**Innovative Monitoring System for TeleICU Patients Using Video Processing and Deep Learning**

*Submitted by*

*TEAM ELECTRONAUTS*

PALAK SONI

VARTIKA SHARMA

VATSALA MISRA

****

**VIT Bhopal University**

**Bhopal**

**Madhya Pradesh**

**July, 2024**

****

**Bonafide Certificate**

Certified that this project report titled **“Innovative Monitoring System for TeleICU Patients Using Video Processing and Deep Learning”** is the bonafide work of **“**Palak Soni, Vartika Sharma, Vatsala Misra**”** who carried out the project work under my supervision.

**Supervisor**

**Sl. No. Topic**

1. Problem Statement
2. Solution
   1. Model-1
   2. Model-2
   3. Model-3
3. Technologies Used
4. Conclusion

**PROBLEM STATEMENT**

ICUs are critical care units in hospitals where patients with severe and life-threatening illnesses or injuries are closely monitored. The concept of a TeleICU involves using technology to extend the reach of ICU care to patients who might not be in a traditional ICU setting, such as those in remote areas or smaller hospitals without specialized ICU staff.



**SOLUTION**

By leveraging video processing and deep learning, the innovative monitoring system for TeleICU patients can significantly enhance patient care by providing continuous, real-time monitoring and alerting medical staff to potential emergencies, thereby improving patient outcomes and optimizing healthcare resources.

*MODEL -1: Person Identification and Classification*

*PROCESS EXPLAINATION:*

1. Installation of YOLO and Roboflow:

- You start by installing the `ultralytics` package, which includes the YOLO (You Only Look Once) model, a popular real-time object detection system.

- Next, you install `roboflow`, a tool to manage datasets for machine learning tasks.

2. Environment Setup:

- You import the necessary libraries and clear any previous outputs to keep your workspace clean.

- Basic checks are run to ensure the environment is properly set up.

3. Accessing and Downloading the Dataset:

- Using the Roboflow API, you access your specific project and version to download the dataset required for training the YOLO model.

- The dataset is downloaded in a format compatible with YOLO (e.g., YOLOv8).

4. Training the YOLO Model:

- The YOLO model is trained using the downloaded dataset.

- Training involves specifying the task (object detection), the mode (train), the model architecture (`yolov8n.pt` for YOLOv8 Nano), the dataset location, the number of epochs (iterations over the dataset), and the image size.

- The training command initializes the model, loads the dataset, and begins training, adjusting the model's weights to learn from the data.

5. Inference with the Trained Model:

- After training, the model is used to make predictions (inference) on new images.

- You specify the path to the trained model weights and the directory containing images for testing.

- The predictions are saved, and you can visualize the results, where the model highlights the detected objects in the images.

*INPUT:*

# Pip install method (recommended)

!pip install ultralytics==8.0.196

from IPython import display

display.clear\_output()

import ultralytics

ultralytics.checks()

!pip install ultralytics==8.0.196

from IPython import display

display.clear\_output()

import ultralytics

ultralytics.checks()

from ultralytics import YOLO

from IPython.display import display, Image

import cv2

import os

!pip install roboflow

from roboflow import Roboflow

rf = Roboflow(api\_key="AdoSMIM9QeiJJzZaQZkk")

project = rf.workspace("vartika-wpiaz").project("teleicu-cumfn")

version = project.version(1)

dataset = version.download("yolov8")

!yolo task=detect mode=train model=yolov8n.pt data={dataset.location}/data.yaml epochs=10

imgsz=640

model=YOLO(yolov8n.pt)

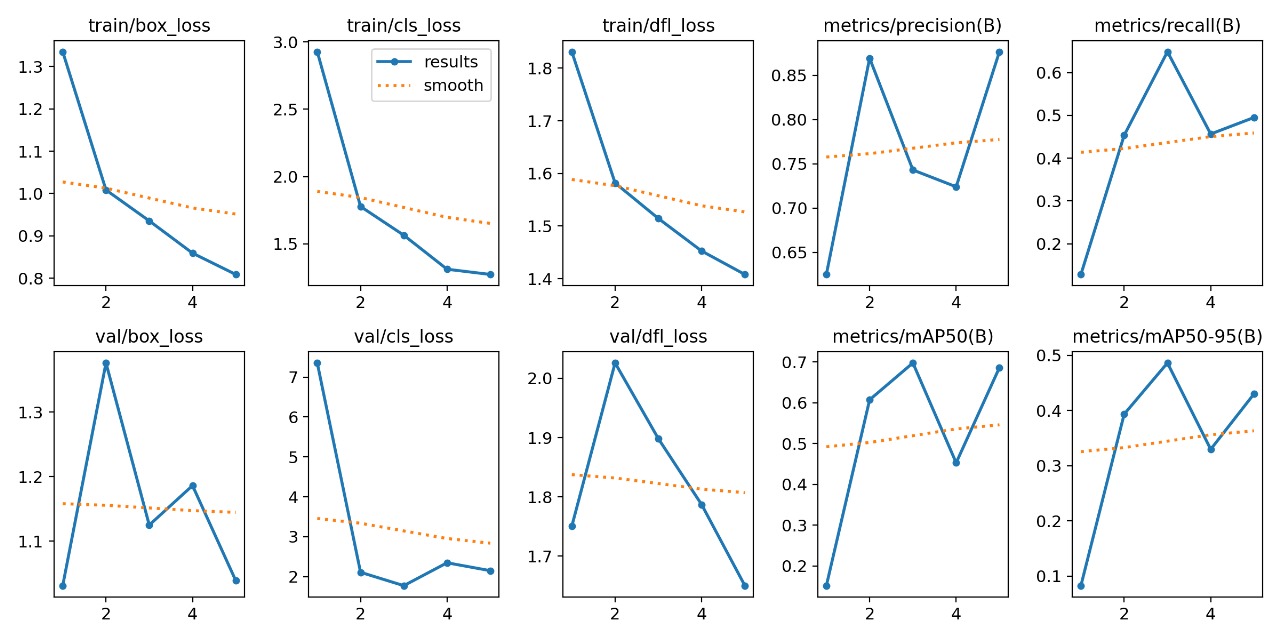
results=model.train(data=os.path.join(dataset.location, "data.yaml")), epochs=100

infer=YOLO("/content/runs/detect/train2/weights/last.pt")

infer.predict("/content/TeleICU-1/test/images", save= True)

OUTPUT:

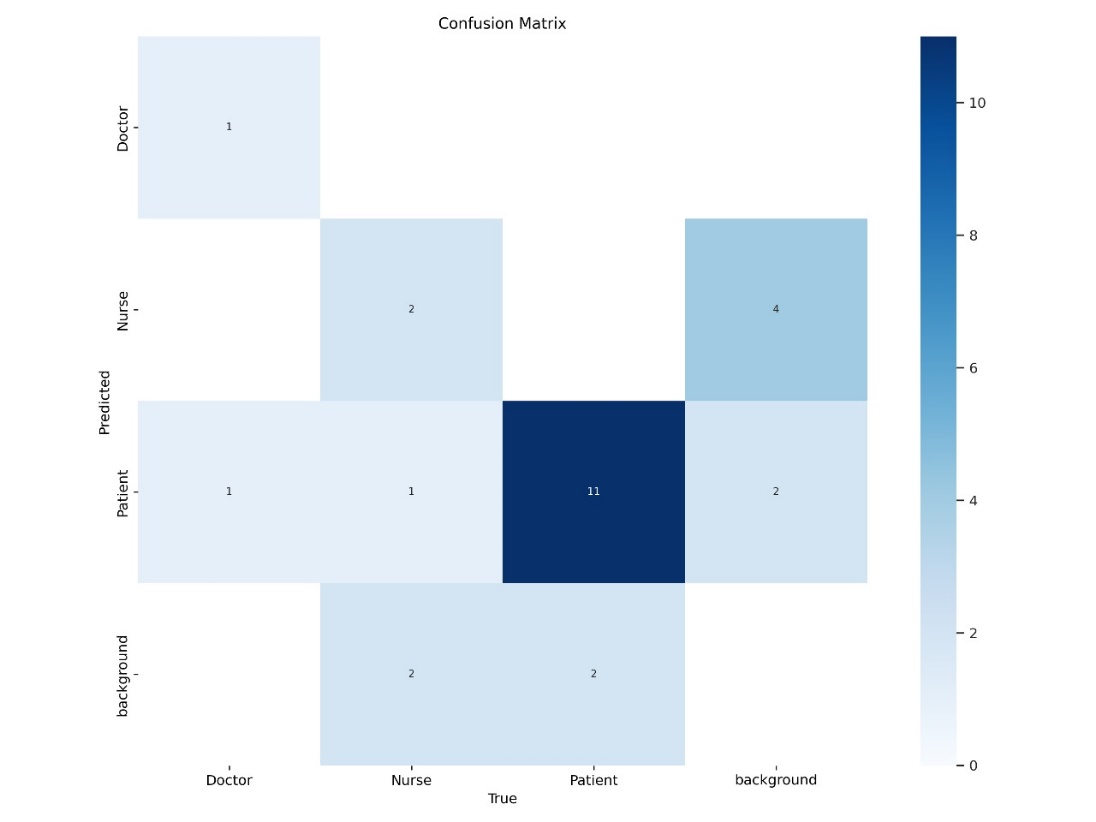
Figure: Training and Validation metrics

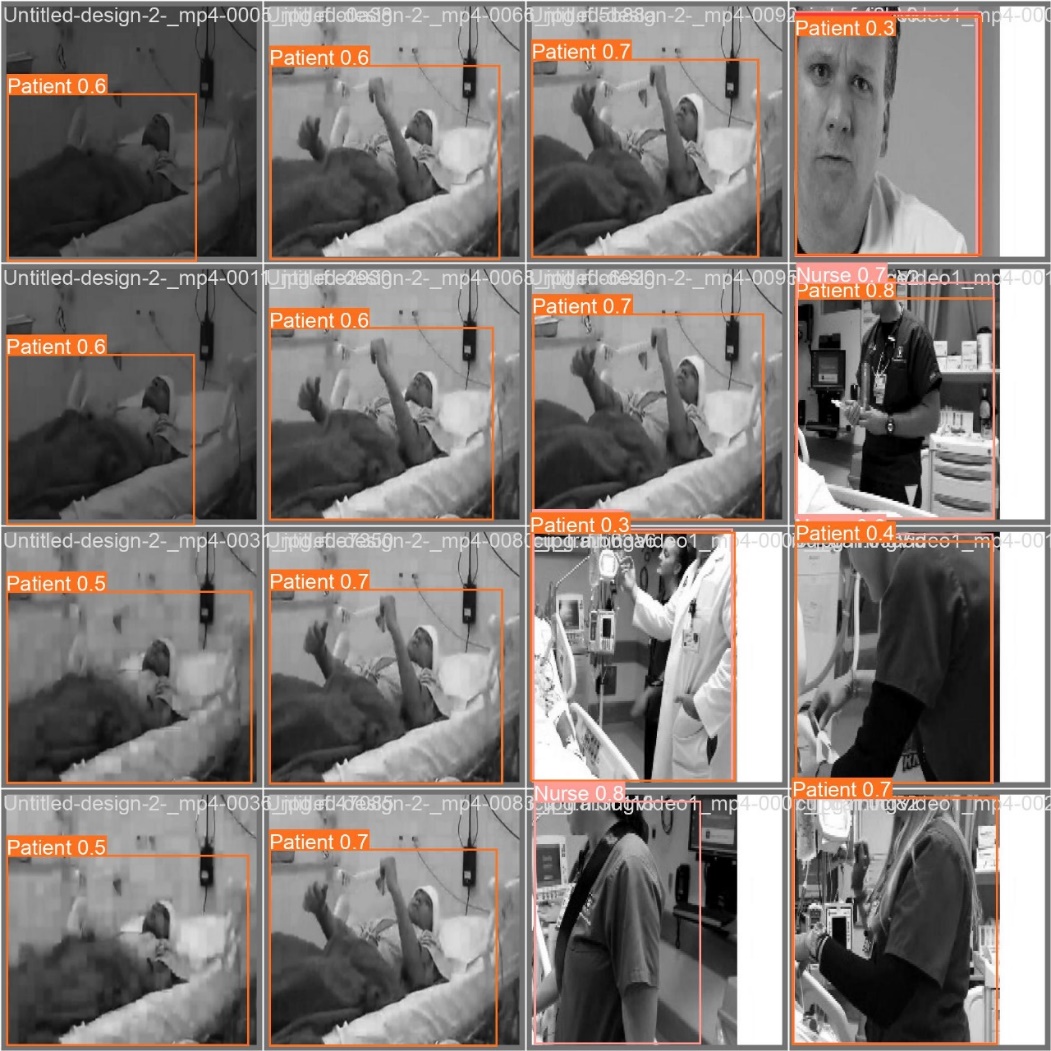


Where,

x-axis: epochs or iterations during training

y-axis: different loss values (unitless) or performance metrics (ranging from 0 to 1)

Figure: Confusion Matrix

OUTPUT:

*MODEL-2: Movement Identification*

*PROCESS EXPLAINATION:*

1. Initialization:

* Import necessary libraries (cv2 for OpenCV and Video for displaying video in Jupyter).
* Define the output video file (output\_video.avi).

1. Video Capture Setup:

* Open the video capture from the file 'FNSEIZURE.mp4'.
* Retrieve the frame dimensions (width and height).
* Define the codec and create a VideoWriter object to write the processed frames to the output file.

1. Frame Processing:

* Read the first two frames of the video.
* Loop while the video capture is open:
  + Calculate the absolute difference between consecutive frames.
  + Convert the difference to grayscale and apply Gaussian blur.
  + Apply binary thresholding to highlight significant changes.
  + Dilate the thresholded image to fill gaps.
  + Find contours in the dilated image.

1. Motion Detection:

* For each contour, calculate the bounding rectangle.
* If the contour area is above a certain threshold, draw a rectangle and add a 'MOVEMENT' label on the frame.
* Write the processed frame to the output video file.
* Update the frames for the next iteration.

1. Cleanup and Display:

* Release the video capture and writer objects.
* Destroy all OpenCV windows.
* Display the output video in the Jupyter Notebook.

*INPUT:*

import cv2

from IPython.display import Video, display

# Define the output video file

output\_file = 'output\_video.avi'

# Open the video capture

cap = cv2.VideoCapture('FNSEIZURE.mp4')

# Get the frame width and height

frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

# Define the codec and create VideoWriter object

out = cv2.VideoWriter(output\_file, cv2.VideoWriter\_fourcc(\*'XVID'), 20.0, (frame\_width, frame\_height))

ret, frame1 = cap.read()

ret, frame2 = cap.read()

while cap.isOpened():

    diff = cv2.absdiff(frame1, frame2)

    gray = cv2.cvtColor(diff, cv2.COLOR\_BGR2GRAY)

    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    \_, thresh = cv2.threshold(blur, 20, 255, cv2.THRESH\_BINARY)

    dilated = cv2.dilate(thresh, None, iterations=3)

    contours, \_ = cv2.findContours(dilated, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

    for contour in contours:

        (x, y, w, h) = cv2.boundingRect(contour)

        if cv2.contourArea(contour) < 1000:

            continue

        cv2.rectangle(frame1, (x, y), (x+w, y+h), (0, 255, 0), 2)

        cv2.putText(frame1, 'MOVEMENT', (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 3)

    # Write the frame to the output video

    out.write(frame1)

    frame1 = frame2

    ret, frame2 = cap.read()

    if not ret:

        break

# Release everything

cap.release()

out.release()

cv2.destroyAllWindows()

# Display the video in Jupyter Notebook

display(Video(output\_file))

OUTPUT: 

*MODEL-3: Movement Classification*

*PROCESS EXPLAINATION:*

* 1. Initialization:
* The script starts by importing necessary libraries: cv2 for computer vision tasks and numpy for numerical operations.
* The video file "FNSEIZURE.mp4" is loaded using cv2.VideoCapture.
  1. Error Handling:
* It checks if the video file is successfully opened. If not, it prints an error message and exits the script.
  1. Output Video Writer Setup:
* A video writer is set up to save the processed frames into "TESToutput\_labeled1.avi" using the XVID codec with a frame rate of 20.0.
  1. Background Subtraction and Motion Detection:
* A background subtractor (MOG2) is created to detect moving objects in the video frames.
* A minimum contour area is defined to filter out small, irrelevant movements.
* A kernel for morphological operations is defined to help clean up the motion masks.
  1. Processing Frames:
* The script reads frames from the video one by one.
* For each frame, background subtraction is applied to create a foreground mask.
* The foreground mask is processed using thresholding, median blur, and morphological operations (closing and opening) to reduce noise and enhance motion detection.
  1. Contour Detection and Classification:
* Contours are found in the foreground mask.
* For each significant contour (larger than the defined minimum area), the bounding box is calculated.
* Based on the dimensions of the bounding box, the motion is classified as "Walking," "Standing," or "Seizure." The classification criteria are:
  + "Walking" if height > 200 and width > 100.
  + "Standing" if height > 50 and width < 50.
  + "Seizure" for all other cases.
  1. Annotation and Display:
* Bounding boxes and labels are drawn on the original frame using different colors for each label.
* The annotated frame is written to the output video file.
* The frame is also displayed in a window named 'Frame'.
  1. Cleanup:
* The script waits for the 'q' key to be pressed to exit the loop.
* All resources, including the video capture and writer objects, are released, and any OpenCV windows are closed.

*INPUT:*

import cv2

import numpy as np

# Load video

video\_path = "FNSEIZURE.mp4"

cap = cv2.VideoCapture(video\_path)

if not cap.isOpened():

    print("Error: Could not open video.")

    exit()

# video writer to save output

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

out = cv2.VideoWriter('TESToutput\_labeled1.avi', fourcc, 20.0, (int(cap.get(3)), int(cap.get(4))))

fgbg = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=16, detectShadows=True)

min\_contour\_area = 1000

kernel = np.ones((5, 5), np.uint8)

while cap.isOpened():

    ret, frame = cap.read()

    if not ret:

        break

    # Motion detection using background subtraction

    fgmask = fgbg.apply(frame)

    fgmask = cv2.threshold(fgmask, 200, 255, cv2.THRESH\_BINARY)[1]  # Thresholding

    fgmask = cv2.medianBlur(fgmask, 5)  # Apply median blur to reduce noise

    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH\_CLOSE, kernel)  # Morphological closing

    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH\_OPEN, kernel)  # Morphological opening to remove small objects

    contours, \_ = cv2.findContours(fgmask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    for contour in contours:

        if cv2.contourArea(contour) > min\_contour\_area:

            x, y, w, h = cv2.boundingRect(contour)

            roi = frame[y:y+h, x:x+w]

            # Classification logic based on bounding box size

            if h > 200 and w > 100:  # condition for 'walking'

                label = "Walking"

            elif h > 50 and w < 50:  # condition for 'standing'

                label = "Standing"

            else:

                label = "Seizure"

            # Draw bounding box and label

            color = (0, 255, 0) if label == "Walking" else (255, 0, 0) if label == "Standing" else (0, 0, 255)

            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

            cv2.putText(frame, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

    out.write(frame)

    cv2.imshow('Frame', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

# Release resources

cap.release()

out.release()

cv2.destroyAllWindows()

OUTPUT: 

**TECHNOLOGIES USED:**

1. Open CV: OpenCV is used for various computer vision tasks such as video capture, image processing, and object detection.
2. IPython Display: Used to display video files within a Jupyter Notebook environment.
3. NumPy (Numerical Python): NumPy is used for numerical computations, such as creating kernels for morphological operations.

**CONCLUSION:**

The development of an innovative monitoring system for TeleICU patients using video processing and deep learning presents a promising advancement in critical care. This system offers continuous, real-time monitoring of ICU patients through high-definition video feeds, leveraging state-of-the-art deep learning models to detect and alert medical staff of critical events. By integrating various technologies such as OpenCV for video processing, model training and scalable data processing and storage, the system ensures timely and accurate detection of patient distress and medical emergencies.

This innovative monitoring system has the potential to revolutionize TeleICU care by providing enhanced patient monitoring, early detection of critical events, and improved patient outcomes. By harnessing the power of video processing and other technologies, healthcare providers can offer more proactive and responsive care, ultimately saving lives and optimizing the utilization of healthcare resources.

