fuse | machines

BONE FRACTURE DETECTION

FUSE AI FELLOWSHIP
2023-2024

1. TEAM MEMBERS

a) Joaquin Ayzanoa (Perú)

2. PROBLEM STATEMENT

a) A data set of bone fractures with which we want to train a machine learning model, specifically focusing on algorithms to automatically detect and classify bone fractures in X-ray images for better and fast registration for doctors and patients.

3. OBJETIVES

a) Detect and classify bone fractures in X-ray images for better and fast registration for doctors and patients.

b) Deploy the ML model with an interface so it can predict the type of fracture when someone upload an image.

4. GITHUB REPOSITORY LINK

This is the link of the repository:

https://github.com/JoaquinAyzanoa/PortafolioJA/tree/e96351fa14b03b0788fd2a7d78b996a5b3a9c38d/Fracture%20Bone%20Detection

GitHub

5. RESNET MODEL

The Resnet model was created from scratch using pytorch, no pretrained model or downloaded model was used here.

- checkpoints
- data
- losses
- losses1
- model
- prepare_data
- resnet_checkpoint_131.ckp
- train
- trainer

- data
  - test
    - images
    - labels
  - train
    - images
    - labels
    - data_train.csv
    - labels.cache
  - valid
    - images
    - labels
    - data_valid.csv
    - labels.cache
  - data.yaml
  - README.dataset.txt

Fracture Bone Detection > Resnet Model > model.py > {} nn

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride):
        super(ResBlock, self).__init__()

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=s
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=
        self.bn2 = nn.BatchNorm2d(out_channels)

        # Skip connection
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride
                nn.BatchNorm2d(out_channels)
            )

```

5. RESNET MODEL

The Resnet model was created from scratch using pytorch, no pretrained model or downloaded model was used here.

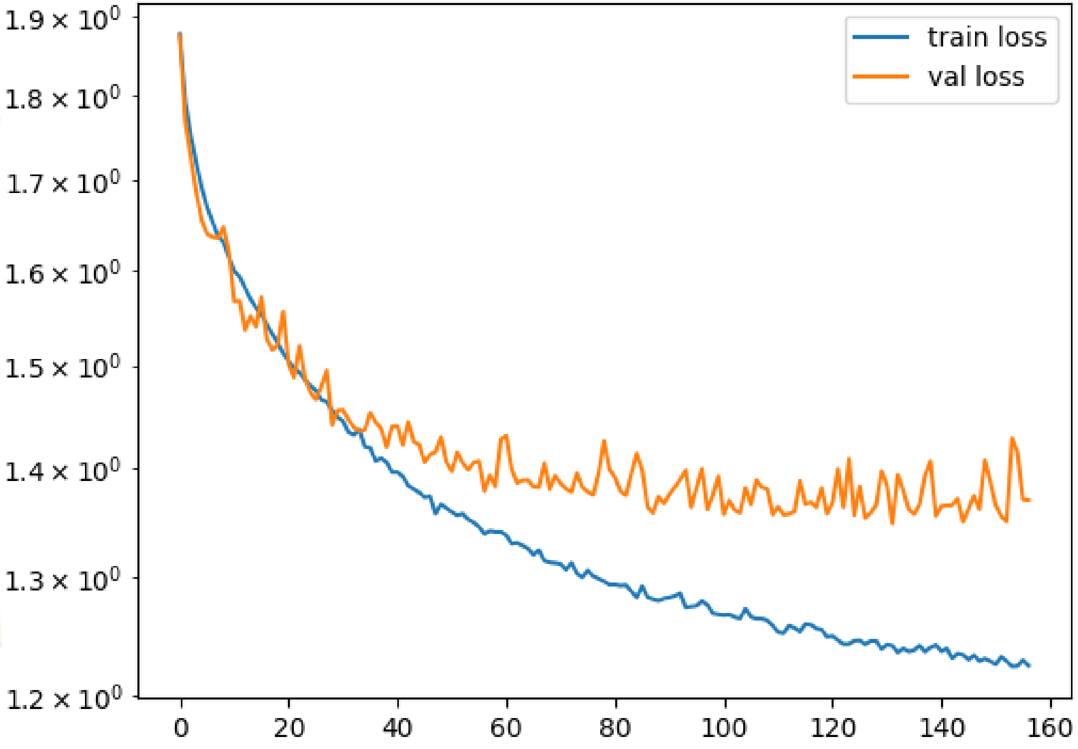
Fracture Bone Detection > Resnet Model > prepare_data.py > {} pd

```python
import glob
import os
import pandas as pd

def prepare_data(root, mode):
    file_dataname = os.path.join(root, mode)
    file_dataname = file_dataname + f'/data_{mode}.csv'
    if os.path.exists(file_dataname):
        df = pd.read_csv(file_dataname, delimiter=';')
        print(f'data_{mode}.csv' +' already exists')
    else:
        files = sorted(glob.glob(os.path.join(root, mode) + '/images/*.jpg'))
        df= pd.DataFrame(columns=['filename','label', 'coords'])
        for file in files:
            text_file = file.replace(".jpg", ".txt").replace("images", "labels"

            with open(text_file, mode="r") as f:
                lines = f.readlines()
                for line in lines:
                    values = [float(value) for value in line.split()]
                    label = int(values[0])
                    coords = values[1:]
                    row_data = [file]
                    row_data.append(label)
```

Fracture Bone Detection > Resnet Model > train.py > ...

```python

# set up data loading for the training and validation set each using t.utils.data
train_dataset = ChallengeDataset(train_data, mode='train')
val_dataset = ChallengeDataset(val_data, mode='val')
train_dataloader = t.utils.data.DataLoader(train_dataset, batch_size=48, shuffle=
val_dataloader = t.utils.data.DataLoader(val_dataset, batch_size=48, shuffle=Fals

# create an instance of our ResNet model
resnet_model = model.ResNet()
resnet_model.to(device)
# set up a suitable loss criterion (you can find a pre-implemented loss functions
# set up the optimizer (see t.optim)
# create an object of type Trainer and set its early stopping criterion
criterion = t.nn.CrossEntropyLoss()  # Assuming it's a multi class classificatio
optimizer = t.optim.Adam(resnet_model.parameters(), lr=0.00001)
trainer = Trainer(model=resnet_model, crit=criterion, optim=optimizer,
                  train_dl=train_dataloader, val_test_dl=val_dataloader,
                  cuda=True, early_stopping_patience=25)

#trainer.restore_checkpoint(79)
res = trainer.fit(epochs=200)

```

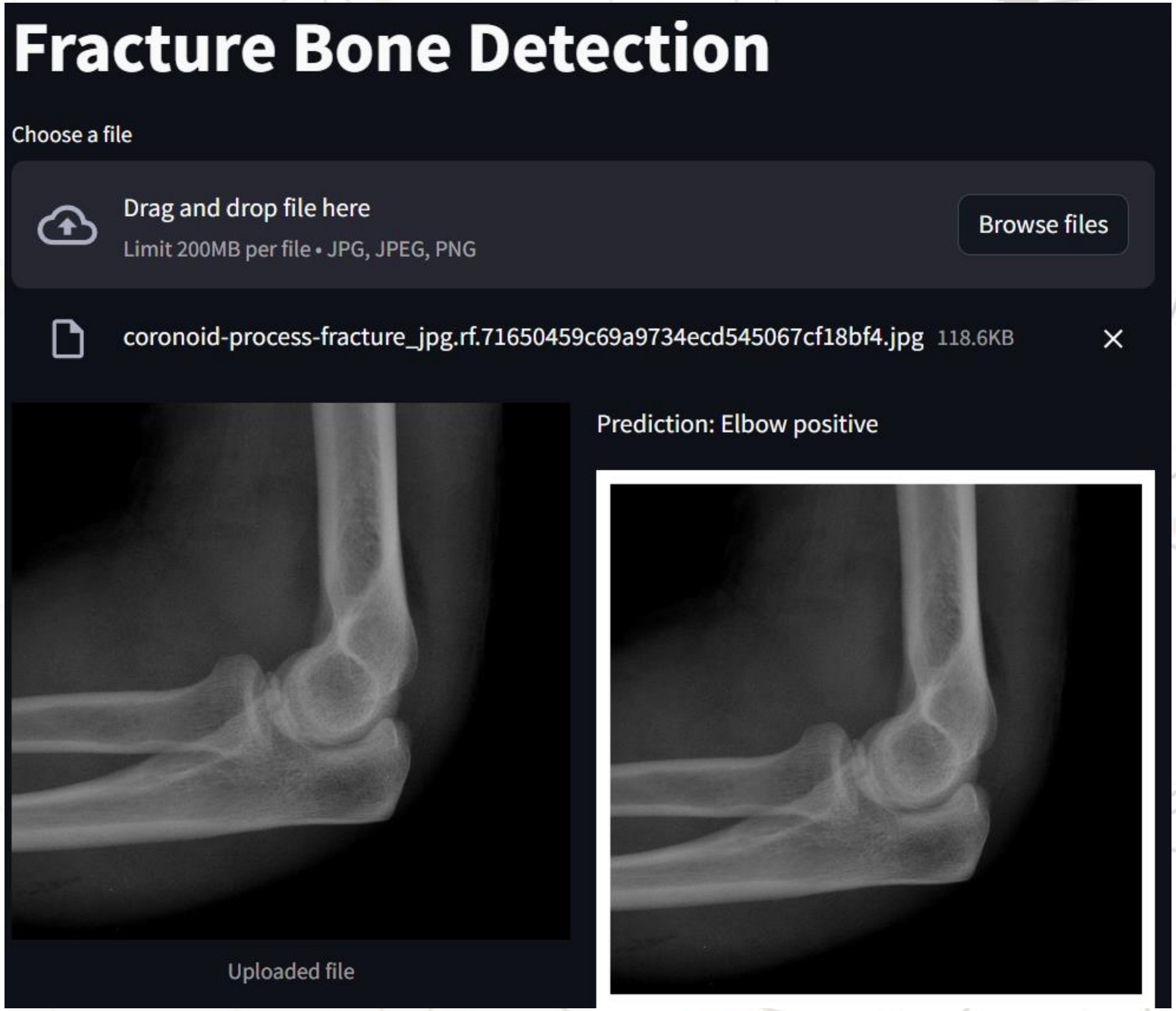
5.1 Results



TERMINAL

```
Epoch 130/200: Train Loss: 1.2447, Val Loss: 1.3684, F1 Score: 0.7963
Epoch 131/200: Train Loss: 1.2428, Val Loss: 1.3483, F1 Score: 0.8056
Epoch 132/200: Train Loss: 1.2381, Val Loss: 1.3751, F1 Score: 0.7709
Epoch 133/200: Train Loss: 1.2427, Val Loss: 1.3700, F1 Score: 0.7849
Epoch 134/200: Train Loss: 1.2407, Val Loss: 1.3798, F1 Score: 0.7667
Epoch 135/200: Train Loss: 1.2460, Val Loss: 1.4967, F1 Score: 0.6689
Epoch 136/200: Train Loss: 1.2396, Val Loss: 1.3573, F1 Score: 0.7905
Epoch 137/200: Train Loss: 1.2312, Val Loss: 1.3677, F1 Score: 0.7814
Epoch 138/200: Train Loss: 1.2327, Val Loss: 1.3985, F1 Score: 0.7460
Epoch 139/200: Train Loss: 1.2372, Val Loss: 1.3769, F1 Score: 0.7764
Epoch 140/200: Train Loss: 1.2328, Val Loss: 1.3675, F1 Score: 0.7830
Epoch 141/200: Train Loss: 1.2302, Val Loss: 1.4016, F1 Score: 0.7574
Epoch 142/200: Train Loss: 1.2360, Val Loss: 1.3950, F1 Score: 0.7550
Epoch 143/200: Train Loss: 1.2355, Val Loss: 1.3733, F1 Score: 0.7756
Epoch 144/200: Train Loss: 1.2303, Val Loss: 1.4430, F1 Score: 0.7138
Epoch 145/200: Train Loss: 1.2395, Val Loss: 1.3915, F1 Score: 0.7705
Epoch 146/200: Train Loss: 1.2372, Val Loss: 1.3624, F1 Score: 0.7839
Epoch 147/200: Train Loss: 1.2335, Val Loss: 1.3638, F1 Score: 0.7819
Epoch 148/200: Train Loss: 1.2293, Val Loss: 1.4029, F1 Score: 0.7375
Epoch 149/200: Train Loss: 1.2357, Val Loss: 1.4042, F1 Score: 0.7306
Epoch 150/200: Train Loss: 1.2290, Val Loss: 1.3804, F1 Score: 0.7608
Epoch 151/200: Train Loss: 1.2305, Val Loss: 1.4086, F1 Score: 0.7491
Epoch 152/200: Train Loss: 1.2302, Val Loss: 1.3770, F1 Score: 0.7741
Epoch 153/200: Train Loss: 1.2270, Val Loss: 1.3767, F1 Score: 0.7921
Epoch 154/200: Train Loss: 1.2288, Val Loss: 1.3628, F1 Score: 0.7973
Epoch 155/200: Train Loss: 1.2262, Val Loss: 1.3854, F1 Score: 0.7686
Epoch 156/200: Train Loss: 1.2275, Val Loss: 1.3872, F1 Score: 0.7690
Early stopping after 156 epochs without improvement.
```

5.1 Results

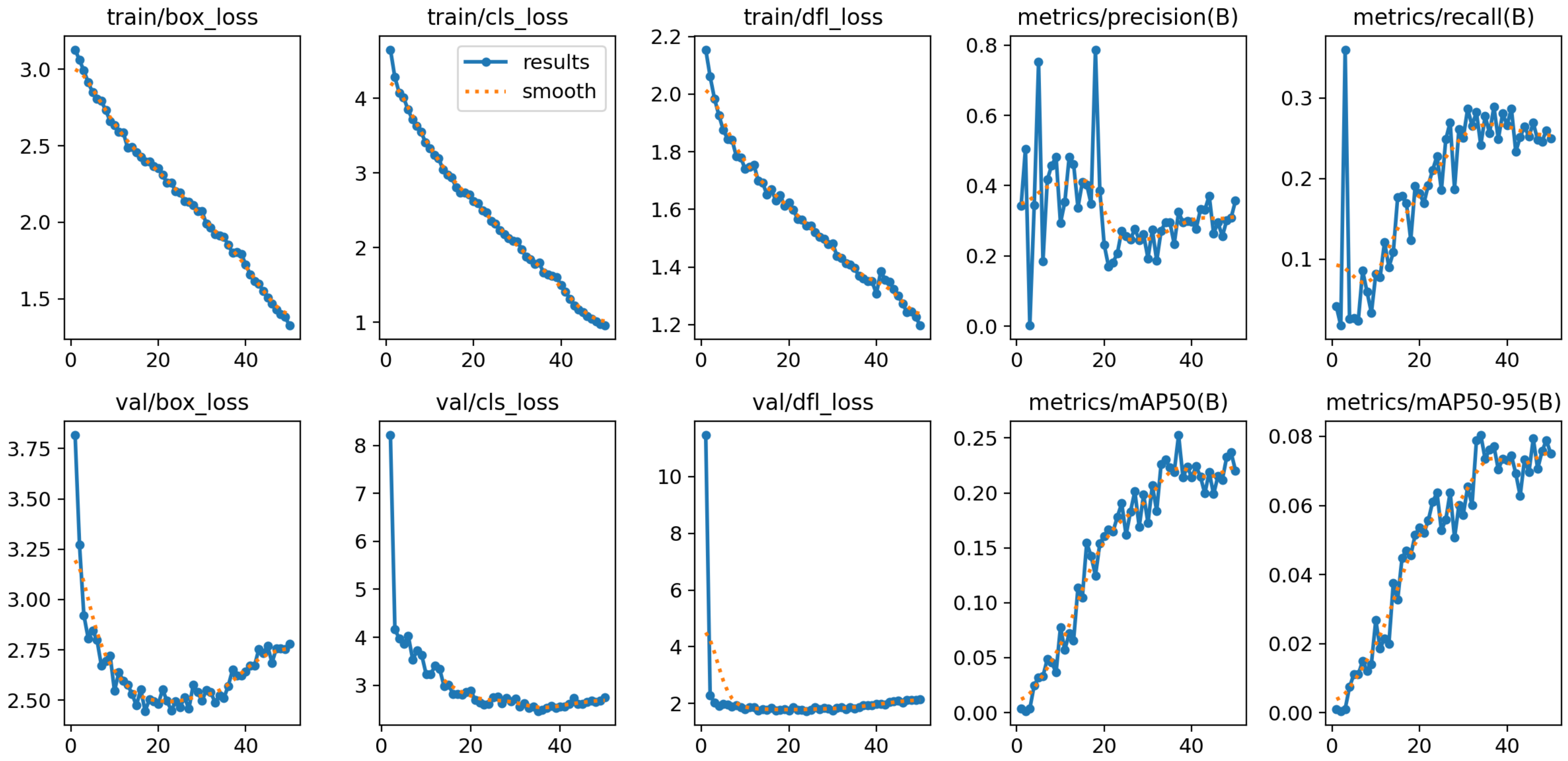

Fracture Bone Detection
Choose a file
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files
coronoid-process-fracture_jpg.rf.71650459c69a9734ecd545067cf18bf4.jpg 118.6KB
Prediction: Elbow positive
Uploaded file

6.YOLO-V8 FOR OBJECT DETECTION

The YoloV8 model is a pretrained model downloaded from the library "Ultralytics".
The YoloV8 was finetuned to bone fracture detection task.

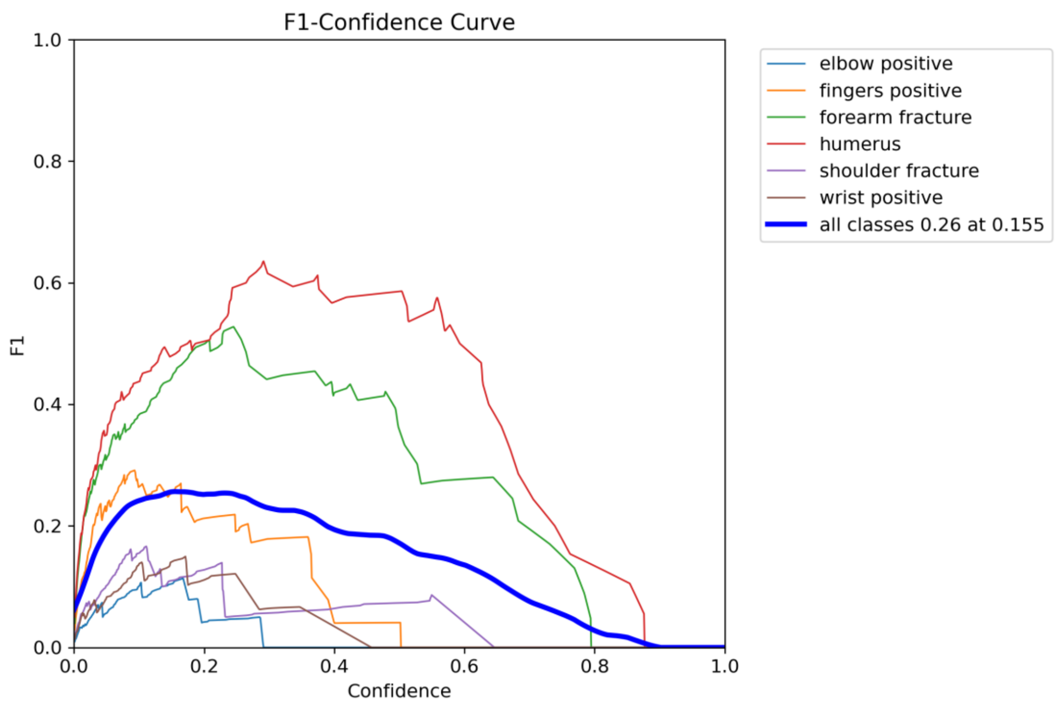
```
# Run inference on an image with YOLO
!yolo detect train data='./data.yaml' model=yolov8x.pt epochs=50 imgsz=320 device=0
                                                                                    Python

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      49/50      5.51G      1.381      0.975      1.226         11        320: 100% 227/227 [01:2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 11
                   all        348        204      0.309       0.26      0.237     0.0789

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      50/50       5.5G      1.325     0.9576      1.197          7        320: 100% 227/227 [01:2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 11
                   all        348        204      0.357       0.25       0.22     0.0749

50 epochs completed in 1.312 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 136.7MB
Optimizer stripped from runs/detect/train/weights/best.pt, 136.7MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.1.27 🚀 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 68130309 parameters, 0 gradients, 257.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 11
                   all        348        204      0.296      0.242       0.23     0.0802
        elbow positive        348         29     0.0521     0.0345     0.0265     0.0094
      fingers positive        348         48      0.389      0.146      0.204     0.0617
      forearm fracture        348         43      0.538      0.465      0.438      0.175
               humerus        348         36      0.442      0.583      0.593      0.204
     shoulder fracture        348         20      0.111       0.15     0.0631     0.0137
        wrist positive        348         28      0.244     0.0714     0.0568     0.0166
Speed: 0.1ms preprocess, 5.7ms inference, 0.0ms loss, 1.3ms postprocess per image
```

6.1 Results

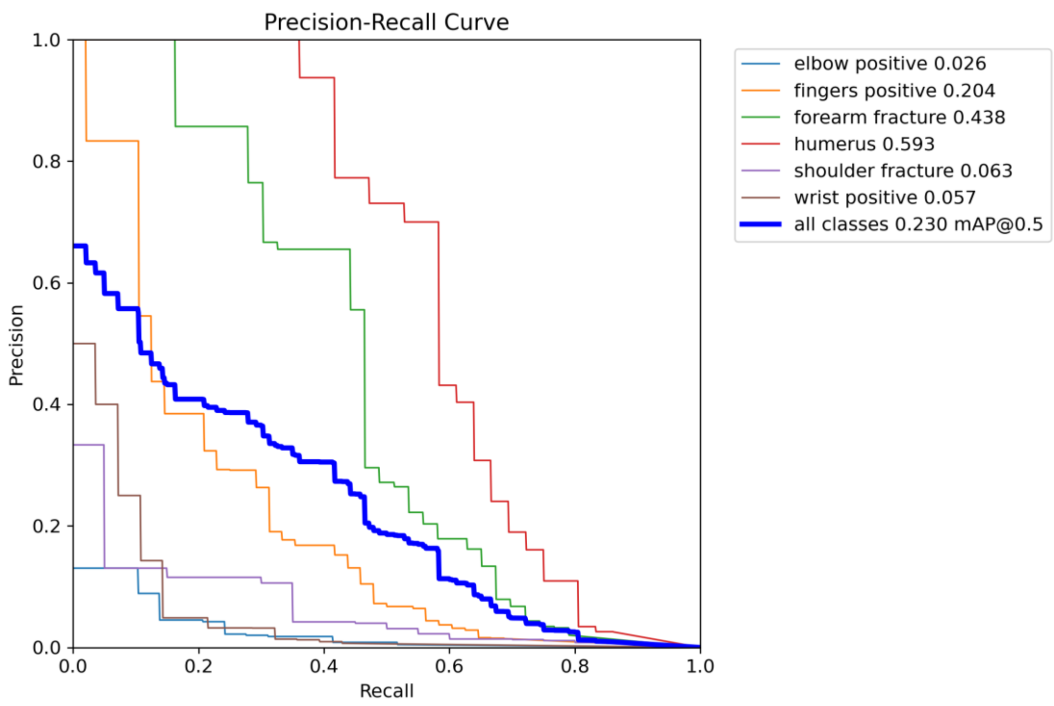


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)

6.1 Results

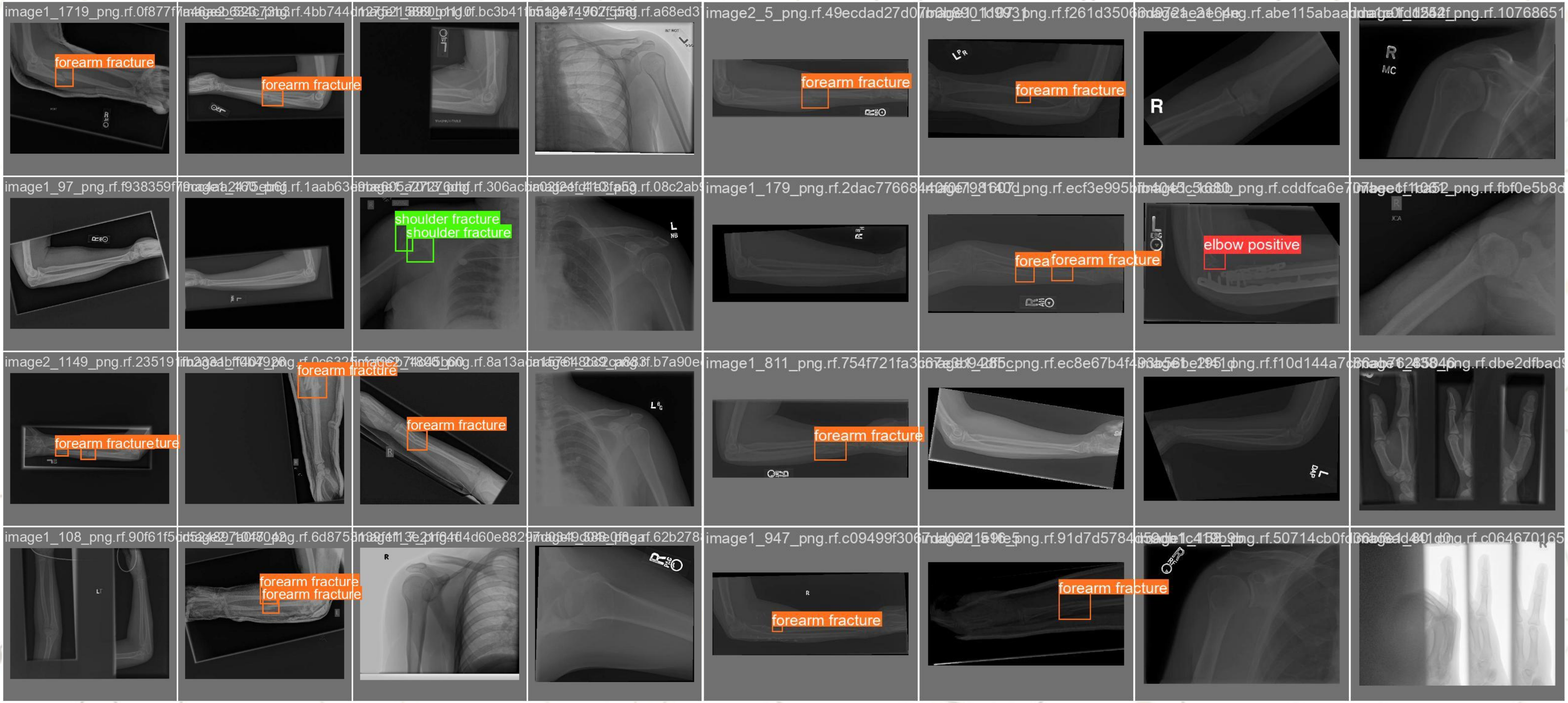


F1-Confidence Curve
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
elbow positive
fingers positive
forearm fracture
humerus
shoulder fracture
wrist positive
all classes 0.26 at 0.155

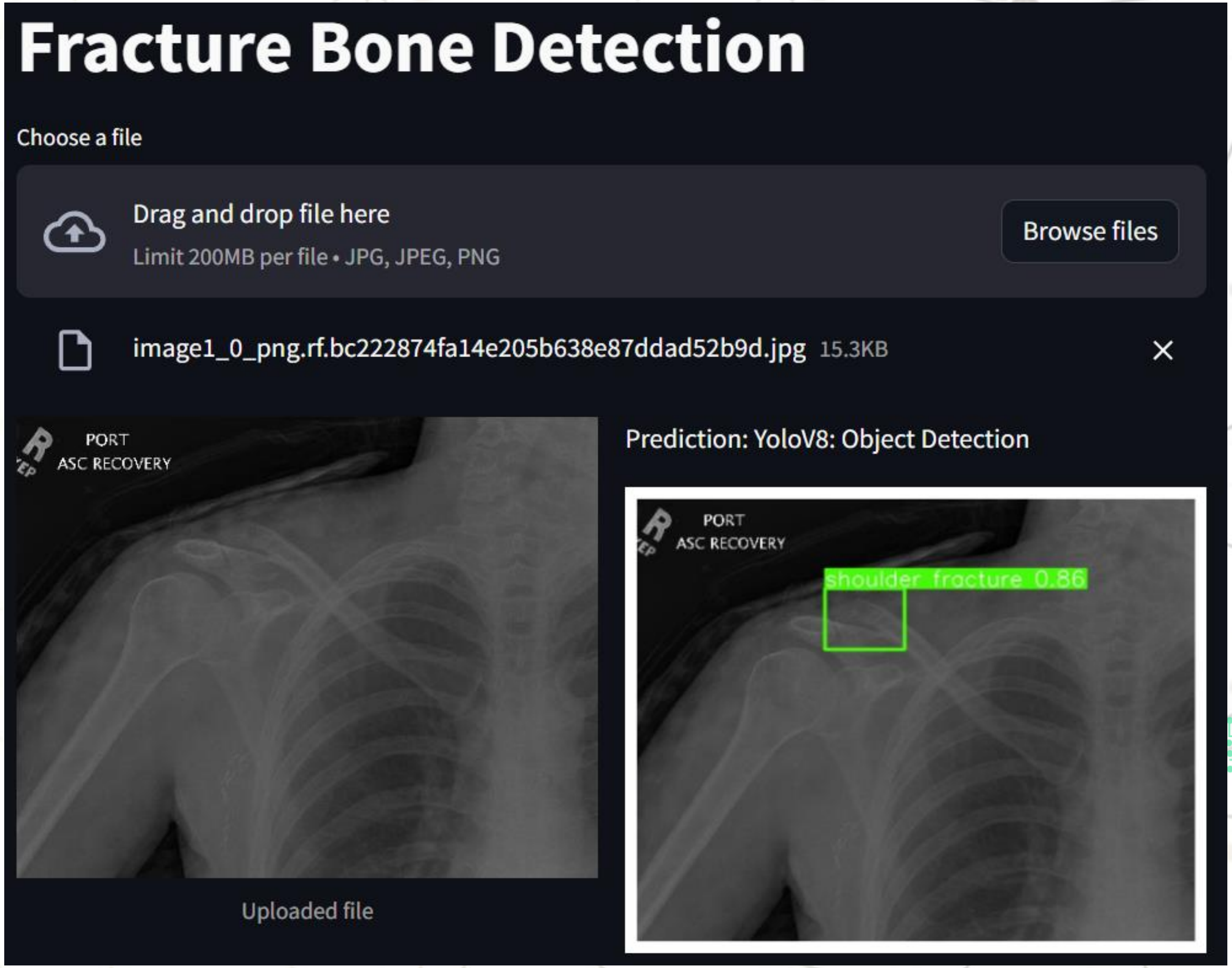


Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
elbow positive 0.026
fingers positive 0.204
forearm fracture 0.438
humerus 0.593
shoulder fracture 0.063
wrist positive 0.057
all classes 0.230 mAP@0.5

6.1 Results



6.2 Results with Streamlit and FastAPI



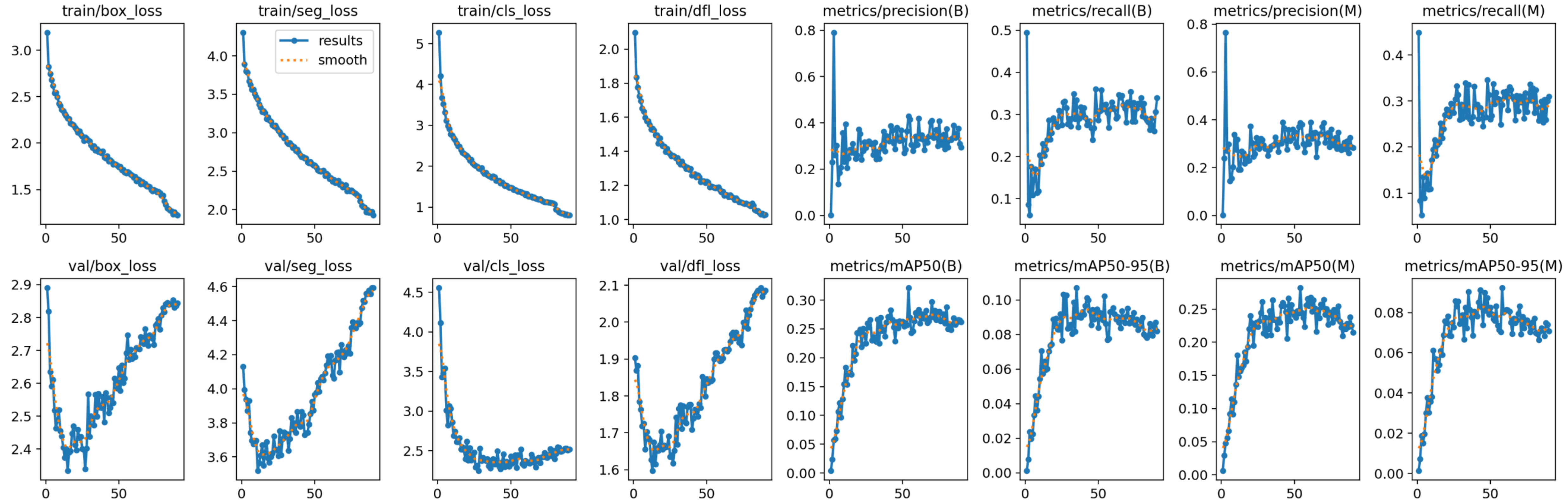
7.YOLO-V8 WITH INSTANCE SEGMENTATION

The YoloV8-SEG model is a pretrained model downloaded from the library “Ultralytics”.
The YoloV8 was finetuned to bone fracture instance segmentation.

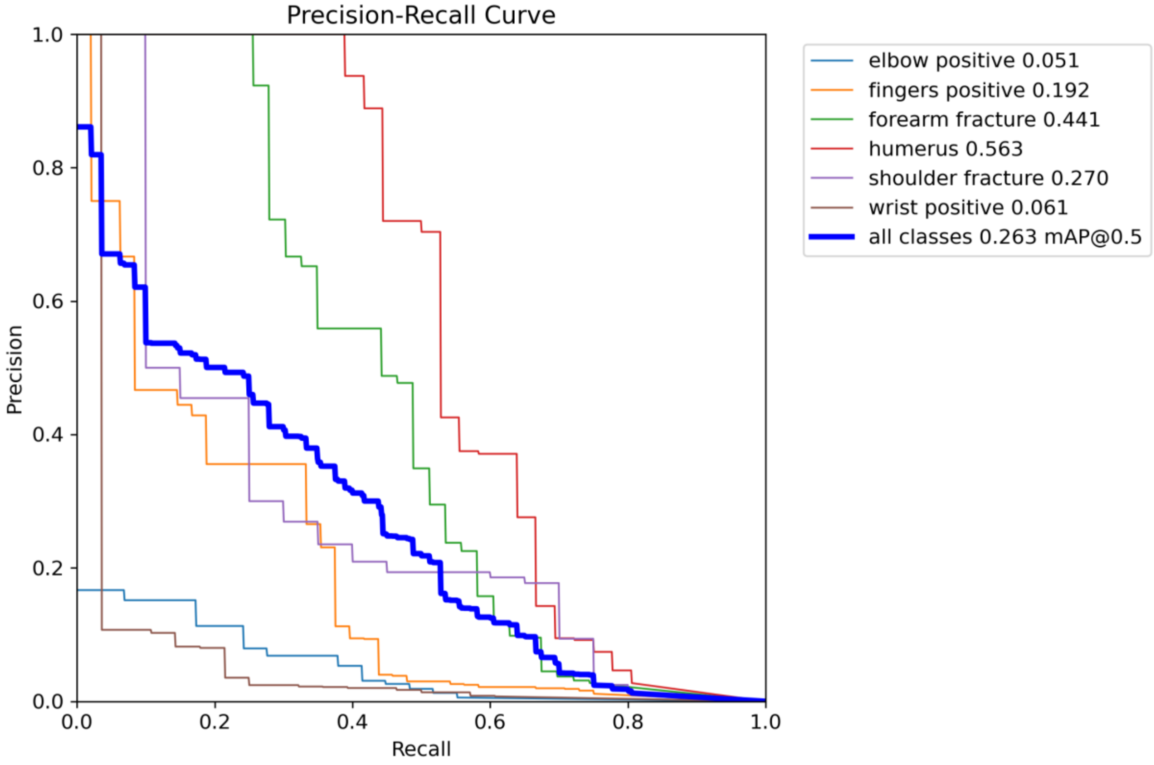
```
# Run inference on an image with YOLO
!yolo detect train data='./data.yaml' model=yolov8n-seg.pt epochs=90 imgsz=320 device=0
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.1.0/yolov8n-seg.pt to 'yolov8n-seg.pt'...
100% 6.73M/6.73M [00:00<00:00, 24.7MB/s]
WARNING ⚠ conflicting 'task=detect' passed with 'task=segment' model. Ignoring 'task=detect' and updating to 'task=segment' to match model
Ultralytics YOLOv8.1.29 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=segment, mode=train, model=yolov8n-seg.pt, data=./data.yaml, epochs=90, time=None, patience=100, batch=16, imgsz=320, s
2024-03-20 22:10:23.819812: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to
2024-03-20 22:10:23.819872: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to r
2024-03-20 22:10:23.832882: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting t
Overriding model.yaml nc=80 with nc=7

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
```

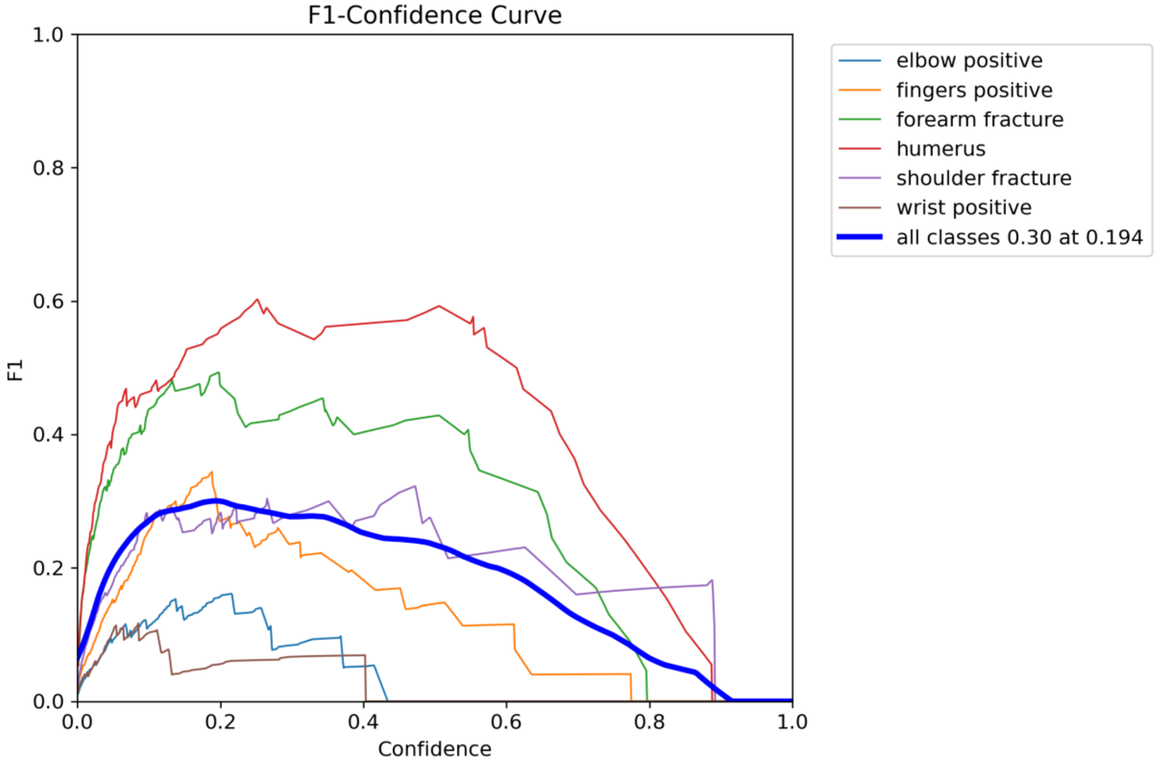
7.1 Results


train/box_loss
train/seg_loss
results
smooth
train/cls_loss
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
metrics/precision(M)
metrics/recall(M)
val/box_loss
val/seg_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)
metrics/mAP50(M)
metrics/mAP50-95(M)

7.1 Results

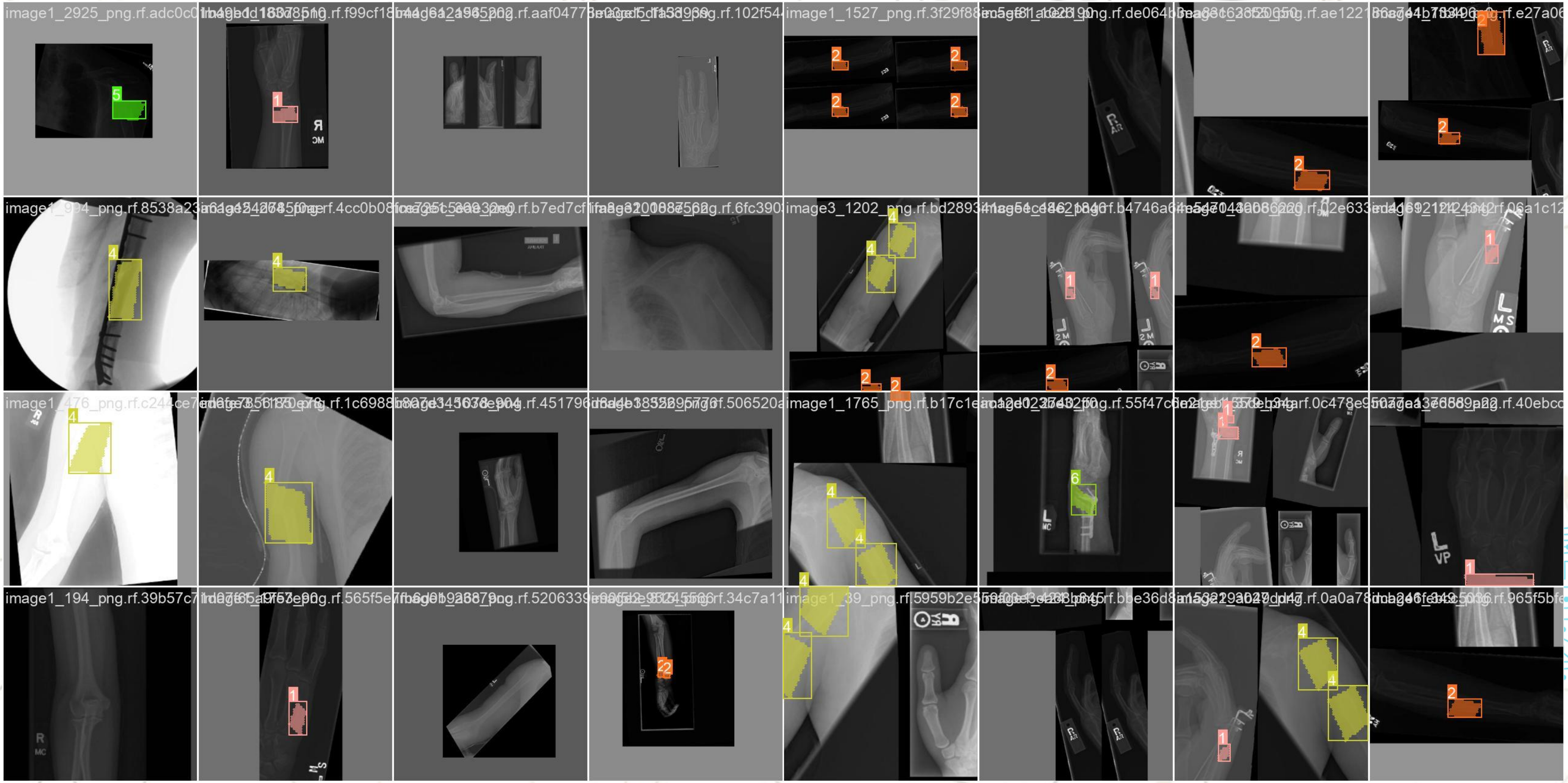


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
elbow positive 0.051
fingers positive 0.192
forearm fracture 0.441
humerus 0.563
shoulder fracture 0.270
wrist positive 0.061
all classes 0.263 mAP@0.5

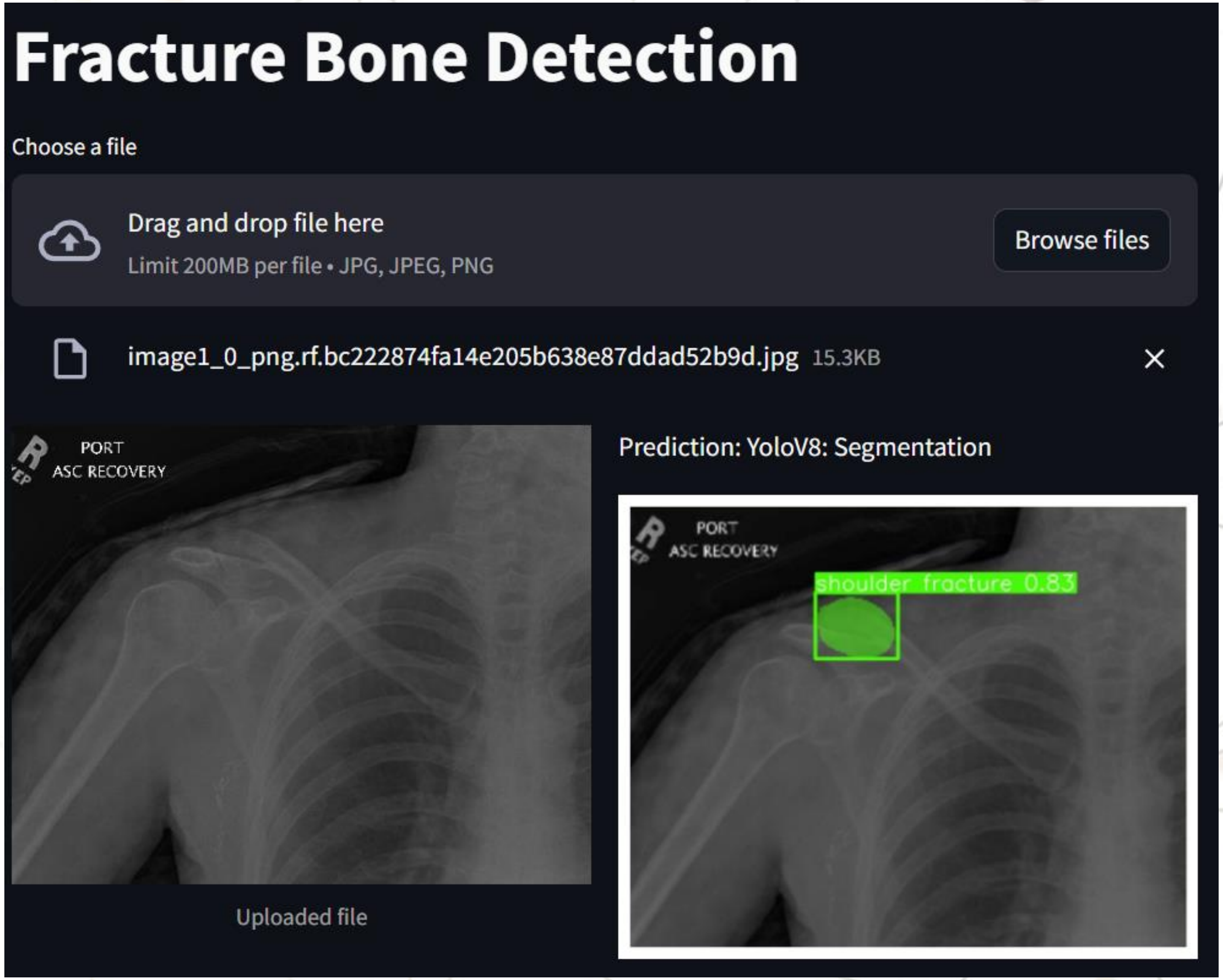


F1-Confidence Curve
F1
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
elbow positive
fingers positive
forearm fracture
humerus
shoulder fracture
wrist positive
all classes 0.30 at 0.194

7.1 Results

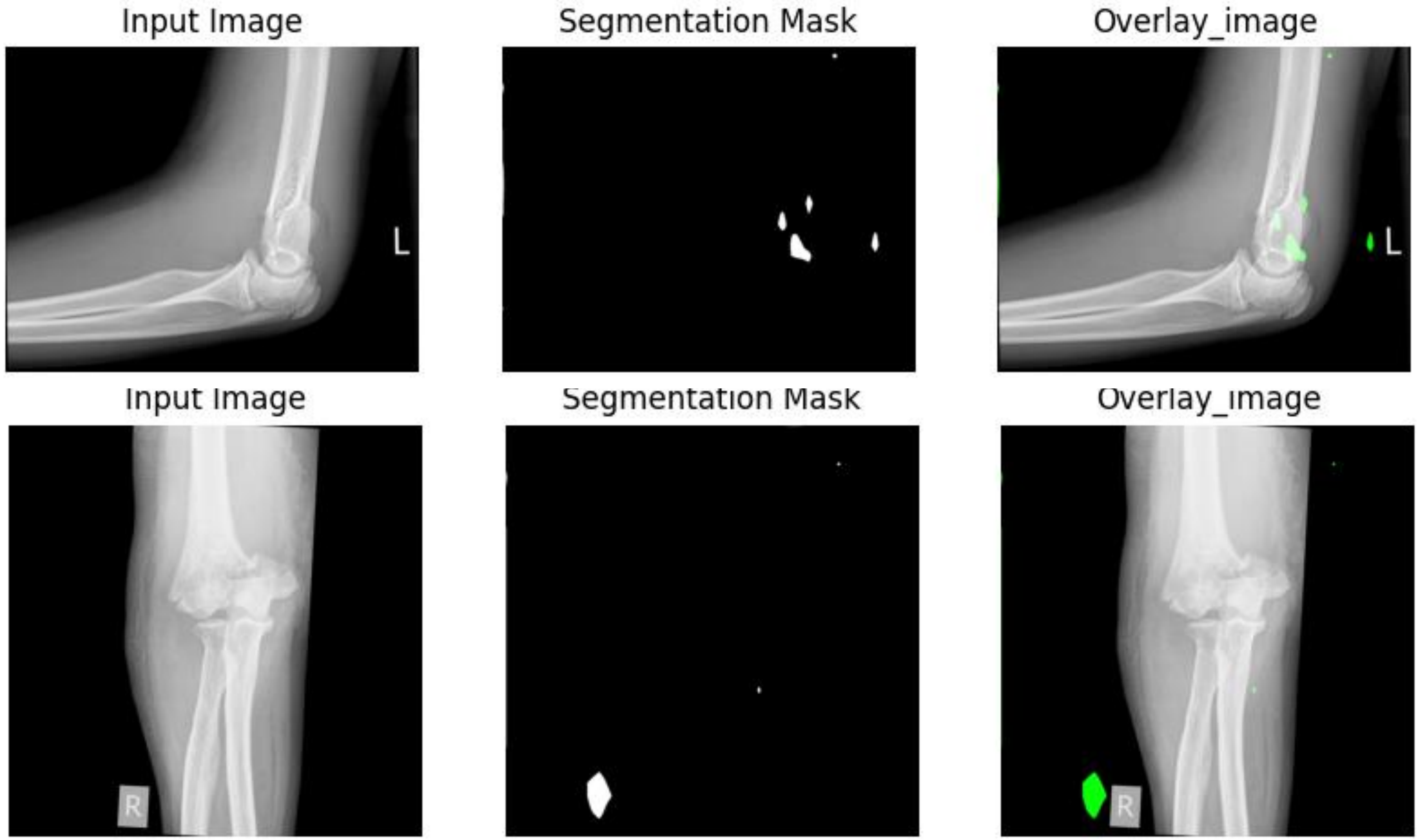


7.3 Results with Streamlit and FastAPI



8. Conclusions

1. Resnet strugle to make instance segmentation as is shown bellow:



8. Conclusions

2. Using Resnet with FastAPI has a lower response time in the server.

3. Using Yolo model has a relative higher response time in the server using streamlit and FastAPI.

4. Segmentation using YOLO could be improve by fine-tuning the model for more epochs or with more data for the training.

5. The deployment of the models on the server was successfully completed.