Ben-Gurion University of the Negev

Faculty of Engineering Science

School of Electrical and Computer Engineering

Dept. of Electrical and Computer Engineering

Fourth-Year Engineering Project

Final Report

Project Name: DL Traffic Signs Overlays by Artificial Data Set

Project number:	p-2024-080
Students:	Shireto Adi - 318277514
	Tsalovich Alon - 208473629
Supervisors:	Prof. Hadar Ofer
	Mr. Dror Itai
Submitting date:	29/09/24

Contents

1.Abstract

1.1 Hebrew abstract

כיסויי תמרורי תנועה מבוססים למידה עמוקה באמצעות מערך נתונים מלאכותי

שמות הסטודנטים: צלוביץ אלון, שרטו עדי

shiretoa@post.bgu.ac.il

מנחים: פרופסור הדר עופר, מר דרור איתי

תקציר:

כיום, חברות רבות בעולם עובדות על פיתוח מגוון פתרונות ומוצרים לדבר הבא בעולם הרכב, הרכב האוטונומי. אתגר מרכזי בנהיגה אוטונומית מלאה הוא זיהוי תמרורים. כאשר מערכות אוטונומיות נתקלות במגבלות, שליטה אנושית מרחוק יכולה להוות פתרון בר-קיימא, המחייבת העברת וידאו איכותית ויעילה בין מצלמת הרכב למסך הנהג המרוחק.

פרויקט זה נותן מענה לצורך לשפר את איכות הווידאו ולהפחית את השימוש ברוחב הפס עבור תרחישי נהיגה מרחוק, תוך התמודדות עם אתגרי וידאו בזמן אמת כגון תמרורים הנצפים מזוויות שונות, גדלים שונים עקב מרחק ובעיות ראות הנגרמות מתנאי מזג אוויר, סנוור מהשמש, תמרורים דהויים או שעברו השחתה (מדבקות ,גרפיטי) אבל שעדיין ניתן לזהות אותם.

הגישה שלנו כוללת אלגוריתם שיוצר מסכות צבע ייעודית עבור כל תמרור ומודל אוטונקודר שמשחזר את התמרורים ממסכות הצבע המתאימות להם. לצורך אימון המודל, יצרנו מאגר מידע המכיל זוגות של תמרורים ומסכות צבע מתוך תמונות מהחיים האמיתיים, תוך התחשבות באתגרים הקשורים לזוויות, גדלים, ראות מוגבלת וסנוור.

אנו מצפים שהמערכת תוכל להתמודד עם תנאי ראות שונים עבור מגוון רחב של תמרורים ולספק תמונה ברורה יותר של התמרור לנהג המרוחק.

מילות מפתח:

זיהוי תמרורים, YOLO, מקודד אוטומטי, שיפור וידאו, הפחתת רוחב פס, נהיגה מרחוק.

1.2 English abstract

DL Traffic Signs Overlays by Artificial Data Set

Students: Tsalovich Alon, Shireto Adi

shiretoa@post.bgu.ac.il

supervisors: Prof. Hadar Ofer, Mr. Dror Itai

Abstract:

Today, many companies in the world are working on developing a variety of solutions and products for the next thing in the automotive world, the autonomous vehicle. One major challenge in fully autonomous driving is traffic sign recognition. When autonomous systems encounter limitations, remote human control can be a viable solution, requiring high-quality and efficient video transmission between the vehicle's camera and the remote driver's screen.

This project addresses the need to enhance video quality and reduce bandwidth usage for remote driving scenarios, while also tackling real-time video challenges such as traffic signs observed from various angles, different sizes due to distance, and visibility issues caused by weather conditions, glare from the sun, faded or vandalized signs (stickers, graffiti) but still recognizable.

Our approach involves an algorithm that creates colored masks for each designated traffic sign and a trained autoencoder model that reconstructs the traffic signs from their appropriate color masks. For training the model, we created a database containing pairs of traffic signs and colored masks from real life images, while taking into consideration challenges related to angles, sizes, limited visibility, and glare.

We expect that the system will be able to cope with different visibility conditions for a wide variety of road signs and provide a clearer picture of the road sign for the remote driver.

Keywords:

Traffic sign recognition, YOLO, Autoencoder, Video enhancement, Bandwidth reduction, Remote driving.

2. Project Goals

1. Utilize real-world traffic signs: Ensure the system can recognize and reconstruct traffic signs from real-world environments, moving beyond simulated environments like Cognata. This is crucial to making the system applicable to actual driving scenarios, where the conditions are unpredictable and diverse. The **visual clarity** of the reconstructed road signs is the key measure of success.

2. Expand the variety of traffic signs: Train the system to handle a broader set of road signs, not limiting it to common or standardized signs. This expansion allows the system to operate in more locations, including areas with unique or less common road signs. The model should achieve a low **L1 loss**, indicating accuracy in the reconstruction process. Additionally, **SSIM** should be used to measure the similarity between the original and reconstructed images, ensuring that the reproduced signs are as close as possible to the originals.

3. Handle challenging visibility conditions: Design the system to be robust under various conditions that impact visibility, such as signs viewed from different angles, varying distances, and in adverse weather or lighting conditions (e.g., glare from the sun, fog, or rain). This is essential for ensuring reliable recognition of road signs in real-life driving situations.

4. Simulate the system in two demonstration videos:
   The first video will show the colored masks that replace the traffic signs, demonstrating how the system perceives the road signs in simplified form.
   The second video will show the reconstructed road signs, highlighting how the autoencoder model transitions from colored masks to clear, recognizable signs for the remote driver.
   In the first video, the colored masks should effectively replace the traffic signs.
   In the second video, the autoencoder model must successfully transition the colored masks back into clear, recognizable road signs, demonstrating the system's accuracy and effectiveness.

3. Introduction

3.1 Motivation

As autonomous vehicles continue to evolve, one of the primary challenges they face is the accurate and reliable recognition of traffic signs in real-time. While autonomous systems have made great strides, they can still encounter limitations in adverse conditions such as poor weather, glare, or unexpected obstacles. In such cases, remote human drivers can intervene to maintain safety and control. However, the transmission of high-quality video from the vehicle's camera to the remote driver can suffer from bandwidth constraints, leading to delays or reduced image quality, which may impair the driver's ability to correctly interpret traffic signs.

3.2 Purpose of the Project

This project aims to enhance the visibility and clarity of traffic signs displayed on a remote driver's screen, particularly under challenging conditions. The key approach involves classifying traffic signs into color-coded masks, which not only helps improve sign visibility but also reduces the bandwidth required to transmit video. By transmitting simplified mask data rather than full high-resolution images, we aim to optimize both image clarity and transmission efficiency, making the system more suitable for real-world remote driving scenarios.

3.3 Previous Work and Approaches

Previous approaches in traffic sign recognition often focused on single traffic signs without the need for classification between different signs. For instance, earlier work developed an autoencoder model that could reconstruct only one type of traffic sign at a time. These models did not face the challenge of distinguishing between multiple signs, limiting their applicability in real-world scenarios where a wide variety of signs must be recognized and reconstructed accurately.

Moreover, the previous project utilized Cognata simulator, which significantly simplified the environment. By using simulated traffic signs and controlled conditions, they were able to generate highly accurate masks, but this approach lacked the complexity and unpredictability of real-world driving. Simulators do not fully replicate challenges such as varying angles, lighting conditions, or visual obstructions, making them less practical for deployment in real-world scenarios.

To overcome the limitations of earlier works, our approach focuses on handling real-world traffic signs and accounting for multiple sign classifications. By using actual traffic sign images from varied conditions, we ensure the system can cope with diverse, unpredictable situations that autonomous vehicles encounter on real roads. This allows for a more robust and reliable solution compared to previous models based on simulators.

4. Final product specifications

Our project is built from two main parts:

4.1 Autonomous vehicle:

On the vehicle side have an algorithm that receives a video shot from the vehicle's camera while driving. The algorithm automatically cuts the video into frames, recognizes the traffic signs, classifies them and creates a colored mask for our designated signs: stop sign, give way sign, no entry sign, other.

The algorithm then places the masks on top of the frames in the appropriate relevant places and as a final step on this side the algorithm compresses the masked frames into a video that is sent to the remote driver.

4.2 Remote driver:

On the remote driver's side, we have an algorithm that receives the compressed video with the masks, cuts the video into frames, detects the masks and passes each mask through a trained autoencoder. The autoencoder reconstructs the traffic signs from the colored masks. After this step the algorithm puts the reconstructed traffic signs on top of the original video in the appropriate relevant place.

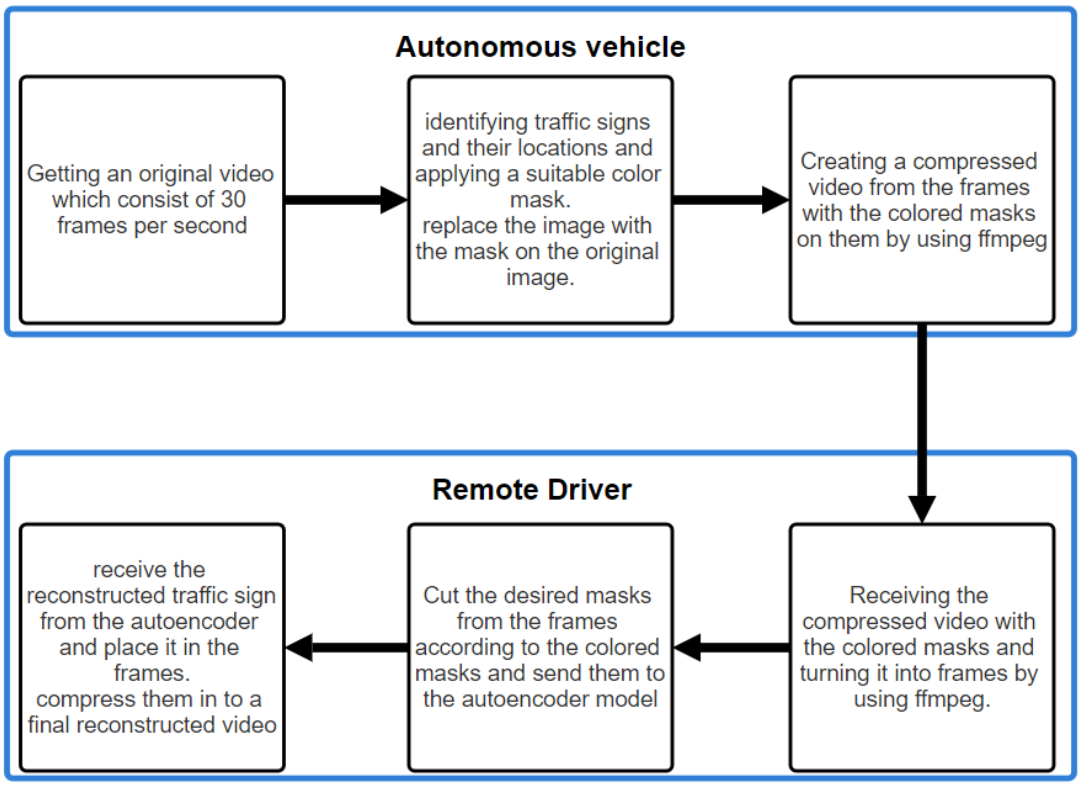

Figure 1: system block diagram

5. Final Product

During the project we created a process of reconstructing traffic signs from a suitable color mask. We used many live videos that were shot in a variety of different environmental conditions and difficult visibility conditions, we classified traffic signs into color masks and then on the decoding side using a trained model that we implemented we were able to reconstruct the appropriate traffic signs.

The process is described in the following diagram:

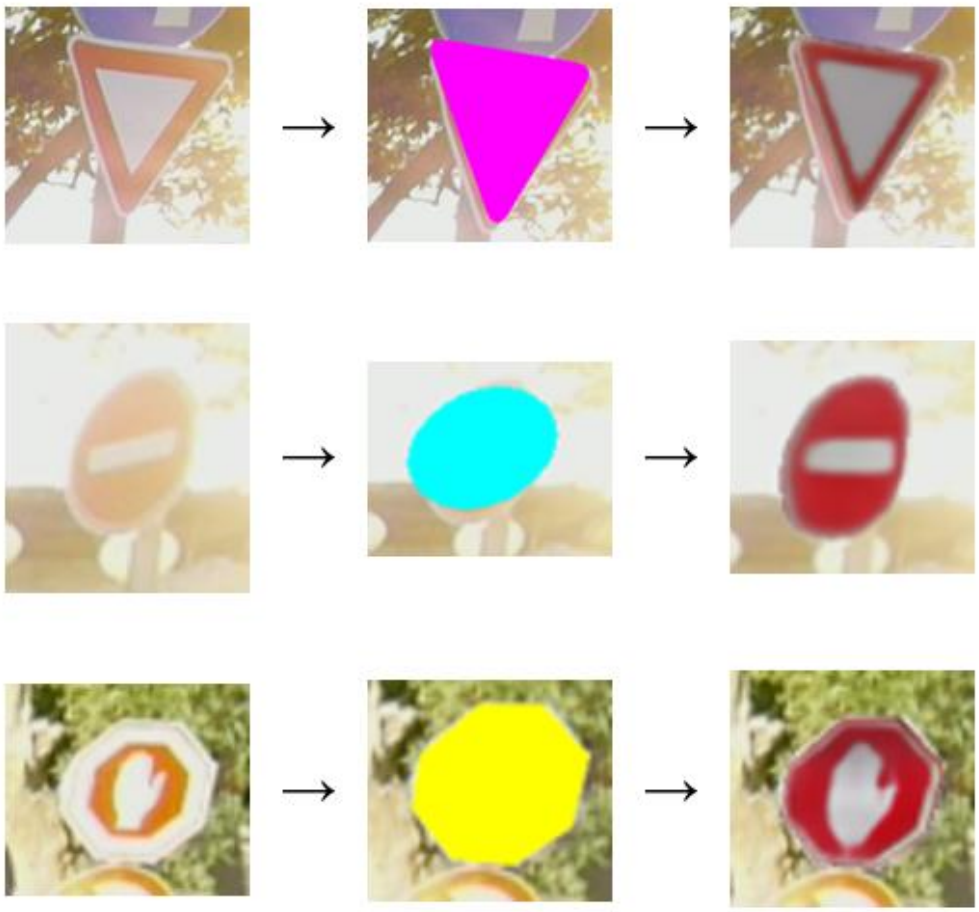

Figure 2: Example of the system on our traffic sign.

6. The Method

6.1 Yolo:

YOLO (You Only Look Once) is a real-time object detection model known for its speed and accuracy, capable of processing an entire image in one pass to predict bounding boxes and class probabilities simultaneously. To enhance detection capabilities and provide a backup, we utilized two different YOLO model APIs: one [1] trained on a large dataset of traffic signs via Roboflow for detecting traffic signs, and another [2] to ensure redundancy and improve overall detection performance.

Figure 3: YOLO detected and label example.

6.2 Classification model:

6.2.1 Data Collection:

To develop a robust traffic sign classification model, we initially leveraged the YOLO object detection model to identify and classify road signs in images. YOLO's preliminary classification categorized the signs into general road sign categories. Following this, we manually reviewed and processed these images, categorizing them into three specific classes: Stop Sign, Give Way Sign, and No Entry Sign. This manual labeling ensured precise classification and significantly enhanced the dataset's quality and the model's performance. Additionally, we enriched our dataset by including images of road signs from other categories sourced from platforms such as Roboflow, Kaggle, and Google Maps. We applied projective transformations to simulate various angles, which further improved the model's robustness and versatility in real-world scenarios.

6.2.2 Architecture and Training:

The development of our traffic sign classification model involved a comprehensive approach encompassing dataset preparation, advanced preprocessing, and rigorous training strategies to achieve optimal performance.

1. Dataset Preparation: The process began with the systematic organization of images into distinct categories—Stop Signs, Give Way Signs, No Entry Signs, and None—using the TrafficSignDatasetPreparation class. This class automated directory creation and ensured balanced representation by splitting the dataset into training, validation, and test subsets.

2. Data Augmentation: To enhance the model's generalization capabilities, we applied a series of transformations to the training dataset. This included resizing images to a standard resolution of 224x224 pixels and applying adjustments to brightness, contrast, saturation, and hue through ColorJitter. Additionally, pixel values were normalized based on precomputed mean and standard deviation values. These augmentations helped the model learn to recognize traffic signs under varying conditions and angles, reducing the risk of overfitting to specific training samples.

3. Model Architecture: The TrafficSignModel class managed the training process, utilizing a Convolutional Neural Network (CNN) architecture defined in the CNNModel class. Below is a visual representation of the architecture:

```
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d( in_channels: 3,  out_channels: 64, kernel_size=6, stride=4, padding=2)
        self.pool = nn.MaxPool2d(kernel_size=3, stride=2)
        self.conv2 = nn.Conv2d( in_channels: 64,  out_channels: 128, kernel_size=4, padding=2)
        self.conv3 = nn.Conv2d( in_channels: 128,  out_channels: 256, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d( in_channels: 256,  out_channels: 512, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(512 * 2 * 2,  out_features: 2048)
        self.fc2 = nn.Linear( in_features: 2048,  out_features: 1024)
        self.fc3 = nn.Linear( in_features: 1024,  out_features: 4)
        self.dropout = nn.Dropout(0.5)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))
        x = self.pool(self.relu(self.conv4(x)))
        x = x.view(-1, x.shape[1] * x.shape[2] * x.shape[3])
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

Figure 4: Architecture of the CNN used for traffic sign classification.

6.2.3 The architecture includes:

- Convolutional Layers: conv1 through conv4 that progressively extract hierarchical features from input images.
- Max-Pooling Operations: These reduce dimensionality and capture essential features.
- Fully Connected Layers: fc1, fc2, and fc3, with dropout and ReLU activation applied to prevent overfitting and introduce non-linearity.

The model was trained using the Adam optimizer with a learning rate of 0.0001 and a Cross-Entropy Loss function. Training was conducted over 20 epochs with early stopping implemented to halt training if validation loss ceased to improve, ensuring efficient learning. The final model demonstrated high accuracy and robust performance on unseen data, validated through detailed evaluation metrics, including accuracy, precision, recall, F1-score, and a confusion matrix.

6.3 Autoencoder Model:

The autoencoder model developed in this project is based on an autoencoder model [3] used for anomaly detection. With modification to this model, we designed a more compatible model suited to our needs, reconstructing traffic signs from their masked images. This model consists of an encoder and a decoder that work together to encode the masked input into a compact latent representation and then decode it back into an image that closely resembles the original traffic sign.

6.3.1 Data collection

To prepare and standardize the dataset for training the autoencoder, we followed a structured process:

Mask Generation:

After classifying and cropping images of Stop Signs, Give Way Signs, and No Entry Signs, we generated colored masks based on the shape and category of each sign, setting the background to black. Masks were created using:

- **Custom Contour Detection Algorithm**: An algorithm to detect contours and accurately color the shapes of the signs.
- **Rembg Tool**: Initially detected and colored signs white, after which colors were adjusted and the background was kept black.

After generating masks with both methods, we combined them by applying a bitwise OR operation to merge the results, resulting in a third mask. All tree masks were evaluated using Intersection over Union (IoU) metrics to determine the most accurate representation. The mask with the highest IoU value was selected as the final mask. This comprehensive approach ensured the most precise and consistent

representation of each sign with a black background, enhancing the dataset's quality and improving the autoencoder's training efficacy.

6.3.2 Architecture

The autoencoder consists of an encoder and a decoder:

Encoder: Compresses the masked image into a lower-dimensional latent representation.

- Compression and Feature Extraction: Uses a series of convolutional layers, followed by ReLU activation, batch normalization, and max pooling. These operations progressively reduce spatial dimensions and extract essential features, producing a compact latent representation.

Decoder: Reconstructs the original traffic sign from the latent representation.

- Expansion: Employs convolutional layers with ReLU activation and batch normalization to upsample the latent representation, increasing the spatial dimensions to match the original size of the traffic sign. The use of F. interpolate helps with upsampling.
- Reconstruction: Additional convolutional layers refine features during upsampling. Skip connections from the encoder are utilized to preserve details. The final layer uses a sigmoid activation function to ensure pixel values are in the appropriate range.

Below are images illustrating the model architecture and layer visualization:

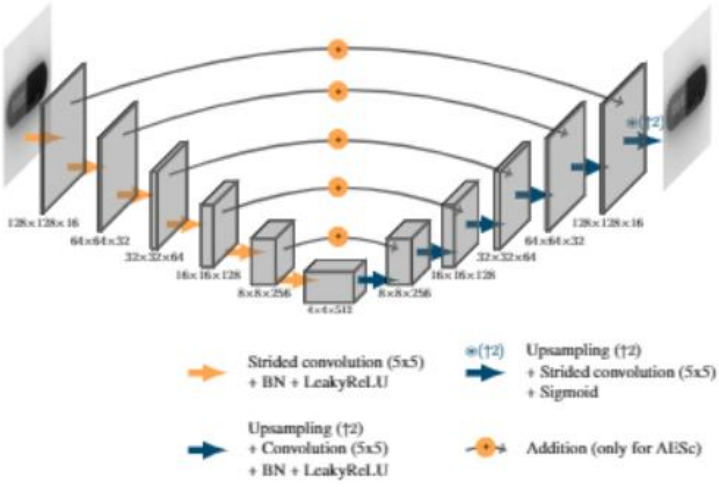

Figure 5: The autoencoder model for anomaly detection used as the base for out model.

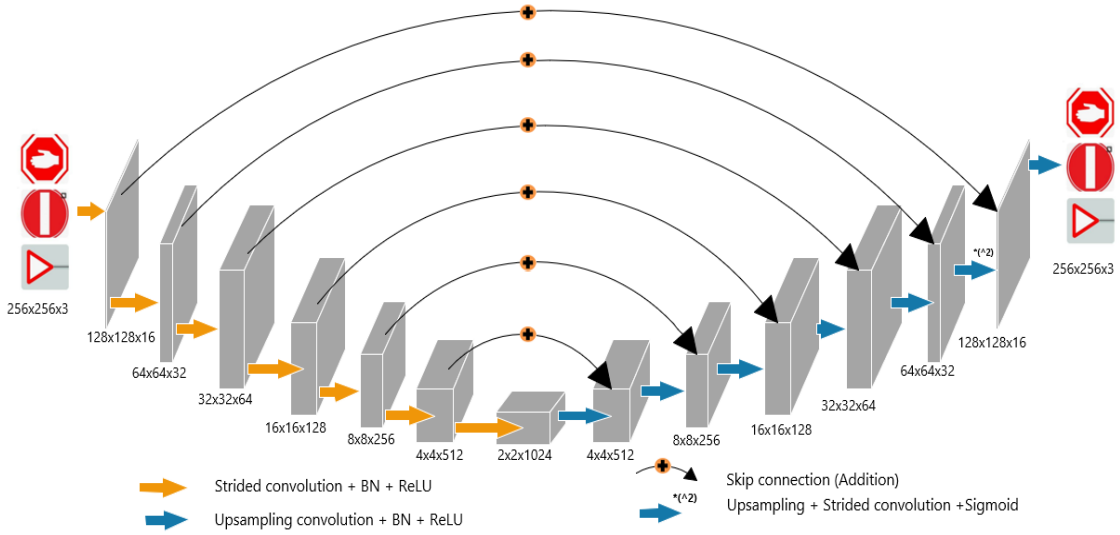

Figure 6: Visualization of each layer in the Autoencoder model.

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        # Encoder
        self.encoder = nn.ModuleList([
            nn.Sequential(nn.Conv2d( in_channels: 3,  out_channels: 16, kernel_size=7, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(16), nn.MaxPool2d( kernel_size: 2, stride=2, padding=0)),   # 256 -> 128
            nn.Sequential(nn.Conv2d( in_channels: 16,  out_channels: 32, kernel_size=7, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(32), nn.MaxPool2d( kernel_size: 2, stride=2, padding=0)),   # 128 -> 64
            nn.Sequential(nn.Conv2d( in_channels: 32,  out_channels: 64, kernel_size=7, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(64), nn.MaxPool2d( kernel_size: 2, stride=2, padding=0)),   # 64 -> 32
            nn.Sequential(nn.Conv2d( in_channels: 64,  out_channels: 128, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(128), nn.MaxPool2d( kernel_size: 2, stride=2, padding=0)),  # 32 -> 16
            nn.Sequential(nn.Conv2d( in_channels: 128,  out_channels: 256, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(256), nn.MaxPool2d( kernel_size: 2, stride=2, padding=0)),  # 16 -> 8
            nn.Sequential(nn.Conv2d( in_channels: 256,  out_channels: 512, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(512), nn.MaxPool2d( kernel_size: 2, stride=2, padding=0)),   # 8 -> 4
            nn.Sequential(nn.Conv2d( in_channels: 512,  out_channels: 1024, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(1024), nn.MaxPool2d( kernel_size: 2, stride=2, padding=0))   # 4 -> 2
        ])
        # Decoder
        self.decoder = nn.ModuleList([
            nn.Sequential(nn.Conv2d( in_channels: 1024,  out_channels: 512, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(512)),  # 2 -> 4
            nn.Sequential(nn.Conv2d( in_channels: 512,  out_channels: 256, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(256)),  # 4 -> 8
            nn.Sequential(nn.Conv2d( in_channels: 256,  out_channels: 128, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(128)),  # 8 -> 16
            nn.Sequential(nn.Conv2d( in_channels: 128,  out_channels: 64, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(64)),  # 16 -> 32
            nn.Sequential(nn.Conv2d( in_channels: 64,  out_channels: 32, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(32)),  # 32 -> 64
            nn.Sequential(nn.Conv2d( in_channels: 32,  out_channels: 16, kernel_size=3, stride=1, padding='same'), nn.ReLU(),
                          nn.BatchNorm2d(16)),  # 64 -> 128
            nn.Sequential(nn.Conv2d( in_channels: 16,  out_channels: 3, kernel_size=2, stride=2), nn.Sigmoid())  # 128 -> 256
        ])
```

Figure 7: Architecture of the Autoencoder used for traffic sign reconstruction.

```python
def forward(self, x):
    skips = []
    for layer in self.encoder:
        x = layer(x)
        skips.append(x)

    skips = skips[::-1][1:]  # Reverse and remove the last element

    for i, layer in enumerate(self.decoder):
        x = F.interpolate(x, scale_factor=2, mode='nearest')  # Use F.interpolate from torch.nn.functional
        x = layer(x)
        if i < len(skips):
            x += skips[i]

    x = F.interpolate(x, scale_factor=2, mode='nearest')  # Use F.interpolate from torch.nn.functional
    return x
```

Figure 8: Skip connection (Additions) that were used on the model.

6.3.3 Training Process:

Initialization:

- Device Setup: Configured to run on GPU if available, otherwise on CPU.
- Model and Loss Functions: Initialized an instance of the Autoencoder class and set up L1 and MSE loss functions to quantify the difference between reconstructed images and original traffic signs.

Hyperparameters:

- Learning Rate and Optimizer: Adam optimizer with a learning rate of 1e-3. A learning rate scheduler, StepLR, adjusts the learning rate every 10 epochs.
- Batch Size and Loss Weights: Batch size set to 32; combined loss function uses a weighted sum of L1 and MSE losses with weights alpha (0.6) and beta (0.4).

Data Preparation:

- Dataset and DataLoader: Loaded masked images and corresponding traffic signs, resized and transformed into tensors. A custom PairedDataset was used for training.
- Data Splitting: Dataset split into training, validation, and test sets with corresponding DataLoader objects for batch processing and shuffling.

Training Loop:

- Epochs and Early Stopping: Trained for up to 100 epochs with early stopping to prevent overfitting, halting training if validation loss does not improve for 50 consecutive epochs.
- Loss Calculation and Backpropagation: Loss computed using both L1 and MSE metrics; optimizer updated model weights to minimize loss.

Validation:

- Evaluation: Model evaluated on the validation set after each epoch. Metrics such as loss and SSIM (Structural Similarity Index) were calculated to assess reconstruction quality.

6.4 Compressor:

FFMPEG is a powerful open-source multimedia framework capable of decoding, encoding, transcoding, muxing, demuxing, streaming, and playing almost anything that humans and machines have created in the realm of video and audio. It is widely used for video processing tasks due to its flexibility, efficiency, and broad support for various formats.

This approach utilizes FFMPEG to efficiently extract frames from a video and create a new video from those frames. The process involves two key functions: one for extracting frames and another for compiling them into a video.

6.4.1 Frame Extraction:

The extract_frames_from_yuv function extracts individual frames from a specified video file and saves them as PNG images in a designated output folder. The steps involved are:

- **Output Folder Creation:** The function checks if the output folder exists, and if not, it creates it.
- **Frame Extraction:** FFMPEG is executed to extract frames from the input video. The frames are saved using a sequential naming pattern (e.g., frame_0.png, frame_1.png, etc.).

```
def extract_frames_from_yuv(video_path, output_folder):
    # Ensure the output folder exists
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Output pattern for the frames
    output_pattern = os.path.join(output_folder, 'frame_%d.png')

    # Run FFMPEG to extract frames from the video
    (
        ffmpeg
        .input(video_path)  # No need for framerate here
        .output(output_pattern)
        .run(cmd=r'D:\ffmpeg-7.0.2-full_build-shared\bin\ffmpeg.exe')  # Full path to ffmpeg.exe
    )
```

Figure 9: Function for extracting frame from YUV video using FFMPEG.

6.4.2 Video Creation:

The create_video_using_ffmpeg function compiles the extracted frames into a new video file. It specifies the frame rate and uses the H.264 codec for efficient compression. The steps include:

- **Input Validation:** The function checks if the folder containing the extracted frames exists.
- **Video Compilation:** FFMPEG is executed to combine the frames into a video file, specifying the codec and pixel format for compatibility with most media players.

```python
def create_video_using_ffmpeg(folder_path, output_file, framerate=30):
    # Ensure the folder exists
    if not os.path.exists(folder_path):
        raise ValueError(f"Folder '{folder_path}' does not exist.")

    # Frame input pattern
    frame_pattern = os.path.join(folder_path, 'frame_%d.png')

    # Run FFMPEG with the full path to ffmpeg.exe
    (
        ffmpeg
        .input(frame_pattern, framerate=framerate)
        .output(output_file, vcodec='libx264', pix_fmt='yuv420p')
        .run(cmd=r'D:\ffmpeg-7.0.2-full_build-shared\bin\ffmpeg.exe')  # Full path to ffmpeg.exe
    )
```

Figure 10: Function for creating YUV video from frames using FFMPEG.

6.4.3 Key Advantages of Using FFMPEG:

- **Efficiency:** FFMPEG directly handles both the extraction and compilation processes, streamlining the workflow.
- **Quality Control:** By saving frames as PNG images, the highest quality is maintained during extraction.
- **Compatibility:** The use of the H.264 codec and YUV420p pixel format ensures the output video is widely compatible with various playback systems and platforms.

7. Models Results

7.1 Classification model:

Test set – the test set consisted of a total of 6,302 images. The categories count was: 1,234 stops sign images, 1,227 no entry images, 1,207 give way images and 2,635 none images.

Confusion Matrix Overview - The confusion matrix is a tool to evaluate the performance of the classification model by comparing the actual labels to the predicted labels for a multi-class classification problem. Each row of the matrix represents the instances in a true class, while each column represents the instances in a predicted class.

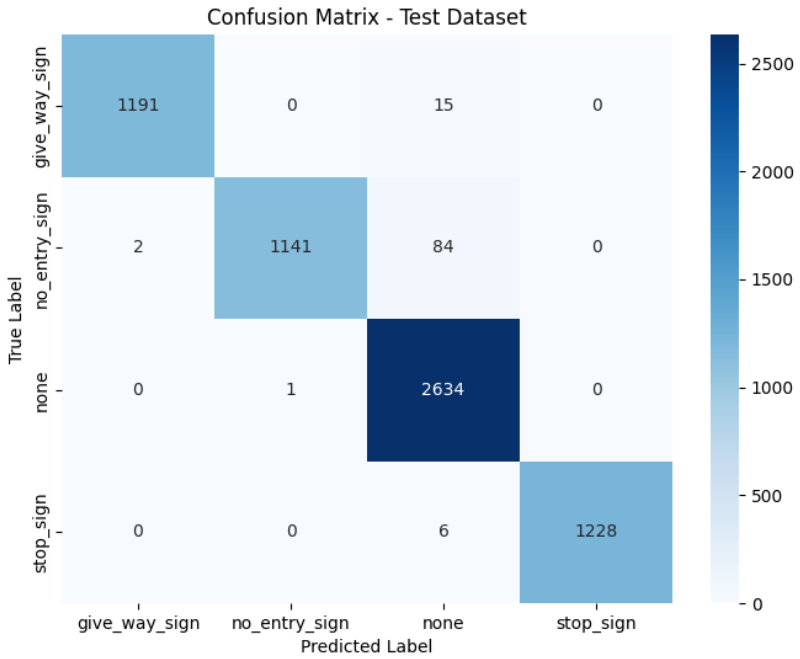

Figure 11: Confusin matrix of our classification model on test set.

Overall Performance: The confusion matrix demonstrates that the road sign classification model performs exceptionally well across all categories. The overall accuracy of the model is approximately 98.3% (6194 correct predictions out of 6302 total predictions). This high accuracy indicates that the model is highly reliable in classifying road signs.

Class-specific Performance:

a) Give Way Sign:

- Precision: 100% $(1191/(1191 + 0 + 0 + 0))$
- Recall: 98.75% $(1191/(1191 + 15))$

- The model perfectly identifies give way signs when it predicts them, but occasionally misses some, classifying them as "none".

b) No Entry Sign:

- Precision: 92.99% $(1141/(1141 + 84 + 1))$
- Recall: 99.82% $(1141/(1141 + 2))$
- The model rarely misclassifies other signs as no entry, but it sometimes fails to identify no entry signs, mostly classifying them as "none".

c) None (No Sign):

- Precision: 99.74% $(2634/(2634 + 1 + 6))$
- Recall: 100% $(2634/2634)$
- The model excels at identifying when no sign is present, with perfect recall and very high precision.

d) Stop Sign:

- Precision: 99.51% $(1228/(1228 + 6))$
- Recall: 100% $(1228/1228)$
- The model identifies all stop signs correctly and rarely misclassifies other signs as stop signs.

e) Misclassification Patterns:

- The most common error is misclassifying signs as "none", particularly for no entry signs (84 instances) and give way signs (15 instances).
- There are very few misclassifications between different types of signs, with only 2 no entry signs misclassified as give way signs.

Conclusion: The road sign classification model demonstrates excellent performance with high accuracy, precision, and recall across all classes. Its strongest performance is in correctly identifying the presence of signs, with slight room for improvement in distinguishing between signs and no signs in some cases. The model's high reliability makes it a valuable tool for automated road sign recognition, with potential applications in driver assistance systems and autonomous vehicles.

7.2 Autoencoder Model:

7.2.1 Performance during training:

1. Autoencoder Loss Graph (figure 12):

This graph shows how well the model is learning to reconstruct images over time.

- Blue line (Training Loss): This shows the error on the training data. It starts high but quickly drops and continues to decrease slowly, showing the model is improving its reconstruction ability on the training set.
- Orange line (Validation Loss): This represents the error on data the model hasn't seen during training. It's jumpier at first but eventually stabilizes, staying higher than the training loss.

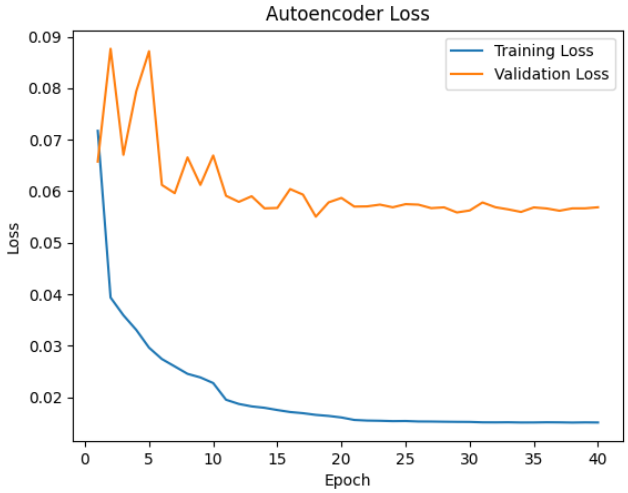

Figure 12: The autoencoder train and validation loss.

2. Autoencoder SSIM Graph (figure 13):

This graph shows how similar the reconstructed images are to the original ones. Higher values mean better reconstruction.

- Blue line (Training SSIM): This shows similarity for training images. It starts around 0.75 and improves to about 0.90, with some ups and downs. This indicates the model is getting better at reconstructing training images over time.
- Orange line (Validation SSIM): This shows similarity for validation images. It starts lower but improves to about 0.75 and stays steady.

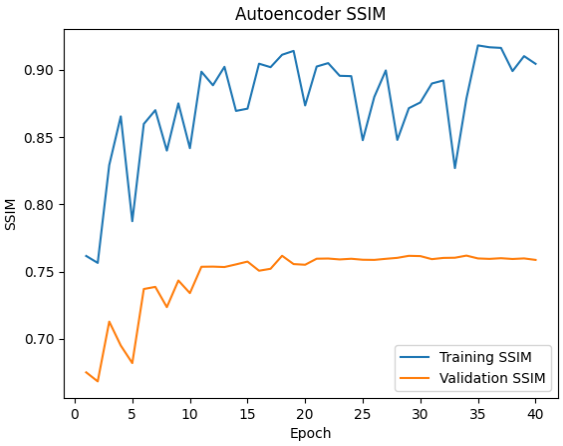

Figure 13: The autoencoder SSIM train and validation.

7.2.2 Visualization of reconstruction improvement during training:

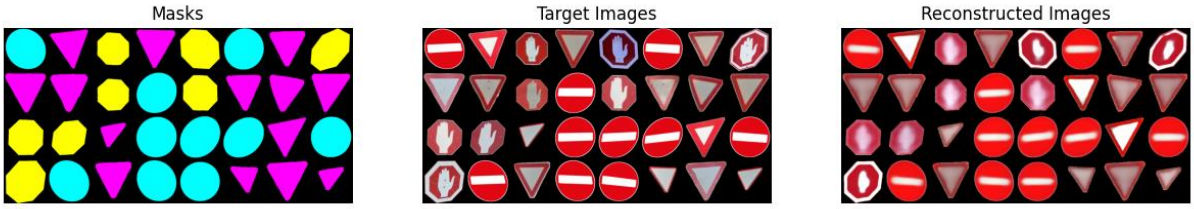

Figure 14: Input masks for the autoencoder, target images and the output reconstructed images after the first training epoch.

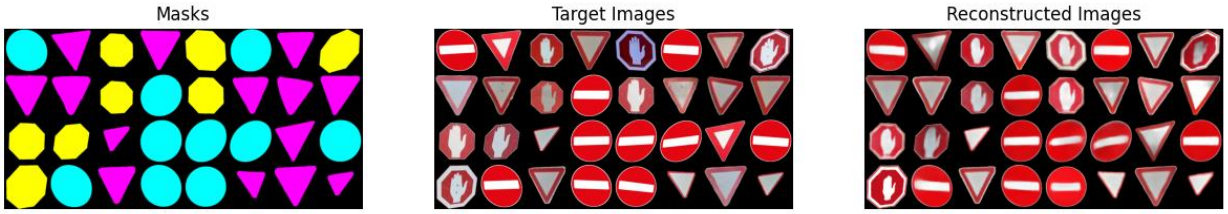

Figure 15: Input masks for the autoencoder, target images and the output reconstructed images after the last training epoch.

7.2.3 Performance of autoencoder on test set:

SSIM (Structural Similarity Index): The SSIM score of 0.7396 reflects a moderate level of similarity between the original and reconstructed images. SSIM is a measure that evaluates the perceived quality of the images based on luminance, contrast, and structure. A score in this range indicates that while the autoencoder effectively preserves some structural details of the images, there is still room for improvement. The goal is to increase the SSIM score closer to 1.0, which represents perfect similarity.

MSE (Mean Squared Error): The MSE value of 0.0269 signifies that the average squared difference between the original and reconstructed images is relatively low. This suggests that the autoencoder is performing well in minimizing pixel-wise differences, resulting in a good reconstruction quality overall. Lower MSE values generally correspond to better reconstruction performance, but the SSIM provides additional insight into the perceptual quality.

In summary, the autoencoder demonstrates a solid ability to reconstruct images with a reasonable level of detail preservation and accuracy. However, further refinement of the model could enhance these metrics, potentially improving both the structural similarity and overall reconstruction quality.

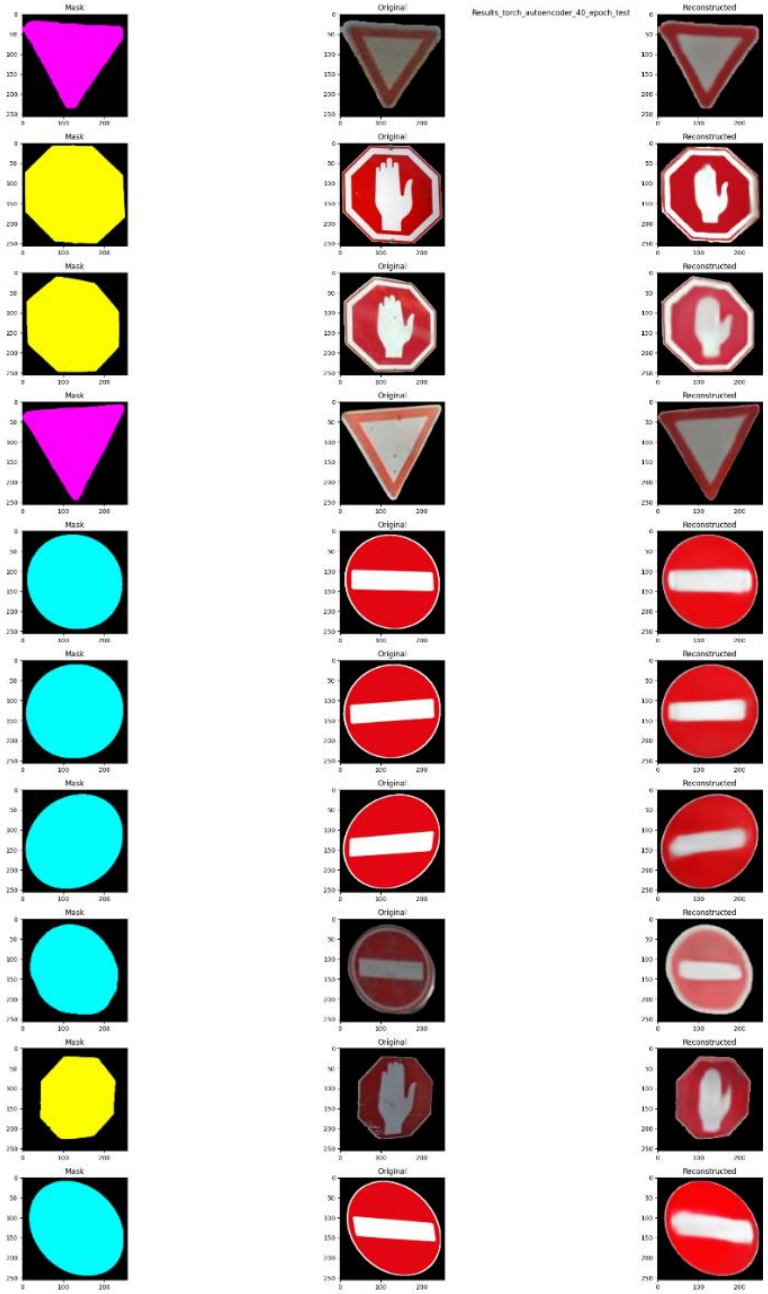

Figure 16: Autoencoder performance on test set.

8. Final Results

In this simulation, some traffic signs are obscured due to sun glare. The project workflow involved detecting the traffic signs, generating matching masks, replacing the obscured signs with these masks, and then accurately reconstructing the original traffic signs using our model.

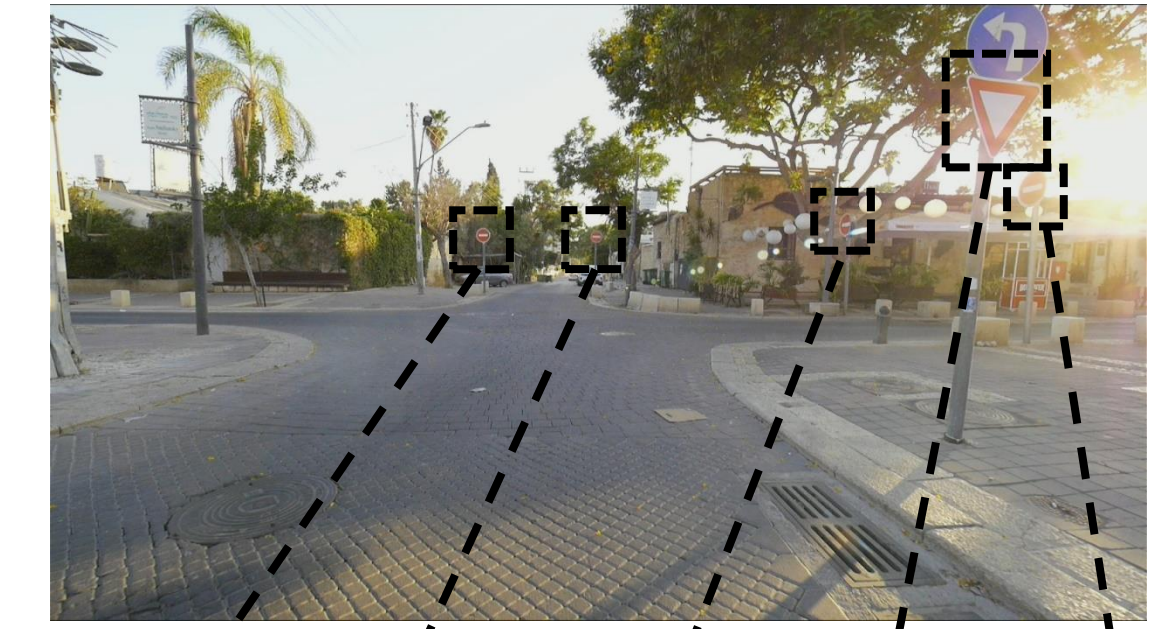

Figure 17: Pre-Processing Real-time image.

Figure 18: Detected and cropped the traffic signs from the input image.

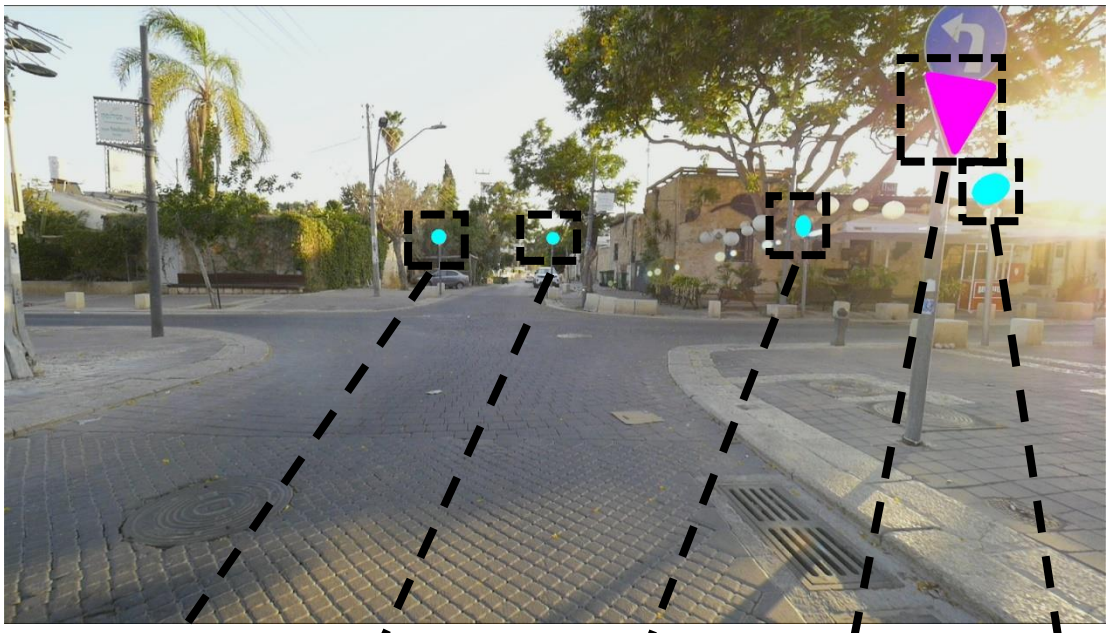

Figure 19: Image with the colored mask before the autoencoder.

Figure 20: For each traffic sign created a coresponding colored mask and replaced the sign with it's mask.

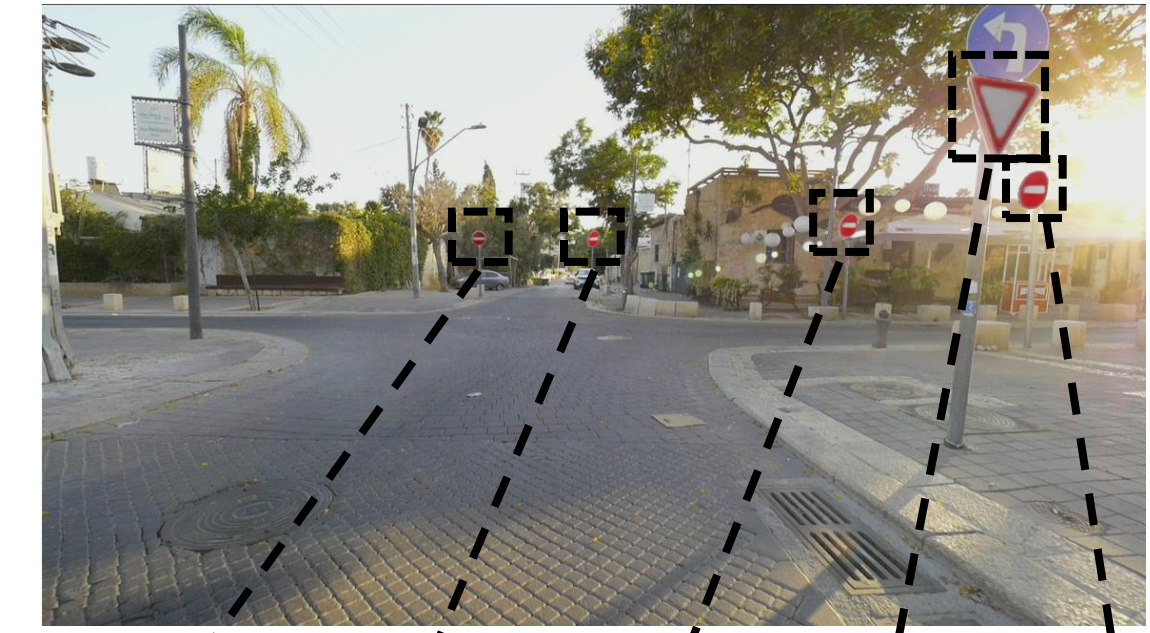

Figure 21: Reconstruct image after the autoencoder.

Figure 22: Reconstructed the traffic sign from each mask and replaced the masks with the reconstructed images.

8.1 H.265 Compressed Video Frame vs. Reconstructed Traffic Sign Image:

Figure 23: Video frame after H.265 compressed with 2Mbit bit rate.

H.265 Compressed Video Frame:

- **Loss of Quality Due to H.265 Compression (2 Mbit)**: The original image is a frame from a video compressed using the H.265 codec at a 2 Mbit rate. While H.265 is efficient for reducing file sizes, it introduces visual artifacts that degrade image quality, particularly when using lower bit rates. In this case, the compression causes a loss of fine details and reduces the sharpness of the frame, which is evident in the traffic signs.
- **Impaired Traffic Sign Visibility**: The compression significantly impacts the readability of traffic signs, especially those affected by sunlight or situated farther away. Key signs, such as the "yield" and "no entry" signs, appear blurry and lack clear borders due to both the compression artifacts and environmental factors like glare.
- **Artifacts and Noise**: Compression introduces blocking and noise around high-detail areas, such as the edges of the traffic signs. This distortion obscures important information and makes it difficult for both human drivers and computer vision systems to accurately interpret the scene.

Reconstructed Traffic Signs (Using Autoencoder):

- **Targeted Reconstruction of Traffic Signs**: The autoencoder focuses specifically on reconstructing traffic signs, rather than the entire image. The rest of the frame remains as it was in the compressed video, while the traffic signs are enhanced and restored to near-original clarity.

- **Restored Sharpness and Visibility**: The autoencoder successfully improves the clarity and sharpness of the traffic signs, which were previously blurred by compression. Signs like the "Give way" and "no entry" symbols are reconstructed with precise edges, clear borders, and accurate colors, making them easily readable, even from a distance or in the presence of glare.
- **Preserved Environmental Context**: While the traffic signs are reconstructed to improve their visibility, the rest of the scene is left unchanged, retaining the characteristics and compression artifacts of the original H.265 frame. This ensures that the reconstructed signs fit naturally within the existing environment without altering other elements of the frame.

Here is a visual comparison:

Figure 24: Reconstructed signs.

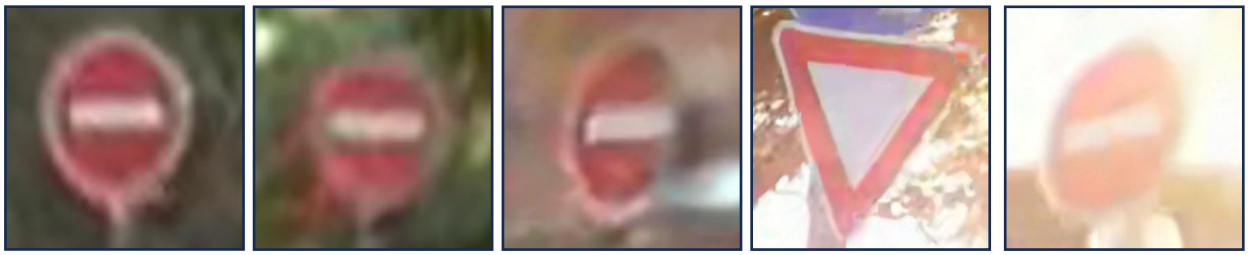

Figure 25: Compressed signs.

This selective reconstruction improves the reliability of the traffic signs, making them easier for drivers and computer vision systems to interpret accurately, despite the overall compression of the video.

9. Challenges and Solutions

In the previous project, Cognata was used to create short simulation videos that included various angles and different weather conditions as a substitute for real-life footage. Cognata also automatically generated a segmented video matching the simulation. In our project, however, we did not have access to Cognata, so we had to film our own videos, gather diverse data, and develop our own segmentation method.

Here are several challenges we encountered:

- **Replacing Cognata with Cityscapes Semantic Segmentation**:

Initially, we tried using Cityscapes for semantic segmentation, but the results were unsuitable for our needs. The segmentation lacked accuracy, made frequent errors, and could not properly handle multiple signs in close proximity.

For example:

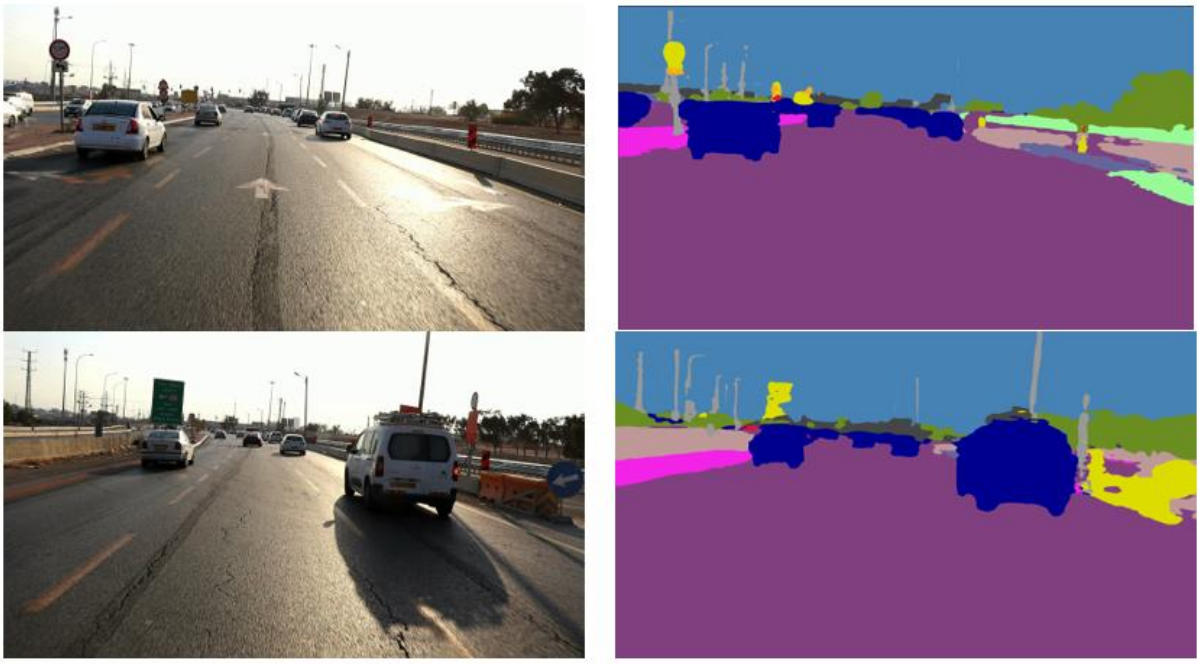

Figure 26: Semantic segmentation using Cityscapes.

- **Difficulty in Distinguishing Specific Traffic Signs with YOLO**:

After using YOLO, we faced challenges extracting three specific traffic signs from live images because YOLO labeled all traffic signs with one general category and did not distinguish between them. Our solution was to train a classification model. Initially, we struggled to achieve high accuracy and experienced several misclassifications, such as:

Figure 27: A street sign classified incorrectly as a "give way" sign.

To improve accuracy, we identified the specific signs the model misclassified and incorporated more variations of these signs into the training dataset.

- **Creating Accurate Colored Masks**:
  For generating color masks, we wrote a custom script, but it encountered issues with smaller signs in live images (approximately 50x50 pixels), making it difficult to create accurate masks. To address this, we applied several methods for improving mask creation.
  For example, the masks we got at first:

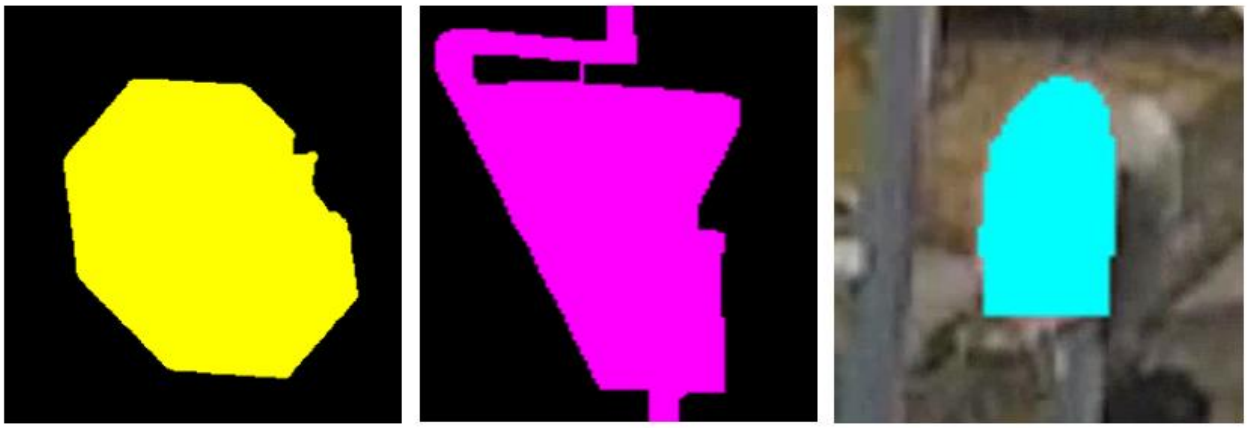

Figure 28: Example illustrating the mask creation problem.

- **Size Mismatch Between Reconstructed Signs and Masks**:
  In some instances, the reconstructed traffic signs were smaller than the original masks, leaving a visible outline when the sign was placed back into the image.
  For example:

Figure 29: Reconstructed sign placed onto the original image with a visible outline of the mask.

To fix this issue we applied techniques to clean up the mask, remove noise and close gaps, and reconstructed traffic sign to perfectly match the size of the original mask ensuring the mask was more accurate and smooth.

10. Conclusions and future work

10.1 Conclusions

- Our algorithm successfully processes live video frames, effectively detecting, classifying, and generating appropriate color masks for the relevant traffic signs in each frame.

- The autoencoder consistently and accurately reconstructs traffic signs from their masks under various environmental conditions and from a wide range of camera angles.

- As a result of the project, we successfully generated a compressed video that displays the reconstructed traffic signs with the correct orientation and tilt.

10.2 Future work

- We would like to train a YOLO model to detect and label traffic sign types and not use one general category for all traffic signs. This step will eliminate the need of using YOLO and then classify each result with a classification model. It will make the use of our whole algorithm simpler and have less room for mistakes.

- We would like to extend the validity of the model from three specific traffic signs to a larger variety of traffic signs.

- We would like to find a way to improve the quality of the reconstructed traffic sign.

- We would like to improve the mask creation process, make it simpler and more accurate.

10.3 Budget

Description	Total Working Hours	Hourly wage (NIS)	Total Cost (NIS)
Manpower			
Student Salary	800 (per student)	80	64,000
Adviser	80	400	32,000
Total Salary			160,000

11. References

[1] Universe-Data-Test Computer Vision Project by Spinview posted on Roboflow.

[2] Logistics Computer Vision Project by Large Benchmark Datasets posted on Roboflow.

[3] A. Collin and C. De Vleeschouwer, "Improved anomaly detection by training an autoencoder with skip connections on images corrupted with Stain-shaped noise," in 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 2021 pp. 7915-7922

12. Final Grade Recommendation Form

המלצות ציון (ע"י מנחה אקדמי) לדו"ח מסכם

אם יש צורך, לכל סטודנט/ית בנפרד

מספר הפרויקט: P-2024-080

הפרויקט: כיסויי תמרורי תנועה מבוססים למידה עמוקה באמצעות מערך נתונים מלאכותי

שם המנחה מהמחלקה: פרופ' הדר עופר, מר דרור איתי

שם הסטודנט/ית: עדי שרטו ת.ז: 318277514

שם הסטודנט/ית: אלון צלוביץ ת.ז: 208473629

אחוז %	קריטריון	חלש 55-64	בינוני 65-74	טוב 75-84	ט"מ 85-94	מצוין 95-100
20	הצגת הגישה והתכנון ההנדסי					
20	הצגת התוצאות וניתוח השגיאות					
20	הסקת מסקנות					
10	גילוי יוזמה וחריצות					
20	פתרון בעיות, מקוריות ותרומה אישית (מעבר למילוי ההנחיות)					
10	עמידה בלו"ז ורמת הביצוע המעשי					

אם יש כוונה לפרסם/ יפורסם מאמר, שם כתב העת ומועד משוער להגשה:

ציין אם יש כוונה לשקול המלצה כפרויקט מצטיין:

הערות נוספות: