Automatic Vehicle Plate Number Blurring System

Ahmed Khaled - 236664
Faculty of Informatics & Computer Science
The British University in Egypt
Cairo, Egypt

Submitted to Dr. Ahmed Fahmi

1. Abstract

In an era where surveillance and street photography have become very popular, ensuring the privacy of individuals captured in public images has become increasingly more important. Vehicle license plates are personal information and require protection, particularly when images are shared publicly. This paper presents an automated image processing system for detecting and blurring vehicle license plates using deep learning. We explored the performance of multiple YOLO (You Only Look Once) object detection models, YOLOv8n, YOLOv10n, and YOLOv11n. Each model was trained across different epoch configurations (3, 5, and 10 epochs) to determine the most accurate and efficient model. The best-performing model was further trained to 50 epochs to maximize detection reliability. A blurring mechanism is then applied over the detected license plates, ensuring it's hidden and meets privacy standards. The system is tested on a diverse set of images, including varying angles, lighting conditions, and plate styles, to evaluate its robustness. This project imposes a practical and scalable approach to license blurring and protecting private information online.

2. Keywords

License Plate Detection, YOLO, Object Detection, Image Privacy, Deep Learning, Blurring, Visual Computing, Egyptian License plates.

3. Introduction

The increasing number of surveillance cameras, dashcams, and public photography raises significant concerns around visual privacy. Among the most sensitive elements in such imagery are vehicle license plates, which can be used to trace identity, movement, or ownership. Protecting license plates in images especially when they are shared or stored is both an ethical and legal requirement in many countries.

Traditionally, license plate blurring has been done manually or through exhaustive detection methods, both of which lack scalability and robustness. With the advancement of deep learning, particularly object detection models, it is now possible to automate plate detection with high accuracy, even in challenging scenarios like poor lighting, motion blur, or angled views.

This project addresses the challenge of developing a fully automated system to detect and blur vehicle license plates in static images or dynamic videos. We will use YOLO object detection models, known for their speed and accuracy, and compare several versions such as YOLOv8n, YOLOv10n, and YOLOv11n across multiple epochs to identify the most effective architecture for this task. Once detected, the plates are automatically blurred using image processing techniques.

4. System Description

4.1 Problem Overview

License plates in images are identifiers that can be linked back to individuals or vehicles. Public sharing of such content without blurring could violate privacy laws. Our system focuses on detecting license plates in static images and dynamic videos and applying a blur over them to make the data not visible. This project will focus on Egyptian license plates which contain nine different styles of license plate that looks like the following:

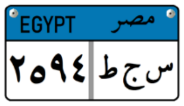


Figure 1.

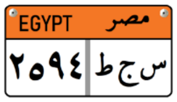


Figure 2.

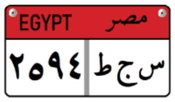


Figure 3.

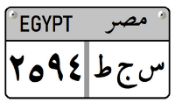


Figure 4.

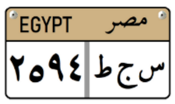


Figure 5.

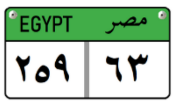


Figure 6.

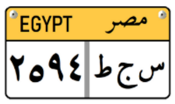


Figure 7.

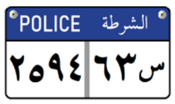


Figure 8.

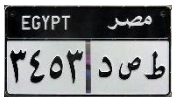


Figure 9.

Figure #	Designated for	Color
1	Private Vehicles	Light Blue
2	Taxis	Orange
3	Truck	Red
4	Buses	Gray
5	Limousines	Beige
6	Diplomatic Vehicles	Green
7	Foreign vehicles	Yellow
8	Police Vehicles	Dark Blue
9	Honoring the human	Black

Goal:
In a city surveillance photo or a video, our system will:

1. Detect the location of all license plates using a trained YOLO model.
2. Apply a Gaussian blur effect to each license plate.
3. Output the privacy-compliant version of the image.

4.2 Dataset

We used a dataset of labelled vehicle license plates to train and evaluate our object detection models. The dataset contains exactly 2,085 images, offering a diverse range of license plate examples. It features various plate styles and fonts, and includes images captured under different lighting conditions such as daylight, nighttime, and shadows. The dataset also includes multiple camera angles such as frontal, diagonal, and partial views providing a realistic scenario that improve the model's robustness. To ensure balanced training and unbiased evaluation, we split the dataset into training and validation sets using an 80 to 20 ratio. As part of our training process, we used the Egyptian Cars Plates dataset published by Mahmoud Bahaa on Kaggle. This dataset contains labelled images of Egyptian license plates, it presents a valuable resource for studying license plates.

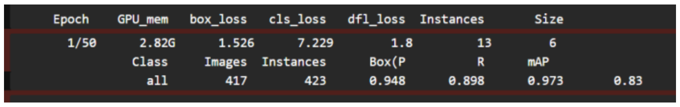
4.3 Model Selection and Comparison

To identify the most effective object detection architecture for license plate recognition, we trained and evaluated three versions of the YOLO model: YOLOv8n, YOLOv10n, and YOLOv11n. Each model was trained at 3, 5, and 10 epochs to analyze learning behavior, speed, and detection accuracy across early training phases. This allowed us to observe how quickly each model adapted to the dataset and which model demonstrated the best performance with minimal training. The training was conducted on the same setup to ensure consistency and efficient experimentation. The system specifications were as follows:

- **CPU**: Intel Core i9-11900H
- **GPU**: NVIDIA GeForce RTX 3070
- **RAM**: 32 GB DDR4
- **Operating System**: Windows 11
- **Training Environment**: PyTorch 2.6.0 with CUDA 12.4 support

All training was executed on the GPU using CUDA acceleration to reduce training time and benefit from parallel computation. Based on the results from the initial epoch evaluations, the best-performing model will be selected and trained for 50 epochs to reach its full potential before final testing and deployment.

4.4 Results Histograms



```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/50      2.82G      1.526      7.229        1.8         13          6
                 Class     Images  Instances      Box(P          R        mAP
                   all        417        423      0.948      0.898      0.973       0.83
```

Figure 10.

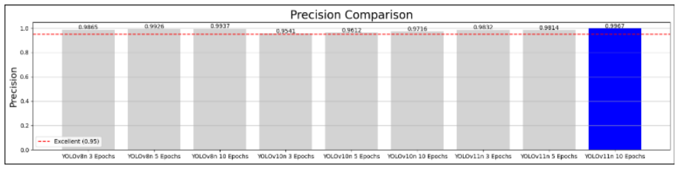


Figure 11.

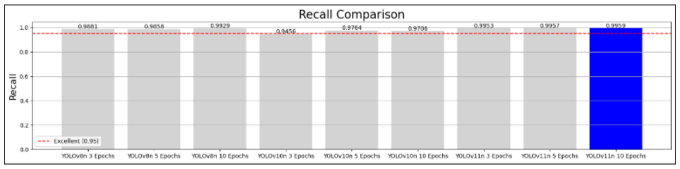


Figure 12.

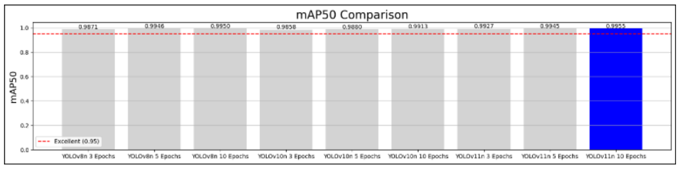


Figure 13.

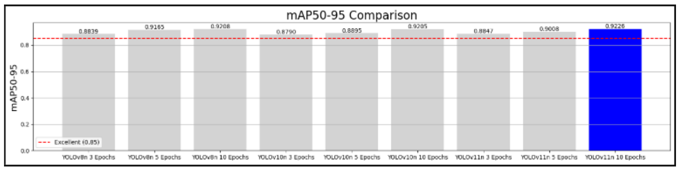


Figure 14.

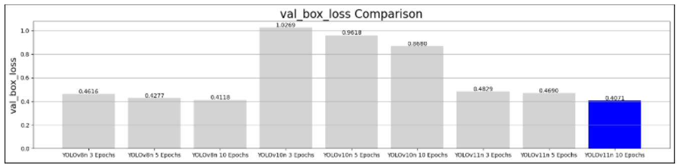


Figure 15.

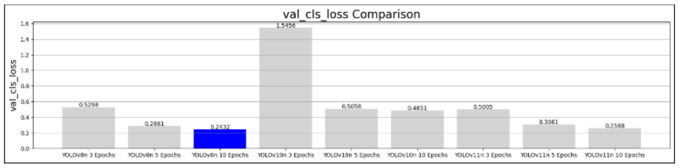


Figure 16.

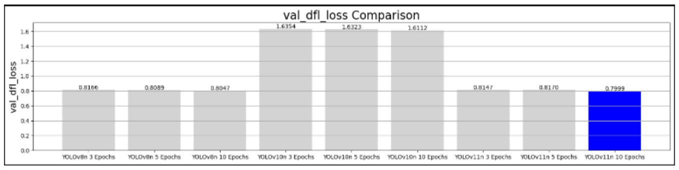


Figure 17.

Explanation:

- Figure 10, Training Output Example: This image shows how the training processes looks like inside the Jupyter Notebook.
- Figure 11, Precision Comparison: This image shows the best score of precision recorded for each model on every epoch.
- Figure 12, Recall Comparison: This image shows the best score of Recall recorded for each model on every epoch.
- Figure 13, mAP50 Comparison: This image shows the best score of mAP50 recorded for each model on every epoch.
- Figure 14, mAP50-90 Comparison: This image shows the best score of mAP50-90 recorded for each model on every epoch.
- Figure 15, val_box_loss Comparison: This image shows the best score of val_box_loss recorded for each model on every epoch.
- Figure 16, val_cls_loss Comparison: This image shows the best score of val_cls_loss recorded for each model on every epoch.
- Figure 17, val_dfl_loss Comparison: This image shows the best score of val_dfl_loss recorded for each model on every epoch.

4.5 Evaluation Criteria

The criteria that are used to know which model the best one is based on 7 key scores which are:

1. **Precision:** Measures of how accurate the model's positive predictions are. High precision means most of the detected license plates are actually correct. Higher values mean better model.

2. **Recall:** Measures of how well the model finds all relevant objects. High recall means the model detects nearly all license plates. Higher values mean better model.

3. **mAP50:** The mean Average Precision at IoU ≥ 0.5. This metric combines localization and classification performance. High mAP50 indicates the model is good at placing bounding boxes correctly and identifying the objects inside. Higher values mean better model.

4. **mAP50-90:** A stricter version of mAP, averaging over multiple IoU thresholds (from 0.5 to 0.95). It reflects the model's ability to make precise detections across varying strictness. Lower than mAP50, but more reliable for fine-tuning. Higher values mean better model.

5. **val_box_loss:** Validation loss related to the bounding box regression. Indicates how closely the predicted boxes match the actual object positions. Lower values mean more accurate localization. Lower values mean better model.

6. **val_cls_loss:** Classification loss during validation. Measures how well the model predicts the correct class (license plate). High values could mean misclassifications. Lower values mean better model.

7. **val_dfl_loss:** Distribution Focal Loss — used to refine bounding box predictions in YOLO. It helps improve boundary alignment. Lower values indicate smoother and more precise box outputs. Lower values mean better models.

Result Summary:

After evaluating the graphs, it appears that YOLO11n at 10 epochs outperformed all other models across nearly all criteria, except for criteria 6, where YOLOv8n had a slight advantage. However, after a closer examination, the data reveals that the difference between YOLO11n and YOLOv8n on these criteria was minimal. In fact, YOLO11n was ranked second in that category, outperforming all other models and epochs. This reinforces the conclusion that YOLO11n remains the most promising option overall. Given its strong performance at just

10 epochs, it is reasonable to expect that YOLO11n will significantly surpass YOLOv8n and all the other models as the number of epochs increases especially at 50 epochs and beyond.

5. Methodology and Implementation

In this section, we will provide a detailed overview of the methodology used in this project, including the setup process, model selection, and evaluation. We also will walk through the implementation steps, highlighting how the YOLO models were integrated, tested, and compared using custom test images. This will help replicate the process or adapt it to similar object detection tasks.

5.1 Step & Installation

We will begin by downloading all the required packages for this project.

```
!pip install opencv-python
!pip install matplotlib
!pip install ultralytics
```

5.2 Importing Libraries

After downloading the packages, we will now import our needed libraries with any prefix we desire

```
from matplotlib import pyplot as plt
from ultralytics import YOLO
import pandas as pd
import numpy as np
import torch
import cv2
```

5.3 Initialization & Configuration

After completing the setup process, we will now initialize any variable that we want to use a lot further in the notebook, in our case we will only initialize the path of the dataset.yaml file of the dataset used for the training.

```
path = r"Egyptian License
Plate\dataset.yaml"
```

5.4 Model Training for Comparison

In this part, we will begin training different versions of YOLO on varying numbers of epochs to evaluate and compare their performance. The models being trained include YOLOv8n, YOLOv10n, and YOLOv11n, each trained on 3, 5, and 10 epochs.

Note: This step is not necessary, and it was only carried out to identify the most promising model to proceed with.

We start by loading the default YOLO model provided by the Ultralytics library and training it for an initial 3 epochs. Afterward, we will resume training the same model for 2 more epochs to obtain a version trained for a total of 5 epochs. The same approach is used to extend training to 10 epochs. This progressive training strategy is applied consistently across all YOLO versions being tested.

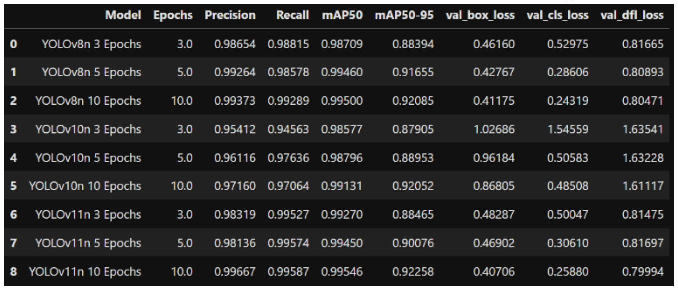
```
Yolov8n_3_Epochs = YOLO("yolov8n.pt")
result = Yolov8n_3_Epochs.train(data =
path, epochs = 3, imgsz = 640)!pip

#Train YOLOV8n using only 3 epochs.

Yolov8n_5_Epochs =
YOLO(r"runs\detect\train1\weights\last.pt
")
result = Yolov8n_5_Epochs.train(data =
path, epochs = 2, imgsz = 640, resume =
True)

#Loads the previous YOLOV8n and adds 2
more epochs to reach a total of 5 epochs.

Yolov8n_10_Epochs =
YOLO(r"runs\detect\train2\weights\last.pt
")
result = Yolov8n_10_Epochs.train(data =
path, epochs = 5, imgsz = 640, resume =
True)

#Loads the previous YOLOV8n and adds 5
more epochs to reach a total of 10
epochs.
```

This code sample shows the previously described process but only for model YOLOv8n. This code will also be preformed two more times to get YOLOv10n and YOLOv11 trained.

Foreshadowing: Finally, after our tests we found out that YOLOv11 was the best model to work on. So, we will repeat this sample again but this time using 50 epochs.

5.5 Evaluation

After completing the training for each model, a results.csv file is generated containing detailed information about the model's accuracy and performance. These files were opened, and the best values from each were selected to represent the overall performance of each model. The extracted data was then stored in a Data Frame to make the information easier to interpret and more accessible for the next step of the analysis. The data extraction process was automated using a loop to iterate through each file and collect the relevant metrics. Here is how the data frame looked like after the data extraction processes.



	Model	Epochs	Precision	Recall	mAP50	mAP50-95	val_box_loss	val_cls_loss	val_dfl_loss
0	YOLOv8n 3 Epochs	3.0	0.98654	0.98815	0.98709	0.88394	0.46160	0.52975	0.81665
1	YOLOv8n 5 Epochs	5.0	0.99264	0.98578	0.99460	0.91655	0.42767	0.28606	0.80893
2	YOLOv8n 10 Epochs	10.0	0.99373	0.99289	0.99500	0.92085	0.41175	0.24319	0.80471
3	YOLOv10n 3 Epochs	3.0	0.95412	0.94563	0.98577	0.87905	1.02686	1.54559	1.63541
4	YOLOv10n 5 Epochs	5.0	0.96116	0.97636	0.98796	0.88953	0.96184	0.50583	1.63228
5	YOLOv10n 10 Epochs	10.0	0.97160	0.97064	0.99131	0.92052	0.86805	0.48508	1.61117
6	YOLOv11n 3 Epochs	3.0	0.98319	0.99527	0.99270	0.88465	0.48287	0.50047	0.81475
7	YOLOv11n 5 Epochs	5.0	0.98136	0.99574	0.99450	0.90076	0.46902	0.30610	0.81697
8	YOLOv11n 10 Epochs	10.0	0.99667	0.99587	0.99546	0.92258	0.40706	0.25880	0.79994

Figure 18.

```
For model_name, file_path in {

    'YOLOv8n 3 Epochs':
r'C:\Users\gosta\Desktop\Yolo Models
Comparison\Yolo8n\Yolo8n - 3
Epoch\results.csv', YOLOv8n 5 Epochs':
r'C:\Users\gosta\Desktop\Yolo Models
Comparison\Yolo8n\Yolo8n - 5
Epoch\results.csv', . . .

df = pd.read_csv(file_path)
    last = df.iloc[-1]
    model_metrics.append({
        'Model': model_name,
        'Epochs': last['epoch'],
        'Precision':
last['metrics/precision(B)'],
        'Recall':
last['metrics/recall(B)'],
        'mAP50':
last['metrics/mAP50(B)'],
        'mAP50-95': last['metrics/mAP50-
95(B)'],
        'val_box_loss':
last['val/box_loss'],
        'val_cls_loss':
last['val/cls_loss'],
        'val_dfl_loss':
last['val/dfl_loss'],
    })
```

5.6 Evaluation

With the data prepared, the next step is to visualize it using Matplotlib histograms. Each evaluation criterion will be compared across all models and their corresponding epochs. This provides a clear and intuitive way to analyze model performance side by side. Examples of the expected histogram format can be seen in Figures 11 through 17.

Seven histograms were created one for each criterion, the criteria's explanation can be found above in section 4.4

```
plt.figure(figsize=(20, 4))

# Highlight best value
colors = ['lightgray'] *
len(comparison_df)
colors[comparison_df['Precision'].idxmax(
)] = 'blue'

# Create bar chart
bars = plt.bar(comparison_df['Model'],
comparison_df['Precision'], color=colors)

# Labels & title
plt.title('Precision Comparison',
fontsize = 20)
plt.ylabel('Precision', fontsize = 16)

# Add benchmark line
plt.axhline(0.95, color='red',
linestyle='--', label='Excellent (0.95)')
plt.legend()

# Add value labels
for bar in bars:
    plt.text(bar.get_x() +
bar.get_width()/2, bar.get_height() +
0.005, f'{bar.get_height():.4f}',
ha='center')

# Add grid lines on y-axis
plt.grid(True, axis='y')
plt.show()
```

5.7 Applying The Best Model On Images

After completing all the previous steps, we now have a YOLOv11n model trained for 50 epochs. It's time to put this model to the test and evaluate its performance in real-world scenarios.

Firstly, we will load the model and images that we are going to test.

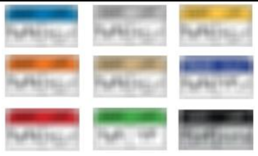
```
BestModel = YOLO(r"weights/best.pt")
image_paths = [
    r"image1.png",
    r"image2.png", . . . ]
```

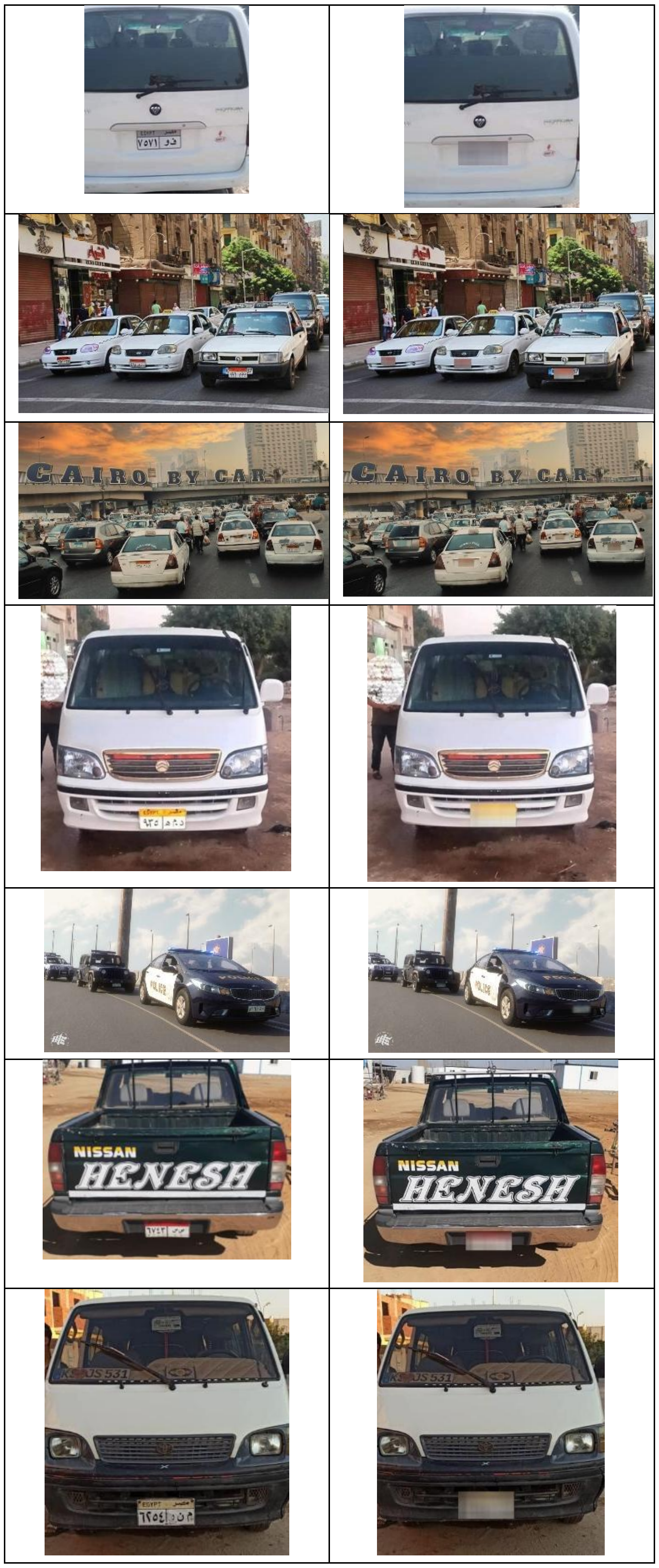
Secondly, we'll run the code that iterates over the selected images and apply Gaussian Blur to the license plates detected by the model. The original and blurred versions of each image are then stored as pairs in an array, allowing us to easily display the before-and-after results side by side.

```
# Store pairs of original and blurred
images
image_pairs = []

for img_path in image_paths:
    original = cv2.imread(img_path)
    image = original.copy()

    # Give image to model
    results = BestModel(img_path)

    # Apply Gaussian blur
    for r in results:
        if r.boxes is not None:
            for box in r.boxes.xyxy:
                x1, y1, x2, y2 = map(int,
box.tolist())
                roi = image[y1:y2, x1:x2]
                blurred =
cv2.GaussianBlur(roi, (65, 65), 30)
                image[y1:y2, x1:x2] =
blurred

    # Convert BGR to RGB
    original_rgb = cv2.cvtColor(original,
cv2.COLOR_BGR2RGB)
    blurred_rgb = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)
    image_pairs.append((original_rgb,
blurred_rgb))

# Display the images before and afer
n = len(image_pairs)
plt.figure(figsize=(12, 5 * n))

for i, (orig, blur) in
enumerate(image_pairs):
    plt.subplot(n, 2, 2 * i + 1)
    plt.imshow(orig)
    plt.title(f"Original Image {i+1}")
    plt.axis('off')

    plt.subplot(n, 2, 2 * i + 2)
    plt.imshow(blur)
    plt.title(f"Blurred Image {i+1}")
    plt.axis('off')

plt.tight_layout()
plt.show()
```

After running this code, you should get an output of images side by side, showing the image before and after the blurring effect. In this area I have selected 15 photos with different angles, lighting conditions, and plate styles. These tests should show the capabilities of the Model.

Here is the sample of the images I used.

Images before	Images After
	



5.8 Applying The Best Model On Videos

Since our model performs flawlessly on static images, it can also be applied to videos after all, a video is simply a sequence of individual frames (images). The video can be broken down into multiple frames, and the model can be used to generate a blurred version of each frame. Once all frames have been processed, they are recompiled to produce a fully blurred output video.

```
input_video = r"Input.mp4"
output_video = r"Output.mp4"

# Open video
video = cv2.VideoCapture(input_video)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
fps = int(video.get(cv2.CAP_PROP_FPS))
frame_width =
int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height =
int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
out = cv2.VideoWriter(output_video,
fourcc, fps, (frame_width, frame_height))

# Process each frame
while video.isOpened():
    ret, frame = video.read()
    if not ret:
        break

    # Run detection
    results = BestModel(frame)

    for r in results:
        if r.boxes is not None:
            for box in r.boxes.xyxy:
                x1, y1, x2, y2 = map(int,
box.tolist())
                roi = frame[y1:y2, x1:x2]
                blurred =
cv2.GaussianBlur(roi, (25, 25), 30)
                frame[y1:y2, x1:x2] =
blurred

    # Write processed frame to output
video
    out.write(frame)

cap.release()
out.release()
cv2.destroyAllWindows()
print("Video processing complete.")
```

A sample of the tested video can be found in the following link: https://www.youtube.com/watch?v=bEtzdi2-8Vg

6. Conclusion

This project successfully demonstrates the development of an automated vehicle license plate blurring system using deep learning, with a focus on ensuring visual privacy in publicly shared images and videos. By leveraging multiple versions of the YOLO object detection architecture (YOLOv8n, YOLOv10n, and YOLOv11n), we conducted a thorough comparison based on precision, recall, mAP, and loss metrics to determine the most effective model.

Through iterative training across different epoch configurations (3, 5, and 10), YOLOv11n emerged as the most promising candidate. Its strong performance even at early training stages suggested excellent scalability, which was confirmed after extending the training to 50 epochs. The final model demonstrated robust license plate detection across a variety of challenging conditions, including varying lighting, angles, and plate styles.

Using Gaussian blur, we implemented a reliable anonymization method that effectively concealed license plates in both static images and dynamic videos. The system maintained real-time processing capabilities and high accuracy, making it suitable for integration into surveillance, public monitoring, or content-sharing platforms where user privacy must be preserved.

Overall, this project presents a practical and scalable solution for automated privacy enforcement in visual content, setting the foundation for future improvements such as real-time video deployment, multi-region plate recognition, and expanded datasets for broader applicability.