(transformed_xmax))

    return json_data
```

2.2 Inversion de Coordonnées en JSON

La fonction `flip_coordinates_in_json(jsondata, image_width)` effectue une inversion des coordonnées dans un fichier JSON par rapport à la largeur de l'image.

```
def flip_coordinates_in_json(jsondata, image_width):
    json_data = copy.deepcopy(jsondata)

    for obj in json_data['annotation']['object']:
        xmin = int(obj['bndbox']['xmin'])
        xmax = int(obj['bndbox']['xmax'])

        flipped_xmin = image_width - xmin
        flipped_xmax = image_width - xmax

        obj['bndbox']['xmin'] = str(flipped_xmin)
        obj['bndbox']['xmax'] = str(flipped_xmax)

    return json_data
```

2.3 Inversion de Coordonnées

La fonction `flip_coordinates(points, image_width)` effectue une inversion des coordonnées par rapport à la largeur de l'image.

```
def flip_coordinates(points, image_width):
    flipped_points = [(image_width - x, y) for x, y in points]
    print(flipped_points)

    return flipped_points
```

3. Boucle d'Augmentation de Données

La dernière partie du code démontre comment utiliser les fonctions précédentes pour augmenter un ensemble de données. Il itère sur des incréments spécifiés, applique des transformations de redimensionnement et d'inversion au tenseur de données, puis imprime la taille du jeu de données augmenté.

Ensuite, à l'intérieur de la boucle, le code manipule des fichiers JSON en lisant, transformant les coordonnées, créant des répertoires et écrivant les données JSON augmentées dans de nouveaux fichiers, à la fois pour les coordonnées redimensionnées et inversées.

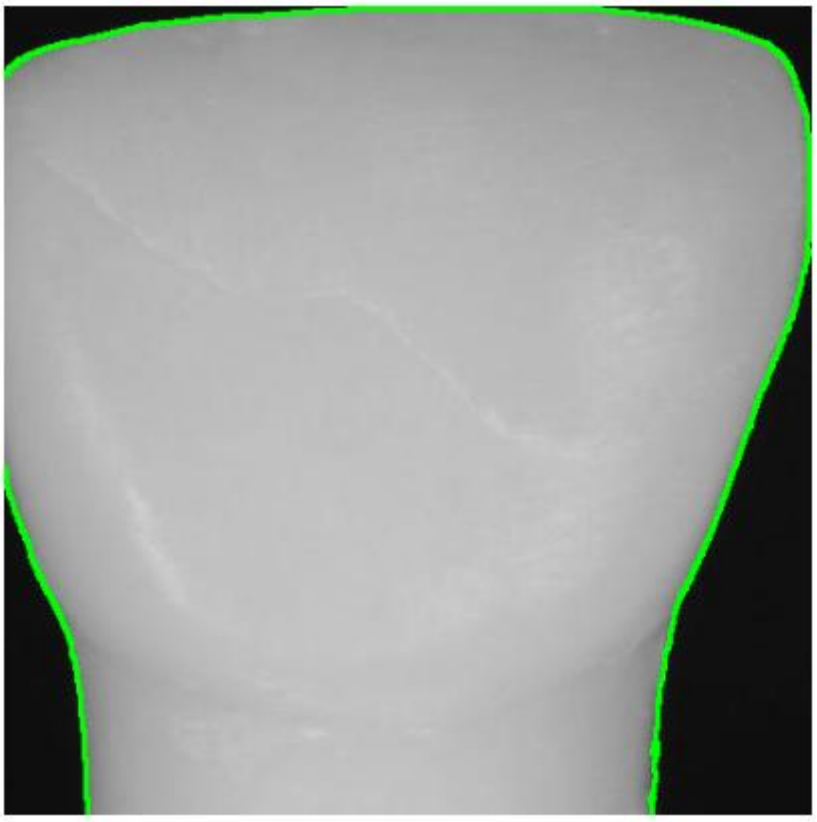
```
def plot_transformations(image_path, points, increment):
    original_img = Image.open(image_path)
    original_width = original_img.size[0]
    print(original_width)
    transformed_img = resize_and_crop_image(image_path, increment)
     transformed_img_pil = transforms.ToPILImage()(transformed_img)  # Convert to PIL for
flipping

    flipped_img = flip_image_horizontal(original_img)

    transformed_points = transform_coordinates(points, original_width, increment)
    flipped_points = flip_coordinates(points, original_width)

    fig, axs = plt.subplots(1, 3, figsize=(15, 5))

    axs[0].imshow(original_img)
    axs[0].set_title('Original Image')
    for point in points:
        axs[0].plot(point[0], point[1], 'ro')

    axs[1].imshow(transformed_img_pil)  # Show the transformed image PIL object
    axs[1].set_title('Transformed Image')
    for point in transformed_points:
        axs[1].plot(point[0], point[1], 'ro')

    axs[2].imshow(flipped_img)  # Show the flipped image PIL object
    axs[2].set_title('Flipped Image')
    for point in flipped_points:
        axs[2].plot(point[0], point[1], 'ro')

    plt.tight_layout()
    plt.show()

image_path = folp+'/0.jpg'  # Replace with the path to your image
points = [(50, 50), (100, 150), (200, 100), (300, 200)]  # Replace with your points
increment = 50  # Replace with the increment value for resizing

# Plot transformations
plot_transformations(image_path, points, increment)
```

Outline

Outline script

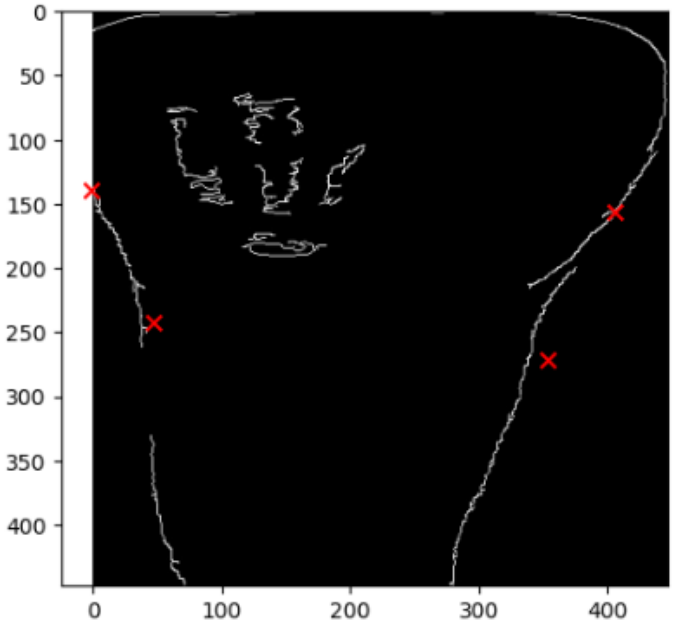
Ce code Python illustre le processus pour obtenir les contours d'une image, mettant ainsi en évidence les éléments importants. Tout d'abord, l'image est lue en niveaux de gris, puis une technique d'amélioration de la netteté, appelée unsharp masking, est appliquée pour renforcer la clarté de l'image. Ensuite, le détecteur de bords Canny est utilisé pour détecter les contours dans l'image améliorée. Les contours sont ensuite identifiés à l'aide de la fonction findContours de la bibliothèque OpenCV. Enfin, les contours sont dessinés sur l'image originale en couleur, et le résultat est affiché. Ce script permet de visualiser clairement les contours des objets dans une image, ce qui peut être utile pour se concentrer sur des détails importants dans le domaine de l'analyse d'images dentaires.

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

def draw_teeth_contours(image_path):
    # Read the image
    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)

    # Find contours
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Draw contours on the original image
     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()


# Replace 'your_image_path.jpg' with the actual path to your image file
draw_teeth_contours(afolp+"/operation_5/32.jpg")
```



Saving Outlines

Ce code Python concerne la sauvegarde des contours d'images. Il utilise la bibliothèque OpenCV pour lire une image en niveaux de gris, détecter les bords avec l'algorithme de Canny, trouver les contours externes, puis les dessiner sur une image noire. Ces contours sont ensuite sauvegardés dans un répertoire spécifique. Le script parcourt plusieurs images en boucle, générant des contours pour chacune et les sauvegardant dans des chemins spécifiques en fonction de l'opération et de la valeur de l'image.



Model training

Ce code définit un réseau neuronal simple, appelé CoordinatesPredictor, destiné à prédire les coordonnées à partir d'images en noir et blanc. Le réseau est composé de couches convolutionnelles (conv1, conv2), de couches de max-pooling (pool) et de couches entièrement connectées (fc1, fc2). La couche de sortie (fc2) produit deux coordonnées (x, y). Le modèle utilise la bibliothèque PyTorch, avec des opérations de convolution, de pooling et de couches entièrement connectées pour apprendre à prédire les coordonnées à partir des images d'entrée. Ce modèle est conçu pour fonctionner sur un processeur graphique (GPU).

```
import torch.nn as nn
import cv2
import numpy as np
device = torch.device("cuda")

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)

    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
```

Image Preprocessing Function

La fonction `get_image_from_path` traite une image d'entrée à partir d'un chemin spécifié. Elle lit l'image en niveaux de gris, applique un seuil binaire, puis la convertit en un tenseur PyTorch. Plus précisément, la fonction utilise la bibliothèque OpenCV pour lire l'image, la convertit en niveaux de gris, applique un seuil binaire, et enfin, elle crée un tenseur PyTorch représentant l'image traitée en ajoutant des dimensions supplémentaires. Ce processus permet d'obtenir une représentation adaptée à l'utilisation dans des modèles d'apprentissage automatique.

```
# Function to get a black and white image as a 2D tensor from a given path
def get_image_from_path(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    _, binary_image = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)
    return torch.tensor(binary_image, dtype=torch.float).unsqueeze(0).unsqueeze(0)
```

Training Loop

Ce code représente la boucle d'entraînement pour le modèle CoordinatesPredictor. La boucle itère sur un nombre spécifié d'époques, charge les données JSON, extrait les coordonnées, et entraîne le modèle en utilisant l'optimiseur Adam et la perte Mean Squared Error (MSE). Il initialise le modèle, la fonction de perte et l'optimiseur, puis parcourt les époques, les valeurs d'opération et les valeurs JPG, charge les données JSON, extrait les coordonnées, et effectue la rétropropagation pour ajuster les poids du modèle. Enfin, il sauvegarde le modèle entraîné.

```
import torch.optim as optim
model = CoordinatesPredictor().to(device)

# Initialize your dataset and dataloader
transform = transforms.Compose([transforms.ToTensor()])
# Initialize the model, loss function, and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
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)

            # Extract coordinates from JSON
            coordinates = [[float(obj["bndbox"]["xmin"]), int(obj["bndbox"]["ymin"])] for obj
in data["annotation"]["object"]]

            # Get the input image as a 2D tensor
            image_path = f"{acolp}/operation_{operation_value}/{jpg_value}.jpg"
            input_image = get_image_from_path(image_path).to(device)

            # Assuming input_image is a torch tensor
            optimizer.zero_grad()
            outputs = model(input_image)
            target_coordinates = torch.tensor(coordinates).view(-1).to(device)  # Flatten the
coordinates list, move to GPU
            loss = criterion(outputs, target_coordinates)
            loss.backward()
            optimizer.step()

    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')

# Save the trained model
torch.save(model.state_dict(), 'coordinates_predictor_model.pth')
```

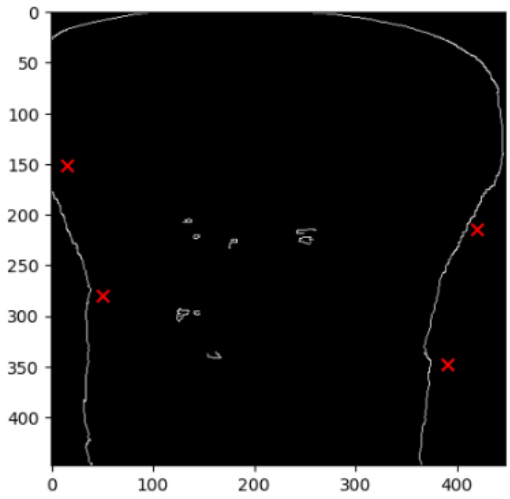
La boucle parcourt chaque image dans l'ensemble de données, calcule la perte, et met à jour les paramètres du modèle. Le modèle entraîné final est sauvegardé sous le nom 'coordinates_predictor_model.pth'. Cette démarche permet d'optimiser le modèle pour prédire avec précision les coordonnées des points spécifiques sur les dents, contribuant ainsi à l'amélioration des performances du modèle dans diverses applications dentaires.

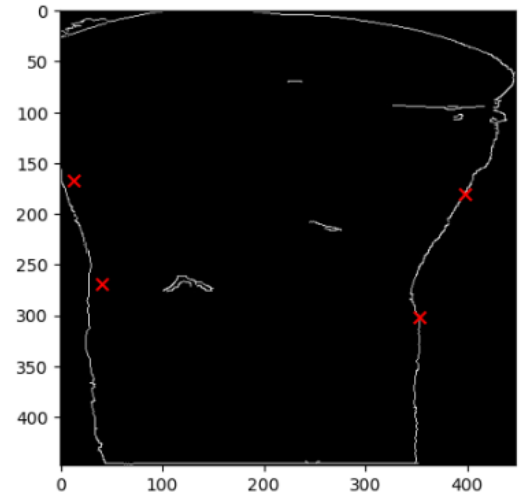
```
/usr/local/lib/python3.10/dist-packages/torch
  return F.mse_loss(input, target, reduction=
Epoch [1/10], Loss: 37912.7891
Epoch [2/10], Loss: 16765.1172
Epoch [3/10], Loss: 28989.5840
Epoch [4/10], Loss: 7781.4482
Epoch [5/10], Loss: 6753.6079
Epoch [6/10], Loss: 3831.1450
Epoch [7/10], Loss: 2350.6182
Epoch [8/10], Loss: 2210.8333
Epoch [9/10], Loss: 841.1655
Epoch [10/10], Loss: 6073.4912
```

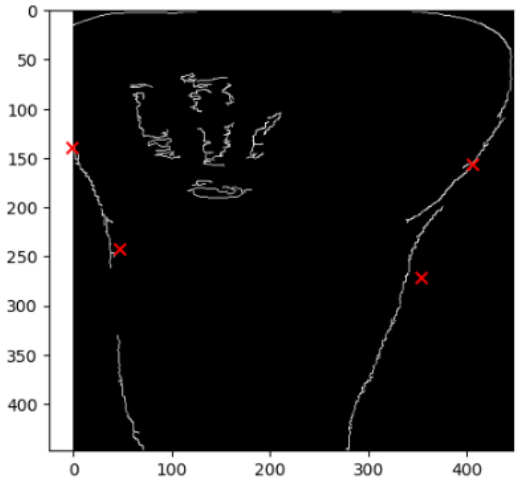
```
Epoch [1/10], Loss: 201.2026
Epoch [2/10], Loss: 195.1530
Epoch [3/10], Loss: 202.1035
Epoch [4/10], Loss: 153.1508
Epoch [5/10], Loss: 148.5576
Epoch [6/10], Loss: 130.4602
Epoch [7/10], Loss: 77.5002
Epoch [8/10], Loss: 66.0755
Epoch [9/10], Loss: 54.5706
Epoch [10/10], Loss: 72.6252
```

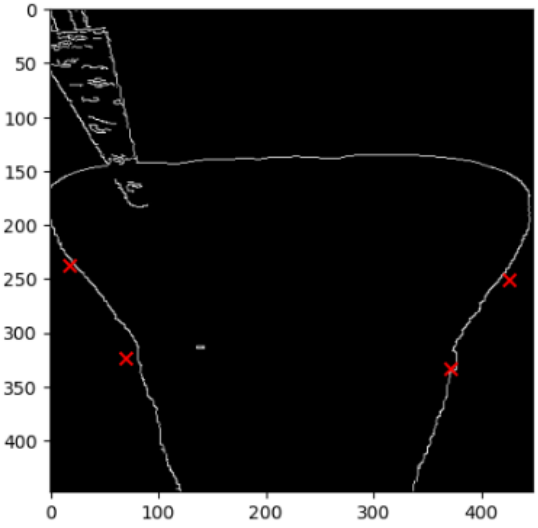
Résultats

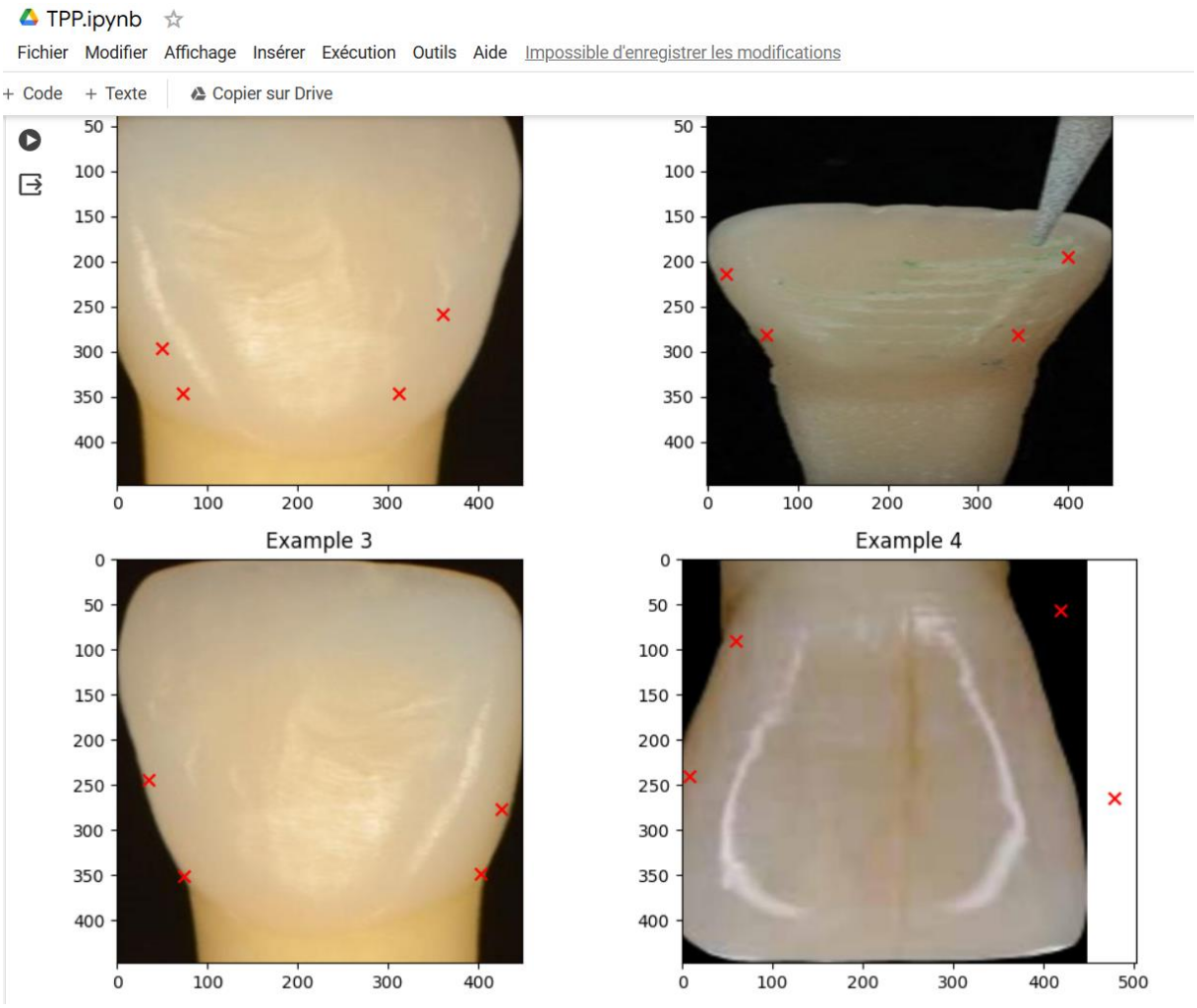
Voici les résultats du modèle de prédiction tracés au-dessus de l'image.


0
50
100
150
200
250
300
350
400
0
100
200
300
400


0
50
100
150
200
250
300
350
400
0
100
200
300
400


0
50
100
150
200
250
300
350
400
0
100
200
300
400


0
50
100
150
200
250
300
350
400
0
100
200
300
400


TPP.ipynb
Fichier Modifier Affichage Insérer Exécution Outils Aide Impossible d'enregistrer les modifications
+ Code + Texte Copier sur Drive
Example 3
Example 4

Conclusion

En conclusion, ce projet démontre avec succès l'efficacité de l'analyse d'images dentaires en utilisant des techniques avancées de traitement d'images et d'apprentissage profond. En suivant une approche en plusieurs étapes, nous avons collecté un ensemble diversifié d'images de dents, augmenté de manière significative la robustesse de notre modèle grâce à une augmentation appropriée des données, et utilisé des fichiers JSON pour guider le modèle dans l'apprentissage des relations spatiales. Le modèle final, sauvegardé sous le nom 'coordinates_predictor_model.pth', se montre prometteur pour la prédiction précise des coordonnées des points sur les dents, ouvrant ainsi la voie à des applications dentaires avancées telles que la planification et l'évaluation des traitements. Ce projet illustre de manière convaincante le potentiel de la combinaison du traitement d'images et de l'apprentissage profond dans le domaine dentaire.