ReadMe:

Task1 : Landmark Detection
Ultrasound Landmark Detection using PyTorch

This ultrasound landmark detection task is implemented in PyTorch. The task involves preprocessing ultrasound images, normalizing coordinates, defining a custom dataset class, training a convolutional neural network (CNN), and evaluating the model's performance.

Prerequisites
Python 3.x
PyTorch
NumPy
pandas
matplotlib
scikit-learn
tqdm
torchvision
PIL
OpenCV
Google Colab (for execution in a Colab environment)

The task is organized as follows:

data/: Contains datasets and annotations.
models/: Defines the CNN architecture for landmark detection.
utils/: Utility functions for data preprocessing and evaluation.
training: Code to train the landmark detection model.
evaluatation: Code to evaluate the trained model on test data.

Data Preparation: Organize your ultrasound images and annotations. Update the file paths in the code accordingly.

Training: Run the train script to train the CNN model. Adjust hyperparameters and network architecture as needed.

Visualization: Visualize the results using matplotlib or other visualization libraries.

Testing:
-Definition model
-Loading Model from saved weights
-Load image through image path
-Predict landmarkPoint
-Draw predicted points on given image

Task2: Segmentation Identification of cranium and bymetric points
Ultrasound Image Segmentation using DUCKNet and UNet
This task contains training and evaluating deep learning models for ultrasound image segmentation using the DUCKNet and UNet architectures. The models are trained to segment ultrasound images into regions of interest

Training
- Two models are provided: DUCKNet and UNet. You can choose either of these models for training.
- Adjust the hyperparameters such as learning rate, batch size, and number of epochs as per your requirements.
- Train the model using the `train` function, which takes care of both training and validation.

Evaluation

- After training, the model can be evaluated on a validation set using the `visualize_segmentation` function, which visualizes the original image, ground truth mask, and predicted mask side by side.
- You can also visualize the raw model output and convert it into a binary mask using the provided functions.

Saving and Loading Models

- Trained model weights can be saved using `torch.save` and loaded using `torch.load`.
- Ensure compatibility between the model architecture and the saved state dict.

#Testing Steps:
-Model Definition
-Model Loading:Pre-trained weights for DUCKNet are loaded from Google Drive.

-Predicting Output Mask
- The model predicts the output mask for an ultrasound image.
- Visualization functions display the original image and predicted mask.

-Fitting Ellipse on Predicted Mask
- Ellipse fitting is performed on the predicted mask.
- Major and minor axis endpoints are extracted from the fitted ellipse.

-Drawing Predicted Biometric Points on Original Image
- Biometric points are drawn on the original image.
- Major and minor axis endpoints are displayed with different colors.

-Scaling Predicted Points to Original Image Dimensions

- Predicted points are rescaled to match the original image size.