Mr. X

Team Members

1. Sahil Ali
2. Shambhavee Sinha
3. Vandana Singh
4. Sanjana Kumari
5. Shivam Kumar
6. Toufiq Hussain

Submitting to

Dumka Engineering College

Under the mentorship of

Mrs. Sunidhi Priyadharshini

Chapter	Page No.
Problem Address & Proposed Solution and Overview	1
Algorithm's Definition	1
Object tracking Algorithm	5-6
Crime Detection Algorithm	6-8
Facial Recognizer Algorithm	9
Vehicle Tracker Algorithm	10
Mr. X 0.0.1 Algorithm	11-13

Problem Addressed

Till now Law Enforcements have to search for criminals/suspects in each CCTV cameras and investigate from person to person which definitely gives time to criminals to run away, resulting in delay of the Day of Justice.

Proposed Solution

Overview

An Ai Vision Camera, Mr. X, which analyzes the environment, predicts crime, generate an alert and tracks the suspect within its vision. Now if the camera is able to get facial data and vehicle number plate of the suspect, then it will store these data in the server and starts a Search Operation in all camera with same software installed.

Now all the cameras that recognize the suspect's face or same vehicle number plate will give the path history of the suspect, allowing Law enforcement officers to catch the suspect as soon as possible.

Algorithm's Definition

Programming Laguages Used: - Python and C++

Algorithms used: - YOLO, DeepSORT, Kalman Filter, CNN + LSTM

IDE – Visual Studio Code

YOLO

YOLO is an object detection algorithm that can detect and locate objects in images and video frames. It's known for its speed and accuracy and is widely used in various applications, including surveillance, autonomous vehicles, and more. The primary idea behind YOLO is to perform object detection in a single pass through the neural network, making it very efficient for real-time processing.

DeepSORT

DeepSORT is an extension of the SORT (Simple Online and Realtime Tracking) algorithm, which is designed to track objects in video streams, such as surveillance cameras. The key enhancement in DeepSORT is the use of deep neural networks to improve tracking accuracy and robustness.

Kalman Filter

The Kalman Filter is an optimal recursive algorithm for estimating the state of a linear dynamic system. It's particularly useful when you have a system that evolves over time and is observed through noisy sensors. The filter predicts the current state of the system and updates it as new measurements become available

We are using a combination of YOLO and DeepSORT algorithm with Kalman Filter, to build an algorithm to track humans and vehicles in a Video.

OCS

Optical Character Recognition (OCR) is a technology that converts printed or handwritten text and characters into machine-encoded text.

ConvLSTM or CNN + LSTM

CNN (Convolutional Neural Network):

CNNs are well-known for their ability to extract spatial features from images. They consist of convolutional layers that learn filters to detect patterns like edges, textures, and more complex visual features.

LSTM (Long Short-Term Memory):

LSTMs, on the other hand, are recurrent neural networks (RNNs) that are excellent at handling sequential data. They can capture long-range dependencies and store information over time.

Why using combination of CNN and LSTM:

- Feature Extraction: You can use a CNN to extract features from video frames or image sequences. This is important for identifying objects, including suspects, in each frame.
- Temporal Modeling: Once you have extracted spatial features using the CNN, you can feed these features into an LSTM network. The LSTM can model the temporal dependencies and track the movement of objects or suspects across frames.
- Object Tracking: The LSTM can maintain a memory of past frames and use it to predict the location of objects, allowing you to track suspects as they move through the camera's field of view.
- Sequence Classification: If your goal is to classify certain actions or behaviors of suspects over a sequence of frames, the LSTM can be used for sequence classification tasks.

We are using ConvLSTM Model to create a pre-trained Crime Detection model.

Object Tracking Algorithm(YOLO + DeepSORT with Kalman filter)

Libraries used: -

YOLOv8
DeepSort
Tensorflow
Scikit
Touch
Opencv-python

Dataset: -

Coco dataset
|__Yolov3.weights
|__Yolov3.cfg
|__Yolo8n.pt
Weight file is a pre-trained model file on coco dataset (330,000 images)

Algorithm

```
# Objects unique IDs array
objects = []

# Initialize the Kalman Filter for each object
for object in objects:
    object.initialize_kalman_filter()

# Process video frames
for frame in video_frames:
    # Detect objects in the frame
    detected_objects = detect_objects(frame)
    # Update tracked objects with new detections
    for object in objects:
        # Predict the object's new position
        object.predict()
        # Associate the object with detected ojects
        object.update_association(detected_objects)

    # Create new tracking objects for unmatched detections
    new_objects = create_new_objects(detected_objects, objects)

    # Add new objects to the list of tracked objects
    objects.extend(new_objects)

    # Remove lost objects
    objects = remove_inactive_objects(objects)

    # Post process tracked objects
    post_process_objects(objects)

    # Draw bounding boxes on the frame for visualization
    draw_objects_on_frame(frame, objects)

    #save the processed frame
    display_frame(frame)
```

Crime Detection Algorithm (ConvLSTM = CNN + LSTM)

Libraries Used: -

Tensorflow
Scikit-learn
Tensorflow.keras
Opencv-python

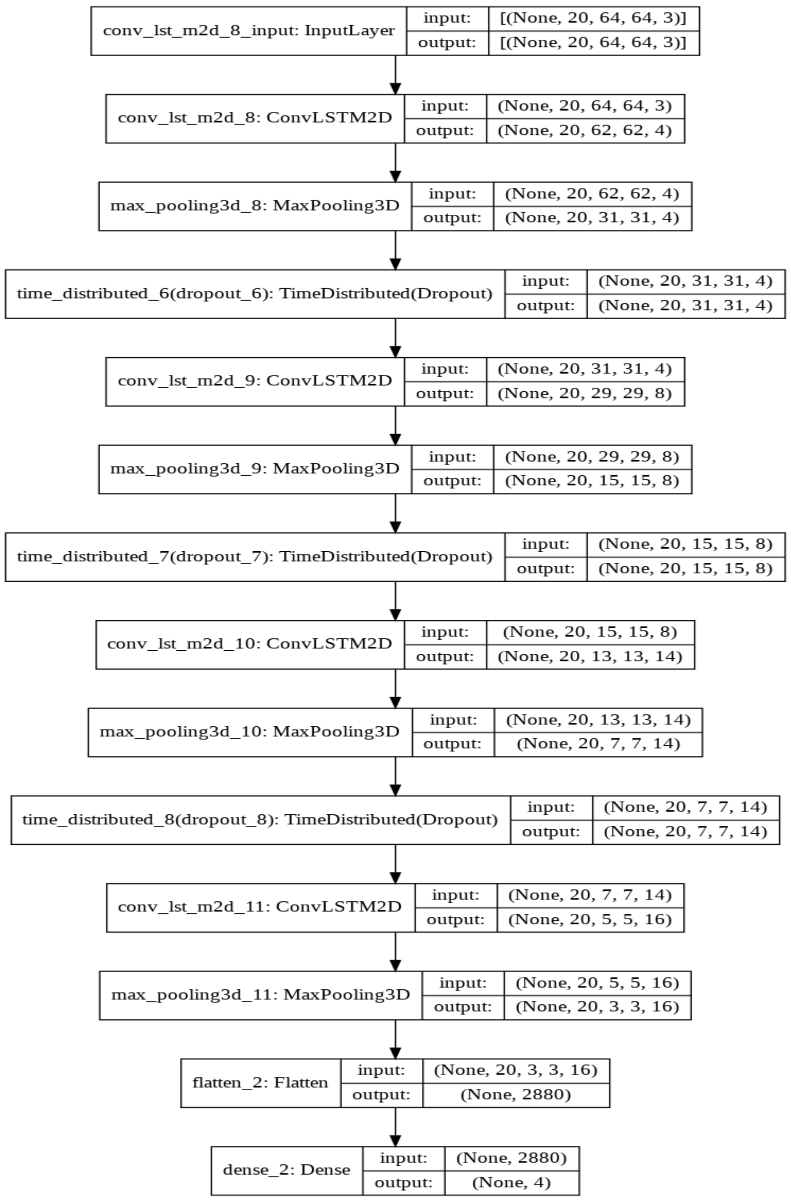
Dataset: -

Video dataset of Gun Shooting, Fighting, Running and smoking with 150 video per class.

Model Creation Algorithm: -

```
# dataset frame collection array
frame_list = []

# dataset preprocessing like frame extraction, augmentation and  data balancing,
def data_preprocessing():
    frame = frame_extraction()
    frame_list.append(frame)
    create_dataset()
    label_dataset()

# dividing dataset into test set and training set (75% and 25% respectivily)
features_train, features_test, labels_train, labels_test = train_test_split(75%,25%)

# creating Model
def create_conv_lstm_model():

    # use a Sequential model for model construction
    model = Sequential()

    # Define the Model Architecture.
    ##################################################################################################
######################

    model.add(ConvLSTM2D(filters = 4, kernel_size = (3, 3), activation = 'tanh',data_format =
"channels_last",
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (SEQUENCE_LENGTH,
                                                                                       IMAGE_HEIGHT,
IMAGE_WIDTH, 3)))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh', data_format =
"channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 14, kernel_size = (3, 3), activation = 'tanh', data_format =
"channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh', data_format =
"channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "softmax"))
  ####################################################################################################
####################

    # Display the models summary.
    model.summary()

    # Return the constructed convlstm model.
    return model

EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min', restore_best_weights = True)

convlstm_model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ["accuracy"])

convlstm_model.fit(x = features_train, y = labels_train, epochs = 50, batch_size = 4,shuffle = True,
validation_split = 0.2, callbacks = [early_stopping_callback])
```

Model Architecture: -


conv_lst_m2d_8_input: InputLayer
input: [(None, 20, 64, 64, 3)]
output: [(None, 20, 64, 64, 3)]
conv_lst_m2d_8: ConvLSTM2D
input: (None, 20, 64, 64, 3)
output: (None, 20, 62, 62, 4)
max_pooling3d_8: MaxPooling3D
input: (None, 20, 62, 62, 4)
output: (None, 20, 31, 31, 4)
time_distributed_6(dropout_6): TimeDistributed(Dropout)
input: (None, 20, 31, 31, 4)
output: (None, 20, 31, 31, 4)
conv_lst_m2d_9: ConvLSTM2D
input: (None, 20, 31, 31, 4)
output: (None, 20, 29, 29, 8)
max_pooling3d_9: MaxPooling3D
input: (None, 20, 29, 29, 8)
output: (None, 20, 15, 15, 8)
time_distributed_7(dropout_7): TimeDistributed(Dropout)
input: (None, 20, 15, 15, 8)
output: (None, 20, 15, 15, 8)
conv_lst_m2d_10: ConvLSTM2D
input: (None, 20, 15, 15, 8)
output: (None, 20, 13, 13, 14)
max_pooling3d_10: MaxPooling3D
input: (None, 20, 13, 13, 14)
output: (None, 20, 7, 7, 14)
time_distributed_8(dropout_8): TimeDistributed(Dropout)
input: (None, 20, 7, 7, 14)
output: (None, 20, 7, 7, 14)
conv_lst_m2d_11: ConvLSTM2D
input: (None, 20, 7, 7, 14)
output: (None, 20, 5, 5, 16)
max_pooling3d_11: MaxPooling3D
input: (None, 20, 5, 5, 16)
output: (None, 20, 3, 3, 16)
flatten_2: Flatten
input: (None, 20, 3, 3, 16)
output: (None, 2880)
dense_2: Dense
input: (None, 2880)
output: (None, 4)

Face Recognizer and data collector Algorithm: -

Libraries Used: -
    Opencv-Python

Dataset: -
    HaarCascade_face.xml (pre-trained model file)

Algorithm: -

```
# Import necessary libraries and load the pre-trained Haar cascade classifier
import cv2

# Load the pre-trained Haar cascade classifier for face detection
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# Load an image or video frame
frame = cv2.imread('input_image.jpg')

# Convert the frame to grayscale (Haar cascades work on grayscale images)
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Detect faces in the grayscale frame
faces = face_cascade.detectMultiScale(gray_frame, scaleFactor=1.3, minNeighbors=5, minSize=(30, 30))

# stores facial data in local server
Store_facial_data()

# Draw rectangles around the detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display or save the frame with detected faces
cv2.imshow('Face Detection', frame)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Vehicle Tracker and data collector Algorithm: -

Libraries: -

    Opencv-python
    OCR
    YOLO

Dataset: -

    license_plate_detector.pt (image to text reading pre-trained Model)

Algorithm: -

```
results = {}
path = get_path()
mot_tracker = Sort()

# load models
coco_model = YOLO("yolov8n.pt")
license_plate_detector = YOLO("license_plate_detector.pt")

# get push data from server to search the number plate
srch_num = get_srch()

# load video
cap = cv2.VideoCapture("C:/Users/toufiqhussain/Downloads/test_2.mp4")
vehicle_number = []
# read Frame
while True:
    frame_nmr += 1
    ret, frame = cap.read()

    # detect vehicle
    detection = vehicle_detection()

    # track vehicle
    tracked_vehicle_id += yolo_track(detection)

    # detect number plate on detection
    number_plates = detect_num_plate(tracked_vehicle_id)

    # read number plate
    for number_plate in number_plates:
        num = easyOCR(number_plates)
        vehicle_number.append(num)
        # check if num match, if match then generate alert and add to path
        if srch_num and  src_vhl_num == num:
            alert()
            report()
            path.append(latitude,longitude)

    cap.release()
```

Models Assembling
(Object Tracker + Crime Detector + Face recognizer + Vehicle Tracker)

Concept: -

We will use threading concept to run all the algorithms while camera is recording live.
We will save the footage locally at every 10 sec, so now all the algorithms will analyze the video and extract all the important data like all the facial and vehicle's data that comes in the frame and if crime happens then crime detection model will generate an alert.

Libraries used: -

OpenCV-Python
Threading
Shutil
OS

Algorithm: -

```
# Crime Detection Model
def crime_pred(video_path):
    return detection

# Vehicle Tracker Model
def vehicle_track(video_path, srch):
    store_data()
    if found:
        return True
    else:
        return False

# Facial Recognizer Model
def facial_recog(video_path, srch):
    store_data()
    if found:
        return True
    else:
        return False

# Run Camera and Start Recording
def main():

    # Creating Thread
    thread_1 = threading.Thread(target=crime_pred)
    thread_2 = threading.Thread(target=vehicle_track)
    thread_3 = threading.Thread(target=facial_recog)

    # Starting Thread
    thread_1.start()
    thread_2.start()
    thread_3.start()


    # starting camera
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("Error: Could not open camera.")
        return
    frame_width = int(cap.get(3))
    frame_height = int(cap.get(4))
    fps = int(cap.get(5))

    fourcc = cv2.VideoWriter_fourcc(*"H264")  # Codec for MP4 format
    stream_1 = None
    stream_2 = None
    s1_fileName = None
    s2_fileName = None
    recording_timer = 0
    file_no = 0
# Crime Detection Model
def crime_pred(video_path):
    return detection

# Vehicle Tracker Model
def vehicle_track(video_path, srch):
    store_data()
    if found:
        return True
    else:
        return False

# Facial Recognizer Model
def facial_recog(video_path, srch):
    store_data()
    if found:
        return True
    else:
        return False

# Run Camera and Start Recording
def main():

    # Creating Thread
    thread_1 = threading.Thread(target=crime_pred)
    thread_2 = threading.Thread(target=vehicle_track)
    thread_3 = threading.Thread(target=facial_recog)

    # Starting Thread
    thread_1.start()
    thread_2.start()
    thread_3.start()

    # starting camera
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("Error: Could not open camera.")
        return

    frame_width = int(cap.get(3))
    frame_height = int(cap.get(4))
    fps = int(cap.get(5))

    fourcc = cv2.VideoWriter_fourcc(*"H264")  # Codec for MP4 format
    stream_1 = None
    stream_2 = None
    s1_fileName = None
    s2_fileName = None
    recording_timer = 0
    file_no = 0

    while True:
        ret, frame = cap.read()

        # Switching between Threads
        if not ret:
            print("Error: Could not read frame.")

            #join thread if camera closes
            thread_1.join()
            thread_2.join()
            thread_3.join()
            break
        if recording_timer == fps * 0:
            stream_1 = cv2.VideoWriter("path.mp4",
                fourcc,
                fps,
                (frame_width, frame_height),
            )
        if recording_timer >= fps * 7 and recording_timer <= fps * 17:
            stream_2.write(frame)
            if recording_timer == fps * 17:
                stream_2.release()

        if recording_timer == fps * 17:
            recording_timer = 0
        else:
            recording_timer += 1

        cv2.imshow("Recording", frame)

        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
    cap.release()
    cv2.destroyAllWindows()

    #join thread if camera closes
    thread_1.join()
    thread_2.join()
    thread_3.join()


if __name__ == "__main__":
    main()
```

Output Example: -

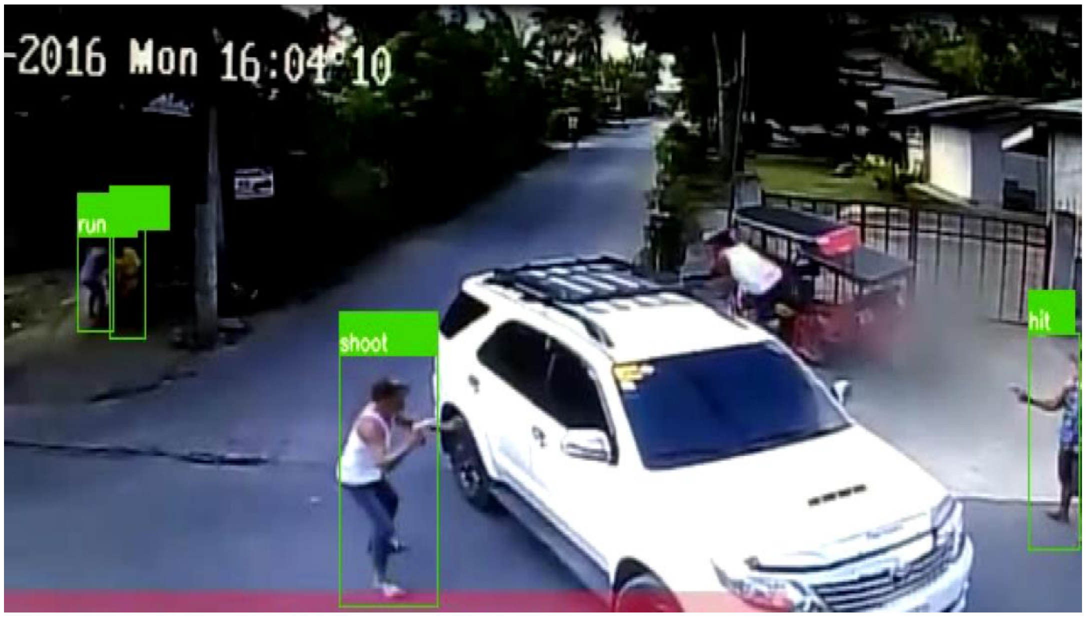


Fig: - Result on CCTV footage of a gun firing in Indore

Conclusion: -

Till now we have created a program to implement it on a camera in real-time, so it can predict if the crime occurs or not, gets the facial data of suspect (if available) and vehicle number plate. It can also search for suspects face data and vehicle number plate if comes in the vision of any camera.

Future of the Project: -

Our program is only use full for tracking if facial data or number plate data is available but what if the suspect hide his face or remove number plate, which happens in most cases.
So, our next target is to build a tracking model that tracks anything without depending on specific data that can be removed to fool the surveillance system and escape.
Most probably we will use a Stereo Vision System and a Reinforcement agent for this purpose.