Motion Detection Algorithm Evaluation Report

Giovanni Beraldi

May 20, 2024

Contents

1 Introduction **2**

2 Methodology **2**
2.1 Datasets . . . 2
2.2 Traditional Approaches . . . 2
2.2.1 Frame Differencing . . . 2
Reference Frame Differencing (RFD) . . . 3
Adjacent Frame Differencing (AFD) . . . 3
Median Frame Differencing (MFD) . . . 3
Median Frame Differencing with Morphology (MFD-M) . . . 3
2.2.2 Mixture of Gaussian . . . 4
2.3 Deep Learning Approaches . . . 4
2.3.1 TransCD: Scene Change Detection via Transformer-based Architecture . . . 4
2.3.2 MU-Net1: Single-stream Spatial-only Detection Using Semantic Segmentation . . . 5

3 Results **7**
3.1 Evaluation Procedure . . . 7
3.2 Qualitative Evaluation . . . 7
3.2.1 Reference Frame Differencing results . . . 7
3.2.2 Adjacent Frame Differencing results . . . 8
3.2.3 Median Frame Differencing results . . . 8
3.2.4 Median Frame Differencing with Morphology results . . . 8
3.2.5 Mixture of Gaussian results . . . 9
3.2.6 TransCD model results . . . 9
3.2.7 MU-Net1 model results . . . 9

4 Comparison of Algorithms **9**
4.1 Qualitative Comparison . . . 9
4.2 Discussion . . . 10

5 Conclusion **11**

1 Introduction

Motion detection is a fundamental task in computer vision, with applications ranging from surveillance to traffic monitoring. Traditional methods for motion detection often rely on background subtraction techniques, while deep learning approaches offer promising results by learning features directly from data. In this report, we evaluate the performance of various motion detection algorithms on real-world datasets to understand their strengths and limitations.

2 Methodology

2.1 Datasets

Two datasets were used for evaluation:

1. Change Detection 2014 (CD2014)
2. Scene Background Modeling (SBMnet)

Both datasets are widely used benchmarks in the field of computer vision, specifically in the area of background subtraction and motion detection algorithms. The datasets consist of several video sequences captured in different environments, such as indoor scenes, outdoor scenes, and scenes with varying lighting conditions and levels of clutter and occlusion. Each video sequence is accompanied by ground truth annotations that label pixels or regions in the frame as either foreground (moving objects) or background.

2.2 Traditional Approaches

Traditional algorithms for background subtraction typically involve techniques that rely on handcrafted features and predefined algorithms. These methods often utilize mathematical operations and heuristic rules to detect changes between consecutive frames in a video sequence.

2.2.1 Frame Differencing

Frame differencing is a technique where two frames are subtracted from each other to detect changes in pixel values. Pixels with significant differences between frames are classified as foreground, indicating potential motion. The `get_mask` function provides a simple yet effective implementation of frame differencing for motion detection. The function takes two input frames, `frame1` and `frame2`, and returns a binary mask highlighting the areas of motion between them.

```python
def get_mask(frame1, frame2, adaptive=True):
    # Convert to grayscale
    frame1 = cv2.cvtColor(frame1, cv2.COLOR_RGB2GRAY)
    frame2 = cv2.cvtColor(frame2, cv2.COLOR_RGB2GRAY)

    # Compute the absolute difference between the two frames
    frame_diff = cv2.absdiff(frame2, frame1)

    # Apply a median blur with a 3x3 kernel to the frame difference
    frame_diff = cv2.medianBlur(frame_diff, 3)

    # Apply a simple threshold to create a binary mask of moving pixels
    if adaptive:
        mask = cv2.adaptiveThreshold(frame_diff, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 5, 3)
    else:
        _, mask = cv2.threshold(frame_diff, 30, 255, cv2.THRESH_BINARY)

    return mask
```

- **Frame Comparison**: The function begins by converting the input frames to grayscale. It then computes the absolute difference between the two frames, emphasizing areas where motion has occurred.

- **Post-processing**: To reduce noise and smooth the image, a median blur with a 3x3 kernel is applied to the frame difference. This helps to reduce noise and smooth out the image, potentially eliminating isolated fluctuations
- **Thresholding**: Finally, a thresholding operation is performed to create a binary mask, where pixels above a certain intensity threshold are considered part of the moving object (foregrounds). The ‘adaptive‘ parameter allows for adaptive thresholding, where the threshold value is dynamically adjusted based on the local pixel neighborhood. This can be particularly useful in varying lighting conditions or scenes with different levels of motion.

Reference Frame Differencing (RFD) RFD algorithm compare each frame in a video sequence to a **reference frame**, typically the initial frame, to detect changes indicative of moving objects. The algorithm begins by selecting a reference frame from the input video sequence. This reference frame represents the initial state of the scene or a background model. For this project, the first frame of the video sequence is always selected as the reference frame. For each incoming frame in the video sequence, the algorithm uses the `get_mask` to computes the absolute difference between pixel intensities in the current frame and corresponding pixels in the reference frame and obtain a binary motion mask.

```
    reference_frame = cv2.imread(frames[0])
    for idx in range(1, len(frames)):
        frame = cv2.imread(frames[idx])
        mask = get_mask(reference_frame, frame)
```

Adjacent Frame Differencing (AFD) AFD involves subtracting the pixel values of one frame t from the corresponding pixel values of the next frame $t + 1$. For each frames pair, the algorithm uses the `get_mask` to computes the absolute difference between pixel intensities in the first frame and corresponding pixels in the second frame and obtain a binary motion mask.

```
    for idx in range(1, len(frames)):
        frame1 = cv2.imread(frames[idx - 1])
        frame2 = cv2.imread(frames[idx])
        mask = get_mask(frame1, frame2)
```

Median Frame Differencing (MFD) Instead of directly subtracting one frame from another, MFD involves taking the median pixel value across a set of consecutive frames. This median operation helps to reduce the impact of noise and outliers that might be present in individual frames. For this project, a rolling window of the last 5 frames is used for the median operation. For each incoming frame in the video sequence, the algorithm uses the `get_mask` to computes the absolute difference between pixel intensities in the current frame and corresponding pixels in the median frame and obtain a binary motion mask.

```
buffer_size = 5
frame_buffer = deque(maxlen=buffer_size)
for idx in range(0, len(frames_path)):
    frame = cv2.imread(frames_path[idx])
    frame_buffer.append(frame)
    median_frame = np.median(np.array(frame_buffer), axis=0).astype(np.uint8)
    mask = get_mask(median_frame, frame)
```

Median Frame Differencing with Morphology (MFD-M) This method enhances MFD by applying morphological operations to refine the motion mask. For this project, `cv2.MORPH_CLOSE` is used: this is a combination of **dilation** followed by **erosion** and it is useful for closing small holes or gaps in foreground objects and smoothing the boundaries of objects.

```
buffer_size = 5
kernel = np.ones((3, 3), np.uint8)
frame_buffer = deque(maxlen=buffer_size)
for idx in range(0, len(frames_path)):
    frame = cv2.imread(frames_path[idx])
    frame_buffer.append(frame)
    median_frame = np.median(np.array(frame_buffer), axis=0).astype(np.uint8)
    mask = get_mask(median_frame, frame)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
```

2.2.2 Mixture of Gaussian

Mixture of Gaussians (MoG) for background subtraction is a popular technique to separate foreground objects from the background in a video sequence.

- **Initialization**: Initially, the algorithm estimates the background model using a mixture of Gaussian distributions. Each pixel in the background model is represented by a set of Gaussian distributions, which describe the variability in pixel intensities over time. For this project, the `OpenCV's cv2.createBackgroundSubtractorMOG2` function it used.
- **Adaptation**: As new frames of the video sequence are observed, the background model is continuously updated to adapt to changes in the scene. For each pixel, the algorithm updates the parameters of the Gaussian distributions (mean, variance, and weight) based on the observed pixel values in the recent frames. For this project, the algorithm use information from the last `200` frames to update the background model.
- **Foreground Detection**: Once the background model is established, foreground objects can be detected by comparing the pixel values in each frame with the corresponding Gaussian distributions in the background model. Pixels that significantly deviate from the background model are classified as foreground pixels. For this project, the threshold value for the **Mahalanobis distance** was has been set to `50`.
- **Post-processing**: Additional post-processing steps may be applied to the binary foreground mask to improve the segmentation and remove artifacts. For this project, `cv2.MORPH_OPEN` is used: this is a combination of erosion followed by dilation. It is useful for removing noise and small objects from the foreground regions while preserving the overall structure and connectivity of larger objects.

```
kernel = np.ones((2, 2), np.uint8)
background_model = cv2.createBackgroundSubtractorMOG2(history=200, varThreshold=50)
for idx in range(0, len(frames_path)):
    frame = cv2.imread(frames_path[idx])
    mask = background_model.apply(frame)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
```

2.3 Deep Learning Approaches

2.3.1 TransCD: Scene Change Detection via Transformer-based Architecture

The first deep learning model used for motion detection is TransCD [2], a transformer-based scene change detection model. TransCD aims to identify changes of interest between bi-temporal images acquired at different times. The architecture of TransCD consists of three main components: the CNN Backbone, the Siamese Vision Transformer (SViT), and the Prediction Head. The network was trained on the Change Detection 2014 dataset.

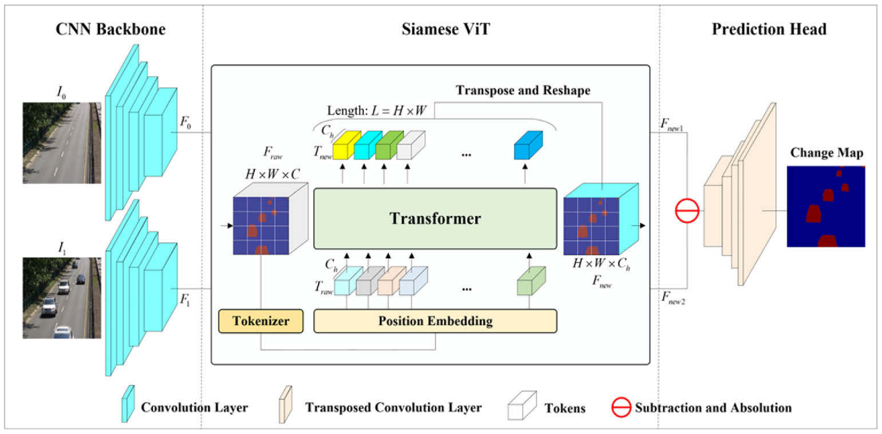


Figure 1: TransCD model architecture

The network input, processing steps, and output can be summarized as follows:

1. **Network input**: The network takes two bi-temporal images, I_0 and I_1, acquired at different times t_0 and t_1 as input. Each image has a spatial resolution of $H_I \times W_I$ with $CI = 3$ channels representing RGB color information.

2. **Processing steps**:

   - **CNN Backbone**: The input images are processed by a CNN backbone to extract high-level features and produce patch embeddings. The CNN backbone functions as a Siamese network with shared weights, mapping the images into high-level feature representations.
   - **Siamese Vision Transformer (SViT)**: The extracted features are then fed into the SViT component, formed by two weight-shared ViT. The SViT establishes global relations, captures long-range dependencies over embedded tokens, and outputs context-rich tokens that enhance the feature representation and model both spatial and temporal relations.
   - **Prediction Head**: The context-rich tokens from the SViT are used as input to the prediction head, which generates a binary change map. The prediction head calculates the feature difference between pairs of context-rich tokens corresponding to the same spatial location in the input images, identifying areas where significant changes have occurred between the two input images. After computing the feature differences, the prediction head maps the high-level feature representations back to the pixel space using a light CNN decoder.

3. **Network output**: The output of the TransCD network is a binary change map with the same spatial resolution as the input images ($H_I \times W_I$). This change map indicates the areas of interest where scene changes have been detected.

For this project the network was used to generate a binary mask for a video sequence, using an approach similar to **Reference Frame Differencing**. The first frame of the video serves as the reference image I_0, while the network analyzes the differences between this reference frame and the successive frames of the video I_1 to identify and visualize scene changes. By comparing each frame with the initial frame, the network detects variations and generates a binary mask that highlights the regions of the video where significant changes have occurred.

2.3.2 MU-Net1: Single-stream Spatial-only Detection Using Semantic Segmentation

The second deep learning model used for motion detection is an image segmentation model. **Image segmentation** is a fundamental task in computer vision and deep learning that involves partitioning an image into multiple segments or regions based on certain characteristics such as color, texture, or intensity. The goal is to accurately delineate the boundaries of objects or regions of interest within the image. In deep learning, image segmentation tasks typically involve training a model to classify each pixel in an image into one of several predefined classes or categories. This is known as pixel-wise semantic segmentation, where the model assigns a label to every pixel in the input image, effectively dividing the image into semantically meaningful regions.

The U-Net architecture is particularly useful for image segmentation. This architecture is a convolutional neural network (CNN) architecture consisting of an encoder-decoder structure with skip connections between corresponding encoder and decoder layers. This design allows for the preservation of spatial information during the upsampling process in the decoder, enabling precise segmentation outputs(concatenating the output from the encoder to the corresponding layers in the decoder). The final layer typically consists of a 1x1 convolution followed by a softmax activation function that produces a segmentation map.

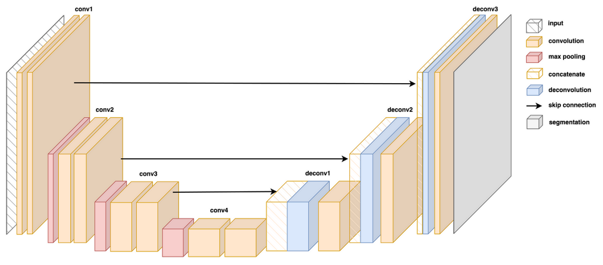


Figure 2: U-Net model architecture

MU-Net1 [1] adopts a U-Net-like architecture for moving object detection, where the encoder module extracts features from the input RGB frames, and the decoder module reconstructs the segmentation masks for detecting moving objects. The skip connections in MU-Net1 help in retaining important spatial details and features throughout the network, contributing to the accurate detection of moving objects in video sequences. MU-Net1 integrates the ResNet-18 backbone architecture to enhance its feature extraction capabilities for moving object detection. The network was trained on the Change Detection 2014 dataset.

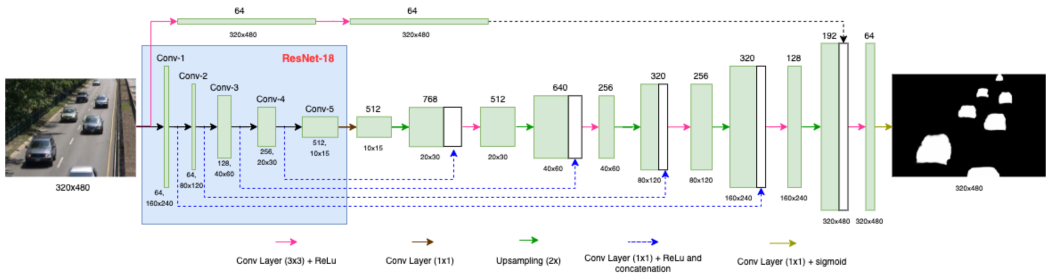


Figure 3: MU-Net1 model architecture

Here is an overview of how MU-Net1 works:

1. **Input**: The input to MU-Net1 consists of RGB frames representing appearance cues in the video sequence.

2. **Encoder**: The encoder part of the network processes the input RGB frames to extract spatial appearance features.

3. **Decoder**: The decoder part of the network reconstructs the spatial appearance features extracted by the encoder to generate segmentation masks for moving object detection.

4. **Output**: The output of MU-Net1 is a segmentation mask that highlights the moving objects in the video.

While effective, object segmentation that rely only on appearance cues suffer from three main problems:

- They cannot detect untrained (moving) objects.
- They fail when appearance cues are limited (e.g. when objects are far from camera).
- They cannot differentiate between moving and stationary objects: the algorithm may segment both moving (foreground) and stationary (background) objects in a scene without distinguishing between them.

For this project, the MU-Net1 network was used to generate a binary mask for a video sequence. The network takes individual frames of the video as input and processes them to produce a binary mask highlighting the moving objects in each frame.

3 Results

3.1 Evaluation Procedure

In this section, we present the results of the motion detection algorithms applied to two benchmark datasets: the CD2014 dataset and the SBM dataset. For evaluation, some videos was randomly selected from each dataset. For each frame of the selected videos, the binary motion mask was obtained by applying the respective algorithm. Using the motion mask, the contours of the moving objects was detected and the bounding boxes drew around them. The bounding box detection script included the following steps:

1. Detect contours of moving objects.
2. Remove bounding boxes contained within larger ones.
3. Apply non-maximum suppression to refine the bounding boxes.
4. Merge bounding boxes that are very close to each other.

Result videos was generated for each algorithm. Each frame of the result video displays the binary motion mask alongside the original frame with the detected moving objects highlighted by bounding boxes.

3.2 Qualitative Evaluation

Qualitative evaluation was performed by visually inspecting the result videos generated by each algorithm. Sample frames from the result videos are shown for each algorithms in the following tables.

For traditional algorithms, the binary mask was obtained using the adaptive threshold.

3.2.1 Reference Frame Differencing results

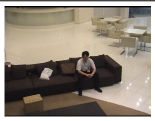
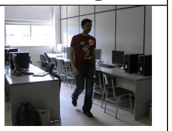
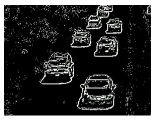
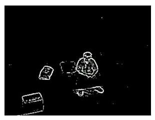
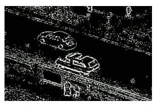
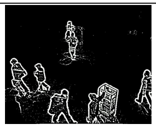
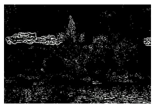
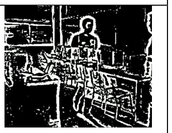
Description	Highway	Sofa	Boulevard	Crowd	Fountain	Illumination change
Test Image						
Binary Mask						

Table 1: Reference Frame Differencing results

3.2.2 Adjacent Frame Differencing results

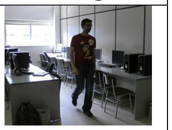
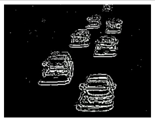
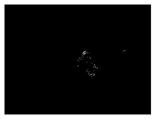
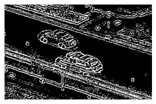
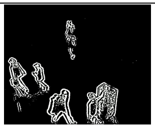
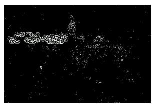
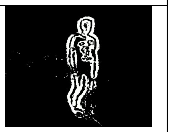
Description	Highway	Sofa	Boulevard	Crowd	Fountain	Illumination change
Test Image						
Binary Mask						

Table 2: Adjacent Frame Differencing results

3.2.3 Median Frame Differencing results

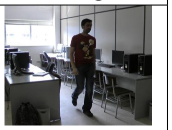
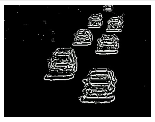
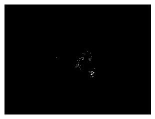
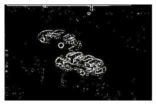
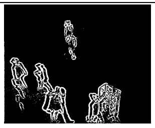
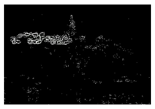
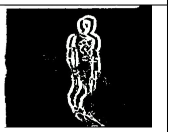
Description	Highway	Sofa	Boulevard	Crowd	Fountain	Illumination change
Test Image						
Binary Mask						

Table 3: Median Frame Differencing results

3.2.4 Median Frame Differencing with Morphology results

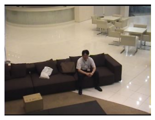
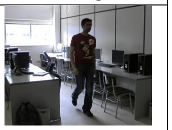
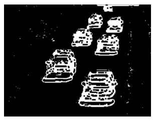
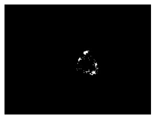
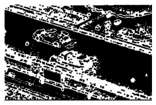
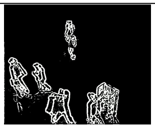
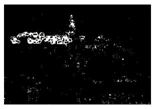
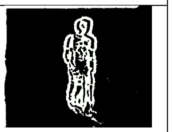
Description	Highway	Sofa	Boulevard	Crowd	Fountain	Illumination change
Test Image						
Binary Mask						

Table 4: Median Frame Differencing with Morphology results

3.2.5 Mixture of Gaussian results

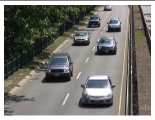
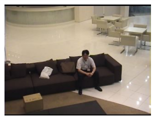
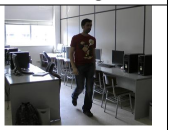
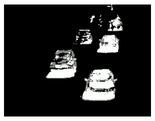
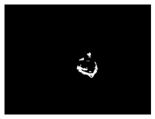
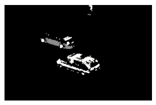
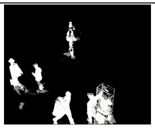
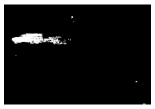
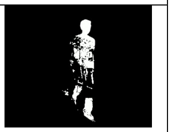
Description	Highway	Sofa	Boulevard	Crowd	Fountain	Illumination change
Test Image						
Binary Mask						

Table 5: Mixture of Gaussian results

3.2.6 TransCD model results

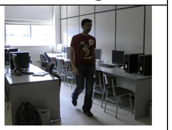
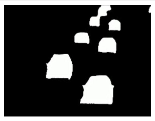
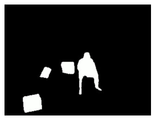
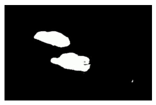
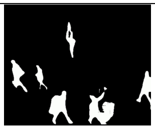
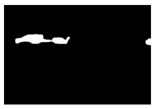
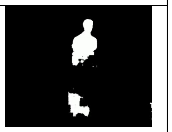
Description	Highway	Sofa	Boulevard	Crowd	Fountain	Illumination change
Test Image						
Binary Mask						

Table 6: TransCD model results

3.2.7 MU-Net1 model results

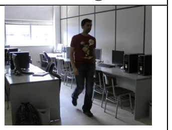
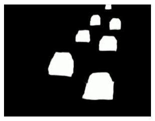
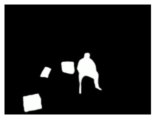
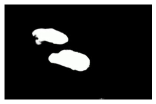
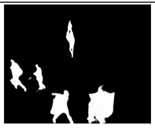
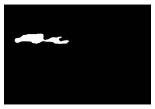
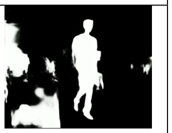
Description	Highway	Sofa	Boulevard	Crowd	Fountain	Illumination change
Test Image						
Binary Mask						

Table 7: Mu-Net1 model results

4 Comparison of Algorithms

4.1 Qualitative Comparison

In Table 8, the results of all the algorithms are shown for comparison using the same random frames from the resulting videos.

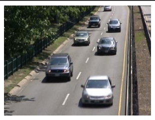
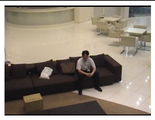
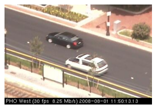
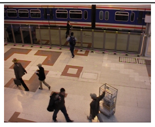
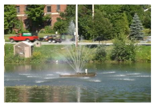
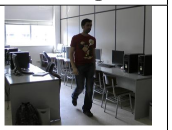
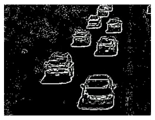
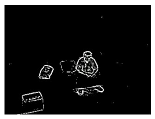
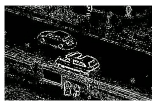
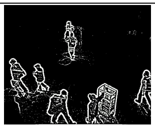
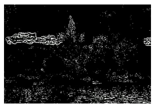
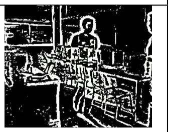
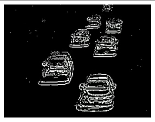
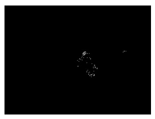
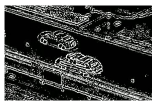
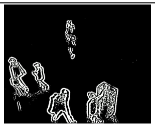
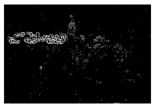
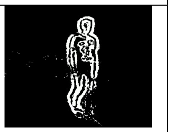
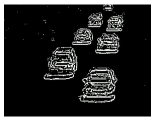
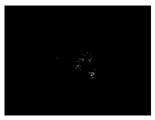
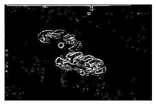
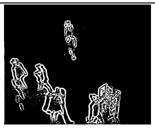
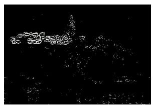
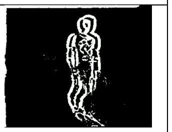
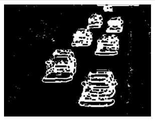
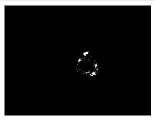
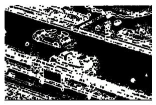
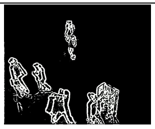
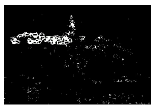
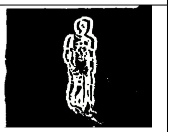
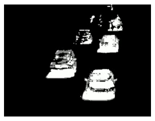
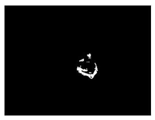
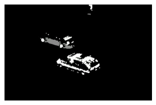
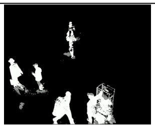
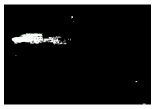
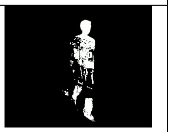
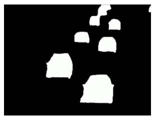
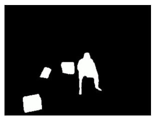
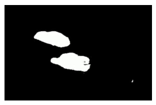
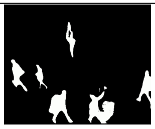
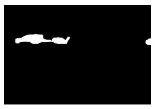
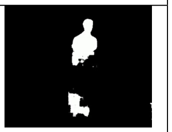
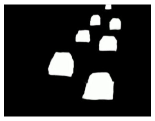
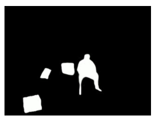
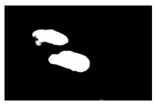
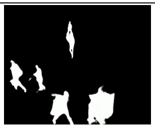
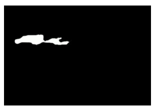
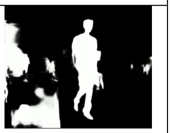
Description	Highway	Sofa	Boulevard	Crowd	Fountain	Illumination change
Test Image						
RFD Binary Mask						
AFD Binary Mask						
MFD Binary Mask						
MFD MORPH Binary Mask						
MOG Binary Mask						
TRANS Binary Mask						
MU-NET1 Binary Mask						

Table 8: Comparison of Algorithms

4.2 Discussion

The evaluation of various background subtraction algorithms for motion detection has yielded significant insights into their respective strengths and weaknesses. Based on the quantitative and qualitative comparisons, it can be noted that:

- RFD is computationally inexpensive and simple to implement, but it struggles with dynamic backgrounds and changes in lighting, leading to false positives and negatives.
- AFD adapts better to gradual changes in the scene, but it is also sensitive to noise, which can cause flickering in the detected motion.
- MFD offers improved robustness to noise and minor environmental changes, but the computational complexity increases due to the need to maintain a buffer of frames and compute the median. Morphological operations can improve the result in some cases, but also make it worse.
- MOG excels in handling scenes with noise, dynamic background and gradual lighting changes, but he algorithm can be computationally intensive, and tuning the parameters can be challenging.

- Deep learning approaches generally outperformed traditional methods, exhibiting better adaptability to complex scenes with varying lighting conditions, occlusions, and background clutter. However, high computational resources are required for both training and inference, making real-time deployment challenging without specialized hardware (e.g., GPUs).

5 Conclusion

In conclusion, the comparison of algorithms highlighted the strengths and weaknesses of each approach in motion detection. While traditional algorithms provide a feasible solution for simple and less dynamic scenarios, deep learning approaches are the preferred choice for high-accuracy motion detection in complex environments. However, the trade-off between computational efficiency and detection performance must be carefully considered, particularly for real-time applications.

References

[1] Gani Rahmon, Filiz Bunyak, Guna Seetharaman, and Kannappan Palaniappan. Motion u-net: Multi-cue encoder-decoder network for motion segmentation. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 8125–8132, 2021.

[2] Zhixue Wang, Yu Zhang, Lin Luo, and Nan Wang. Transcd: scene change detection via transformer-based architecture. *Opt. Express*, 29(25):41409–41427, Dec 2021.