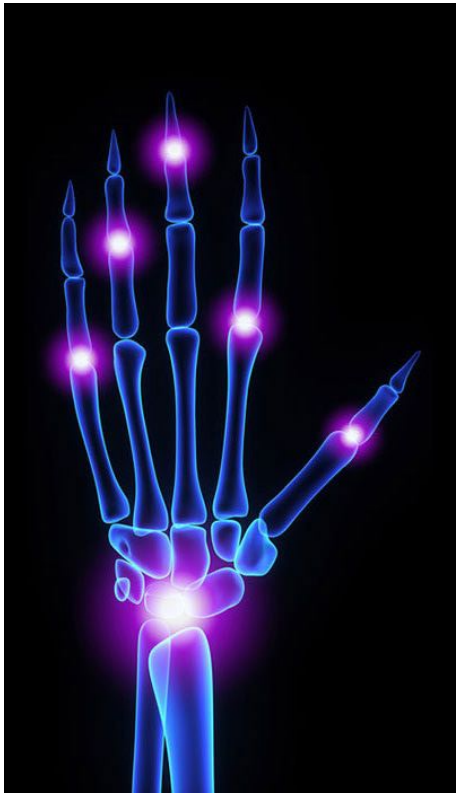
Osteoarthritis Detection in Hand Joints

Course Name-Computer Vision (CS671)
Instructor: Dr Juan Shan

Team Members:
- Anusha Guruprasad
- Darsh Joshi
- Mihika Mishra

Introduction

- Osteoarthritis (OA) of the hand is a systemic autoimmune disease, where early diagnosis and treatment are essential for effective management. It is typically detected using X-ray imaging, which is also employed for identifying conditions like fractures, tumors, foreign objects, and degenerative diseases. Our study focuses on developing an automated system for the detection and classification of hand OA, aiming to enhance the current manual process that radiologists use to mark and measure specific joints in hand X-ray images.
- This proposed system leverages advanced image processing and machine learning algorithms to accurately identify critical finger joints, streamlining the assessment of OA severity and improving the overall efficiency and accuracy of the diagnostic process.



Data & Initial Steps

Requirements & Classes

- **Distal Interphalangeal (DIP)**
- **Proximal Interphalangeal (PIP)**
- **Metacarpophalangeal (MCP)**

Data Preprocessing

- In the data preprocessing phase of our project, we implemented MATLAB script to efficiently prepare our X-ray image dataset for analysis. The primary goals were to convert DICOM files to the more accessible JPEG format and systematically organize the corresponding labeling information.

- These preprocessing steps are vital in ensuring the dataset's compatibility with diverse analytical tools and maintaining an organized, efficient workflow for processing our large-scale medical image dataset.

- We have a folder with several directory to access the DICOM file The data also contains 3577 txt files that are labels .

Conversion of Data to Jpeg files

DICOM to JPEG Conversion: The first script, *convertDicomInFoldersStartingWith9*, targets specific folders (starting with '9') in the source directory to locate DICOM files, particularly in the 'v06' subdirectories. It then uses convertDicomToJpgInFolder and convertDicomToJpg functions to convert these files into JPEG format. This step is essential for making the images compatible with a wide range of image processing and machine learning tools.

Label File Management: The enhanced version of the script adds label file management. After converting DICOM files to JPEG, it moves and renames the associated label files to align with the new JPEG files using the renameLabelFile function. This alignment is crucial for integrating the images with their diagnostic data for automated analysis.

Snippet of the code

```
source_directory = '/Users/mihikamishra/Documents/cv_folder/x_ray_data';
output_directory = '/Users/mihikamishra/Documents/cv_folder/output_folder';

convertDicomInFoldersStartingWith9(source_directory, output_directory);

function convertDicomInFoldersStartingWith9(source_dir, output_dir)
    folders = dir(fullfile(source_dir, '9*'));

    for i = 1:numel(folders)
        current_folder = fullfile(folders(i).folder, folders(i).name, 'v06');

        if exist(current_folder, 'dir') == 7
            convertDicomToJpgInFolder(current_folder, output_dir, folders(i).name);
        else
            disp(['v06 folder not found in ', folders(i).name]);
        end
    end
end

function convertDicomToJpgInFolder(folder, output_dir, folder_name)
    dicom_filename = fullfile(folder, '001');

    % Create a unique filename using the folder name
    jpg_filename = fullfile(output_dir, [folder_name, '.jpg']);

    convertDicomToJpg(dicom_filename, jpg_filename);
end

function convertDicomToJpg(dicomFilename, jpgFilename)
    dicomImage = dicomread(dicomFilename);
    adjustedImage = imadjust(dicomImage);

    if isa(adjustedImage, 'uint16')
        adjustedImage = uint8(255 * mat2gray(adjustedImage));
    end

    imwrite(adjustedImage, jpgFilename, 'jpg');
```

```
% Main code and primary function
source_directory = '/Users/mihikamishra/Documents/work/x_ray_data';
output_directory = '/Users/mihikamishra/Documents/work/output_folder';
label_directory = '/Users/mihikamishra/Documents/work/txt'; % Add your label directory path

convertDicomInFoldersStartingWith9(source_directory, output_directory, label_directory);

function convertDicomInFoldersStartingWith9(source_dir, output_dir, label_dir)
    folders = dir(fullfile(source_dir, '9*'));

    for i = 1:numel(folders)
        current_folder = fullfile(folders(i).folder, folders(i).name, 'v06');

        if exist(current_folder, 'dir') == 7
            image_filename = convertDicomToJpgInFolder(current_folder, output_dir, folders(i
            renameLabelFile(label_dir, output_dir, folders(i).name, image_filename);
        else
            disp(['v06 folder not found in ', folders(i).name]);
        end
    end
end

function image_filename = convertDicomToJpgInFolder(folder, output_dir, folder_name)
    dicom_filename = fullfile(folder, '001');
    image_filename = fullfile(output_dir, [folder_name, '.jpg']);

    convertDicomToJpg(dicom_filename, image_filename);
    return;
end

function convertDicomToJpg(dicomFilename, jpgFilename)
    dicomImage = dicomread(dicomFilename);
    adjustedImage = imadjust(dicomImage);

    if isa(adjustedImage, 'uint16')
        adjustedImage = uint8(255 * mat2gray(adjustedImage));
    end
```

Baseline Method - Manual

Computer Vision - Image Processing Techniques Based Detection

Techniques Used :

- Gaussian Blur
- Gamma Correction
- CLAHE (Contrast Limited Adaptive Histogram Equalization)
- Edge Detection - Canny
- K Means Clustering

Considered but not Used:

- Adaptive Threshold
- Distance Transformation
- Hough Filter for Circular edge detection

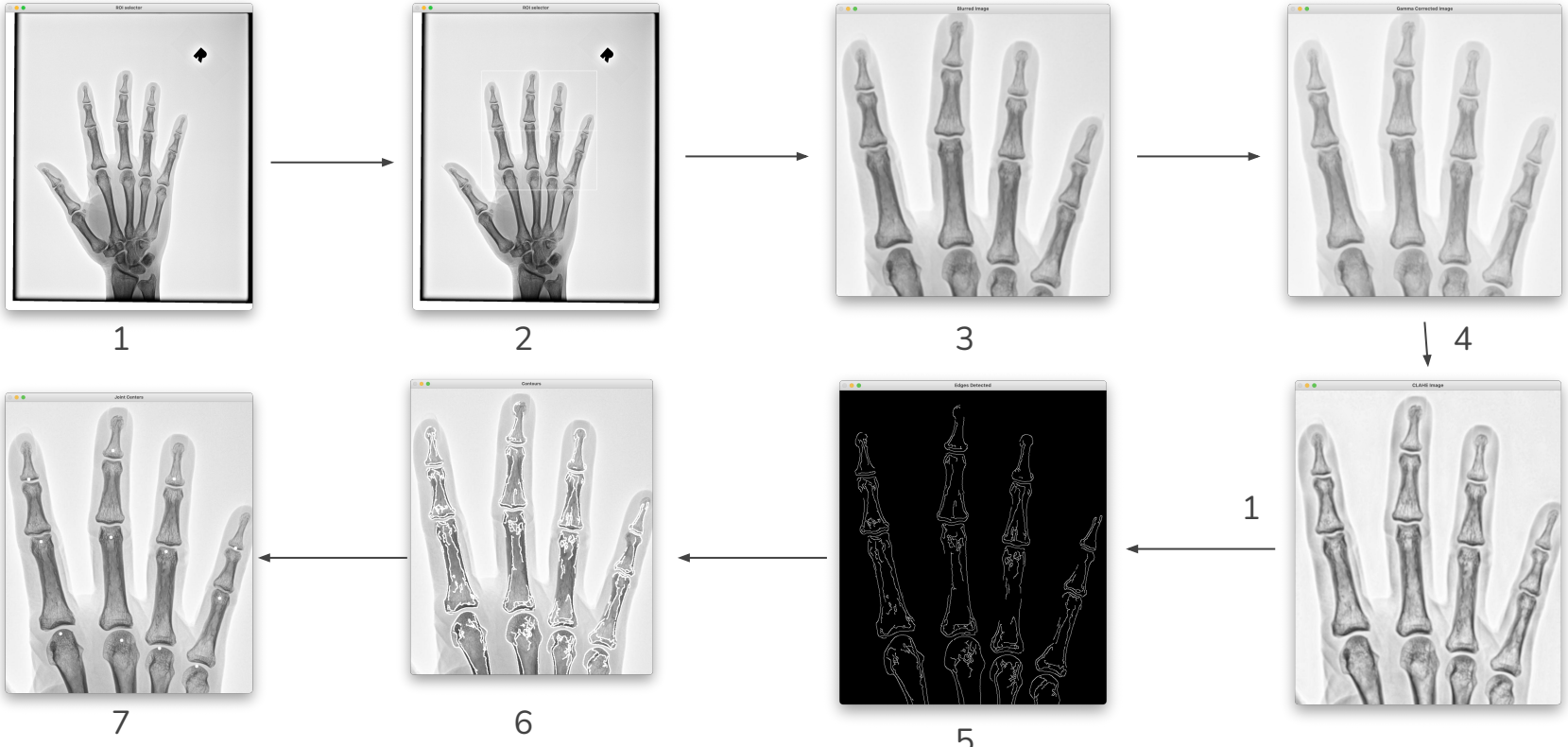
Process Flow

- Manual
  - Dealing with unlabeled data is always difficult. This method is supposed to be a baseline indication on where do we start. Thus, this method has manual cropping.

- Upload via Python Library called ‘tkinter’
  - User will upload the image from their file system.

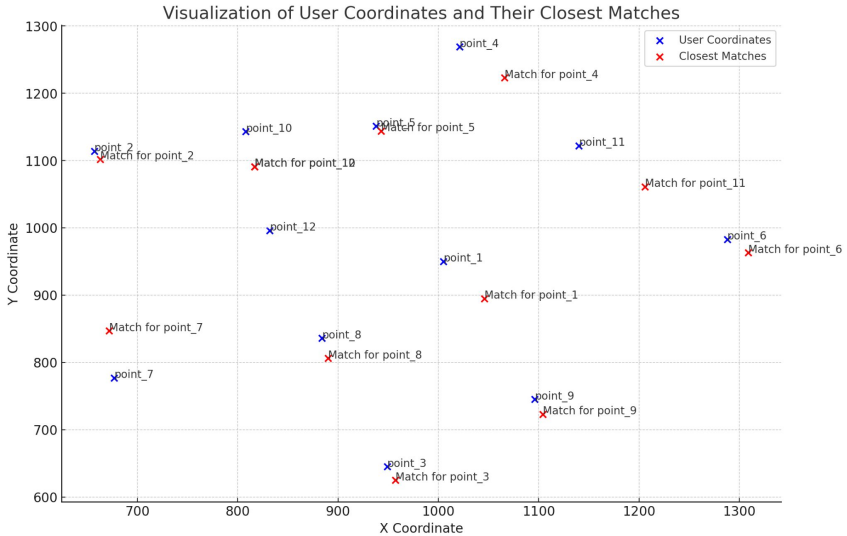
- Manual Cropping
  - User will manually crop the image which then will be processed.

Input to Output Journey



Few more successful examples

Evaluation


Visualization of User Coordinates and Their Closest Matches
User Coordinates
Closest Matches
Y Coordinate
X Coordinate
1300
1200
1100
1000
900
800
700
600
700
800
900
1000
1100
1200
1300
point_4
Match for point_4
point_10
point_5
Match for point_5
point_2
Match for point_2
point_11
Match for point_10
Match for point_11
point_12
point_6
Match for point_6
point_1
Match for point_1
Match for point_7
point_8
Match for point_8
point_7
point_9
Match for point_9
point_3
Match for point_3

Demo

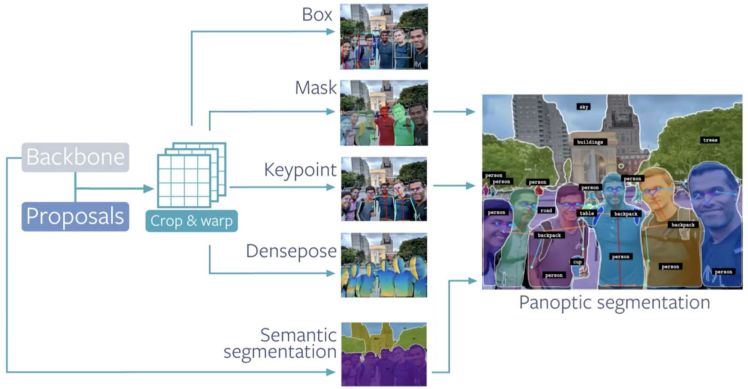
Training the machine learning model

Detectron - Object Detection Model

About the model

Detectron is an open-source software system developed by Facebook AI Research (FAIR) primarily for object detection tasks in images. It provides a flexible and modular implementation of various state-of-the-art object detection algorithms.

Detectron2 is built on PyTorch and offers a more modular and extensible platform for object detection, instance segmentation, keypoint detection, and other computer vision tasks.



- As for the type of model, Detectron is a framework that allows you to implement and train various models for object detection. It supports a wide range of model architectures, including Faster R-CNN, Mask R-CNN, RetinaNet, and many others.

- We have used the Base RCNN model whose weight files can be downloaded directly from the github link.

- As hand detection involves small objects within the image and requires precise localization Faster R-CNN performs better due to its two-stage architecture that typically handles smaller objects more accurately

- RCNN models are used for object localization and obtains better accuracy although it is more complex than other models such as YOLO.

Code Walk through

- Install the pip package for detectron2 from the GitHub link.
- Import all the necessary packages such as Visualizer, DefaultTrainer, DatasetCatalog, MetadataCatalog and cv2_imshow.
- The yaml and weight files according to the version as well as model specs.

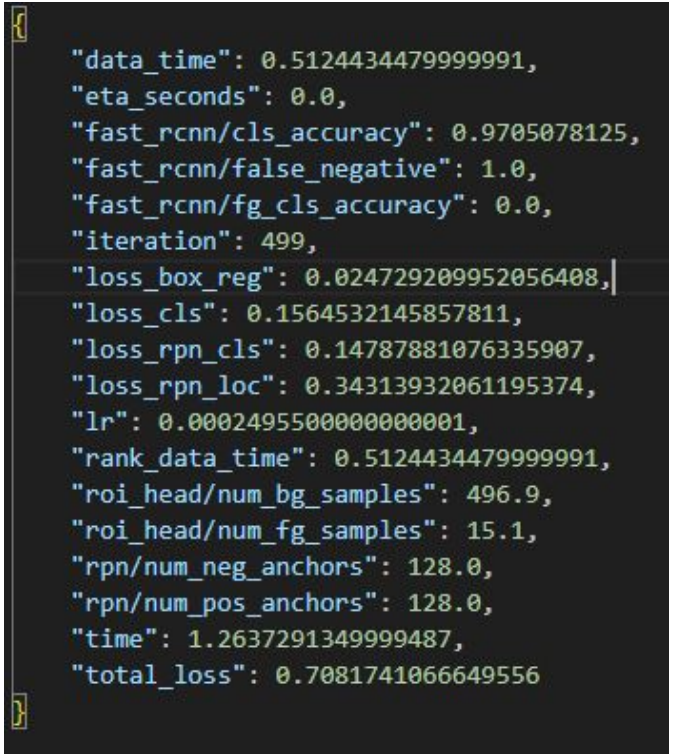
➔ The provided code performs the following tasks:

- ◆ Data Loading and Processing:
  - Loads images and corresponding annotations from a specified directory.
  - Converts annotations into a format suitable for training object detection models in Detectron2.
  - Registers the custom dataset for model training.

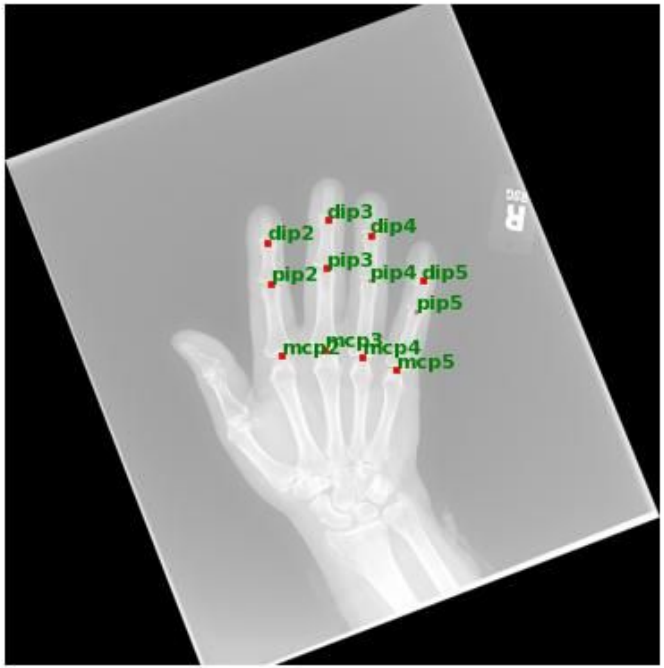
- Visualization for Debugging:
  - Generates visualizations of the loaded dataset for debugging purposes.
  - Renders annotations on sample images, depicting bounding boxes with labels.
- Model Training Preparation:
  - Configures the Detectron2 model by defining its architecture and training settings from a YAML file.
  - Sets up training parameters like dataset paths, model weights, and output directories.
- Model Training:
  - Initializes a DefaultTrainer instance with the specified configuration.
  - Trains the Detectron2 model on the prepared dataset to learn to detect and classify objects based on the provided annotations and images.

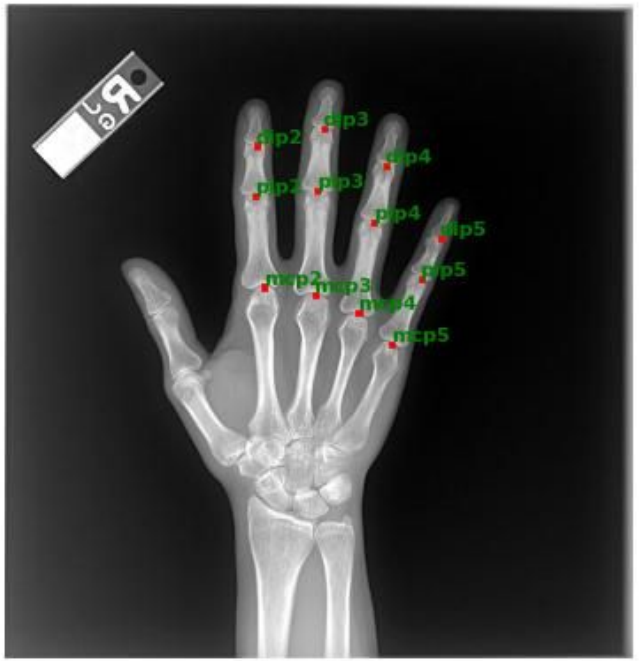
Metrics

- During the training iterations, the code computes and stores various metrics, including accuracy, loss values (such as classification and bounding box regression losses), data loading time, and learning rate, in the metrics.json file.
- This file acts as a log, recording key performance indicators of the model's training progress over successive iterations.
- Our model has been run for 500 epochs with the classification accuracy at the last batch being close to 97%



Visualization of the output





Thank you!!

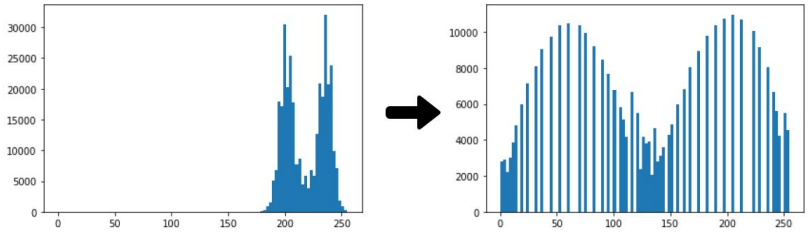
Open for Q&A!

Appendix

CLAHE - Detailed Explanation

Consider an image in which the pixel values are limited to a specified range. A brighter image, for example, will have all pixels restricted to high values.

On the other hand, an excellent image will contain pixels from all sections of the image. So you need to stretch this histogram to either end, which is what Histogram Equalization does (in simple words). This usually increases the image's contrast.



Understanding the model：Detectron with RCNN

- **Initial examination of the entire image:** Just like a detective scanning an entire crime scene, Detectron2 looks at the entire image as a starting point. It doesn’t immediately zoom into specific parts but takes a general look to get an overview.
- **Identification of potentially interesting areas:** Detectron2 then tries to identify areas within the image where objects might be located. It looks for different sections that seem promising and worth investigating further, sort of like marking areas of interest within the crime scene.
- **Zooming in on these areas for detailed analysis:** Once Detectron2 has marked these areas, it zooms in to examine them more closely. It pays attention to the details within these regions to figure out if there are any objects present, like a hand or any other recognizable item.
- **Understanding and recognizing objects:** Within these marked areas, Detectron2 carefully analyzes the visual information, looking for patterns and features that match known objects. It tries to understand if there's something familiar, like a hand gesture or other objects it has learned about during its training.
- **Confirmation and labeling:** After analyzing these areas, Detectron2 makes predictions about what it thinks it has found. It tries to confirm if the objects it recognized are indeed present in those regions. Once confirmed, it labels these objects, indicating what it believes they are.
- **Presenting the findings:** Finally, Detectron2 presents the results by outlining the identified objects with boxes and labels in the image, making it easy to see where they are located.

Model training procedure involves

- Visualizer: Tool to visualize and annotate model predictions on images.
- DefaultTrainer: Class offering a default training loop for object detection models.
- DatasetCatalog: Registry for registering and managing custom datasets.
- MetadataCatalog: Manages metadata associated with datasets, including class information.
- cv2_imshow: Utility for displaying images within Jupyter notebooks using OpenCV (cv2).