Politecnico di Torino

Politecnico di Torino

Master's Degree in Cinema and Media engineering
A.a. 2023/2024
Sessione di Laurea Dicembre 2024

Machine learning for the comparison of synthetic images

as a tool to support the evaluation of rendering exams

Relatori:
Sanna Andrea
Manuri Federico

Candidati:
Bondi Anna

Abstract

The aim of this project is to design and develop a system to compare two similar synthetic images and identify and classify their differences. These images, referred to as Reference and Render, are generated with Blender's Cycles, a ray-trace-based production render engine capable of ultra-realistic rendering. The Reference image and the corresponding 3D meshes are provided to bachelor's students, who should prove their capabilities in rendering for design by reproducing all the visible features of the Reference from the same viewpoint, generating a new Render image. The system should support both students and teachers in identifying and explaining the differences between the Reference and Render image. The problem has been addressed with a machine learning approach: to perform the comparison, a neural network for semantic change detection was trained using a newly annotated dataset generated with Blender. This dataset enabled the network to distinguish both the target features, such as textures, shadows and transparencies and the changes related to them. The model performance is evaluated with metrics as precision, recall and F-score, whereas the extent of the differences has been evaluated by means of similarity measures.

Table of contents

1.Introduction

The 21st century has been characterized by one of the most impacting technologies in history: Artificial Intelligence (AI). Artificial Intelligence refers to the ability of a machine to perform activities usually associated with humans: predictions, decision making, learning. Over the past few years, the development of Artificial Intelligence led to simplification in everyday life and contributed to many fields of application. It has been widely adopted in the field of industry, from banks to marketing, as well as technology and entertainment. Beyond industrial applications, AI generated an exponential increasing of new tools for audiovisual tasks as voice conversion, image and video generation. These applications gained a lot of attention in the past few years, but the process that led to this point was complex.

1.1 Computer Vision and Artificial Intelligence

Research on Computer Vision (CV) has a long tradition. Starting from the 50' scientists started with the idea of teaching computers how to understand images, from edge detection and feature extraction. These tasks are nowadays the basis for most Computer Vision application. We now live in a context where cars can drive without a driver, thanks to a series of cameras that can interpret all the world around to identify danger, pedestrians, street, other cars and so on. This is possible thanks to Machine Learning (ML) and Computer Vision algorithms, that as first step extract features and detect edges.

AI has infiltrated massively also in Computer Graphics for cinema and videogame for example. The generation of historical sites reconstruction and complex scene is nowadays a common use, as well as motion capture and body tracking for animation. In 2017 Disney and Pixar published a work to present a new tool for denoising render images [1]. This allowed the production to render images in lower resolution, in order to save money and time, and enhance image quality with AI. The technology we use today is way faster and precise than in the '50, but the base idea is the same.

1.1.1 Segmentation and Classification

One of the most important tasks in computer vision is the classification and detection of object in images. Those tasks have several important applications, including medical image analysis, autonomous vehicles, security systems and augmented reality. The importance of these tasks has led the attention on the development of algorithms before, and Deep Learning methods after, that reaches performance similar to human eye. This progress was possible also due to an increasing availability of image data and labelled data.

In literature, classification and segmentation have a specific taxonomy, basing on the output: image classification, object detection, instance segmentation and semantic segmentation.

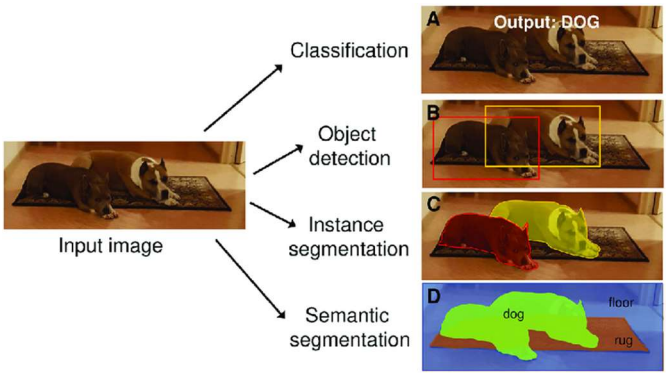


Figure 1.1: Classification Taxonomy

Image classification is a fundamental challenge in computer vision tasks, because it allows to automatically categorize an image. This is a supervised problem, so a label is manually assigned to each image, in order to automatically classify never seen images [Figure 1.1 A] [2]. Usually, the classification network predicts the probability of the image to belong on every possible class and assign the image to the label with the higher probability. Labelling can be employed in different ways, because the classification can have multiple methodologies:

- Binary classification provides only two classes, usually opposite (e.g. benign or malignant tumors, and all the problems that requires a binary response, yes/no).
- Multi-class classification: the classes are more than two and for each image is computed the probability for the image to belong to each class. The class with the higher probability is predicted, but the image can be assigned to only one class.
- Multi-label classification: the image can be assigned to more than one label. In this case, the labels assigned to an image are the ones which prediction is over a threshold. For example, if the labels are the animals, and the image represent more animals, the classes predicted will be more than one.

The object detection task is similar to multi-label classification, because every object in the scene is classified. The difference with image classification is that to every object detected is assigned a boundary box, which establish the location of the object. In this way, also the number of instances of each class are available. In figure 1.1 B, both dogs are detected separately, each one with its position, and the output predicts the class dog for the first object and also the class dog for the second object.

The image segmentation task is slightly different, because it provides a pixel-wise annotation: in the case of instance segmentation each object is defined with all its pixel, and it's classified as in object detection [Figure 1.1 C]. The difference with object detection is that the object is not defined with a bounding box, but with all the pixel that compose the object. Otherwise, with Semantic Segmentation, every pixel of the image is annotated with the class which it belongs [Figure 1.1 D]. Over time, many segmentation algorithms have been developed, such as thresholding, histogram-based bundling, k-means clustering. In the early times, Deep Learning models have shown a better performance, achieving high accuracy rate.

1.2 Image Similarity and Change Detection

One of the problem Computer Vision tries to face, is image similarity. This can be applied in various context, as audio, images, 3D data and more. The concept of similarity is wide and strongly related to the metric used to compute it.

One of the most famous applications for similarity is to understand if two images represent the same scene. Through the years a variety of CV algorithms were born, in order to be independent from all the changes that can occur when we have two images of the same scene: different point of view, scale, rotation, lighting. Features are extracted from both images, compared with some metric, and together with thresholding process determine if the two images represent the same scene.

Figure 1.2: Image Similarity example (*AI generated*)

AI based techniques are able to identify two images as similar if they share the same subject and environment [Figure 1.2]. In such a case, "similarity" other than visual or chromatic matches between images, uses a deeper understanding of the visual content, which includes the identification of the subject (person, object or entity represented) and contextual characteristics (location, outlook and environmental conditions). These features are extracted with deep learning models, as Convolutional Neural Networks (CNN) and other architectures, through which AI can extract a complex representation of the image content, making it possible to recognize similarities even when images differ in detail, viewpoint, lighting.

Basing on the context, instead of similarity it's possible to talk about change detection if I know a priori that my images should be similar but could have some difference and I want to find them. In this context, it's possible only to know if some changes occurred, or also which type of change occurred. To understand which changes occurred, it's necessary to be able to distinguish the different part that compose images. This process is called Semantic Segmentation and Classification. The first case, where only changes has to be found, takes the name of Change Detection (CD), while the second case, with the detection of changing types, is called Semantic Change Detection (SCD).

Change Detection can also be performed with other tools other than AI. Simple CV techniques, as Image Difference, identify changes even when there are difference only in lighting and contrast. Algorithms as histogram comparison, edge detection, techniques based on Fourier's transform, and others, can be used to understand if some changes occurred between two images. None of these algorithms are strong enough to understand the type of changes occurred and where they occurred together.

In general Change Detection is usually used for remote sensing, to check all around the globe where changes occurred through time in the environment. In the last years, the need to understand also the type of changes become stronger, due to deforestation, edification, and all the anthropogenic changes in nature.

1.3 Goals of the project

The main goal of the project is to establish the changes between two images generated by rendering algorithms. The thesis is placed in the context of a rendering course, where the students learn the basis of 3D rendering, the usage of the lights, materials, texture etc. Students are given a blender file, with objects and cameras already set. Their purpose is to set lights, materials and texture, in order to exactly replicate the Reference images the teacher provided.

We refer to the images as Reference and Render: Reference is the image provided by the teacher, while Render is the image generated by the student [Figure 1.3]. The focus of this thesis will be on lighting and materials, particularly on light direction, texture and transparency. One of the most troublesome problems is that these features affect each other sometimes. For example, the changing in light direction can influence other than shadow, also the colour of the texture, or the reflections on the materials. To simplify the problem, as result of the changing of the light direction we only consider the changing in the shadow position; for texture we only consider the part of the image where is present some kind of visual pattern, and for transparency we effectively consider the transparent objects.

With this tool the students have assistance in the learning phase, to understand errors, and it's a support for the professor for evaluating exercises and exams.

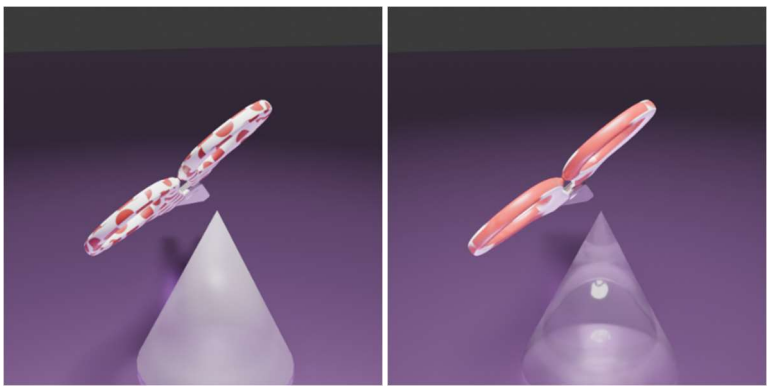


Figure 1.3: Reference and Render examples

1.4 Chapters' organization

The thesis is split into 6 chapters, whose contents are briefly described in the following paragraphs.

In the second chapter, the main works of the problems related to Change Detection are analysed, from image matching to Semantic Change Detection. The limitations and advantages of each task are explained.

The third chapter treats in detail the design of the system. It will be explained basing on the goals of the project together with the existing technologies, the proposed solutions.

The fourth chapter contains a description of the process of implementation, the details of how the dataset was created to train the network and how it was included in the model, together with the testing details.

The results are explained and analysed in the fifth chapter, along with details about the metrics used to evaluate the model and some visual examples of the test results.

The last chapter of the thesis explore potential improvements and new features that could be implemented in the future, comments and conclusions.

2.State of the Art

In computer vision, the task of understanding image similarity is a fundamental challenge and is always increasing. Over the time many different algorithms have been developed, and they are still improving due to more computational power, more efficient algorithms and innovation.

2.1 Image Similarity

Image similarity can be seen as a task where the two images are translated in a numerical representation of their content, and the distance between the numbers is the similarity index between the images.

The definition of similarity is not well-defined and could be interpreted in several ways. The images can differ in terms of contrast, brightness, colours, or they can be semantically identical so that they represent the same instance [Figure 2.1].

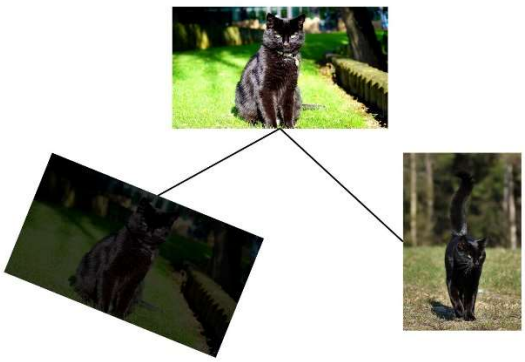


Figure 2.1: Similarity examples

2.1.1 Algorithmic Approaches

One of the easiest but quite inaccurate methods to estimate similarity is the computation of metrics as Mean Square Error (MSE). MSE is defined as the mean of the square of the difference pixel to pixel. If two images are identical, the value of MSE is zero, so theoretically the lower the MSE the more similar are the images. Even if this is a fast way to have a similarity score, the MSE could give a high value if the contrast or brightness are different, while the content is the same, as in the case of figure 2.2. This method, as all the

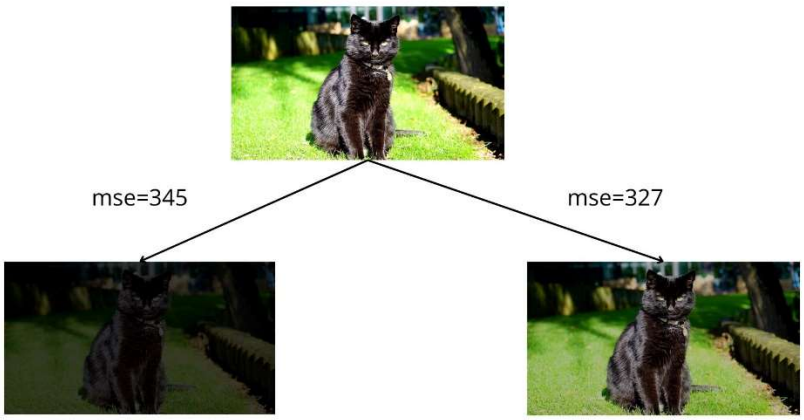


Figure 2.2: MSE

pixel-to-pixel methods, are useless when the images represent two different point of view or there is a difference in lighting conditions.

Considering these problems, it's important finding a way to compare images independently from lighting conditions and structure. To deal with this case, image matching task is the most appropriate. The image matching is based on finding some interest points (keypoint) in the image, which contains meaningful information. Keypoints can be border, edge, or high contrast points.

Image matching is divided into 3 steps: feature detection, feature description and feature matching. The detection of the feature can be done with detectors of different types: gradient, intensity, second order derivative, contour curvature, region segmentation, and learning-based detectors.

The most well-known image matching algorithms are Scale Invariant Feature Transform (SIFT) and Speeded-Up Robust Features (SURF).

SIFT is a powerful feature descriptor, and because of its property is useful in more complex tasks as image matching and object recognition. SIFT [3] was the first scale invariant feature descriptor, as the previous descriptors were only rotation invariant. It uses the second order partial derivatives, in particular the difference of gaussian, and extracts keypoint as the local extrema in a DoG pyramid, filtered using the Hessian matrix of the local intensity values. Over time, different versions of SIFT were presented, and SURF is the most well-known.

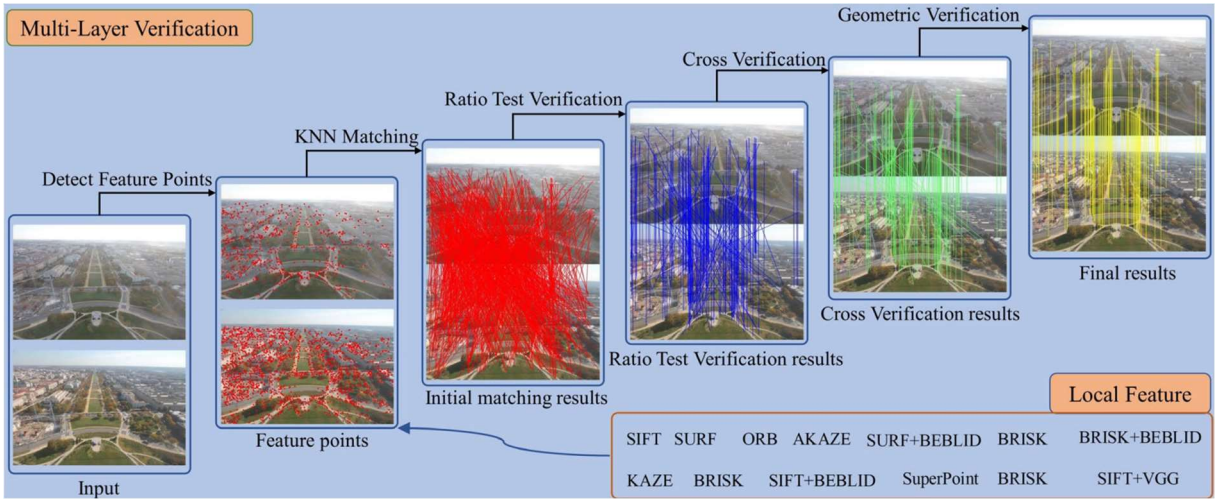


Figure 2.3: Multi Layer Verification

SURF proposed by Bay, Tuytelaars, & Van Gool, in 2006 [4], is a faster version of SIFT that uses a basic Hessian matrix approximation, together with an integral image strategy, thus simplifying calculation of features.

Structure Similarity Index Measure (SSIM), proposed by Wang et al. in 2004 [5], propose a method to measure the perceived changes in structural information in an image. SSIM incorporate perceptual features as luminance and contrast, that may vary across the image.

With changes in illumination and scale, these descriptors tends to generate more false correspondence between features. More complex algorithms as Multi-Layer Verification (MLV) [6] can reach more precise accuracy in these conditions. MLV is a image matching method, based on three verification steps, in order to limit the detection of false features correspondence [Figure 2.3]. Features and descriptors detected with SIFT, then two possible matching points are selected for each query feature point using KNN method. After, three verification tests are adopted. The first is a ratio-test method: the ratio of the distance between the query point and the first candidate to the distance between the query point and the second-best candidate is over a threshold, the feature match is considered as false correspondence. The second verification is a cross verification, useful in scenes where there is a repetitive pattern and texture repetition. In these situations, feature points at different locations can have similar descriptors, leading to a big amount of false correspondence and other matches are removed if not accurate. The last step is geometric validation, that uses homography matrix to set geometric constraints. Correspondence that does not satisfy the matrix are removed, improving the accuracy.

2.1.2 Machine Learning Approaches

Artificial Intelligence based techniques, especially those based on deep learning models, are able to identify two images as similar if they share the same subject and context. Differently from traditional CV algorithms, which compare superficial characteristics of the image and pixels, AI models are able to extract a more complex representation of the image, distinguish features that lead images to be conceptually similar. Convolutional Neural Networks (CNN) and more complex architectures as Siamese Neural Networks (SNN), can extract features vector, numerical representations of the image, which capture structural and context details of the scene. Comparing these vectors, basing on their distance, is possible to understand if images are similar.

In the context of similarity learning, the main goal is to make the machine learn a similarity criterion between images, in order to predict similarity between unseen image data. This purpose can be reached through Siamese Neural Networks. The term "Siamese" refers to the architecture of the network, where two identical sub-networks share the same weights, processing two different inputs. SNN are composed of two main components:

- Feature extraction: usually it is implemented with a Convolutional Neural Network, that gives a representation of the features of the input images. This component also tries to detect meaningful features and ignore less significant ones. In fact, the network should be able to ignore difference of lighting and weather condition. Other than CNN also Multi-Layer Perceptrons (MLP) are sometimes used to extract features. The sub-networks are trained simultaneously, and outputs are compared with some metric.
- Similarity measure: the extracted features are compared with a metric of distance, giving a similarity score or a difference map. Different similarity criteria can be used, such as L1 Normalization, L2 Normalization, Inner Product, Cosine Similarity etc.

Siamese Neural Network were first used in 1994 by Bromley et al. [7] for signature verification, in order to detect valid and forged signatures. Other than signature verification, SNN find space in the application of face verification, object recognition, gait recognition, character recognition, but also speech and natural language processing.

Structure of SNN can vary slightly, but according to Nandy et al. [8] three main types of architecture are used. In the first type of architecture [Figure 2.4 (a)], each sub-network takes one input, and both are merged at the end with a difference metric that compute the similarity index. The second architecture [Figure 2.4 (b)] is similar to the first, but it presents more merged layers at the top of the network. Finally, the third architecture [Figure 2.4 (c)] take the input stacked on top of another, so that the architecture is a simple CNN.

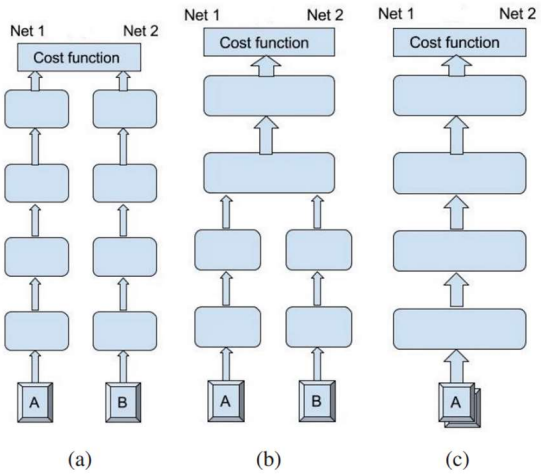


Figure 2.4: SNN Architectures

Nandy et al. show that the third architecture is better in case of comparison detailed structures, while the first and the second are helpful in case of classification.

Another approach to similarity problem is a slightly different structure of Siamese Networks, the so-called Triplet Network. Unlike SSN, Triplet Network is composed by three sub-networks, and as the SSN share the same weights and the same biases. The inputs of this network are three: one sample named “anchor”, a positive sample and a negative sample. The anchor and the positive sample belong to the same class, while the negative sample represent another class. In this way the network tries to minimize the distance between anchor and positive samples, and in the meanwhile tries to enhance the distance between the anchor and the negative sample.

The advantage of SSN is that it perfectly suits the problem of similarity, due to architecture of identical sub-networks with shared weights. This kind of architecture also allow to train the network with a few numbers of examples, trying to achieve good generalization with limited data. Limitations of SSN lies on the difficulties in finding small changes, which makes them not suitable for fine-grained similarity problems.

2.2 Semantic Segmentation

Semantic Segmentation (SS) is a fundamental task in computer vision, used in several applications as autonomous driving, medicine, agriculture, robotics. SS purpose is to assign to each pixel of an image a label, in order to distinguish object or areas of interest and classifying them in predefined categories, such as “roads”, “pedestrian”, “car” in a urban context. Modern semantic segmentation techniques rely mainly on Deep Convolutional Neural Network due to their ability to capture spatial structures and details. In a recent survey of image segmentation [9], an overview of the main segmentation models is performed. The authors show that the most popular segmentation model uses an encoder-decoder architecture. In medical and biomedical applications, a specific type of encoder-decoder model is used, the U-Net, because it needs less data than other architectures, basing on strong data augmentation.

U-Net can be used in other context other than medical and biomedical images, like road segmentation and 3D images. One limitation of encoder-decoder architecture is that the encoding process leads to a loss of resolution, that the decoder cannot restore completely. There are some tricks to adopt to increase the resolution, as computing the non-linear up-sampling or using parallel stream of different resolutions as HRNet.

The first U-Net was proposed by Ronneberger in 2015 [16], and it is composed of a symmetrical structure with a shape of a "U" that includes two main sections [Figure 2.5]:

- Encoder: encoder is used to extract relevant features through convolutional and downsampling operations. In this section, the size of the image is progressively reduced, increasing the number of channels, and the depth of the representation, including features from different abstraction layer.
- Decoder: decoder is used to restore the image to its original size, with upsampling operations followed by convolutions. Moreover, every layer takes as input the output of the corresponding encoder layer, thanks to skip connections. This approach allows to combine high level features with low-level features, in order to have a detailed segmentation.

The usage of skip connections between encoder and decoder are the distinguish part of U-Net architecture. These connections maintain spatial information that would be lost during downsampling, making it easier to recover fine details during the upsampling phase. In practical terms, each encoder block is linked to a corresponding decoder block. The resulting feature maps are concatenated, providing the model with a richer spatial and contextual representation.

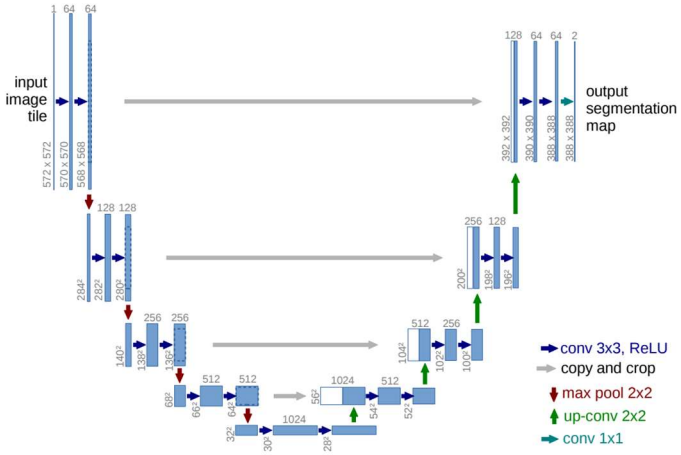


Figure 2.5: U-Net architecture

Through the years several upgrades have been done to SS, especially on multi-class task. In 2019, MultiResU-Net was presented [10]. It is an improvement respect the classic U-Net architecture, using the MultiRes blocks instead of the two convolutional layers. This block uses 3x3, 5x5 and 7x7 filters [Figure 2.6 (a)], factorizing 5x5 and 7x7 filters as a succession of 3x3 filters [Figure 2.6 (b)], to build a sort of Inception block, but adding a residual

connection with 1x1 filter to preserve dimensions [Figure 2.6 (c)]. The multi-scale filters enable multi-resolution feature extraction, improving the ability of the network to manage large scale and fine-grained details. Respect to the base configuration of U-Net, MultiResU-Net achieve higher segmentation accuracy, due to the multi scale feature extraction combined with skip connections and residual blocks. Residual paths also facilitate the training of deeper network, anticipating the vanishing gradient problem. The complexity of this network is higher respect to traditional U-Net, and it's more sensible to hyperparameters as learning rate, depth and number of filters.

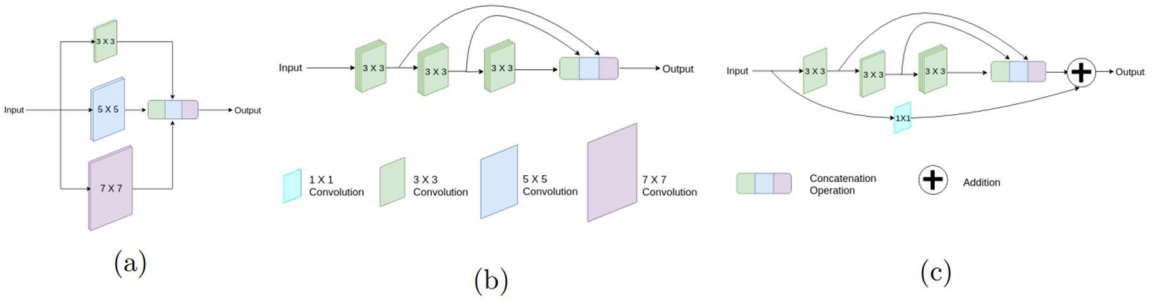


Figure 2.6: MultiResUNet

U-Net is not the only architecture available to do segmentation, but many structures for different purposes are available. One of the most complete segmentation networks is Segment Anything Model (SAM) by Meta [11]. SAM is a is an innovative image segmentation network, an highly versatile model capable of identifying and segmenting nearly any object in a wide variety of contexts, without the need for task-specific fine-tuning. Its architecture uses a segmentation network that can handle different kinds of prompts, such as points, bounding boxes, or even text, to accurately outline objects in an image. SAM's flexibility and precision is possible due to its extensive training, which involved over 1 billion masks collected from 11 million diverse, real-world images. This vast dataset allows SAM to handle complex segmentation tasks across fields like medical imaging, autonomous driving, augmented reality, making high-quality segmentation widely accessible. SAM's architecture is composed of three main components: an image encoder, a prompt encoder, and a lightweight mask decoder.

- Image Encoder: SAM uses as encoder a Vision Transformer (ViT), that processes the input image, creating an high-dimensional feature map that compose the basis for segmentation. By using a ViT, SAM is able to get a significant representation of the image, in order to capture visual details across diverse contexts.
- Prompt Encoder: the prompt encoder in SAM allows the model to receive various types of prompts as input from the user, such as points, bounding boxes, or free-form text descriptions. It encodes these prompts into embeddings that, combined with the image features, allow SAM to adjust its segmentation output based on user input. This allows SAM to adapt the segmentation output on the basis of the request of the user.

- Mask Decoder: the lightweight mask decoder is the final component and turns the combination of features from encoder and prompt embedding into precise object masks. This decoder enables SAM to produce accurate segmentations rapidly, even in real-time applications. The mask decoder uses a transformer-based approach that efficiently maps the combined features to pixel-level masks.

Together, these three components allow SAM to perform accurate and versatile segmentation across a wide range of tasks and applications without specific fine-tuning for each new task. This architecture makes SAM adaptable and accessible for many users, from casual users to professionals working in areas like medical imaging and augmented reality.

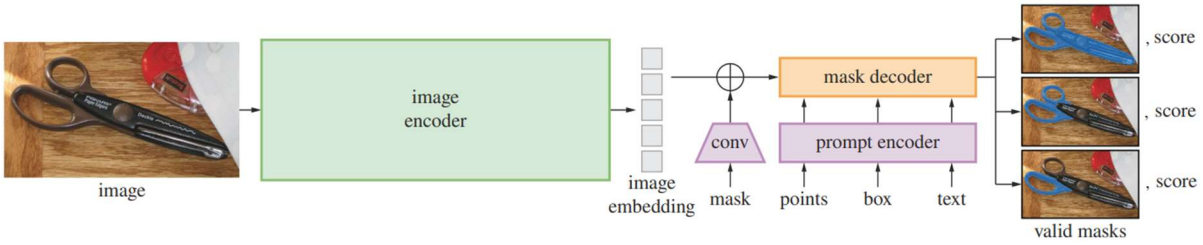


Figure 2.7: SAM overview

2.3 Change Detection

Change detection is a complex and studied task, which finds its main issue in the lack of annotated data, that is a complex and time consuming work. Nevertheless, several valid works were found in the literature, also for the more complex task of Semantic Change Detection (SCD). CD find its major application in land monitoring, using High Spatial Resolution Images with bitemporal supervised learning, focusing on urban planning and disaster monitoring. The main challenge of CD is detecting relevant changes for the given task.

2.3.1 Algorithmic Approaches

In change detection, which is the detection of variations in time sequences of images, there are numerous algorithms that do not rely on artificial intelligence and have been used for decades in geospatial analysis, in environmental monitoring and surveillance. These algorithms use statistical analysis and image processing methods to identify differences between images acquired at different times.

It is essential to note that Change Detection and Image Similarity are complementary tasks, hence all the Algorithmic Approaches described in this work can be used for both purposes, as detecting similarity index it's possible to derive change index and vice versa.

Image Differencing (IM) algorithms are one of the simplest and most effective methods for change detection. IM lies in subtracting pixel by pixel the corresponding values of two images captured at different times, and changes are detected if the difference is over a threshold. The main goal is creating a map of differences: when the difference is close to zero, no notable change occurred, while higher values indicate areas with more variations. This method is

sensible to noise and changes in brightness, and it often require pre-processing or filtering to enhance the accuracy of the results.

Histogram Comparison (HSC) is a technique that compute the distribution of intensity values (or colour) in the two images, independently from spatial position of the pixels, and compare the histograms to detect changes. More two histograms are similar, less changes affected the images. This method is more robust in terms of uniform changes or global intensity changes but has no spatial information. HSC is particularly useful in environmental monitoring scenarios, where changes in light conditions may mask real changes.

Edge detection (AD) based algorithms are effective to identify structural changes between images, such as new constructions or shape variations. Using operators such as Sobel, Canny or Laplacian, these techniques extract the main contours of images, representing the objects in the scene in terms of edges. By comparing the edges extracted from two images, it is possible to determine whether new structures are present or whether previous elements have been removed.

Fourier transform, and all other frequency transforms are powerful tools for detecting changes. Applying the Fourier transform, the image is represented as a combination of spatial frequencies. Differences in frequency components can reveal, for example, new edges or surfaces that were not present in the original image. In addition, Fourier transform techniques are robust to translation and rotation, making them suitable for detecting changes on non-aligned images.

2.3.2 Artificial Intelligence Approaches

Remote sensing is the main field of application of Change Detection (CD) faced with artificial intelligence, as it allows to detect natural and anthropogenic changes occurred over the years in the earth's surface. Change Detection system assign to each pixel a binary label: a positive label is assigned to the pixel if that area is changed between the two images, otherwise the label is negative. These maps are called "change maps" and defines the area where the changes occurred. CD problem is a supervised learning problem, trained in end-to-end manner, so change maps are used to train the network and are also the output of the system. One of the main limitations of Change Detection for remote sensing is the lack of annotated data, which cause the limitation of complexity of the model to avoid overfitting.

The work proposed by Daudt et al. [12] presents a Fully Convolutional Siamese Network for Change Detection (FCCD), with three different architectures that use skip connections to enhance performances. Due to the lack of annotated data, most of recent CD works uses pretrained model or transfer learning from another dataset. Instead FCCD is trainable end-to-end.

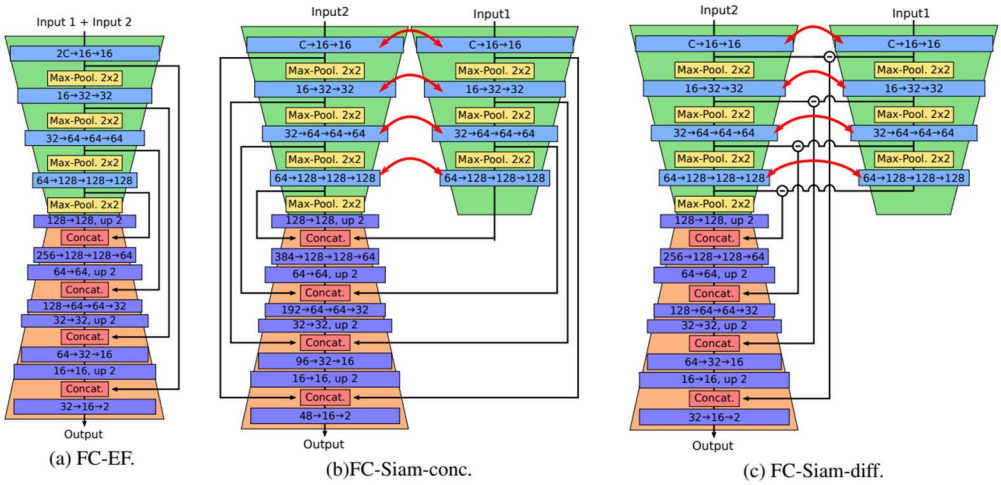


Figure 2.8: FCCD Architectures

These architectures use a patch-based approach, as it enhances accuracy and speed of inference. The first structure [Figure 2.8 (a)] is a single stream network based on U-Net model, and concatenate the two input patches before they are fed to the network. The second and third [Figure 2.8 (b)] [Figure 2.8 (c)] are double-stream architectures, based on Siamese Networks, together with skip connections. In this case, the patches are given separately to two identical branches of the encoder, which share weights and biases. The limitation of this work is that the structure does not permit to detect different types of changes, because it has not a module for Semantic Segmentation.

Parelius in [13], compared through metrics as precision, recall, F1 and OA (overall accuracy) the performance on different change detection architecture on different datasets. The best results come from supervised learning techniques, as UCDNet.

UCDNet [14] is a double stream network that uses attention mechanism to enhance performance and precision. It is composed by two encoders with identical structure and weights, formed by convolutional and pooling layers. UCDNet introduces a version of spatial pyramid pooling (NSSP) between the double-stream encoders and the decoder. NSSP takes ad input the features from the encoders, which goes through four parallel paths, each one composed by a pooling block followed by a global pooling block [Figure 2.9]. The pooling block consist of strided convolution and average pooling in parallel, combined at the end of the block. It helps extracting features in more effective way. Global pooling block helps reducing degradation problem and improve the awareness of global information. The difference in the four streams is the scale of the features. The global pooling block is composed of a mean block and a 1x1 convolution followed by upsampling layer. The decoder takes the features from NSSP and transpose them to pixel-space, with upsampling layers, convolutions and batch normalization. This architecture is optimized to work in urban context, and it achieves good performances on building, streets, vehicles and urban structures. With multi-scale features approach, it can detect both fine-grained changes and large-scale transformations. The limitation of this work lies in the encoder part, where computing the

difference between feature maps reduces the values assigned to pixels, and some of them may be ignored.

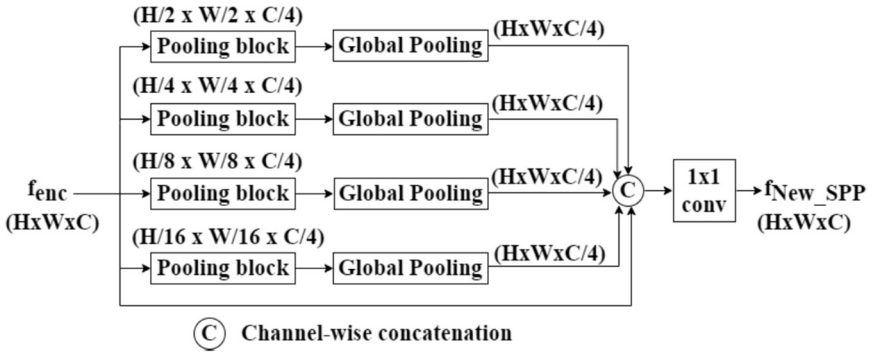


Figure 2.9: NSPP block

2.3.3 Semantic Change Detection

Suzuki first introduced the concept of Semantic Change Detection (SCD) in 2018 [15]. SCD combines the traditional binary Change Detection task with the semantic purpose. The goal is to figure out where changes occurred, and which types of changes are present between the images. This work used multi-class segmentation with three classes, and was applied to the detection of type of changes before and after a tsunami. The segmentation task is approached with hypercolumns and hypermaps with multiscale representation, providing an improved representation of semantic meaning. This work introduces the concept of change detection, but does not directly perform change detection, it perform semantic segmentation on already detected changed areas.

SCD can be divided into two categories: scene-level semantic change detection (SLSCD) and pixel-level semantic change detection (PLSCD). The former, assign semantic labels to object instances in the scene, as cars, while the latter assign a semantic label to each pixel, where a detailed change map can be generated. In this analysis, we focus on PLSCD.

Daudt et al. in 2019 proposed in improvement of FC-EF, the FC-EF-Res [16], a deeper version of the architecture that uses residual blocks. The more complex network is possible due to a new annotated dataset created by the authors, bigger than all the previous remote sensing image dataset. With semantic information and change maps, a variety of strategies to detect changes are available. The segmentation masks are named Land Cover Maps (LCM), and the work presents several strategies to use LCM, CM, and the combination of both.

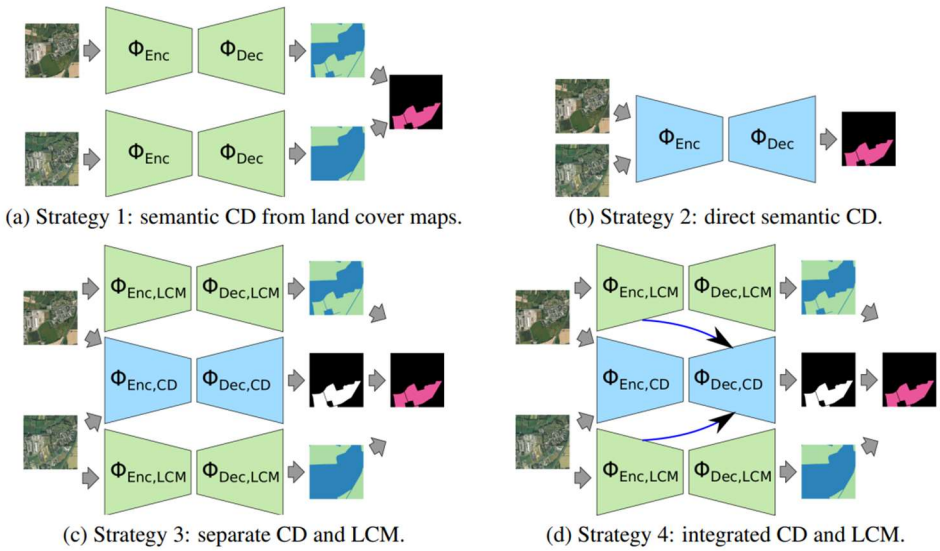


Figure 2.10: FC-ER-Res's strategies

One possibility is to direct compare LCM and create a semantic change map; comparing the LCM of input images, it is possible to detect which area changed label [Figure 2.10 (a)]. The limitation is that this strategy strongly depends on accuracy of LCM, and the changes that occurred in the same semantic area are not detected. This means that if there is a change over an area, which maintains the same label, this change is not detected. Otherwise, it is possible to directly compare binary CM, without using LCM, treating every change map as an independent label [Figure 2.10 (b)]. The weakness is that with the growing of the number of classes the accuracy gets worse, and the training is more complex. Another possibility is to train separately two independent networks, one for Semantic Segmentation and one for Change Detection [Figure 2.10 (c)]. In this way, the inputs produce three outputs: one binary change map and two LCM. With this approach both the pixel change, and the type of changes are predicted, even in the same semantic area. The last possibility presented is to integrate LCM and CM into a single multitasking network, so that LCM can be used for Change Detection [Figure 2.10 (d)]. In this architecture the LCM encoder branches are passed into the decoder of CD, together with the encoder result of CD, in the form of difference skip connections.

In most of the work in the High Spatial Resolution Change Detection, the approach is a bitemporal supervised learning, which uses two images of the same area in different times. The more problematic aspect in this approach is the availability of labelled data, because it is an expensive and time-consuming work define positive and negative samples. These samples are defined positive if the same pixel in different times have the same semantic classification.

After 2019, many other works based on strategy 3 and 4 by Daudt et al. [16], especially for the remote sensing imaging.

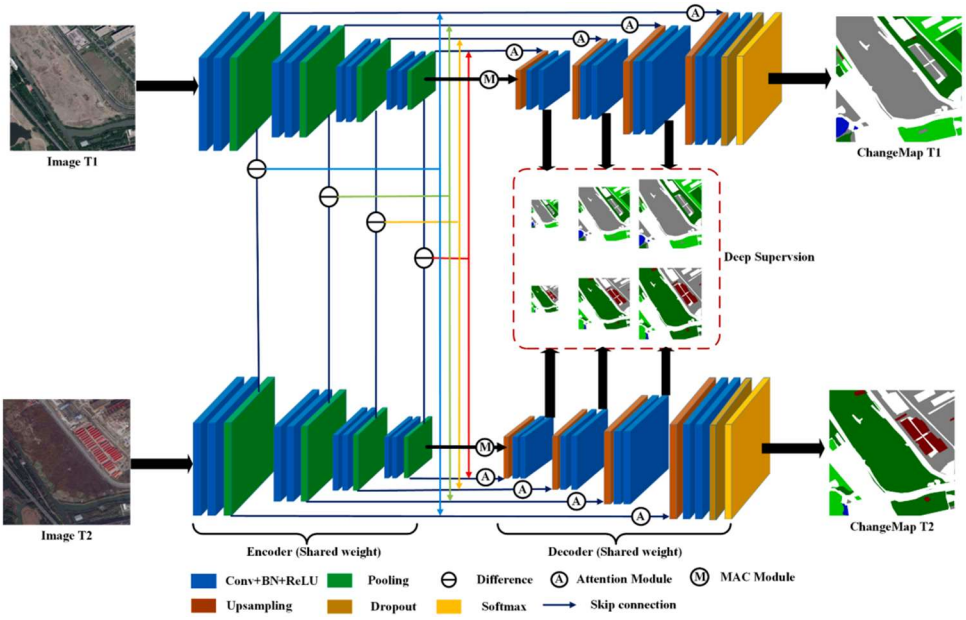


Figure 2.11: SCDNet

In 2021 Peng et al. introduce a PLSCD SCD network, based on Siamese U-Net architecture, the SCDNet [17] .SCDNet is composed by two branch encoder and decoder with shared weights [Figure 2.11]. Pretrained ResNet34 was adopted as backbone for the encoder, to accelerate the convergence of the network. After the encoder, a MAC module is used to enlarge the receptive field, using the sum of dilated convolutions at different dilatation rates. Then the feature maps are expanded with convolutions and upsampling operations, to restore the full-resolution feature map. MAC module is essential to capture objects with different scale. Low-level and high-level feature maps are combined through skip connections with an Attention Module, which takes high level feature map and process it doing the sum of convolution 1x1 of average pooling and convolution 1x1 of max pooling. This result goes through a Sigmoid function and is multiplied element wise with low level feature map. SCDNet output two semantic change maps, one for each decoder.

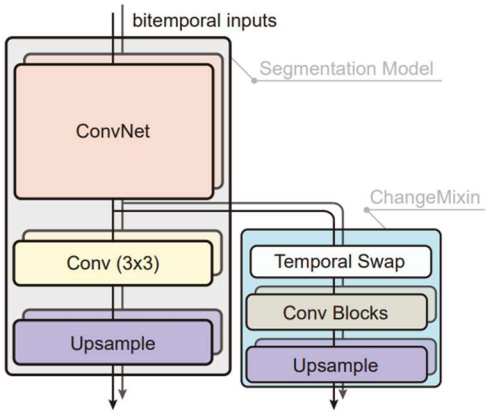


Figure 2.12: ChangeStar architecture

Single Temporal supervised leARning (ChangeStar) [18] is a supervised object change detector, that avoid the problem of collecting paired labelled images. This structure instead of training bitemporal labelled images, train a network able to detect object change in any context, including images of the same area in different times. The idea is to have a generic object change detector from casual image pairs with only semantic labels. The architecture [Figure 2.12] is divided into two parts: a generic segmentation model, used to extract a feature map for every image, and a specific module (ChangeMixin module) that uses a Temporal Swap Module followed by a Fully Convolutional Network. The two branches that process input images share the same weights, following Siamese Networks strategy. The outputs are two semantic predictions, one for each input image, and a change prediction. It is important to note that this work is designed to work only with one type of object of interest. ChangeStar is a flexible model, as it does not use a specific pretrained segmentation module, making it possible to test different backbone as feature extractor, basing on the task. Therefore, it is designed to train the network also on not paired labelled data, allowing a more efficient and flexible training using more data. ChangeStar is available in Torchgeo library from PyTorch, making it easy to implement in any context. However, since it is a SCD architecture, its performances strongly rely on feature extraction and segmentation accuracy.

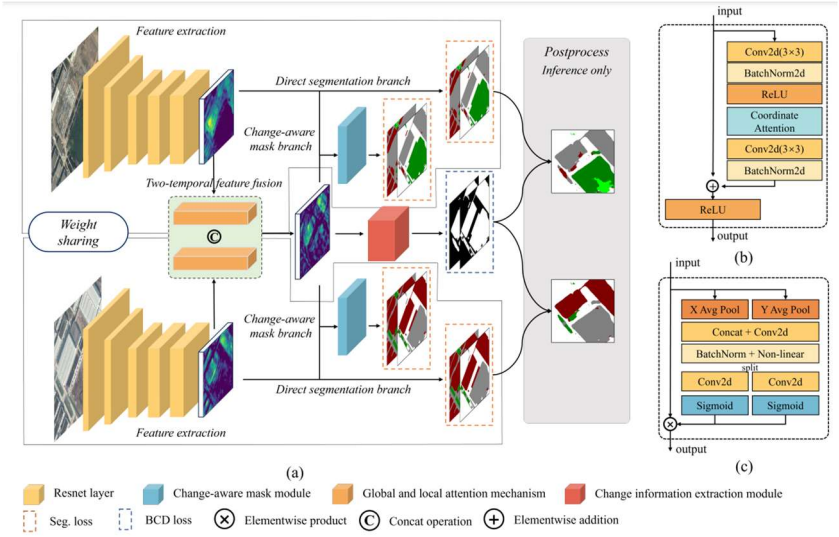


Figure 2.13: CGMNet architecture

One of the latest works related to change detection was proposed in 2024 by Tan et al. [19], which proposed the CGMNet, a SCD change-aware guided multi-task network. This architecture integrates a change aware mask branch, exploiting the knowledge of regions of change to enhance land cover classification. Feature maps are obtained through feature extraction using as backbone a ResNet34, modified to avoid downsampling in some layers, to preserve spatial details. An overview of the architecture is exploited in figure 2.13. Once the feature maps are extracted, they are given to a two-temporal feature fusion block, named GLAM. Due to the characteristics of remote sensing images, Tan et al. developed a global and local attention mechanism (GLAM), which capture both overarching and detail features

through two parallel branches, giving an attention map as output. In order to focus more on the region of change, a change-aware mask branch was introduced. It can improve the ability of the network to concentrate more on changed region, and to extract essential features. This branch takes as input a single-temporal feature map and the output of GLAM. The branch is composed of a change-aware mask module, which takes as input the attention map from GLAM and process it with convolutions, pooling a sigmoid. The output of this module is the change map, which can be combined with segmentation branches.

3.System Design

This chapter has the aim to dig into the design of the system, basing on the analysis of Chapter 2. The characteristics of the architecture are explained together with the modifications of the original structure, for a better understanding of the study.

3.1 The Networks

As detailed in Chapter 2, image similarity algorithmic approaches are not strong enough to detect the type of changes that occur between images. Instead, the majority of machine learning approaches for image similarity uses Siamese Neural Network, which thanks to the sharing of the weighs' technique are the most used architecture to detect similarities. This architecture as it is, does not allow to detect the type of changes, but gives a solid foundation for the networks used for Change Detection and Semantic Change Detection.

The first attempt was performed using basic Change Detection, without annotation, using one change map for each feature, using the FC-Siam-Diff by Daudt [12]. As discussed in previous chapter, adding Semantic Segmentation to Change Detection helps the network to detect changes, as it identifies the area where interested changes can occur. Considering that the main field of application of Semantic Change Detection is remote sensing, which are available limited pair labelled data, the best compromise between Segmentation and Change Detection is the usage of U-Net. For this reason, the majority of Change Detection Network has a U-Net based architecture. As mentioned, U-Net is not the only available architecture for Semantic Change Detection, as other structures relie on pretrained backbone for feature extraction. This project works on synthetic images, which creation and labelling are automatic and does not require the human hand. For this reason, the project explored both ways, U-Net based architecture and the union of Semantic Segmentation and Change Detection. The U-Net based architecture chosen for the project was the FC-ER-Res by Daudt [16], while Semantic Change Detection Network selected was ChangeStar [18].

3.1.1 U-Net based architecture

As mentioned in previous chapter, the skip connection of U-Net allows to use large context information together with local ones, in order to have an overview of all the important feature of the images.

One of the first attempt was done using a simple Change Detection Network, specifically FC-Siam-Diff. This network requires an easier implementation than UCDNet, and the code is public, therefore this architecture was our first choice. FC-Siam-Diff takes as input the Render and Reference images, which pass through identical branches of the encoder. The encoder is composed by 4 steps, each one including a convolutional block, followed by batch normalization and ReLU activation function. The decoder is composed by 4 steps either and

restore the image to the original size. The most significant change we made was increasing the number of output channels in order to equal the number of the feature to monitor. The original version proposes as last layer a LogSoftMax function, because it was designed for binary Change Detection, monitoring all the changes to consider together. The output of the network was composed by two channels, one for changed pixels and the other for non-changed pixels, and the class with higher probability was assigned. In our case, we wanted the type of changes to be separated and independent from each other, so a Sigmoid function was used instead of the LogSoftMax. In this way, if the Sigmoid has a probability minor than a threshold, the pixel is considered not changed for that feature. The input of the network are Render and Reference images, and the loss is computed comparing the output with the change maps created.

Softmax and Sigmoid are the same function, that map the input value in output between 0 and 1 [Figure 3.1]. The difference is that Softmax function works with more than two classes, and for each class it predicts a probability for the pixel to be in that class, and the sum of all the probabilities is 1. Sigmoid is designed to work on binary classification, which is basically our problem. In fact, for each class we have two possible values, 0 if the pixel does not belong to that class, and 1 if it does. With Sigmoid function, we can make every probability for each class independent from the other, enabling the multi-label segmentation.

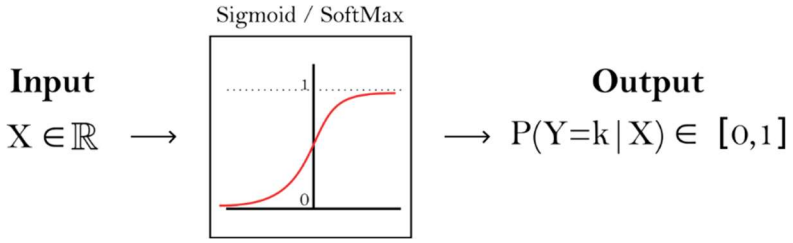


Figure 3.1: Sigmoid and Softmax

In a second time, we used the FresUNet by Daudt et al. [16], as it is an upgrading of FC-Siam-Diff used before, implementing different strategies. The easiest strategy to implement was strategy 2, direct semantic CD. This architecture takes up the Siam-Diff but includes residual blocks in order to increase the spatial accuracy of the results and facilitate training. As the previous network, the LogSoftMax function was replaced with the Sigmoid function and the output channel with the number of features to consider.

With the introduction of labelled data in our database, we had the possibility to implement strategy 3 and 4.

FresUNet with strategy 3 uses two separate networks: the first performs CD, while the second Semantic Segmentation (LCM). The architecture of the two networks is the same, FresUNet, with Sigmoid function instead of LogSoftMax. The only difference between is that the CD architecture takes two inputs, Reference and Render image, while the LCM architecture takes only one input, Reference or Render. Even in this case the number of channels was set equal

to the number of feature (N_FEATURES). This lead to three outputs for each pair of images: two LCM, each with N_FEATURE number of channels, and one change map, also composed of N_FEATURES channels. Training separately the networks, there is the need to use two different losses together with two different backward, one for Change Detection Network and the other for Semantic Segmentation Network.

The 4th strategy proposed by Daudt et al. in [16], is a combination of Semantic Segmentation and Change Detection, that combine the two tasks in the same network. Respect to other architectures for Semantic Change Detection described in Chapter 2, this network is older and easiest to implement, but it is also fully trainable and does not require separate segmentation module. In fact, SCDNet uses a ResNet34 backbone, trained to detect objects as car, trees, while we need to detect less defined features as texture, transparencies, and shadows. CGMNet was excluded for timing reason, in fact this work was presented while experiments were already running.

In strategy 4, the network takes as input Reference and Render image, and output two semantic annotations and one change map for each feature. To do this, the network is composed of three branches, one for CD and two for Semantic Segmentation [Figure 3.2]. Semantic Segmentation uses one encoder-decoder for each image, reconstructing class annotations. Change Detection branch is encoder-decoder, but encoder takes as input both Reference and Render image, while decoder takes CD encoder output and both LCM encoder output. In this way, Change Detection can be performed with Semantic Feature map, other than CD features. The network output three images for each feature: one change map and two semantic segmentation maps. This network was originally developed to perform multi-class segmentation, so every pixel is assigned only to one class. Our problem is a multi-label segmentation problem, as every pixel could be assigned to more than one class. For example, if the shadow is on the floor and the floor has a texture, the network should say that that pixel belongs to both classes. Every architecture was modified in order to have 3 input channels (the RGB image), and 3 output channels (one for shadow, texture and transparency). The conversion from multi-class segmentation to multi-label segmentation was performed modifying the last layer of the network, instead of using a Softmax function it was replaced with Sigmoid function. The architecture thus modified is therefore compatible with multi-label segmentation, as for each output channel, each pixel has a probability between 0 and 1 to belong to each class.

In all strategies proposed by [16], the FresUNet encoder-decoder architecture is the same, and is a U-Net with skip connections with 6 encoder and 6 decoder blocks. Each block is composed of two residual blocks followed by subsampling residual blocks for encoder blocks, and upsampling residual blocks for decoder blocks.

Implementing strategy 4, we needed to have two different decoders, in order to manage the different shape of input. In fact, for LCM decoder there is only one input, while for CD decoder three inputs were concatenated, consequently the input size had to be slightly changed; the encoder is the same both for LCM and CD.

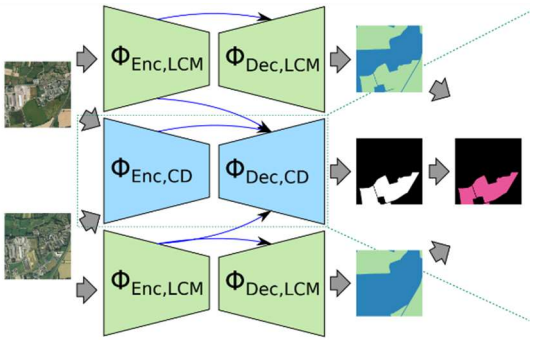


Figure 3.2: Strategy 4 Architecture

With all the U-Net based strategy the features considered were shadows (considering only the direction of the light), texture and transparency. With the use of pretrained networks, there was the possibility to augment the number of data available, and it has been increased the database introducing the new feature of reflections.

3.1.2 Pretrained Network

The reason for choosing a network that requires pre-trained segmentation lies in the results of the U-Net architectures, specially on the ones regarding the Semantic Segmentation. As we shall see in Chapter 5, the lack of significant results in Semantic Change Detection of U-Net based networks is not only in the Change Detection task, but also in the Semantic Segmentation task, which means that the U-Net is not the best strategy to segment the type of features we want to find. For this reason, a pretrained network with a strong features extractor could help instead of completely train a new network.

ChangeStar has a fast and intuitive implementation, because it is present in the library torchgeo. It requires three main components: a pre-trained feature extractor, a classifier and the ChangeMixin module.

As pre-trained feature extractor SAM by Meta [11] was chosen, as it is designed to segment almost anything, and it is trained using over 11M images and over 1.1B segmentation masks. This leads the network to have a strong feature extractor, able to detect global and fine-grained features. A pre-trained version of SAM was set as feature extractor, and no parameters were trained in this section. To perform segmentation of the features, the classifier was designed as a sequence of convolutions and activation functions, together with upsampling layer. In fact, the feature representation coming from SAM has an output of 256 channels, with the dimension of feature maps equal to 64x64. We want our image to be resized to its original shape of 512x512, and this is possible with different combinations of convolutional and upsampling layers. Conceptually this operation can be made with a single convolutional layer, but the dilatation of the feature map from a small size to a big size with one operation encourage the loss of details, making the final image blurred. On the other side, using a long sequence of convolutions and upsampling layers leads to the opposite problem,

the so called *"checkerboard"* artifact. This term refers to a visual artifact that can occur in Neural Networks working on images, in task like image generation and upsampling, as in this case. These patterns are caused by the uneven overlap or stride in convolution operation in upsampling process, where some regions receive more updates than others. To limit these artifacts, techniques like substitute transposed convolutions with classic convolutions followed by upsampling layer are used. In this case, the trade-off chosen was as shown in Figure 3.3. The sequence is composed of three blocks, that contains convolutional layer followed by batch normalization, ReLU activation and upsample; last layer is composed of a convolutional layer that completely restore the image in the shape 512x512x4, where 4 is the number of features to segment.

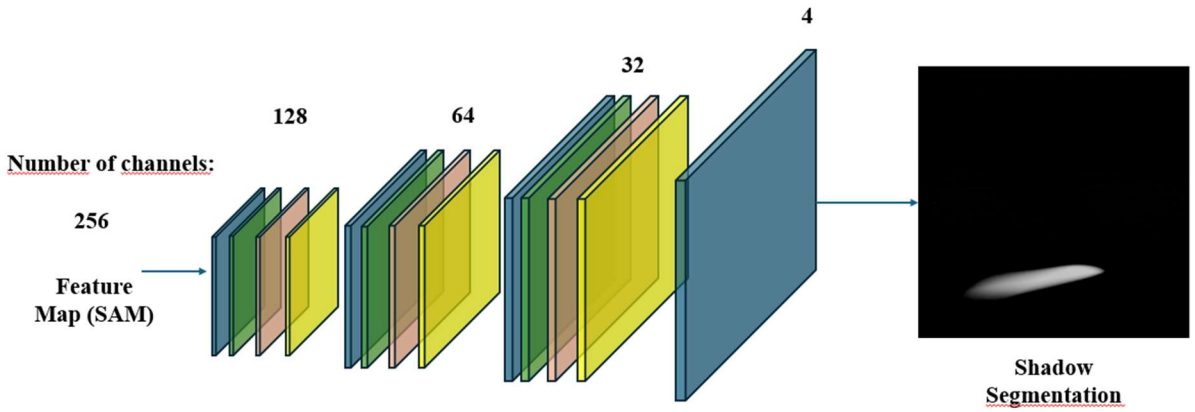


Figure 3.3: deconvolution architecture

The last component of the network is the ChangeMixin module, that takes as input the concatenation of the feature extracted with SAM. ChangeMixin module is composed of a first layer that expand the representation from 256 to 16 channels. After that, a variable number of layers, each one with convolution followed by a batch normalization and ReLU activation function are used, keeping the dept of the model to 16 channels. We used 4 of these layers and after we applied deconvolution with 3 transposed convolutions. ChangeMixin original structure has been slightly modified for the problem previously described, artifacts due to convolutional layers. In this case, the deconvolution is based on two blocks, each composed by convolution, batch normalization, ReLU activation and upsampling layer. At the end of the network the last convolutional layer brings the number of convolutional layer to 4, the number of change map needed. Within ChangeMixin, is performed the concatenation of Reference and Render input features (t1t2), and the concatenation of Render and Reference input feature (t2t1), the so called Temporal Swap Module; then t1t2 is concatenated with t2t1 and given to the layers previously described. Finally, the output is splitted in two change maps: one is the change map between Reference and Render, the other is the opposite, between Render and Reference.

3.2 Dataset

This thesis' aim is to detect the changes occurred between two images of the same scene from the same point of view, focusing on some aspects as light direction, reflections, texture and transparency. No existing dataset is available for this specific purpose, so the creation of a new datasets was necessary to train the network.

The purpose of the dataset is to simulate the potential error a student might make when attempting to recreate the Reference image. Each dataset is structured basing on the format of the exam, with five points of view of the same scene. To generate each dataset, 20 blender files were created, each one containing one camera that display different perspectives of the scene. For each Reference image, 19 render images were created, randomly varying light direction, intensity, radius, texture position, reflections of materials and transparency basing on the database. In total the dataset is composed by 2000 images (5 viewpoint x 20 files x 20 images), with a total number of Reference image equal to 100 and 1900 Reference. Each scene contains a maximum of 3 objects, and at least one object or the floor has a texture. Lighting is present in all scenes, while transparency is not represented in all files.
This amount of data made necessary the scripting of functions to automatically change the camera, light and material properties, in order to generate changes respect to the Reference image. This was done using the API Blender offers, that enable the possibility to access to object and material's properties, and to manipulate them. In total 5 datasets were generated with the following changes:

(a) Only illumination changes, modifying light parameters
(b) Only texture changes, modifying UV map of the material
(c) Only reflections, varying roughness of materials
(d) Only transparency, varying alpha value of the material
(e) Illumination and texture, varying for each image light parameters and UV map simultaneously

Along with Reference and Render images, each dataset includes the annotations of the features. On each dataset the corresponding feature labels (texture, shadow, reflection or transparency) are generated for Reference and for Render image, with a total of 2000 (20 files x 5 viewpoint x 20 images x 1 features) segmentation images. For each pair of labels, a change map is extracted, in the amount 1900 binary change maps (20 files x 5 viewpoint x 19 pairs x 1 features). In the case of the last dataset, there are 4000 segmentation images, because two features are considered, and 3800 change maps. At the beginning of the project a small sized database with 1920x1080 images was generated, and it was used only with FC-Siam-Conc and FresUNet strategy 2. The modalities which annotations and change maps were created is explained in the chapter 4.

3.3 Technologies

In this section we present the technologies used to generate dataset, to refine labels and implement the training and testing functions.

3.3.1 Blender

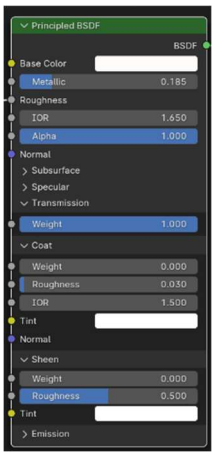


Figure 3.3: material node

Blender [20] is an open-source creation suite that supports the entire 3D rendering pipeline, including modelling, rigging, animation, creation of materials, simulation, compositing, motion tracking, rendering, video editing and game creation. Featuring an extensive Python API (Application Program Interface), Blender offers the possibilities for extending and customizing the scene, allowing users to automate repetitive tasks, develop complex tools or workflows specific to their needs. This is possible due to the possibility to access to nearly every aspect of Blender's functionality, such as modelling, animation, rendering, and simulation. In this project Blender API has been used to randomize the position and settings of lights and cameras, such as properties of the materials. Blender is a cross-platform tool, and runs equally well on Linux, Windows, and Macintosh computers.

Blender offers two main editing methods, Object mode and Edit mode. In Object Mode, it’s possible to adjust global characteristics such as position, rotation, and scale, while in Edit Mode, the geometry of the object can be modified by manipulating vertices, faces, and edges. The more aesthetic aspects of the scene, as materials, lighting, texture and background are controlled by shading nodes [Figure 3.3]. These nodes permit to simulate realistic material properties, and their parameters define how the material interact with the incident light; all the parameters proposed by the nodes modify the material properties and the way the material looks.

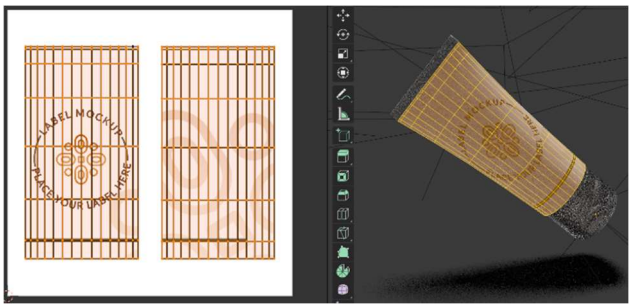


Figure 3.4: UV mapping

As stated above, material's appearance is not influenced only by its properties, but also by the light. Multiple combinations of material settings and lighting might assume the same visual result. Once these aspects are defined it becomes challenging to determine whether the lighting and materials in the reference and rendered images are identical based simply on material visual aspect. To address this, additional parameters are considered to detect changes. Shadow discrepancies allow to detect changes in the direction or angle/radius of the light.

Moreover, the texture can be integrated as a part of the material. The correct placement of texture is possible thanks to UV mapping procedure [Figure 3.4]. This is a possibility of 3D objects to make a planar mapping of the faces, so that 2D image (texture) can be correctly placed in the corresponding 3D face.

Finally, Blender offers a variety of properties for lighting, to achieve the most realistic illumination for every kind of scenery. It's possible to modify the direction, intensity, angle/radius, colour and many other properties [Figure 3.5].

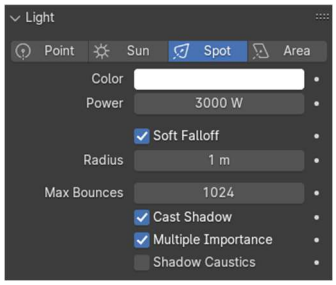


Figure 3.5: Light options

One of the most powerful function Blender offers, is the separation of rendering passes. To render an image, it is necessary to combine several layers, each one representing a specific information of the image. These layers are called passes, and isolate distinct aspects of the scene, as colour, lighting, and shadows Blender renders each pass separately and then combines them to produce the final image, as illustrated in figure 3.6. Blender gives the possibility to edit the render passes allowing more control on the final output. In the present

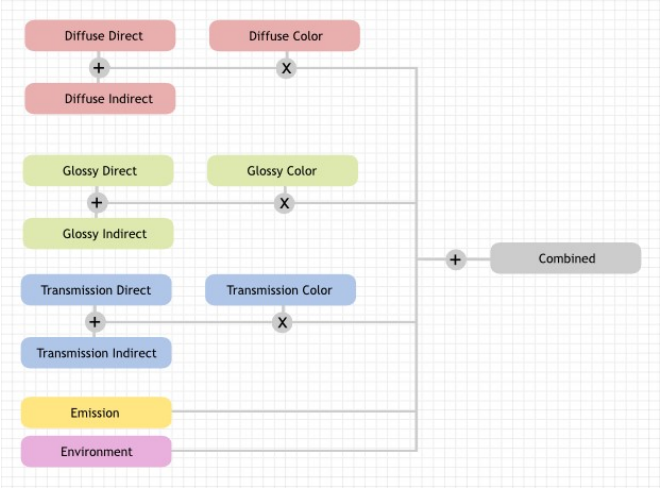


Figure 3.6: rendering passes

case, this possibility is exploited to annotate data. Specifically, the shadow pass was used for shadows, the transmission colour pass for transparency, the material index (or object index in some cases) for texture, and glossy direct pass for reflections [Figure 3.7]. As shown, the usage of two different view layers is necessary to render both shadow pass and image at the same time. A color ramp was set after the glossy direct pass, to eliminate weak reflections and ensure that only bright reflections were annotated. For this reason the lower parameter of color ramp is set to 0.99. Another color ramp was added after the shadow pass. In this case the mode was set to "ease" instead of linear, to avoid the noisy detection of the shadow. In this case each Blender file has its own upper limit value, to ensure a more accurate annotation.

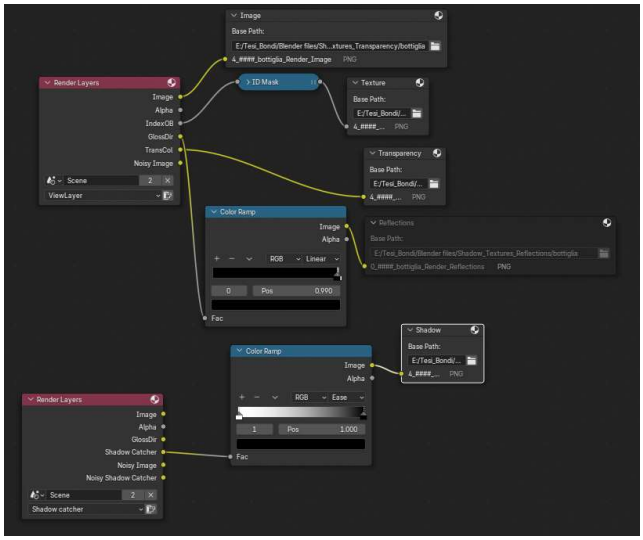


Figure 3.7: dataset annotation

Blender offers two rendering engines, Cycles and EEVE. EEVE is a more recent engine designed for real-time rendering: it's faster but less precise than Cycles. While EEVE is good for gaming and previews, Cycles is recommended for realistic renders. Cycles is a physic-based engine, it tries to match the real-world physics in the best way possible, accounting light bounces and reflections. As a result, Cycles takes longer to render a scene but delivers significantly more accurate results. The dataset was generated using Cycles, to better perform

on lighting and material results. All the images were rendered at 256 samples, with denoising. The aspect ratio of images, labels and change map is 1:1, with a resolution of 512x512.

Blender was selected for the creation of the dataset for three main reasons:

- Blender is the 3D rendering software used in the course related to this thesis
- The structure of Blender allows efficient control over the changes we want to monitor, as parameters of the light, position of the texture and material properties.
- The compositing window in Blender enables the separate rendering of multiple passes, facilitating the creation of annotated datasets for segmentation.
- Blender API offers the possibility to render a customized number of images, automating the manipulation of the properties of the objects that make up the scene, in order to have a wide variety of different combinations of features to monitor.

3.3.2 Python

Python [21] is an interpreted, high-level and object-oriented programming language, suitable for a range of applications. Its dynamic typing and binding, along with its high-level data structures, are especially useful for fast application development and for use as a connection language to connect various components together. Thanks to its simple, readable syntax, Python is easy to learn and helps keep code maintenance costs down. Python's support for modules and packages encourages modular design and makes code reuse simple. The interpreter and extensive standard library are free and widely available across major platforms, making Python both accessible and easy to share.

The advantage of using Python as a programming language, is to use libraries like Pytorch and OpenCV that allow access to advanced features in complex areas such as deep learning and artificial vision, greatly simplifying development work. In particular, Pytorch provides a flexible environment for the implementation and training of neural networks, supporting dynamic computational graph construction and GPU acceleration. Thanks to its scalable architecture, it's easy to test new model's architecture, making Python a fundamental tool in the deep learning field. In the same way, OpenCV is an essential resource for computer vision, offering a wide choice of algorithms for image processing, video manipulation and object recognition. With OpenCV, you can develop complex pipelines of pre-processing and image analysis, applicable in contexts such as facial recognition, semantic segmentation, and motion detection. The integration of such libraries in Python allows to reduce development time, while maintaining high efficiency thanks to the availability of optimized code and, in the case of PyTorch, the possibility of running parallel computing on GPU.

PyTorch [22] is a machine learning library that builds upon the Torch library for applications like computer vision and natural language processing, created by Meta AI. One of the most widely used deep learning frameworks, like TensorFlow and PaddlePaddle, provides free and open-source software that is released under the modified BSD license. Despite the main focus of development being on the polished Python interface, PyTorch also offers a C++ interface.

OpenCV [23] is one of the biggest open-source computer vision libraries for different programming languages, as Python, C++ and Java. It contains more than 2500 algorithms that works on various visual fields, as image and video manipulation, object and face detection, deep learning module and much more. This library is developed mainly for real time processing, making it suitable for applications that requires immediate feedback, as video surveillance, augmented reality, automotive.

4.Implementation

Image semantic Change Detection involves understanding which changes occurred in the same area through time. The purpose of this chapter is to describe the pipeline for detecting semantic change occurred between Render and Reference images. This pipeline integrates data generation, feature extraction, semantic detection, and techniques to identify and quantify changes in features as texture, transparency and shadow. The following sections outline each stage of the pipeline.

4.1 Pipeline

The Semantic Change Detection pipeline for images generated with rendering algorithms consist in the following components:

- Image and Labels generation
- Labels preprocessing
- Change map generation
- Semantic Change Detection
- Evaluation and interpretation

Each stage plays an important role ensuring the correct detection of changes between Render and Reference images. The correct generation of annotated dataset is the first step to ensure the right semantic recognition of changes. With improper labelling the network is tricked to learn incorrect segmentation, which leads to more incorrect Change Detection. It's important to tell that none of the pipeline's steps has a 100% accuracy, consequently the errors of each component sum up.

4.1.1 Image, Labels and Change Map Generation

The first step is the creation of an annotated dataset. For this task, a specific dataset is required, as it needs pairs of images generated by rendering algorithms, each one with specific labels. In order to detect changes in lighting, texture and material properties as reflections and transparency, it's necessary to annotate characteristics on the image that directly or indirectly represent the features. In this case texture, transparency and reflections are the direct feature to monitor, while light has no possibility to be directly annotated from the image, hence it's possible to annotate and use shadows as indirect representation of light.

As mentioned, the correctness of data is fundamental to ensure the correctness of the following steps. Images and labels were generated with Blender, with a combination of render passes and scripted functions thanks to Blender API. Shadow labels needed to be pre-processed before the creation of change maps, as the output of render pass was noisy, due to the nature of shadows, that has not a defined border. The last step before the training of the

network, was the creation of change maps. These were generated starting from annotations, using computer vision algorithms.

Details about the generation of images, annotations and change maps will be described in Chapter 4.2.

4.1.2 Semantic Change Detection

The primary focus on these networks is to set and test the correct parameters and hyperparameters, ensuring the best performance achievable by the network. During the training phase the dataset is divided into three sets: training, validation and test. Training set contains the images on which the model is based to learn the Semantic Change Detection, validation set is an observation set which contains the images used to evaluate the metrics and tuning hyperparameters on data the network has never seen. Finally, test set contains never seen images, with the same distribution of training set data, useful to find out the ability of the model to generalize on never seen data.
To enhance the robustness of the network, is useful to augment the variability of the dataset through data augmentation, with random flipping and rotation. These transformations are applied consistently across the Reference, Render, labels, and change maps to maintain data alignment. In this project data augmentation is crucial, because it allows to render more Render images for the same Reference, and still having a variety of data.
Other parameters to choose are loss function and optimizer. Loss function calculates the distance between the output of the network and the ground truth, and it's used to backpropagate the error through the network, adjusting the weights to minimize the difference between predictions of the network and ground truth. The optimizer has the main function of minimizing the loss by adjusting the network weights through gradient descent. The most common optimizers are Stochastic Gradient Descent (SGD), that offers stability in training, and Adam, a gradient descent method that leads to faster convergence. The learning rate of the optimizer is an hyperparameter that requires fine-tuning; a value too high can cause the training process to get over the optimal solution, while a value too low can slow the convergence of the model. Moreover, techniques like learning rate scheduling are often applied to dynamically adjust the learning rate during training, enhancing model performance and stability. The choice of loss function and optimizer, along with the tuning of their hyperparameters, is essential to achieving best results, as these components directly influence the network's ability to generalize well on new data.

The last parameter to face, is the one that concern class imbalance. Change Detection is usually an unbalanced task, so that the presence of changed pixels and non-changed pixel is strongly disproportionate. It mostly depends on the specific pair of images, but to provide an indication it is usually 10% of changed pixel compared to 90% of non-changed pixels. This involves that network receives high accuracy rate predicting all non-changed pixels and is not able to effectively detect changed pixels. For this purpose, it's crucial to set weighted loss, in order to give more prominence to class change.

Due to its architecture, Semantic Change Detection Network need a slightly different implementation respect traditional networks, starting from the loss. In this network we have two different task, Semantic Segmentation and Change Detection. For this reason, two losses are defined to monitor separately the performance of the branches and were initialized with different weights. To compute the total loss, CD loss and LCM loss are sum up with α factor.

4.1.3 Evaluation and Interpretation

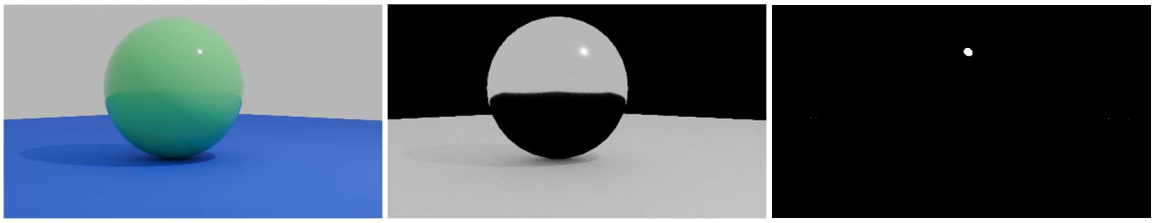
The last step of the pipeline is evaluation and interpretation of results. The above-mentioned loss is one of the metrics we have to define if the network is learning and generalising well, but loss depends on different parameters that can alter the result's comprehension. Loss is a good metric to see if the model is able to learn and generalize but gives no information about the accuracy of the results. Another metric often used is accuracy, which we have seen to be unrepresentative due to class unbalancing in this kind of task. For Semantic Change Detection, one of the best metrics we have to evaluate the performance of the network are precision, recall, F1-score and Intersection over Union (IoU). These metrics are the numerical representation of the correspondence between correct and incorrect predictions. The overview of these metrics, along with visual analysis of test predictions, can give a clue on the problem the network is facing and how to improve the regulation of hyperparameters. These metrics will be better discussed in the following paragraphs, after the details explanation of the generation of annotated dataset, while in Chapter 5 results of experiments are analysed.

4.2 Dataset Generation

The first thing to do to build the dataset is set a scene with a plane on which the objects rest. Some objects are suspended in the air, others are physically resting on the floor. The objects are of various types, maximum three per scene, and were taken partly by Blenderkit and partly built from basic blender shapes, modified in edit mode. In almost every scene there is at least one texture, set on the plane or on the object. To set the texture correctly it was necessary to do UV mapping on objects. In almost all scenes there is a transparency, obtained by modifying the transmission value of the material and the alpha value. Finally, a light was set for each scene, varying from scene to scene the type of light, angle/radius, colour, intensity. Two view layers were set, in order to allow the parallel rendering of shadow and other render passes, because Blender's options make transparent the object that catch the shadows and make them not visible to the camera. In the main view layer were enabled the combined pass, transmission color pass, glossy direct pass and material index/object index pass. In Shadow catcher view layer is enabled the shadow pass.

Two important details have to be considered: Blender does not offer the possibility to render auto-shadow, the shadow an object makes on itself, making the precision of annotations limited. Another thing to consider is the definition of reflections. We wanted to annotate the specular reflections of the light on material, as the white point on the sphere in Figure 4.1, which mostly depends on the roughness parameter of the material but is influenced by other

factors as metallic value and specular IoR value. For this reason, the glossy direct render pass was enabled with shadow catcher, as it excluded from the render the plane. In fact, in many scenes, the plane was fully considered in glossy direct map, even if there were not the reflections we wanted to consider. In figure 4.1, is shown the result of glossy direct render pass. In the sphere is clear the region where the reflection is placed, but on the floor there is not a proper reflection. In this case the plane can be removed from the segmentation mask with a thresholding process, but in most cases the reflection considered by glossy direct was so strong that no thresholding could remove it. For simplicity we didn't consider also the mirror reflections either.



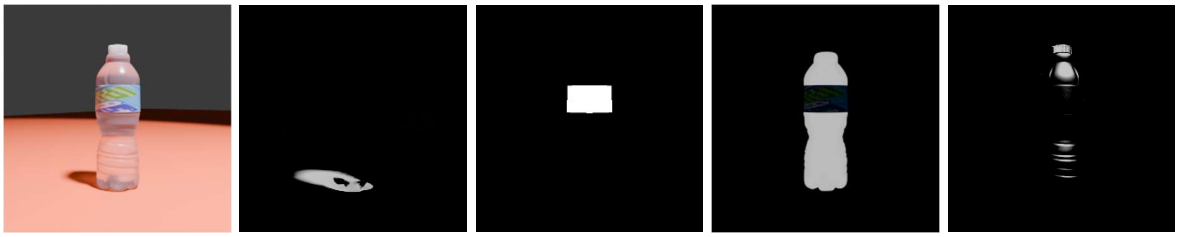
(a) (b) (c)

Figure 4.1: (a) Reference image, (b) glossy direct pass, (c) reflection segmentation

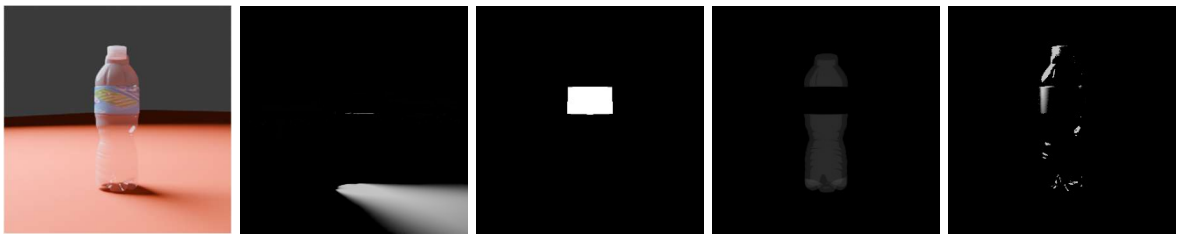
At this point, in the compositing window, are set all the parameters to render passes, the material index to render, and the colour ramp. The thresholding of reflections was made directly on compositing window, with color ramp node.

To generate the variability in material properties, camera position and light settings, Blender API was used. A script function was written to randomize some properties. First of all a *"track to"* constraint was set on the light and camera, in order always have the object in the frame and light up. As mentioned, each scene has 5 Reference image and 19 Render. Camera position in randomized 5 times, and other properties are changed 20 times for each camera position. For dataset (a), light position, angle/radius and intensity were randomized in reasonable range. Regarding dataset (b), to change the texture of the object UV map coordinates, rotation and scale were randomized in reasonable range. In the case of dataset (c) it was necessary to operate on roughness of the material. For dataset (d) alpha value of transparent materials was randomized between one render and the other. From script the save directory were dynamically changed, the view layers were enabled and render were made.

In figure 4.2 the example of annotations. In this case all the features are changing in the same scene, it's only an example to see all the annotations together.



(a) (b) (c) (d) (e)



(f) (g) (h) (i) (j)

Figure 4.2: (a): Reference image, (b): Reference shadow label, (c): Reference texture label, (d): Reference transparency label, (e): Reference reflections label, (f): Reference image, (g): Render shadow label, (h): Render texture label, (i): Render transparency label, (j) Render reflection label.

OpenCV offers a variety of image processing algorithms that allow to elaborate images quickly. The first version of segmentation labels we have is the one that comes out from Blender. These labels are in a grayscale format, and they need to be transformed in binary masks, with value 0 if the pixel does not belong to the feature, 1 otherwise. Before converting the annotations from grayscale to binary, we analysed the grayscale images, to understand how to make this conversion. The way to convert the mask, depend on the feature to consider. For texture and transparency, the grayscale image is pretty clear, because in the former is rendered the material with the texture, and the latter is rendered with transmission colour pass, so only objects with transmission different from zero are rendered. In this case, all the pixels with values over the threshold of 20 were set to 255, ad then to 1 in binary scale. For reflections as we stated before, preprocessing was made directly in Blender compositing window with the colour ramp. In fact, for each scene was set a lower threshold which set to 0 all the values under the threshold.

For shadow annotation the procedure was slightly different. In fact, this kind of feature, has not limited border, and values are not well defined. There are areas in the image, where the shadow pass was stronger, and other areas where the intensity of the shadow is minor [Figure 4.3]. The procedure descripted before is not appropriate for this feature, because with setting as threshold the minimum value the risk is to have a big area representing the shadow, that the human eye don't see in its entirety, or at contrary having a little shadow annotated even if the visible one is bigger. For this reason, shadow annotations from Blender were pre-processed with an erosion function, that eliminated all the *"alone"* white pixels, and threshold was set to 20.

Figure 4.3: shadow grayscale annotation

Figure 4.4: binary and grayscale CM

The last step to build the dataset was to create the change maps. The first part of the process was to mask reference and render image with the associated annotation mask, with a bitwise and operation. Then the masked images were compared computing the absolute difference. In this way, a new grayscale change map was created. In order to limit the detection of small differences, which are not visible by human eye, a threshold equal to 20 was applied to create the binary change map. In Figure 4.4 we can see the comparison between a grayscale change map and a binary change map.

4.3 Training and Testing

We trained two different architectures of Semantic Change Detection, first the Frequent (strategy 4) and after Change Star with pre-trained feature extractor SAM.

Dataset was divided into train, validation and test set with a division 60-20-20: the 60% of the dataset was assigned to training set, 20% to validation and test set each.

4.3.1 Loss function

Since both Semantic Segmentation and Change Detection can be seen as multi-label problem, the last layers on both networks are a Sigmoid function. One of the best choices to work with a multi-label problem is using a BCEWithLogitsLoss, which combine a sigmoid function together with a BCELoss, in a single function. As Pytorch documentation tells us [22], BCEWithLogitsLoss is more stable than using a Sigmoid followed by a BCELoss. Hence the last layer, which was Sigmoid, was removed, and BCEWithLogitsLoss was used. One of the parameters that can be set are the pos_weights, that helps with the problem of unbalanced classes; it accepts a tensor of weights, one per class, that balances negative samples and

positive samples. Two losses were used, one for Semantic Segmentation and one for Change Detection, in order to give the correct weights on both tasks. With the FresUNet architecture the total loss was computed as the sum of CD loss and LCM loss. The latter was weighted with α parameter set to 0.3. The equation 4.1 explain the computation:

$$Total\_loss=Loss\_CD + \alpha * (Loss\_LCM\_Reference+Loss\_LCM\_Render) \quad (4.1)$$

Instead, with ChangeStar architecture α was set to 1, to balance the tasks.
The difference between the two losses is due to the different architecture of the networks.

4.3.2 Weights

Weights were computed for each class as the sum of negative samples divided by the sum of positive samples; therefore, the loss can be weighted on this rate.

The parameter pos_weights was set both for Change Detection and Semantic Segmentation, since these are two unbalanced problems: as mentioned, Change Detection has a low positive rate, in particular considering separated classes, and the same logic can be applied for Semantic Segmentation. In general, the rate for single feature is highest for Semantic Segmentation than not for Change Detection, however both problems are unbalanced.

In this way the CD loss is given the weights of Change Detection, while SS loss is given the Semantic Segmentation weights, in order to better compute the global loss.

4.3.3 Optimizer, Scaler and Scheduler

In order to reduce overfitting, Adam optimizer was implemented in both architectures.

In FresUNet, since one of the first problems encountered was vanishing gradient, we noted that all the layers named Bias had a gradient in the order of 1e-8. So, we decided to deactivate Bias, and to set separate learning rates for bias and other layers. We used a learning rate of 1e-2 for bias and 5e-3 for other layers. During ChangeStar training, instead, the learning rate was set to 5 e-4. In both architectures scheduler was used, in particular the OneCycleLR. This is a learning rate scheduler in PyTorch that optimizes the learning rate following a one-cycle policy. Throughout training, OneCycleLR adjusts the learning rate by gradually increasing it up to a specified maximum (max_lr) and then symmetrically decreasing it. This cyclical approach aids convergence by avoiding local minima, providing faster optimization, and achieving higher final accuracy. Max_lr was set to 1 e-3.
To make the training more efficient, GradScaler was used. GradScaler is a Pytorch's function which primary purpose is to facilitate automatic mixed precision (AMP). By scaling the gradients up before the backpropagation step, it helps prevent the underflow. The scaled gradients help maintain numerical stability and reduce memory consumption, making training faster without compromising model performance. This is especially beneficial for large neural networks where training speed and memory efficiency are critical.

4.3.4 Metrics

The numerical evaluation of the performance is fundamental to find out if the model is learning from data, and if it is able to generalise. Other than loss, which is computed also to do backpropagation, other metrics as accuracy are usually used to evaluate the results. However, in unbalanced problems as Change Detection and Semantic Segmentation, accuracy can be misleading. In fact, if the number of classes is unbalanced, it's easy for the model to predict samples only from that class to reach high accuracy, that's why weighted loss is needed. Metrics that effectively show the real performance of the model, are precision, recall, F-score and IoU.

To understand these metrics, it's necessary explain the concept of true positive (TP), true negative (TN), false positive (FP) and false negative (FN). These metrics are computed pixel-a-pixel: for each pixel, it's compared the class predicted with the ground truth class. These metrics are the sum of all pixels where:

- Predicted and GT class equals to 1 (TP)
- Predicted and GT class are equal to 0 (TN)
- Predicted class is 1 but GT class is 0 (FP)
- Predicted class is 0 but GT class is 1 (FN)

This concept is visually explained in the so called "Confusion Matrix", in Figure 4.5.

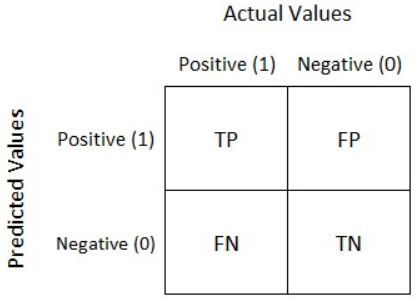


Figure 4.5: Confusion Matrix

Combining TP, TN, FP and FN it's possible to have an overview of the performance of the network, with metrics like F1, precision and recall, and are computed as follows in equations (4.2, 4.3, 4.4, 4.5):

$$P = \frac{TP}{TP + FP} \tag{4.2}$$

$$R = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1 = \frac{2 \text{ x } P \text{ x } R}{P+R} \tag{4.4}$$

$$IoU = \frac{TP}{TP+FP+FN} \tag{4.5}$$

where P and R represent Precision and Recall, respectively.

These metrics were computed both for train and validation set, in order to understand if the model was fitting the training data, and generalising on the validation set. These considerations can lead to figure out if the training is encountering problems as overfitting or underfitting. These metrics range from 0 to 1, and more precise is the model, more these metrics increase to 1.

A new metric could be introduced to evaluate the similarity between the Reference and Render images. This metric is based on the ratio of pixels identified as "changed" on the total number of pixels in the image. In order to consider also the segmentation output, the inverse of the Intersection over Union between Reference and Render feature's segmentation is computed. Equation 4.6 is able to give a percentage of the changes between Reference and Render images:

$$\frac{1}{IoU} * \frac{CP}{TotP} * 100$$

where CP is the number of pixel identified as "changed" in the Change Map, and TotP is the total number of pixel in the Change Map.

This metric is strongly based on the accuracy of change detection and segmentation of the network.

4.3.5 Training Details

The experimental framework for this study was established on a NVIDIA GeForce RTX 2080 Super GPU with 8 GB of memory, operating under the Windows 10 Pro system environment. Code was written in Visual Studio Code utilizing Python as programming language, Pytorch 2.3.0 and OpenCV 4.9.0.80 libraries.

5.Results and Analysis

In this chapter results are analysed, focusing on Semantic Change Detection strategies implemented, especially Strategy 4 of FresUNet and ChangeStar.

5.1 Performance Analysis

5.1.1 FC-Siam-Conc

One of the first attempt was made using the FC-Siam-Conc, which is designed to use only Change Maps without semantic annotations. This experiment was conducted training the network with changes in the direction of the light. Results are showed in Figure 5.1.

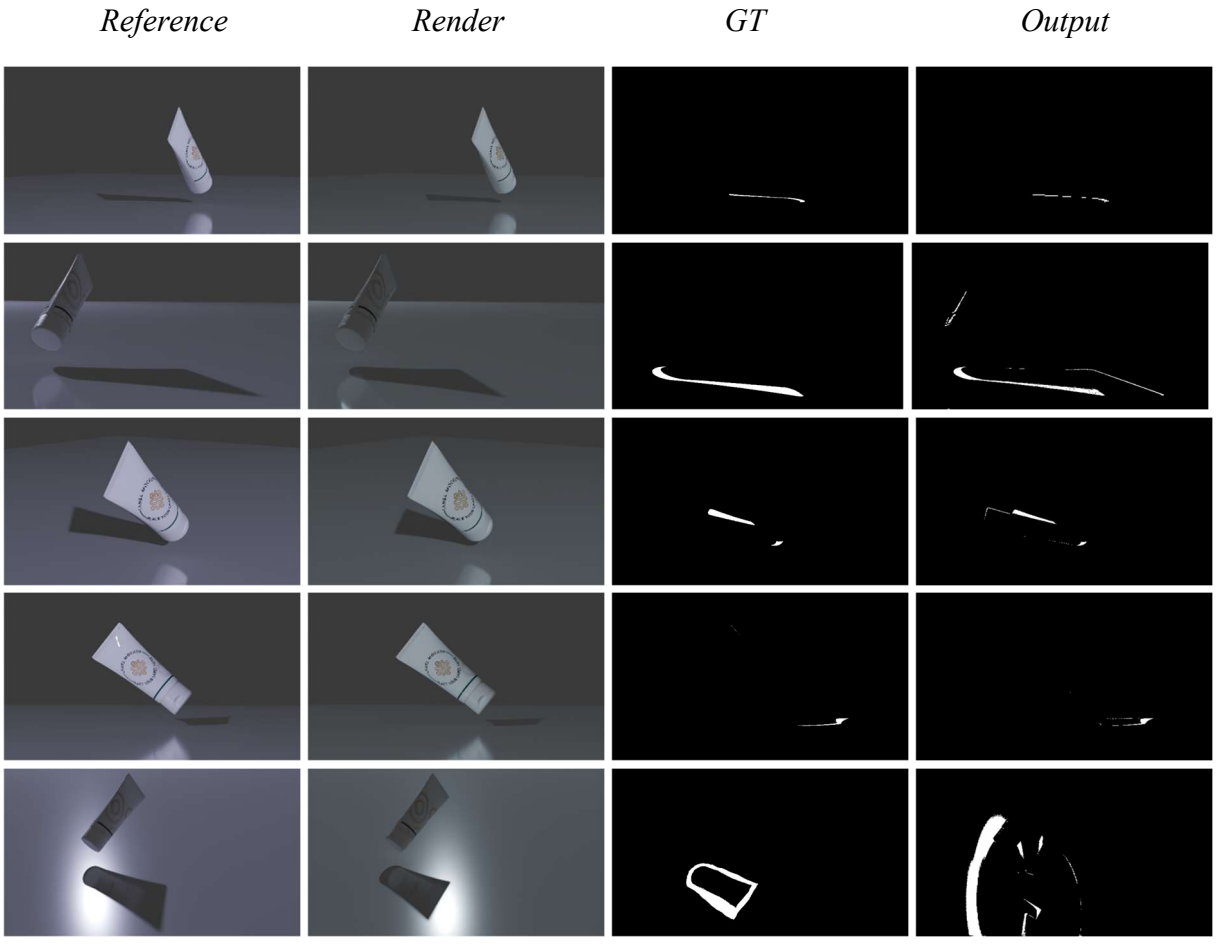


Figure 5.1: FC-Siam-Conc results

As we can see results sometimes detect changes not only in shadows but also in reflections, even if changes in shadows are correctly detected. The same model has been tested on images with multiple features changes.

Figure 5.2 represent the same object and the same viewpoint as the test above mentioned, but in this case two changes were applied: intensity and direction of the light.

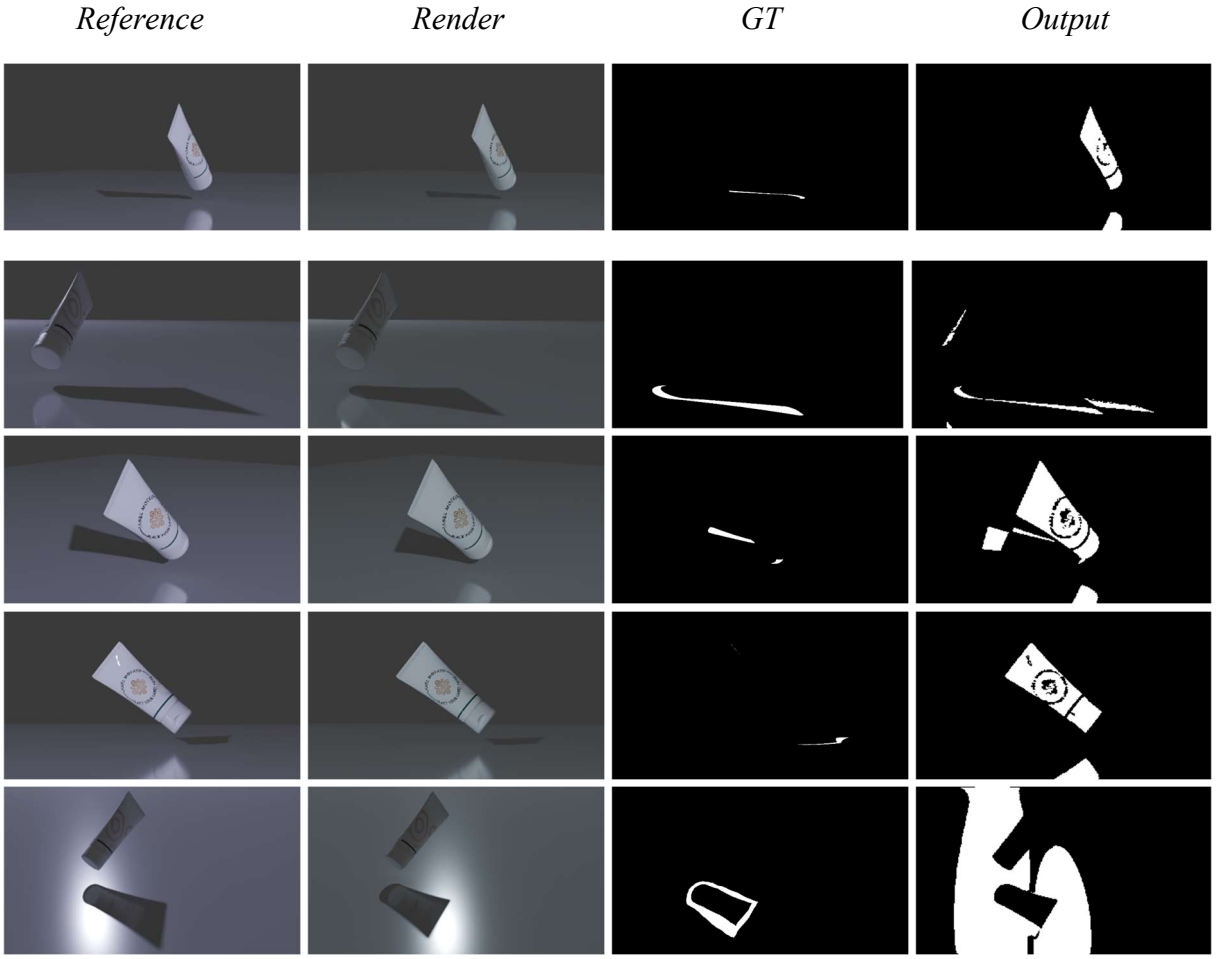


Figure 5.2: FC-Siam-Conc on multiple features changes images

With the same architecture and parameters, the results present a big difference.The results show that the network can detect changes in the image, but is not able to focus on shadows, detecting all the changes in the image, also the one regarding the intensity of the light, visible on the illumination of the tube.

At this point of the project, we decided to augment the dataset, including changes in transparencies and textures. For each pair of Reference and Render images three change maps were generated, one for textures, one for shadows (changing only direction of the light) and one for transparencies. The output of the network was changed in order to have three output channels. Due to the larger amount of images, it was necessary to resize input images to 512x512, in order to reduce computational time. To have an equal benchmark, also GT were resized.

Based on the results of the first experiment, it appeared that the network struggled to discriminate between different features. To assess whether the network was capable of distinguishing these features, a brief training session was conducted.

The results visible in figure 5.3 confirmed the hypothesis that the network is not able to distinguish between different features.

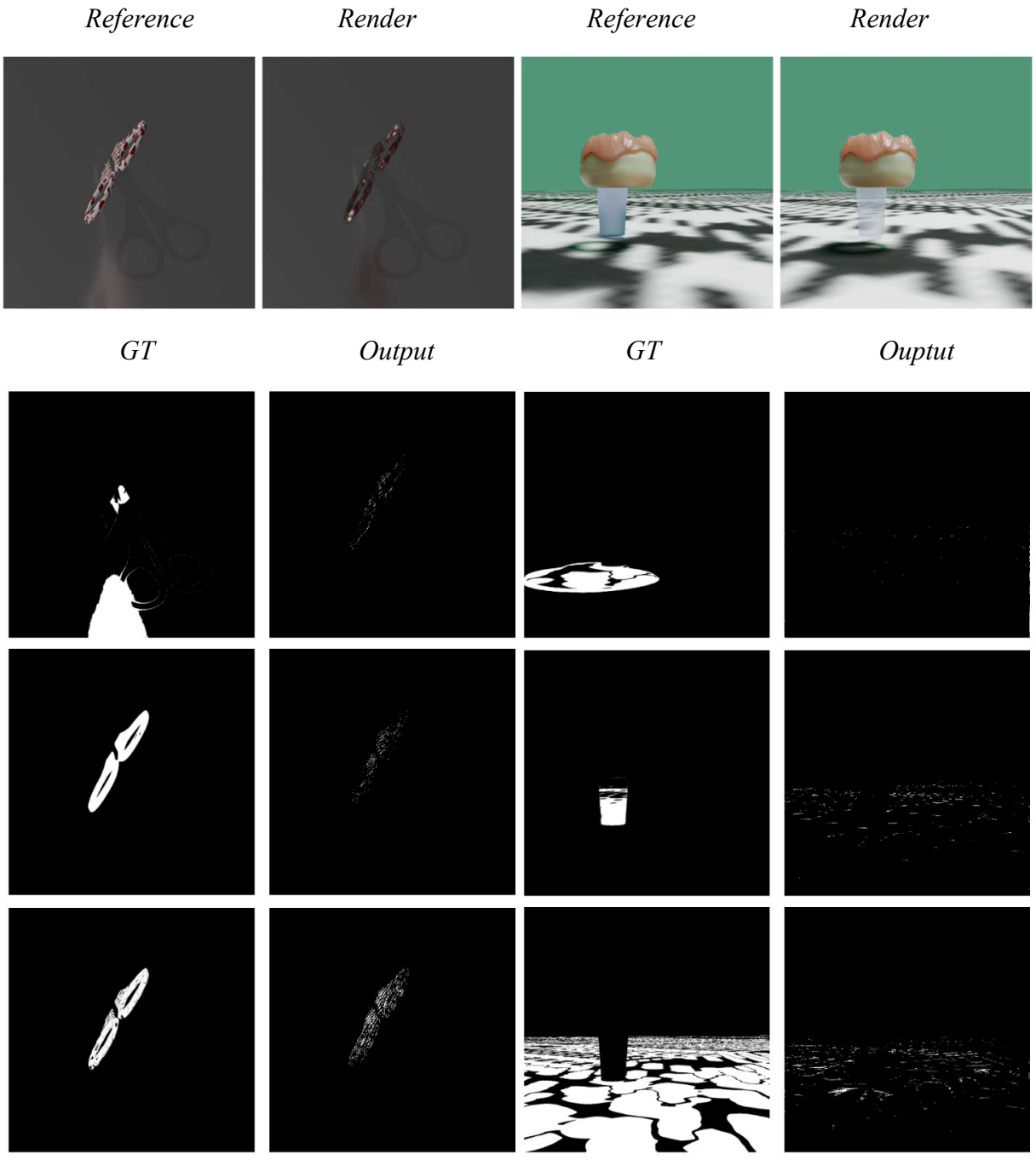


Figure 5.3: in the first raw Reference and Render images, in the second raw shadow CM, in the third raw transparency CM and in the fourth texture CM

It can be observed that the network is not able to distinguish different features, as the output CM are all similar for the same input pair.

Based on the results it was decided to conduct other experiments with FresUNet strategy 2.

5.1.2 FresUNet – Strategy 2

Strategy 2 of FresUNet is designed in order to perform Change Detection without segmentation annotations. The output and the parameters of the network are the same as FC-Siam-Diff, the difference is the architecture.

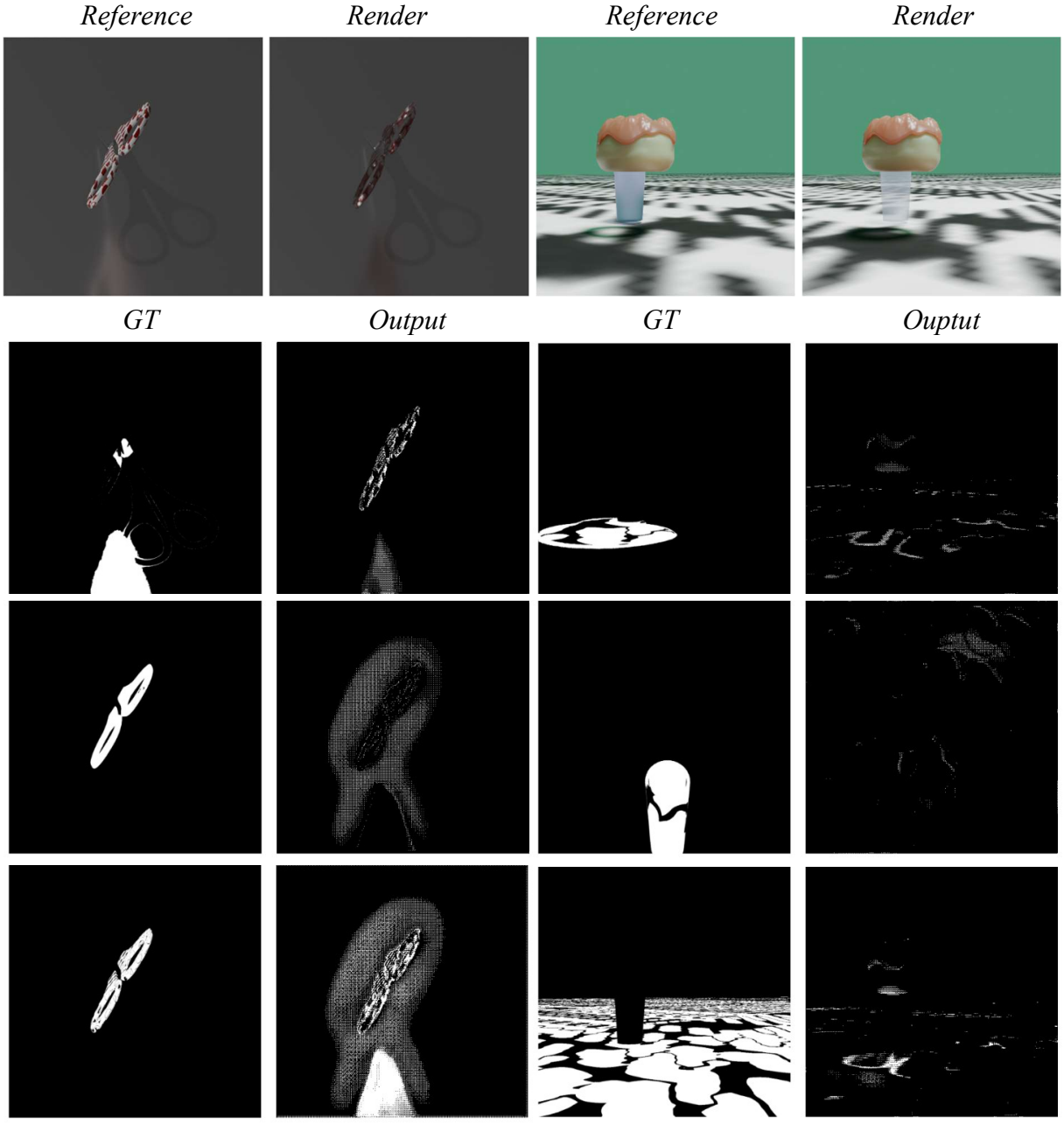


Figure 5.4: in the first raw Reference and Render images, in the second raw shadow CM, in the third raw transparency CM and in the fourth texture CM

It can be observed [Figure 5.4] that in texture and shadows the networks knows what to look for, but it's not able to separate the features. In fact, both in shadow change map and texture

change map bot features are detected. Regarding transparency, it seems that the network struggle in detecting transparent objects.

Relying on the outcomes and derived conclusions it was decided to insert Semantic Segmentation in Change Detection task, in order to enhance the performances and to help the network to understand the region to look for changes.

5.1.3 FresUNet – Strategy 4

Strategy 4 of FresUNet is designed in order to perform Change Detection with segmentation annotations. The network is composed from two parts, one for semantic segmentation and one for change detection. For this architecture four experiments were executed. For each experiment a specific database was build in order to train the network on a different feature. In each dataset only one feature change between Reference and Render image: illumination (detected by shadows), reflections, transparencies and textures. Each trained network was tested both on images with changes on the single feature, and on images with changes in multiple features. The reason behind this choice, lies on the fact that the risk of training a Change Detection Network with images that has only one change, lead the network to learn only to make the difference between images. In this way it's possible to understand if the network effectively learned to combine semantic segmentation and change detection.

To summarize, four experiments were conducted, each one using a different dataset where changes were present in only one feature:

1. Shadows: a new dataset was built, where Reference and Render images has as only difference the illumination. Shadows were annotated and Change Maps were generated. Tests were performed in two sets of images: in the first, belonging to this new dataset, only illumination changed. In the second set of test images, which not belongs to this new dataset, changes were carried out in textures, transparency and reflections.
2. Reflections: a new dataset was built, where Reference and Render images has as only difference the roughness of the material. Reflections were annotated and Change Maps were generated. Tests were performed in two sets of images: in the first, belonging to this new dataset, only roughness changed. In the second set of test images, which not belongs to this new dataset, changes were carried out in textures, transparency and illumination.
3. Texture: a new dataset was built, where Reference and Render images has as only difference the UV mapping of objects with texture applied. Textures were annotated and Change Maps were generated. Tests were performed in two sets of images: in the first, belonging to this new dataset, only UV map changed. In the second set of test images, which not belongs to this new dataset, changes were carried out in illumination, transparency and reflections.
4. Transparency: a new dataset was built, where Reference and Render images has as only difference the alpha value of the material. Transparencies were annotated and

Change Maps were generated. Tests were performed in two sets of images: in the first, belonging to this new dataset, only alpha value changed. In the second set of test images, which not belongs to this new dataset, changes were carried out in textures, illumination and reflections.

Experiment 1 - Shadows

The first experiment was carried out on a database with changes only in illumination, to ensure that the network had the capability to distinguish this specific feature.

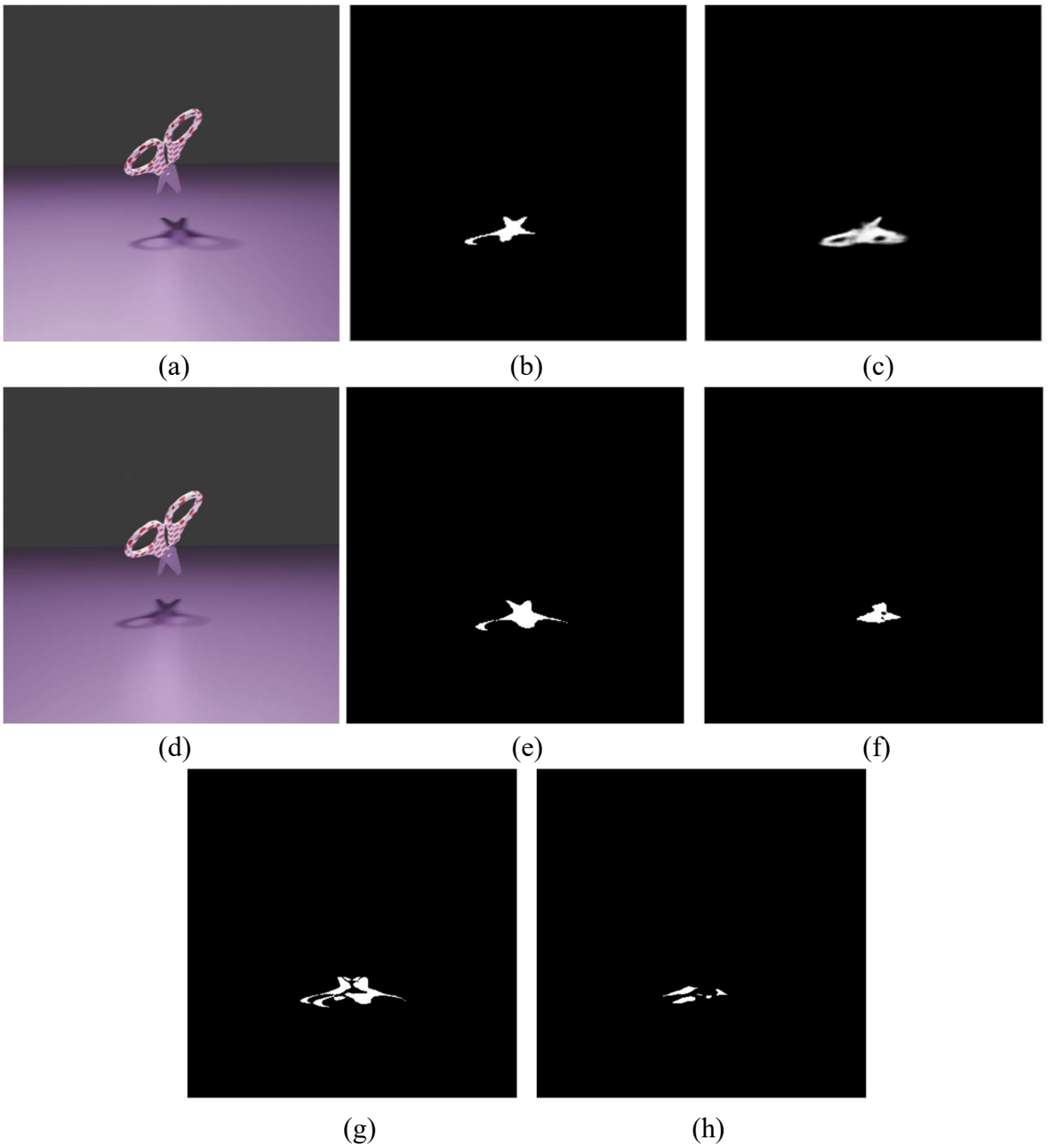


Figure 5.5: (a)-(b)-(c): Reference image - Shadow GT – Shadow Output, (d)-(e)-(f): Render image - Shadow GT – Shadow Output, (g)-(h): GT Change Map – Output Change Map

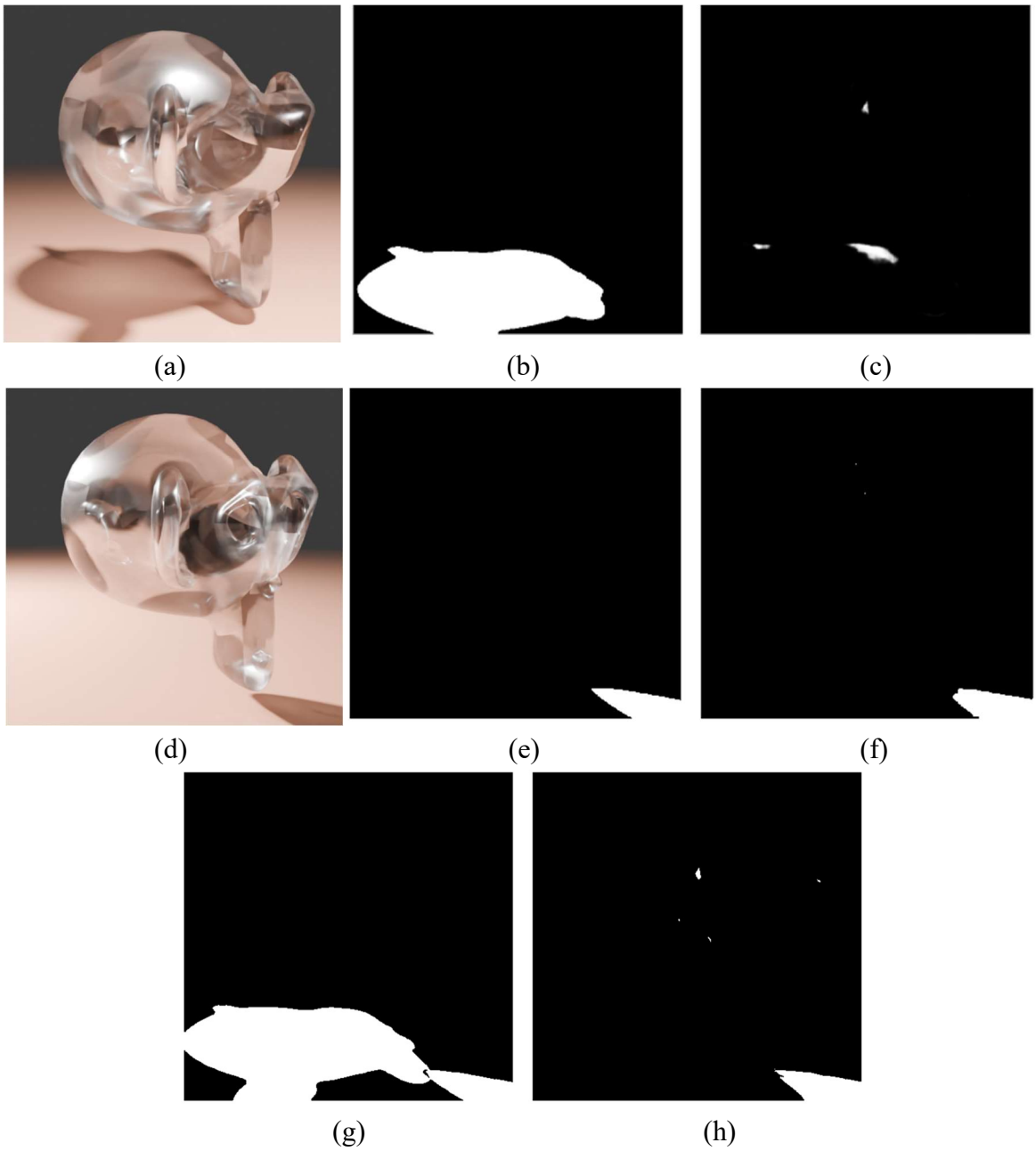


Figure 5.6: (a)-(b)-(c): Reference image - Shadow GT – Shadow Output, (d)-(e)-(f): Render image - Shadow GT – Shadow Output, (g)-(h): GT Change Map – Output Change Map

It can be observed in figure 5.5, that where the shadow is more soft, the network is able to find shadow only in the region where this is more defined. In fact, in figure 5.6, the shadow is well detected in Render image, because it is strongly separated from the floor, while in Reference only the darker parts of the shadow are segmented.

The change map output from the network follow the same logic.Change maps are based on the segmentation output, for this reason if the shadow is not correctly segmented the change map will be partially correct. This can be observed in the second change map: the segmentation of Reference image was only partially correct, in fact the change map is not considering the reference segmentation.

This network has been tested on other images, where the changes were not only in shadows but also in other features. This test was carried out for the purpose of understanding if the network learned only to make the difference between the images or is able to generalize on other features, performing the correct correlation between segmentation and change detection.

To illustrate the performance on difference features, we consider the figure 5.7 and 5.8.

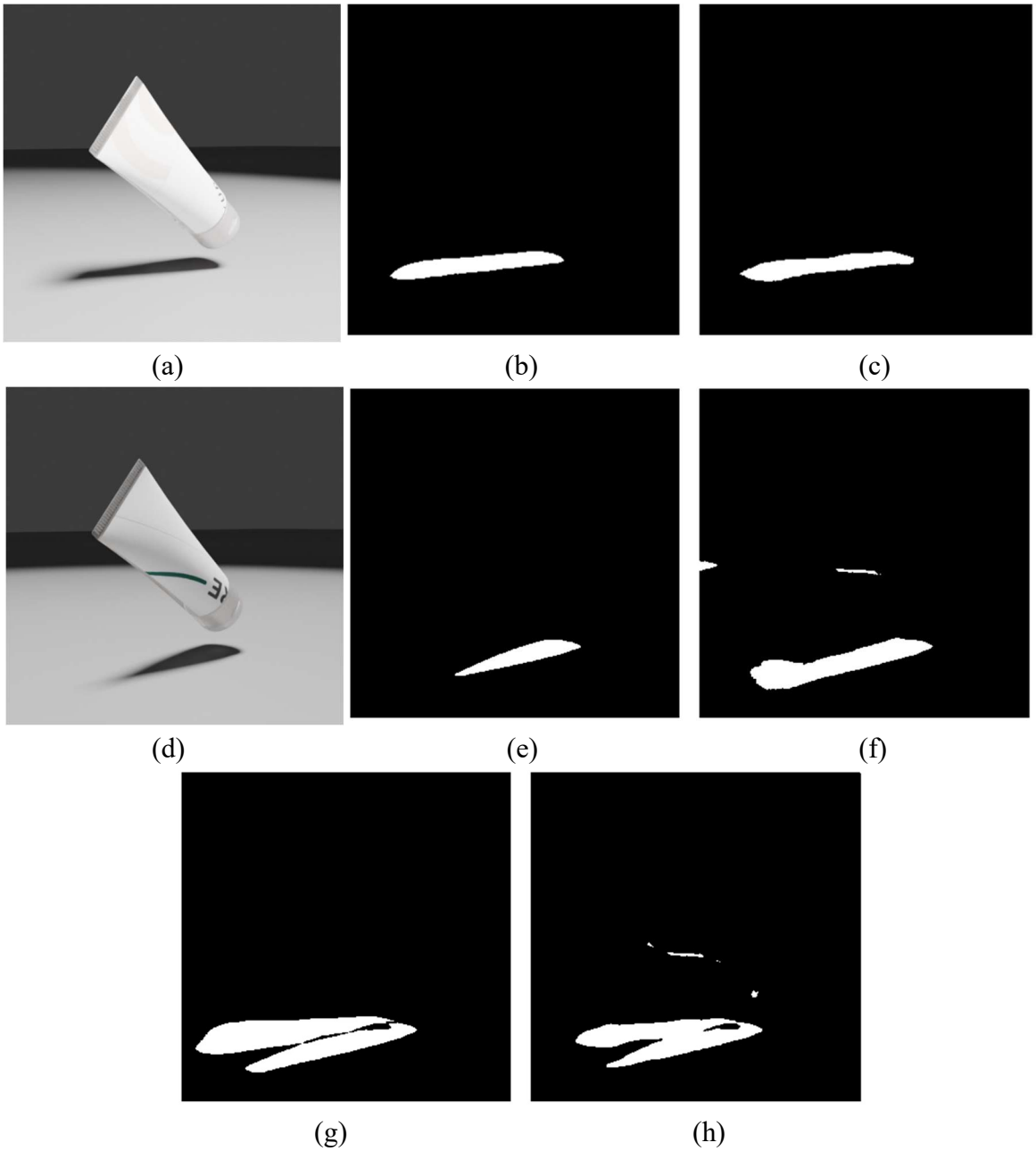


Figure 5.7: (a)-(b)-(c): Reference image - Shadow GT – Shadow Output, (d)-(e)-(f): Render image - Shadow GT – Shadow Output, (g)-(h): GT Change Map – Output Change Map

This image shows changes in other features than shadow: the texture is changed, the transparency is changed and also the material, which has a different reflection. These results show that the network is able to find shadow with strong border, even if it finds as shadow

parts of the texture and of the light cone. The change map slightly finds changes out of shadow but to a lesser extent than correct shadow changes.

To illustrate another example, in the following case the main changes are performed in transparency, texture and light.

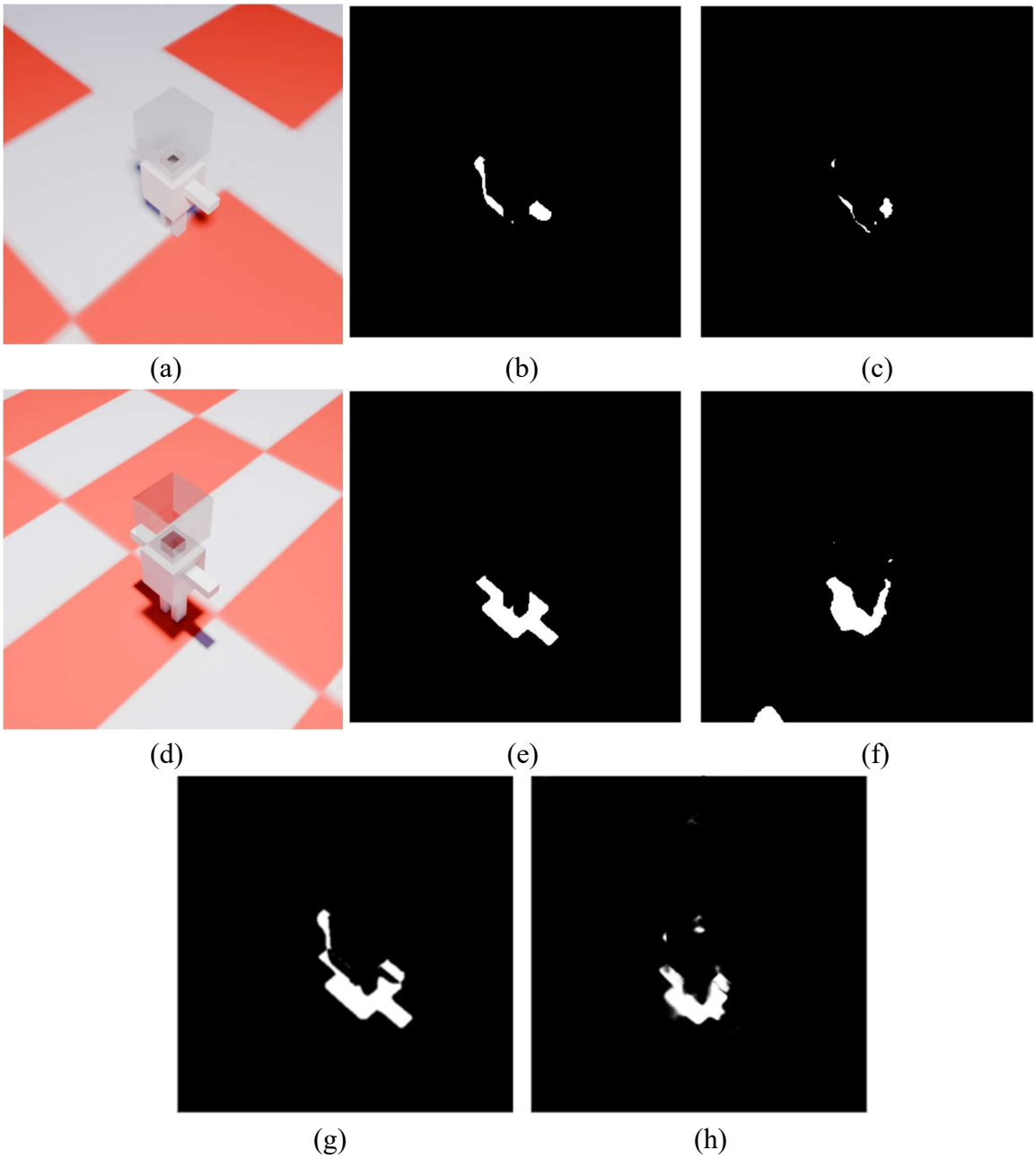


Figure 5.8: (a)-(b)-(c): Reference image - Shadow GT – Shadow Output, (d)-(e)-(f): Render image - Shadow GT – Shadow Output, (g)-(h): GT Change Map – Output Change Map

In this case the shadow is not correctly detected especially on Render annotation, which are irregular even if the shadow in Render image is well defined. Therefore, the Render annotation finds also a small piece of texture, even if it is not present in the change map. It

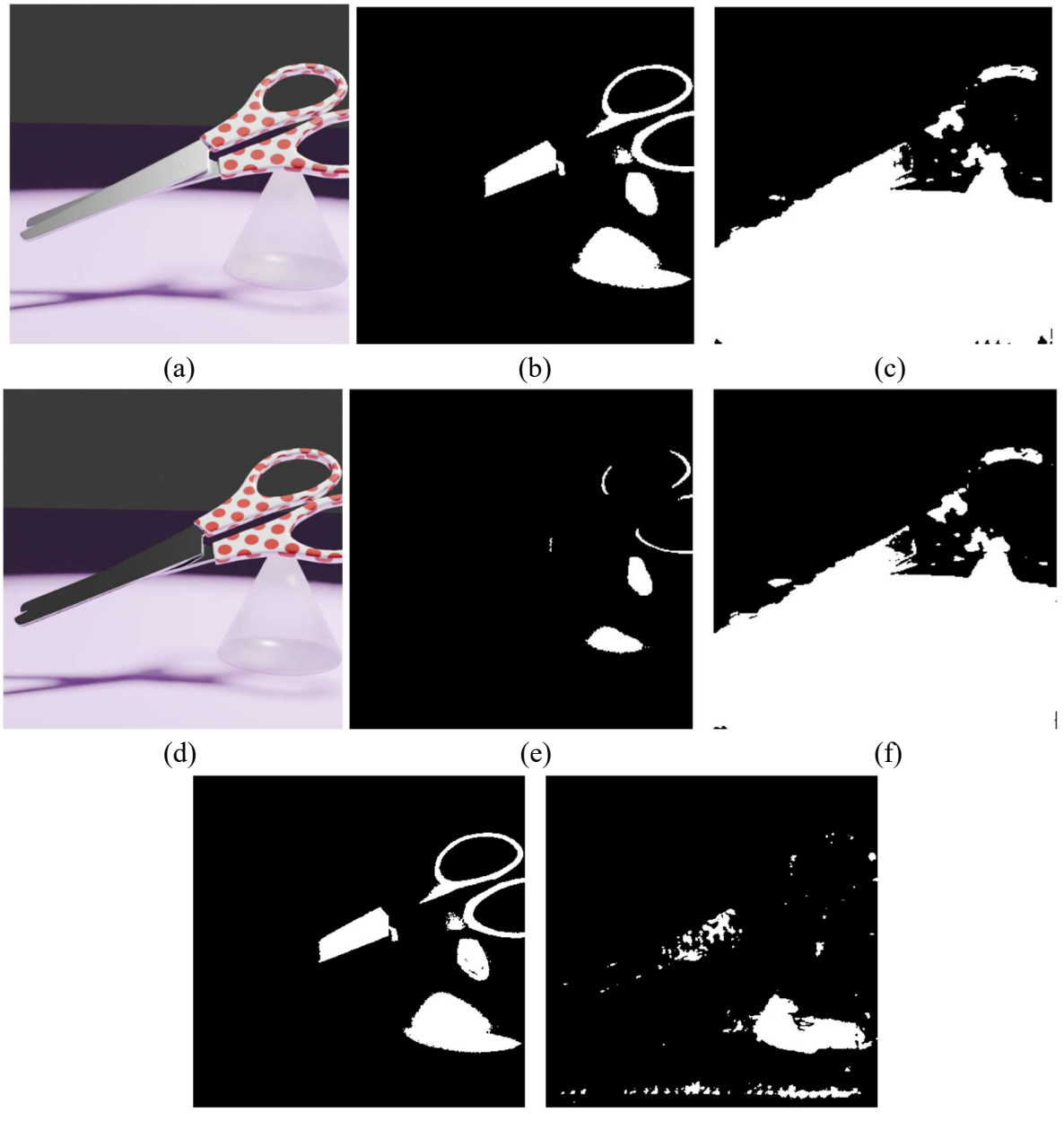
has been noted that shadow are better detected when they are strong and have an high contrast with the floor.

Experiment 2 – Reflections

The second experiment was conducted on images with changes only in reflections.

Reflections are trickier to detect, as also with human eye is not easy to understand is a reflection is soft but cover a large portion of the object. Another reason why reflections are difficult to detect is that they have not defined borders except if they are strong, so it's difficult to understand where the reflection stops.

The tests presented in figure 5.9 and 5.10 are executed on images with changes only in reflections.



(a) (b) (c)

(d) (e) (f)

(g) (h)

Figure 5.9: (a)-(b)-(c): Reference image - Reflections GT – Reflections Output, (d)-(e)-(f): Render image - Reflections GT – Reflections Output, (g)-(h): GT Change Map – Output Change Map

Even if the segmentation of reflections is not precise, as it takes all the floor inside the light cone, the change map is able to detect the region where changes in reflections occurs, even if it takes a small portion of the floor. The border of reflections, both on segmentation and change map, are not precise, but the network can correctly detect the region of the changing, even if not the precise pixels.

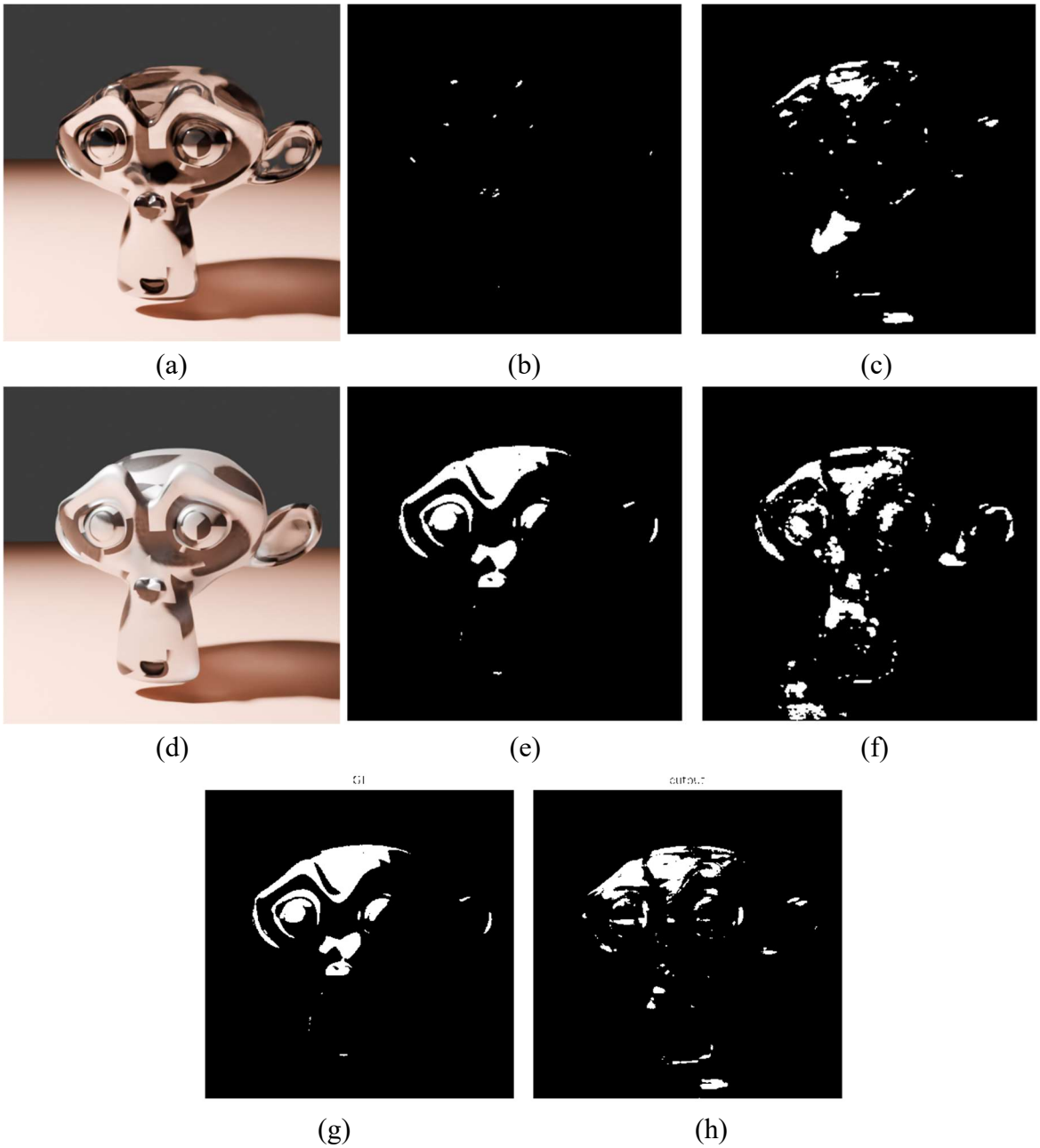


Figure 5.10: (a)-(b)-(c): Reference image - Reflections GT – Reflections Output, (d)-(e)-(f): Render image - Reflections GT – Reflections Output, (g)-(h): GT Change Map – Output Change Map

In this last case, it's possible to note that segmentation is more precise, even if a small section of the floor is annotated as reflection. The change map is similar to the GT in the upper part of the head of Suzanne, but it also determines the changes in sections without reflections.Test on images with other features changing were executed, and here are the results. Figure 5.11 and 5.12 presents changes in illumination, reflections, transparency and texture, in different intensities.

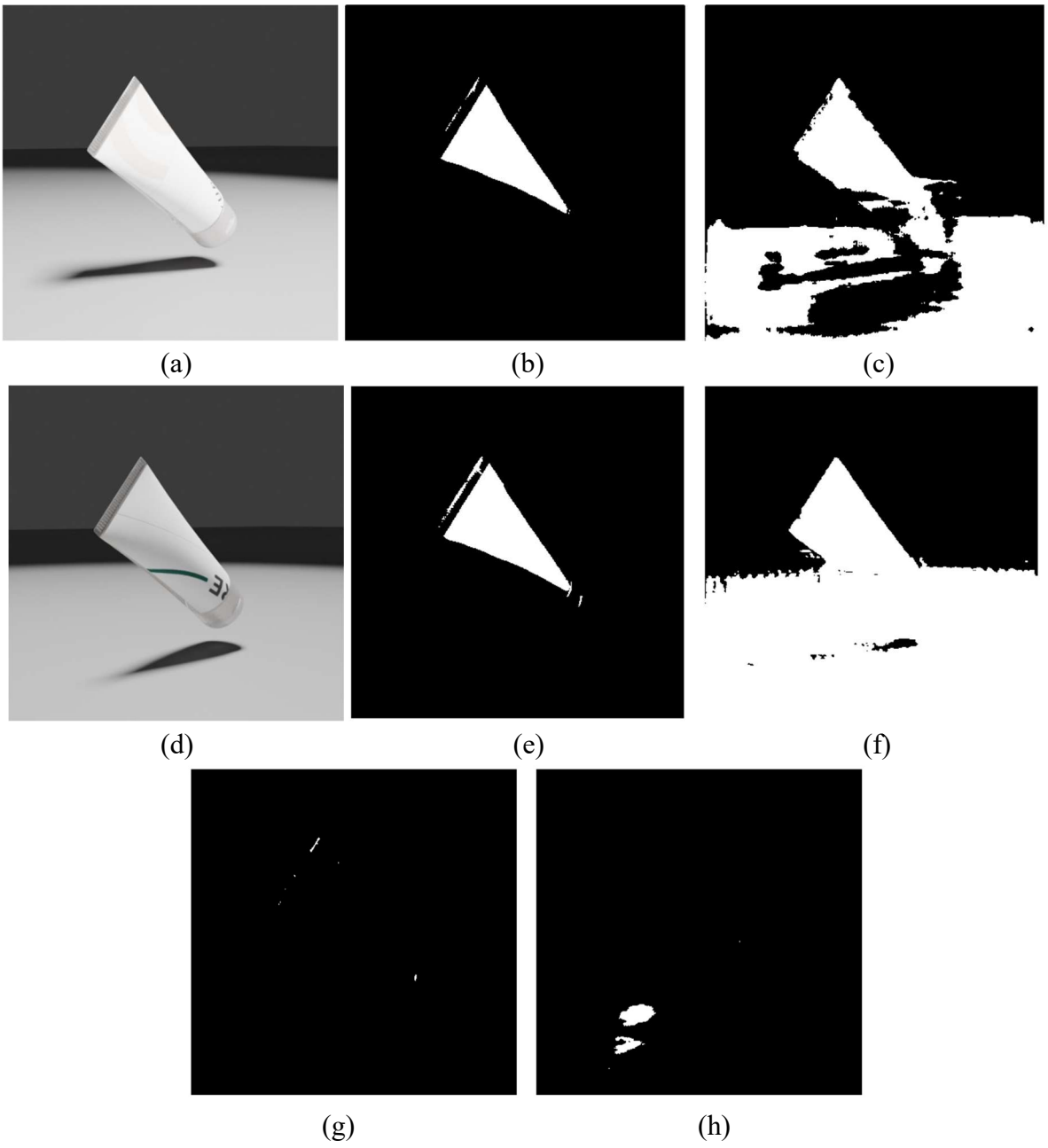


Figure 5.11: (a)-(b)-(c): Reference image - Reflections GT – Reflections Output, (d)-(e)-(f): Render image - Reflections GT – Reflections Output, (g)-(h): GT Change Map – Output Change Map

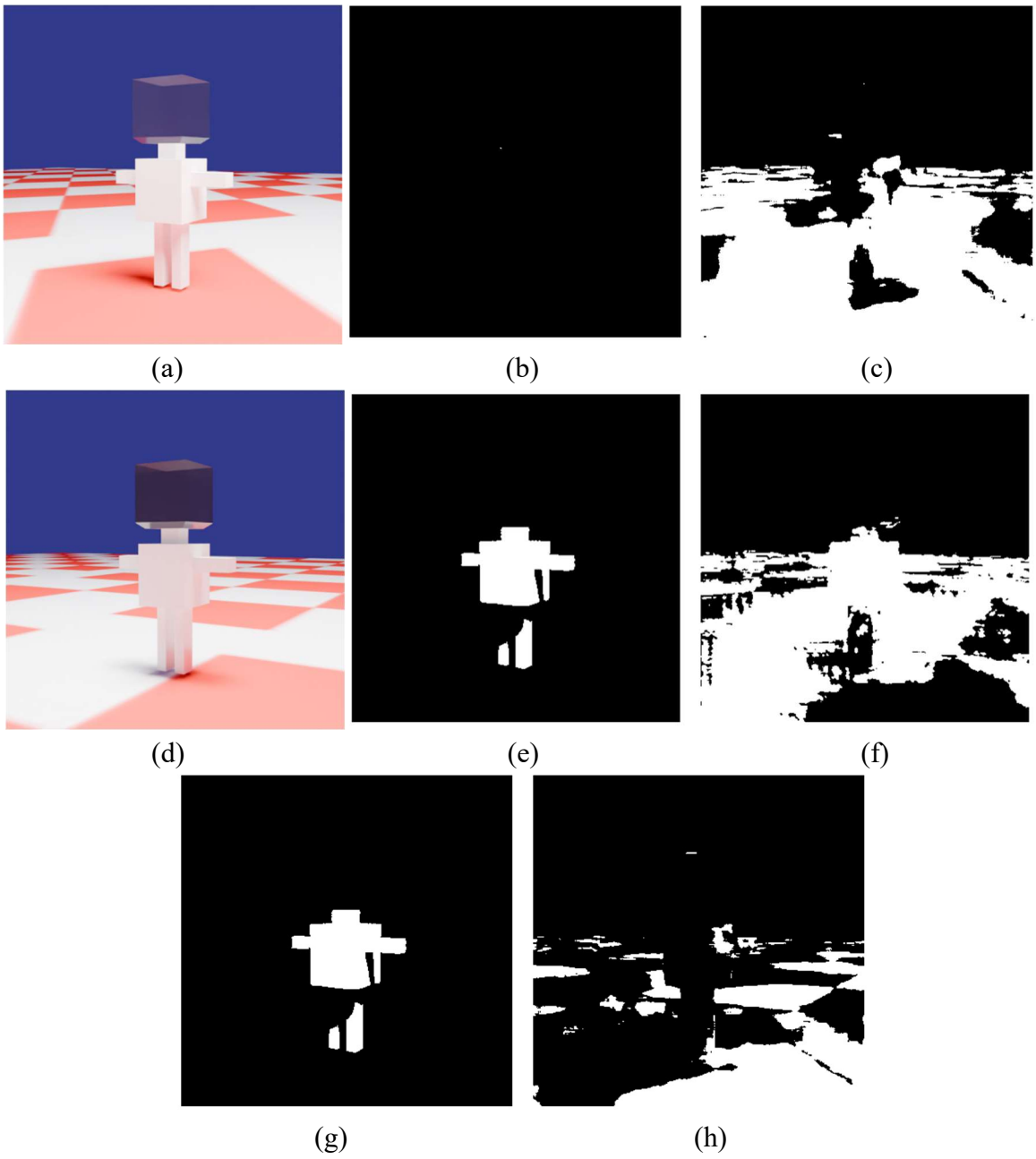


Figure 5.12: (a)-(b)-(c): Reference image - Reflections GT – Reflections Output, (d)-(e)-(f): Render image - Reflections GT – Reflections Output, (g)-(h): GT Change Map – Output Change Map

Results confirm that reflections are a difficult feature to detect, leading to a more evident scarcity of performance where the changes are more than just reflections. It seems that the network is taking as reflection all the significant parts of the image, struggling to isolate reflections from other features.

Experiment 3 – Texture

Experiment three focuses on textures. A new dataset, containing images with only texture's changes was generated, with the associated labels and change maps. The network was trained

and tested on images of the same dataset, that the network had never seen. Then it was tested on images where more features changed.

The results in the first test are shown in figure 5.13 and 5.14.

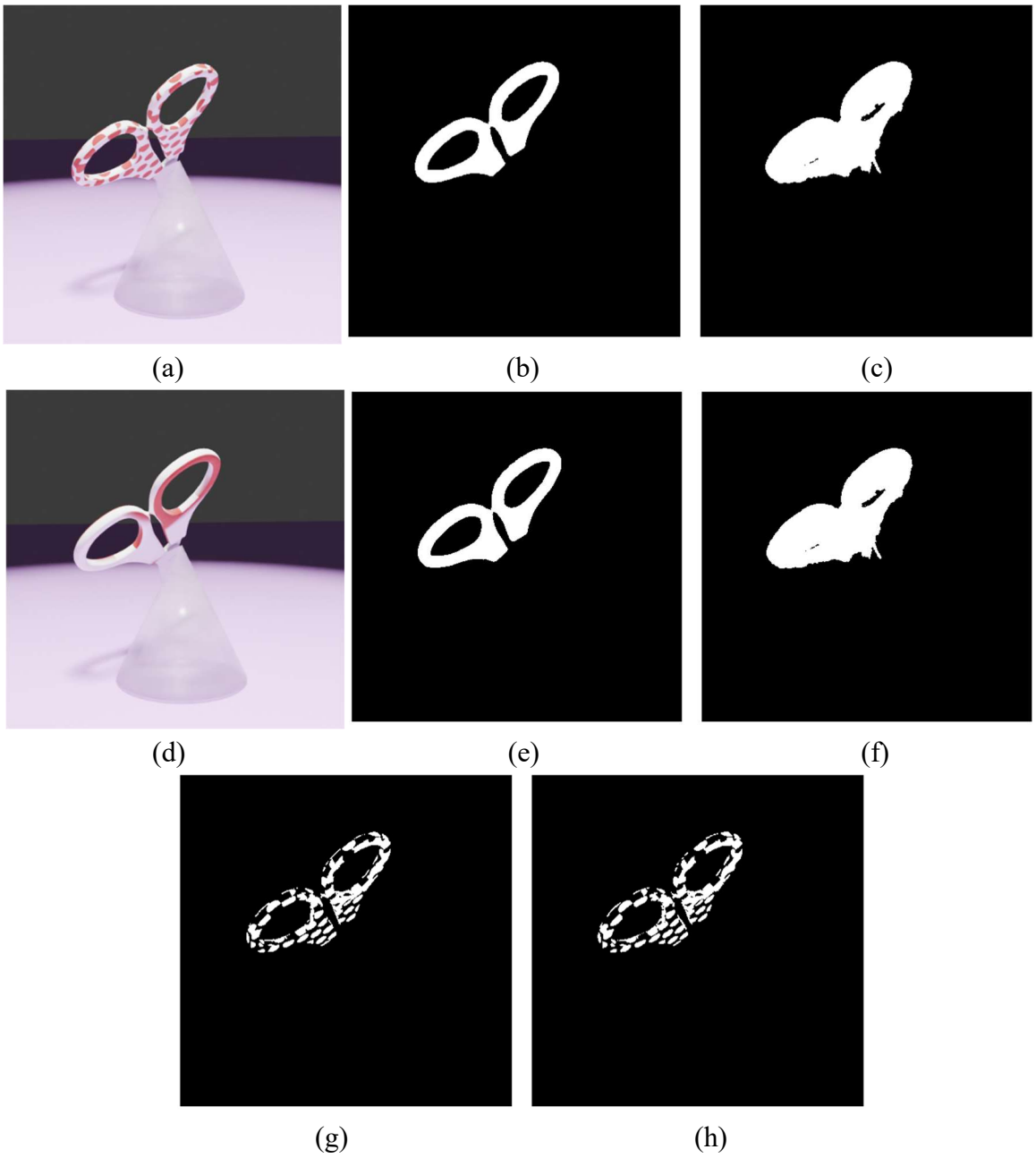


Figure 5.13: (a)-(b)-(c): Reference image - Texture GT – Texture Output, (d)-(e)-(f): Render image - Texture GT – Texture Output, (g)-(h): GT Change Map – Output Change Map

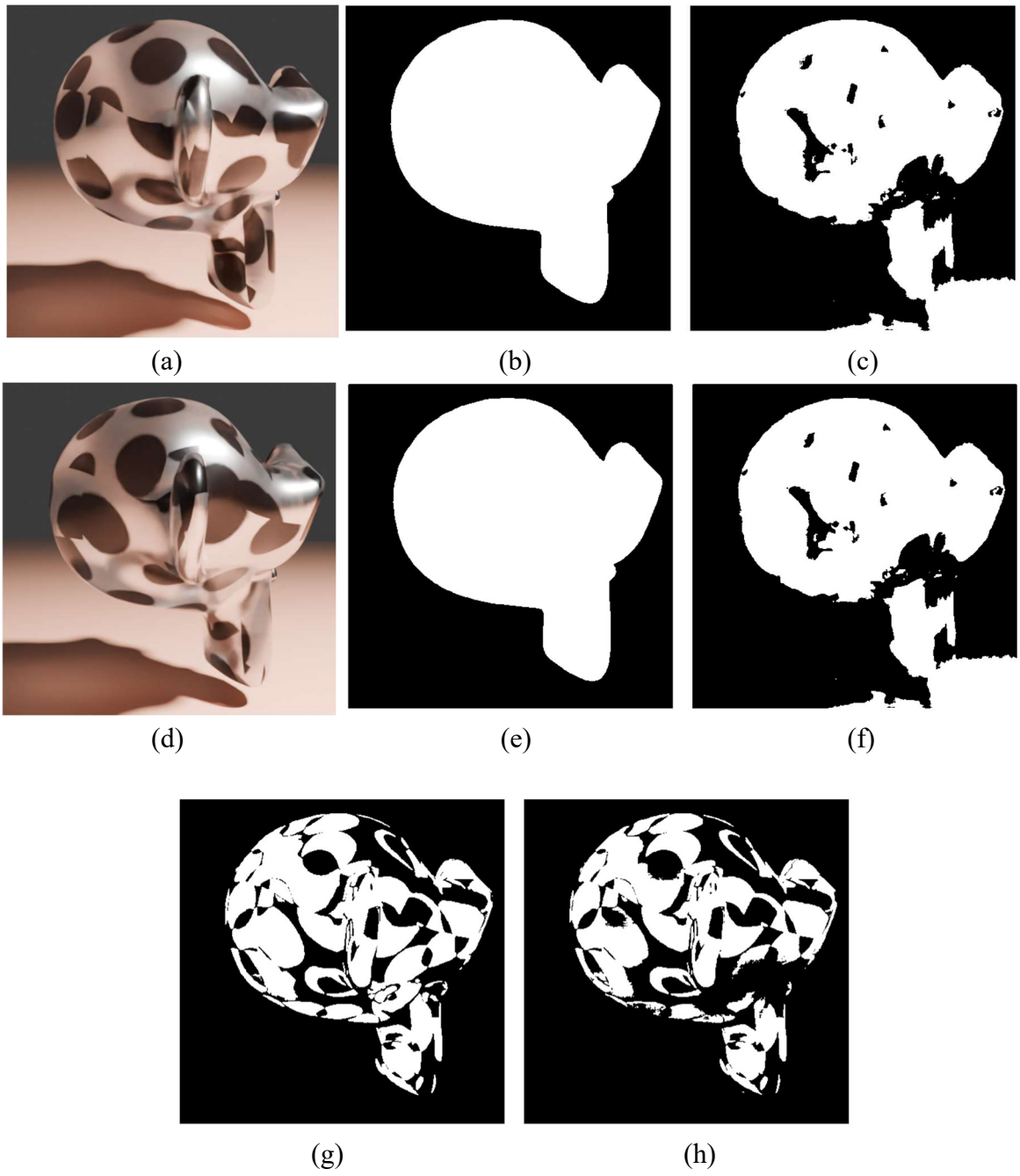


Figure 5.14: (a)-(b)-(c): Reference image - Texture GT – Texture Output, (d)-(e)-(f): Render image - Texture GT – Texture Output, (g)-(h): GT Change Map – Output Change Map

In this test annotations result mostly precise, except for some border that exceeds the real texture. The positive result is that change maps are not detecting these sections and are precise in detecting texture changes.

Figure 5.15 and 5.16 are the results on test made on images with changes in other features.

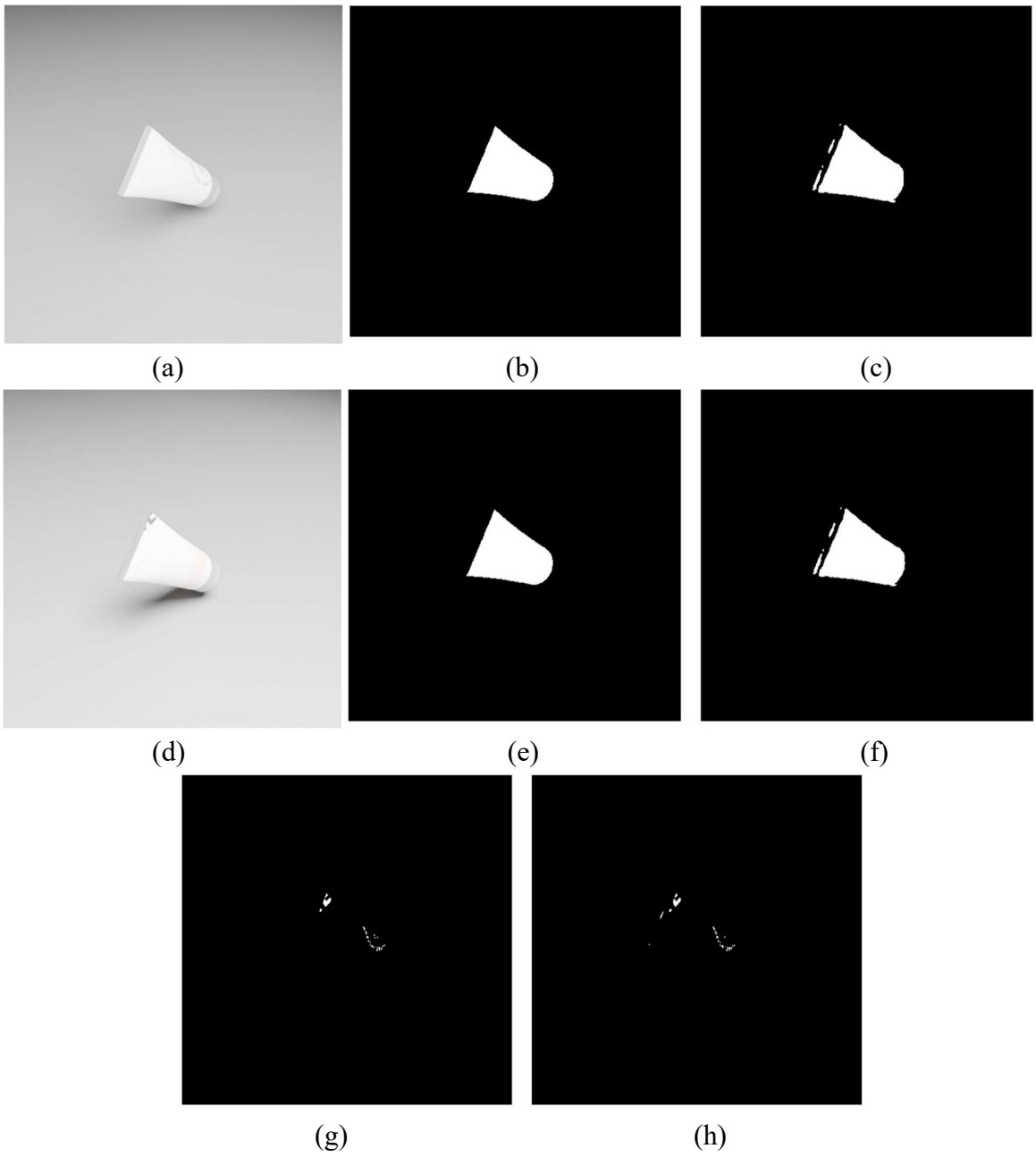


Figure 5.15: (a)-(b)-(c): Reference image - Texture GT – Texture Output, (d)-(e)-(f): Render image - Texture GT – Texture Output, (g)-(h): GT Change Map – Output Change Map

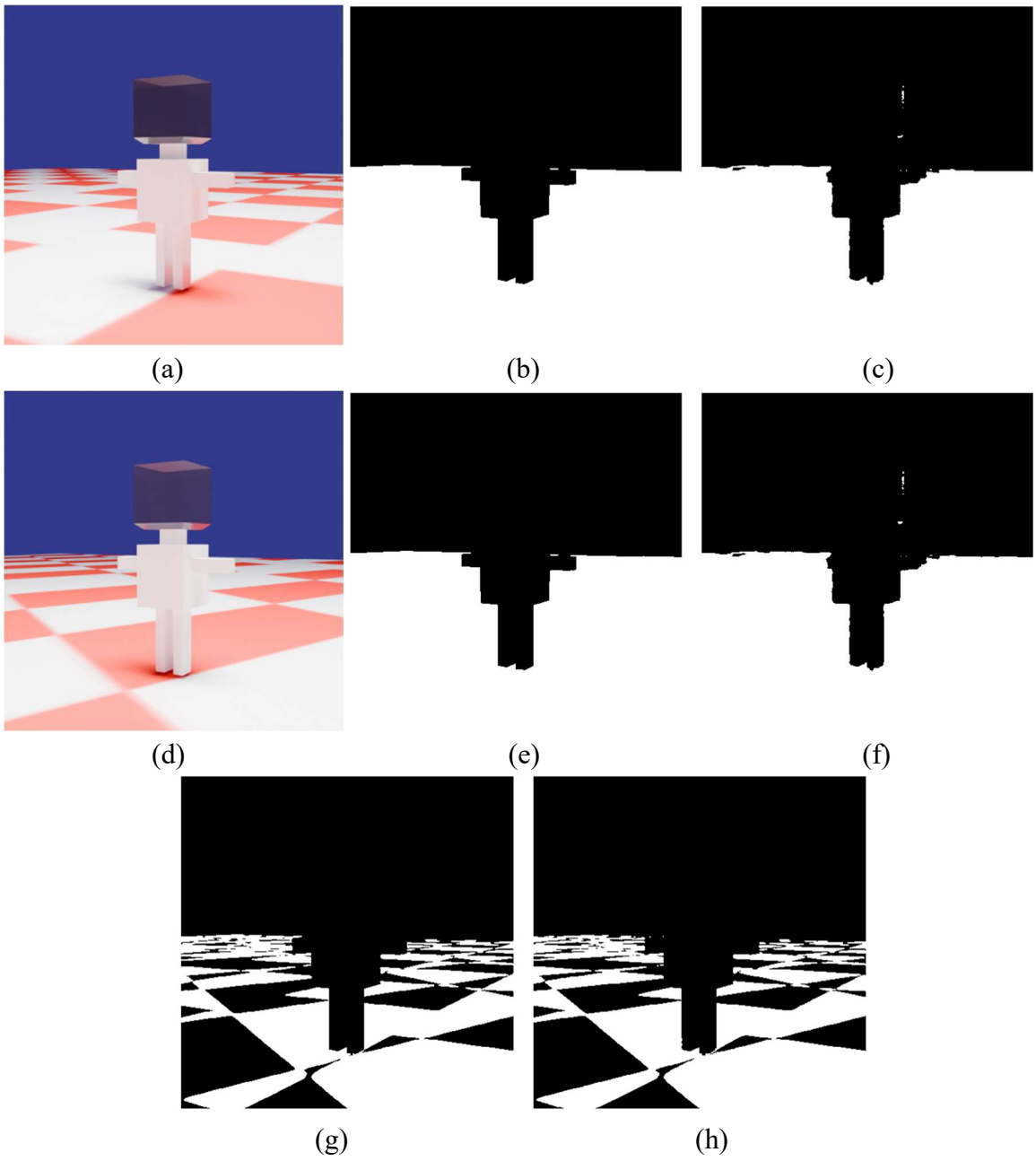


Figure 5.16: (a)-(b)-(c): Reference image - Texture GT – Texture Output, (d)-(e)-(f): Render image - Texture GT – Texture Output, (g)-(h): GT Change Map – Output Change Map

Also on images with more changes, the results are precise. In the first case, even if segmentation detected a small part of transparent material, the change map is equal to the GT, demonstrating that the network has not only learned to perform difference between images, but is able to correlate semantic segmentation with change detection.

Experiment 4 – Transparency

This experiment concern transparency of material, defined as the amount of light that can go through the object. The dataset was build generating Render and Reference images which have as only change the alpha and transmission parameters of the transparent object.

Therefore even there, it was decided to conduct two test, the former on images with only transparency changes, and the latter on images with multiple feature changes. Figure 5.17 and 5.18 show the results.

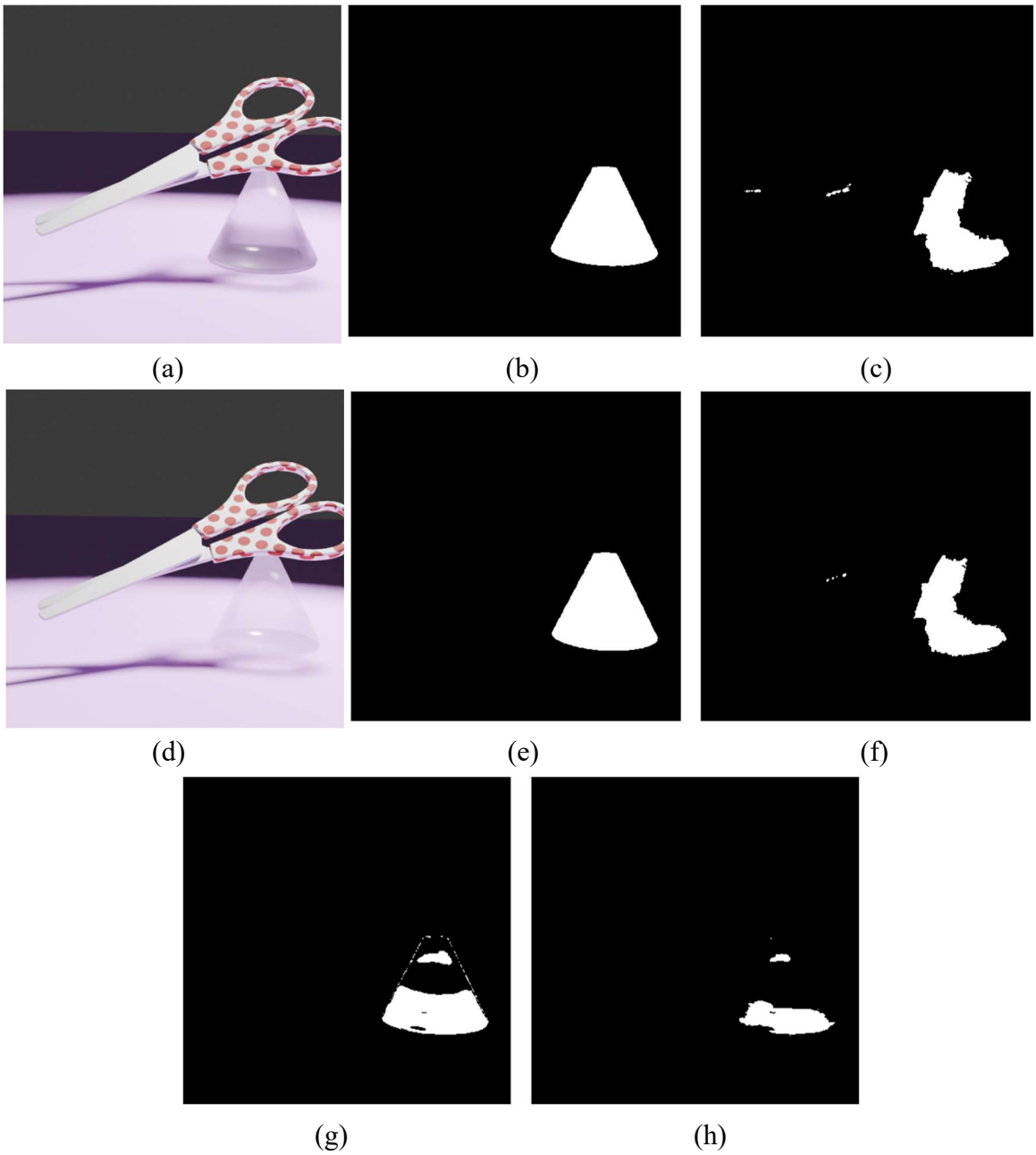


Figure 5.17: (a)-(b)-(c): Reference image - Transparency GT – Transparency Output, (d)-(e)-(f): Render image - Transparency GT – Transparency Output, (g)-(h): GT Change Map – Output Change Map

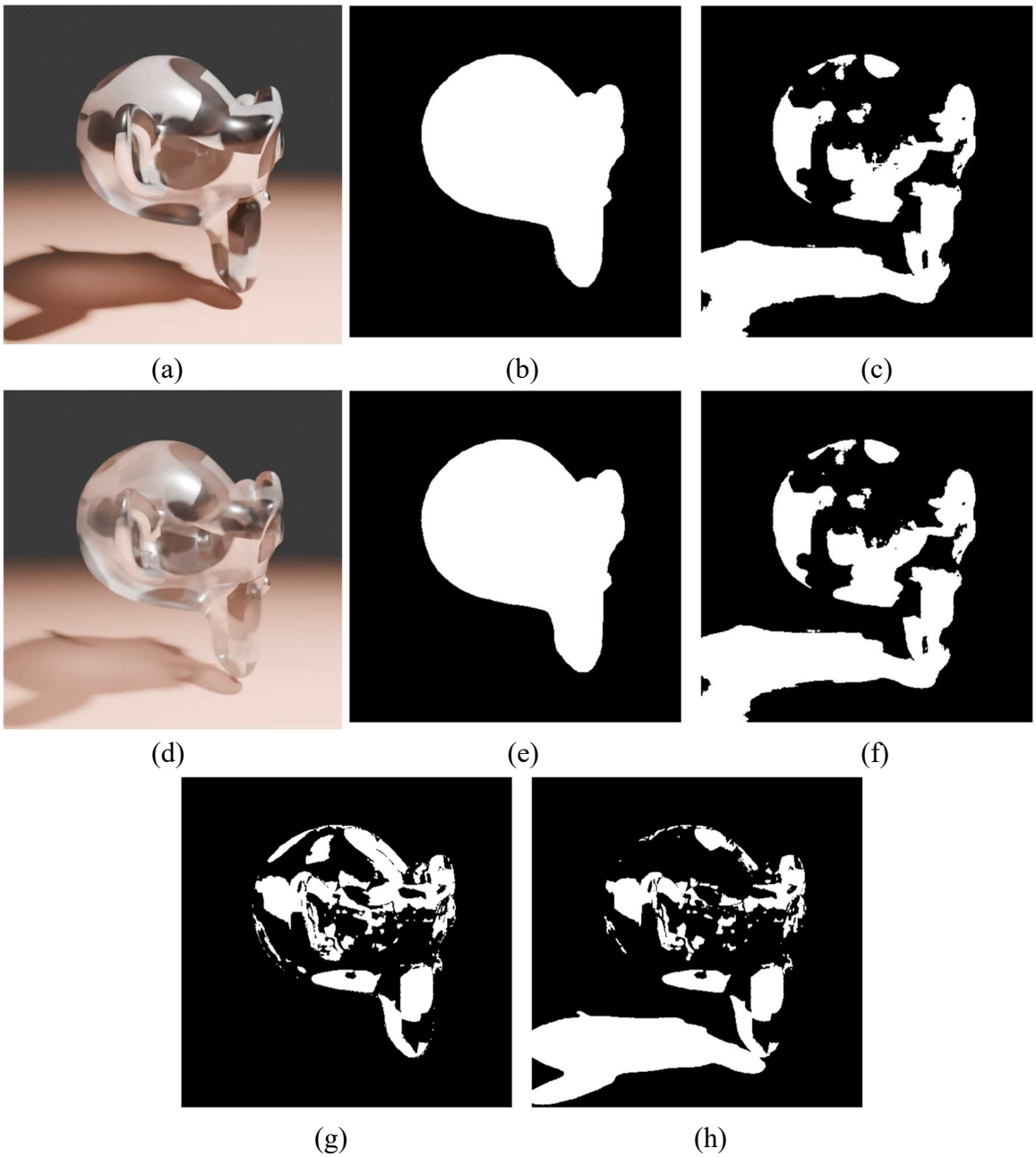


Figure 5.18: (a)-(b)-(c): Reference image - Transparency GT – Transparency Output, (d)-(e)-(f): Render image - Transparency GT – Transparency Output, (g)-(h): GT Change Map – Output Change Map

In scissors case, the annotation is able to detect the cone, even if not in its entirety. Even though this result, the change map is detecting the correct area. In Suzanne's case, the network is not able to separate transparency from shadow in both annotations and change map. This problem was evident also in other images, especially where the object is more transparent, and the shadow is less defined. The more the object is transparent, the less the network is able of detecting it as transparency. This behaviour is caused by the fact that the

network struggle in identifying features with less defined borders, as results on reflections and soft shadows demonstrate.

Images 5.19 and 5.20 presents the results obtained after a test conducted on images with multiple features changes.

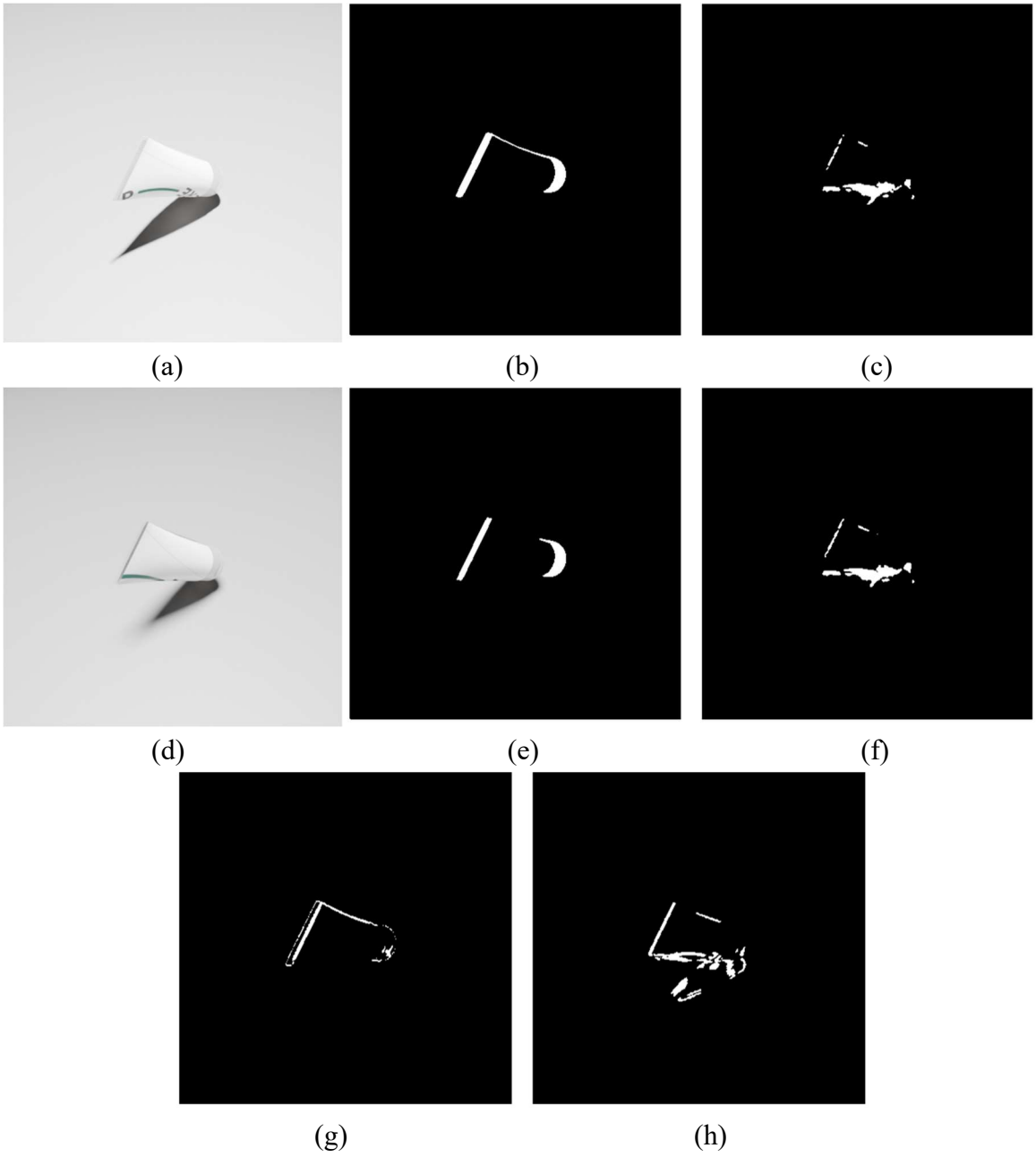


Figure 5.19: (a)-(b)-(c): Reference image - Transparency GT – Transparency Output, (d)-(e)-(f): Render image - Transparency GT – Transparency Output, (g)-(h): GT Change Map – Output Change Map

In this example the transparency of upper part of the tube is detected, but the cap is not detected. Instead, the network detect pieces of texture and shadow, and as the Suzanne's case, is not able to exclude them from change map.

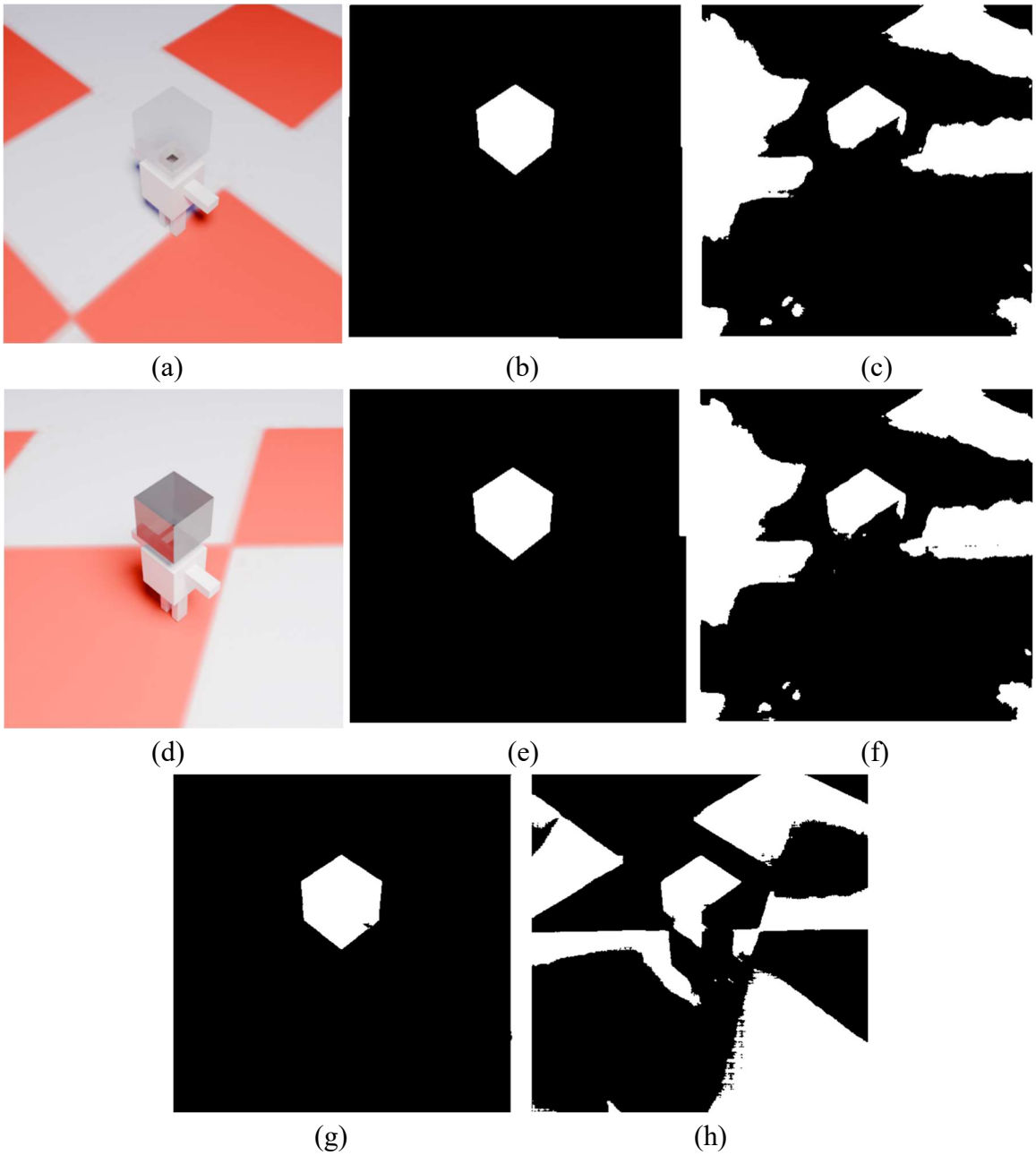


Figure 5.20: (a)-(b)-(c): Reference image - Transparency GT – Transparency Output, (d)-(e)-(f): Render image - Transparency GT – Transparency Output, (g)-(h): GT Change Map – Output Change Map

In this case a big piece of textured floor is labelled as transparency, but also the head of the humanoid is detected, even if not precisely. It can be concluded that transparency, due to its nature of being hard also for human to be seen, can be a difficult feature to detect for the network too.

Experiment 5

This experiment was conducted with a new database, where Reference and Render images differs in terms of illumination and textures. The network has 6 output: two semantic segmentation for shadow, two for texture and two change maps.

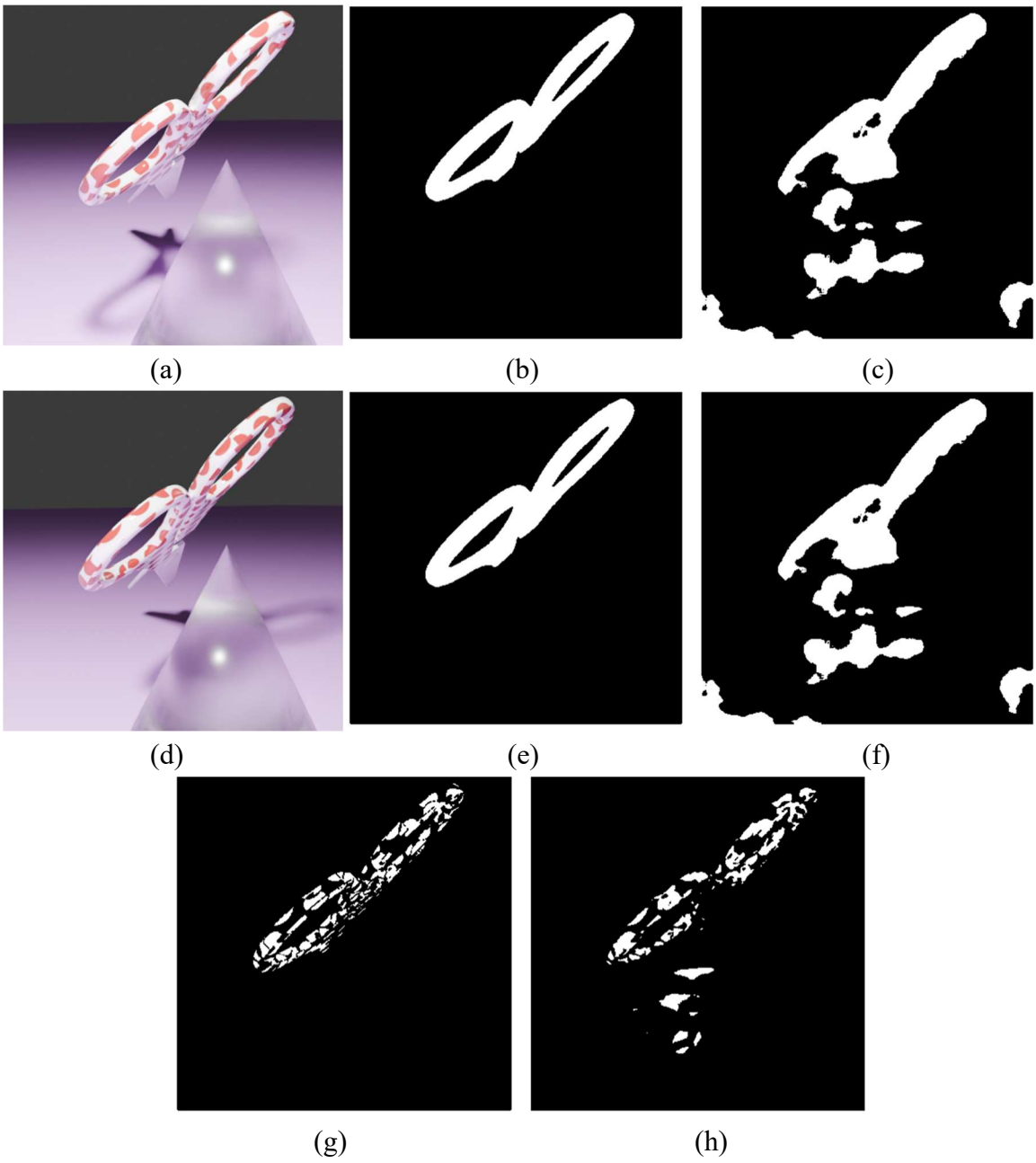


Figure 5.21: (a)-(b)-(c): Reference image - Texture GT – Texture Output, (d)-(e)-(f): Render image - Texture GT – Texture Output, (g)-(h): GT Change Map – Output Change Map

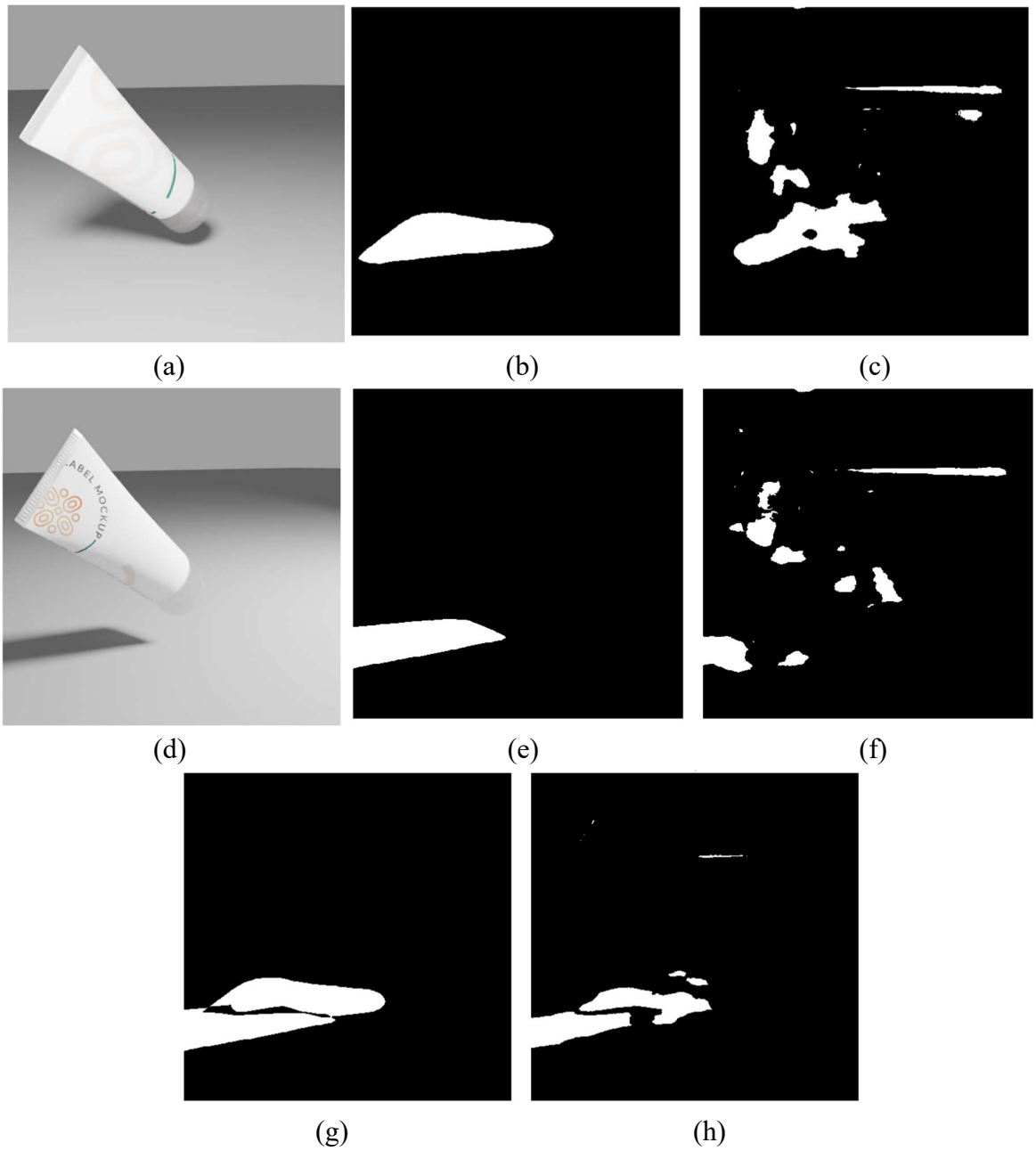


Figure 5.22: (a)-(b)-(c): Reference image - Shadow GT – Shadow Output, (d)-(e)-(f): Render image - Shadow GT – Shadow Output, (g)-(h): GT Change Map – Output Change Map

Figure 5.21 presents the results of texture segmentation and the corresponding change map. While the segmentation successfully detects the texture regions, it also misclassifies certain other areas of the image. Similarly, the change map accurately identifies changes within the texture but also highlights changes in regions outside of the intended texture area.

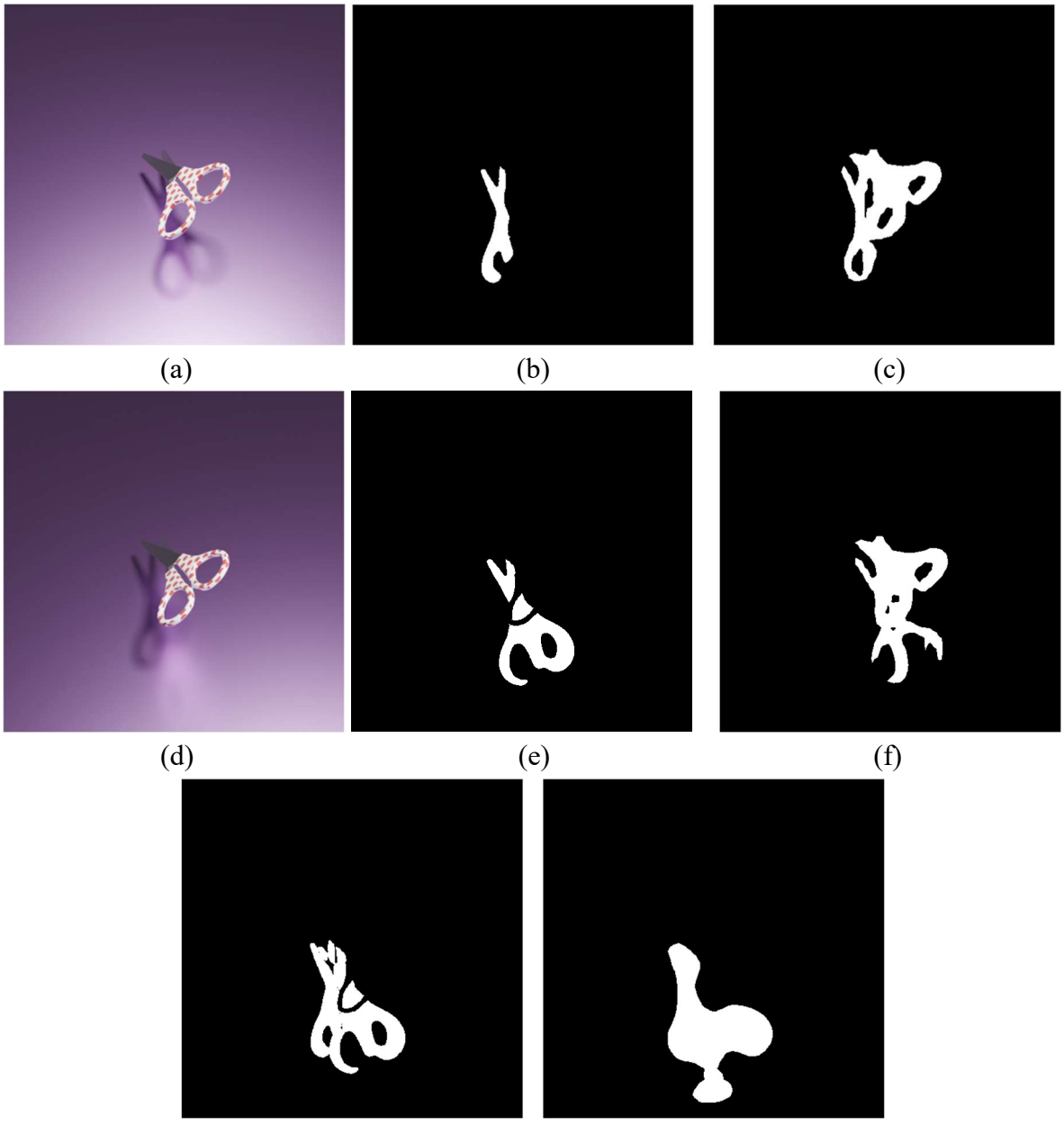
Figure 5.22 show the results of shadow segmentation and the relative change map. Also in this case annotations are not close to GT, as some parts of the tube are identified as shadows. However, change map is able to isolate the shadow bringing the result closer to GT.

5.1.4 ChangeStar

ChangeStar uses a pretrained feature extractor to extract features from images. The feature extracted are taken from a classifier, that apply Semantic Segmentation and from ChangeMixin module, that computes Change Detection. The outputs of the network are two segmentation images and one Change Map. With this architecture, as the one before, four experiments were performed, each one with a different database that represented a single feature as changes. All the four networks as trained, were also tested on images with multiple features differences, but no significant result was obtained.

Experiment 1 – Shadow

The first experiment was conducted on images with changes only in illumination [Figure 5.23, 5.24].

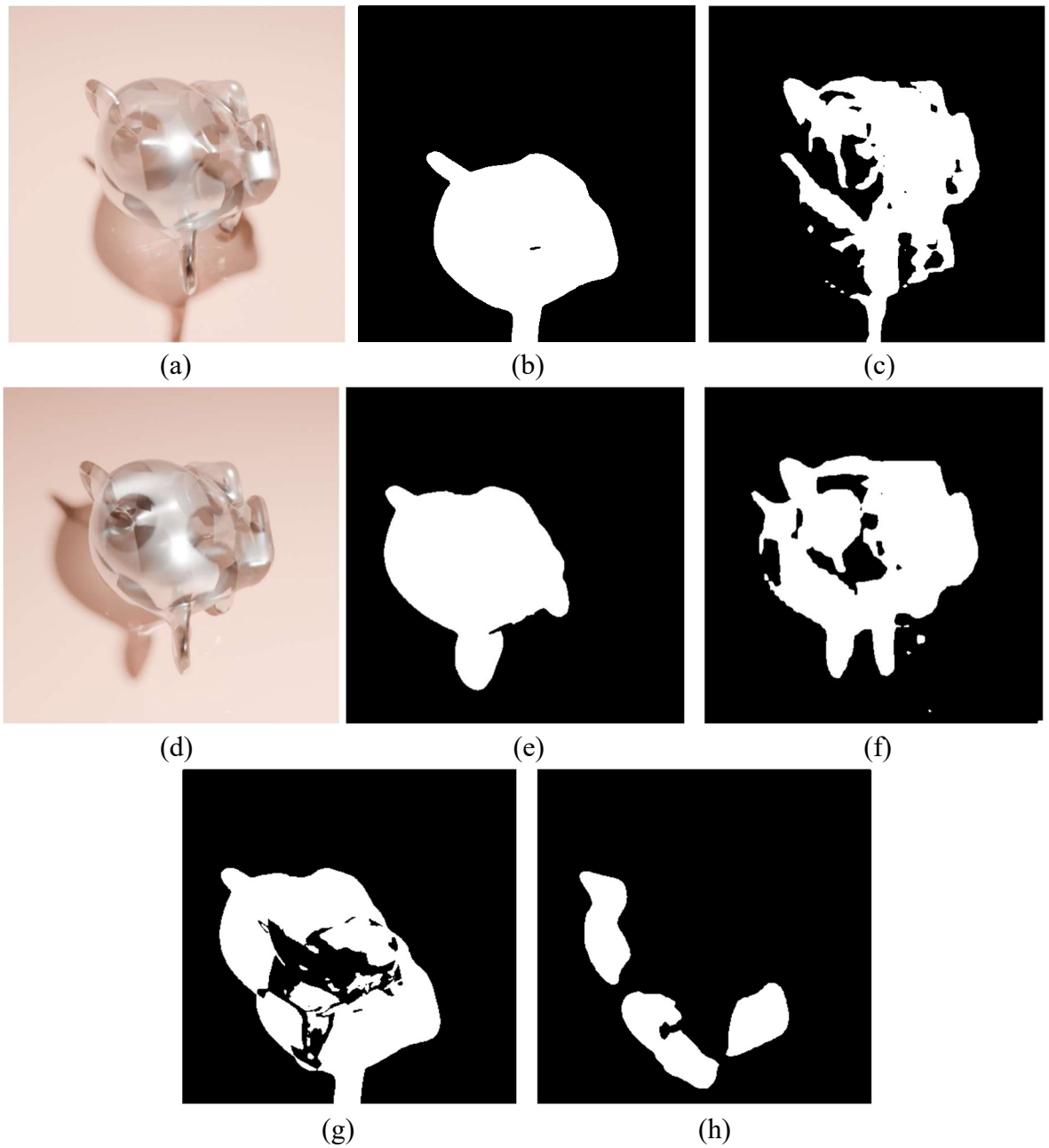


(a) (b) (c)

(d) (e) (f)

(g) (h)

Figure 5.23: (a)-(b)-(c): Reference image - Shadow GT – Shadow Output, (d)-(e)-(f): Render image - Shadow GT – Transparenc Shadow y Output, (g)-(h): GT Change Map – Output Change Map



(a) (b) (c)

(d) (e) (f)

(g) (h)

Figure 5.24: (a)-(b)-(c): Reference image - Shadow GT – Shadow Output, (d)-(e)-(f): Render image - Shadow GT – Transparenc Shadow y Output, (g)-(h): GT Change Map – Output Change Map

Segmentation output demonstrates a capability of the network to learn shadow detection, but also the whole object or a section of it is detected. However Change Maps do not detect the changes in the region labelled which are not shadows, so also change detection part learned to partially distinguish shadows, and it’s useful to refine the result.

Then the network has been tested on images with multiple changes. In this particular case Reference segmentation is missing. Figure 5.25 and 5.26 show the results.

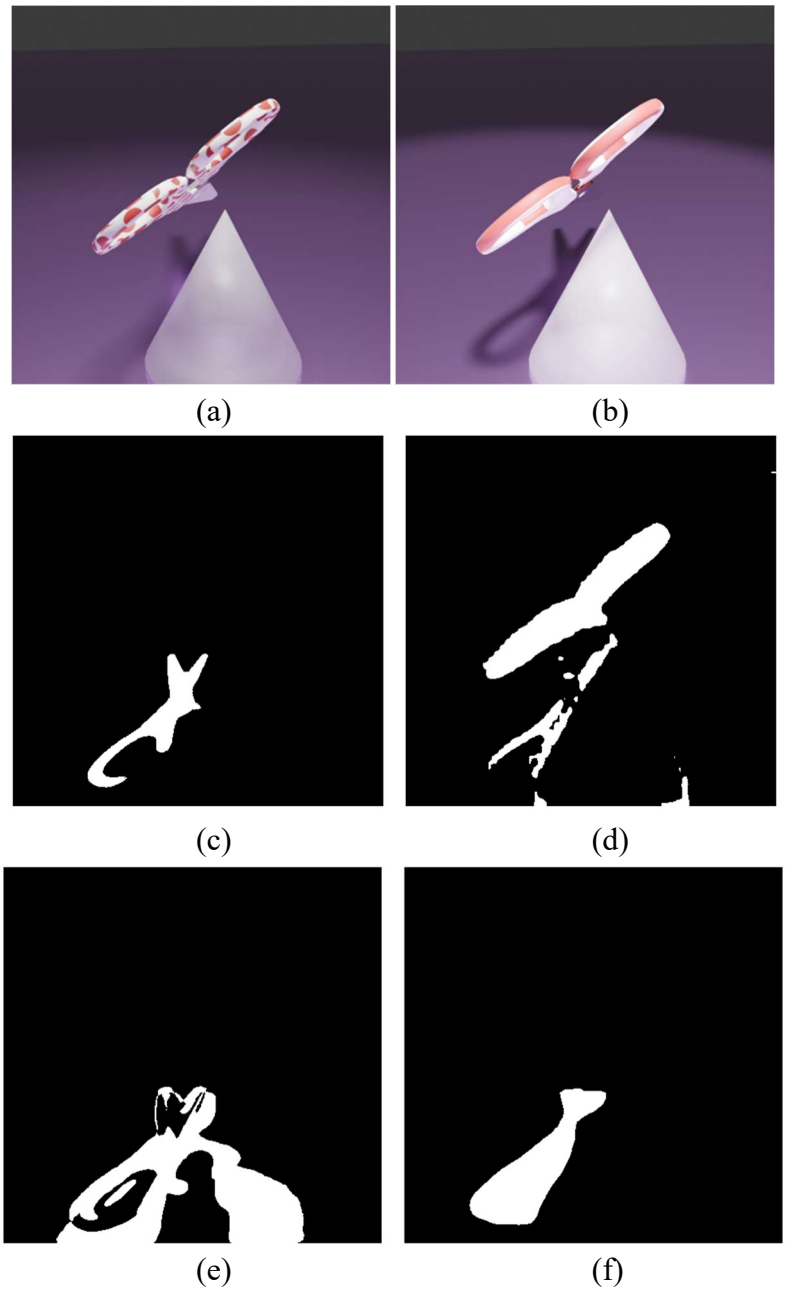


Figure 5.25: (a)-(b): Reference image – Render image, (c)-(d): Render Shadow GT – Render Sadow Output, (e)-(f): GT Change Map – Output Change Map

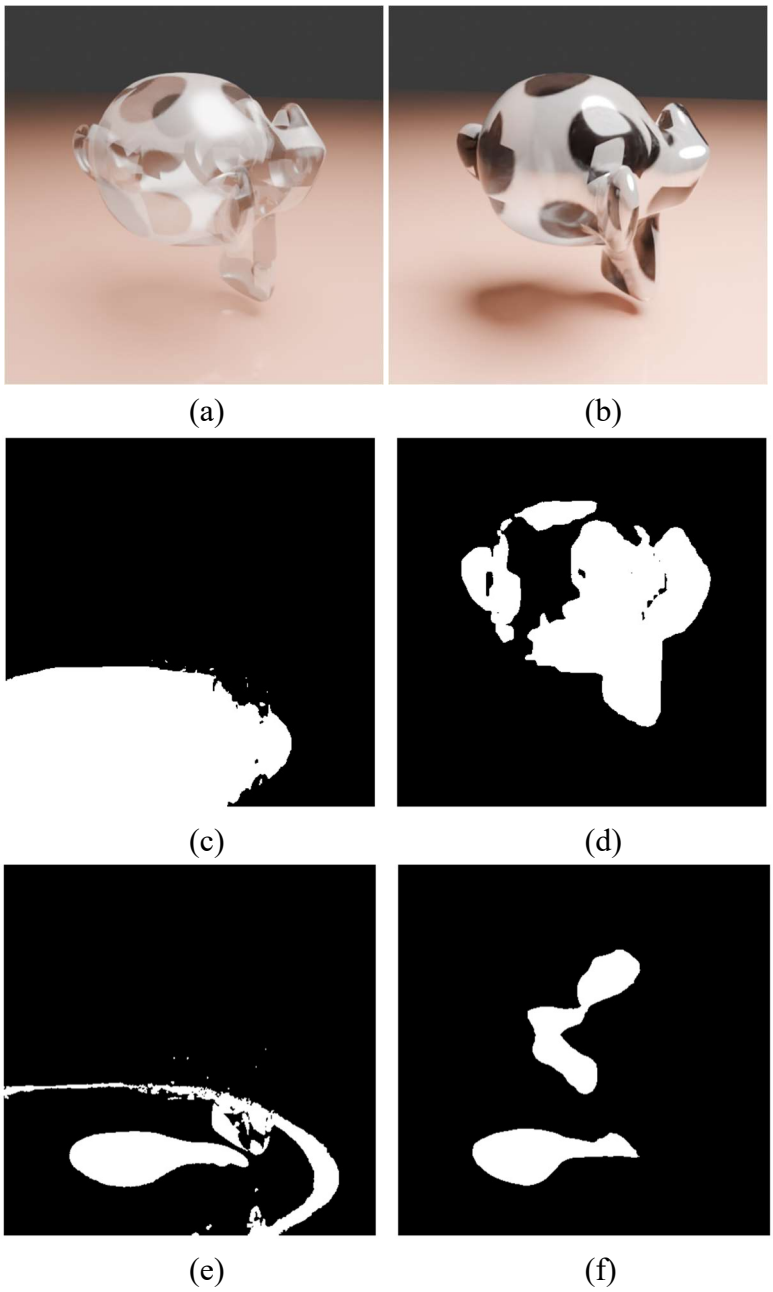


Figure 5.26: (a)-(b): Reference image – Render image, (c)-(d): Render Shadow GT – Render Sadow Output, (e)-(f): GT Change Map – Output Change Map

Even with multiple changes the network is able to detect shadows, but keeps the same defects and detect the object. In Suzanne's case, the shadow is not even detected. Change Detection behaves as the previous test, it is able to eliminate the region assigned to shadows which are not real shadows, and the changes are detected especially where the shadow is stronger. In Suzanne's test also a part of the object is found as changed. It has been noted that shadow are better detected when they are strong and have an high contrast with the floor.

Experiment 2 – Reflections

The first experiment was conducted on images with changes only in material reflections, obtained with the changes of roughness of the materials. Results are shown in figure 5.27 and 5.28.

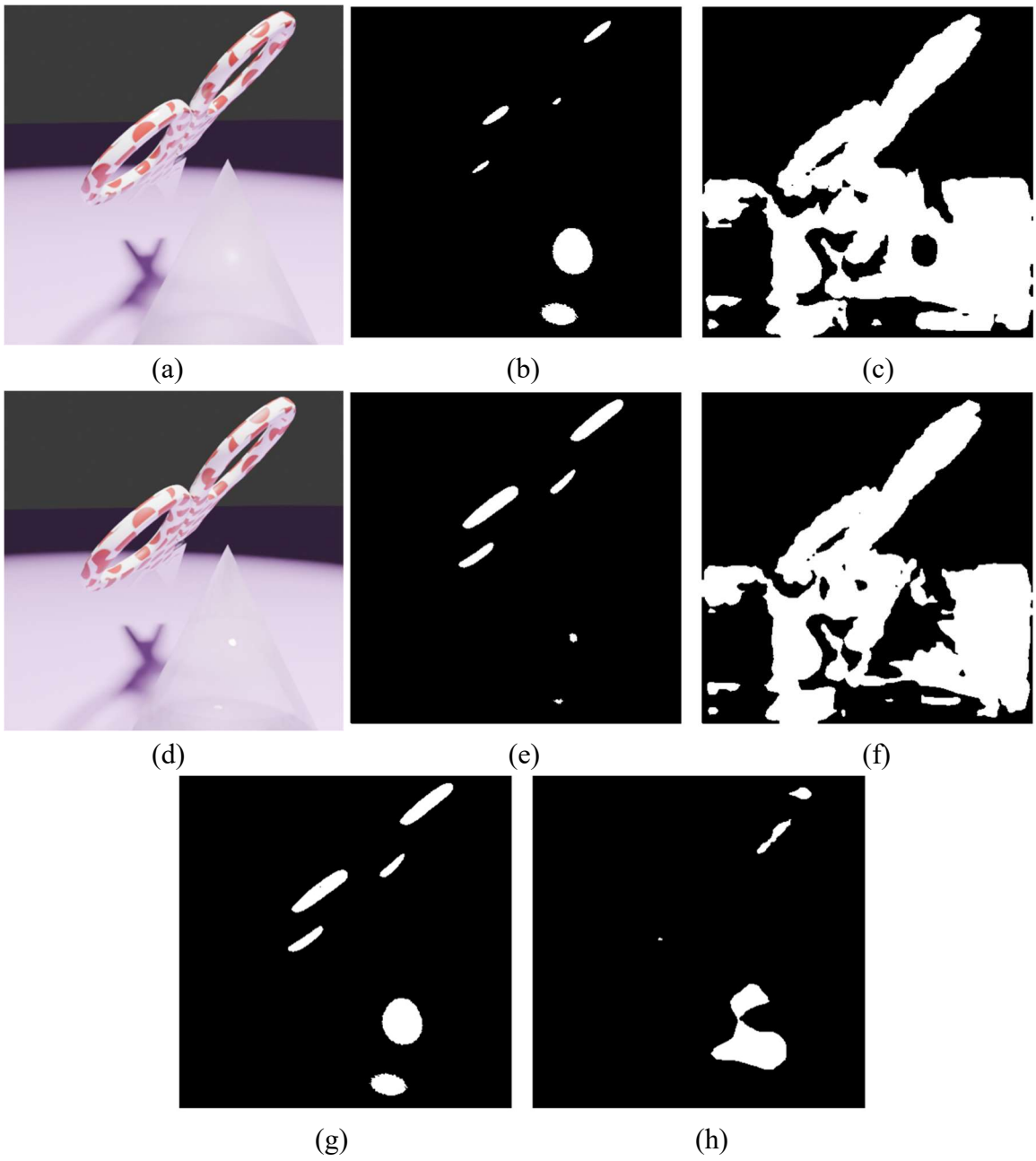


Figure 5.27: (a)-(b)-(c): Reference image - Reflections GT – Reflections Output, (d)-(e)-(f): Render image - Reflections GT – Reflections Output, (g)-(h): GT Change Map – Output Change Map

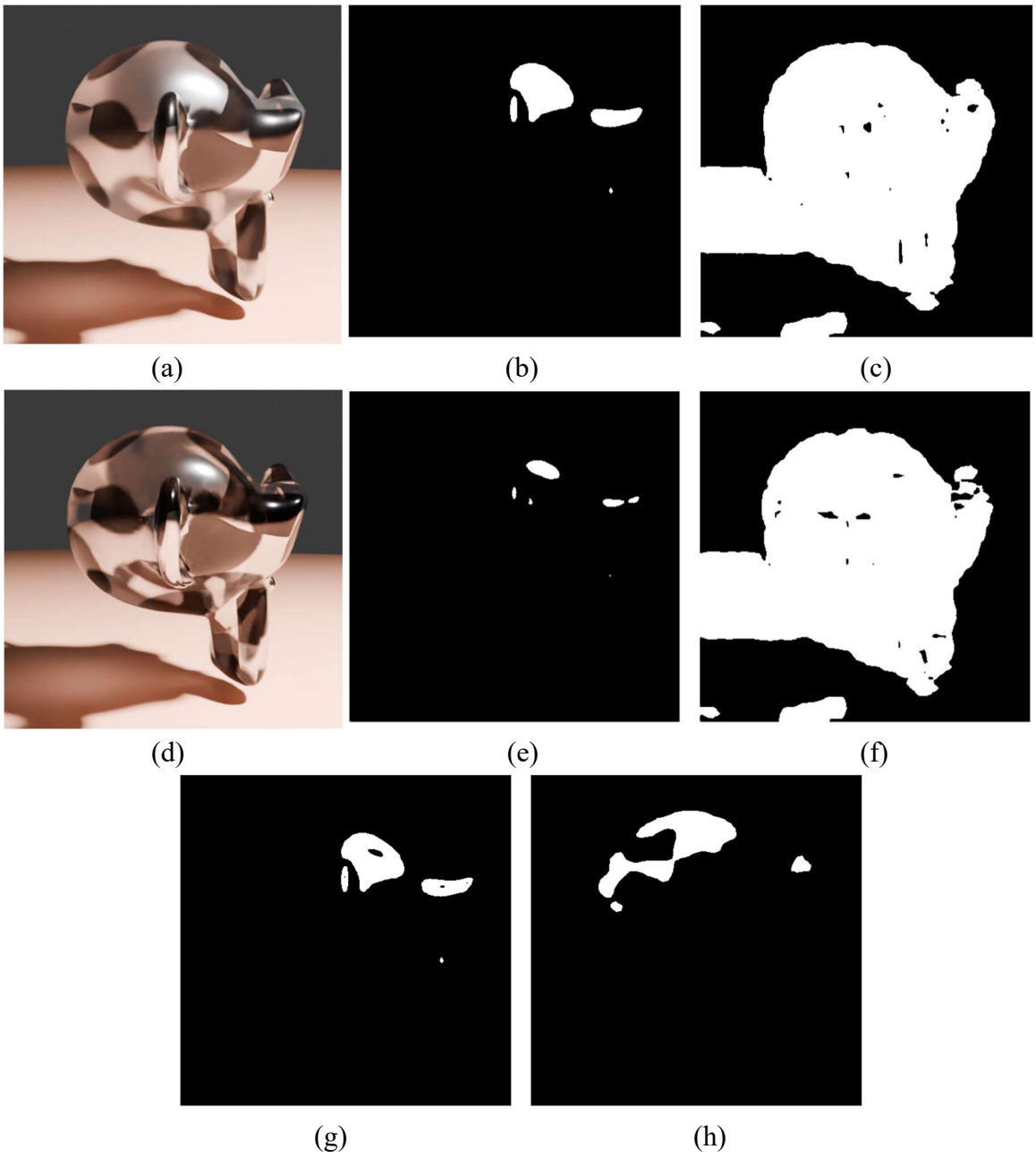


Figure 5.28: (a)-(b)-(c): Reference image - Reflections GT – Reflections Output, (d)-(e)-(f): Render image - Reflections GT – Reflections Output, (g)-(h): GT Change Map – Output Change Map

The segmentation of reflection is not an easy task, and the network struggle in detecting them. The texture placed on the material does not help the detection, but in this feature the network detect a big part of the image, without even distinguishing features or objects, it seems to detect more shadows instead.

The results on images with multiple features changes [Figure 5.29] enhance this problem, detecting shadow and the whole object. Therefore, it seems that the change map detects changes in texture.

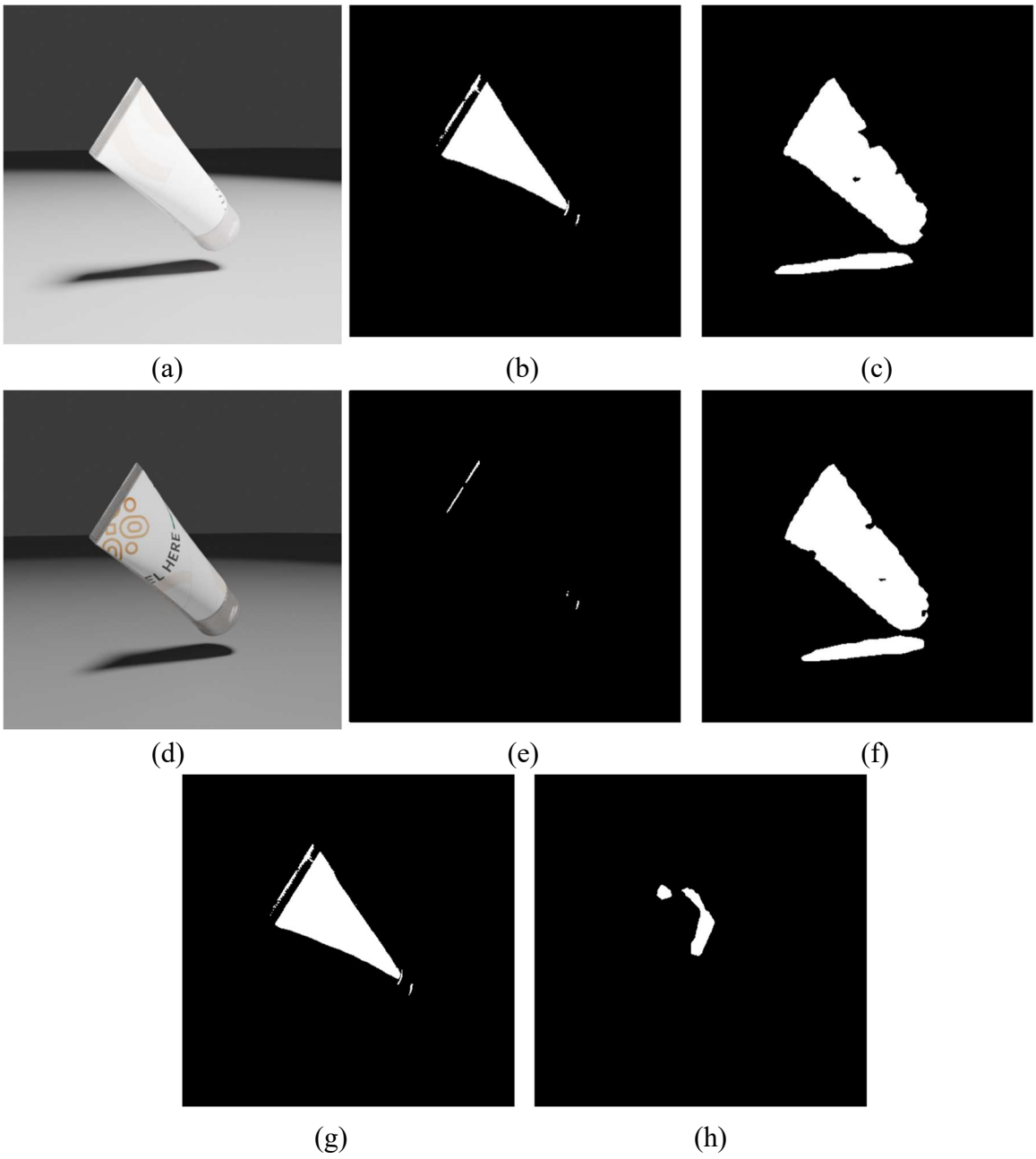


Figure 5.29: (a)-(b)-(c): Reference image - Reflections GT – Reflections Output, (d)-(e)-(f): Render image - Reflections GT – Reflections Output, (g)-(h): GT Change Map – Output Change Map

Experiment 3 - Texture

In this experiment Reference and Render images differs only in textures. Figure 5.30 and 5.31 show the results.

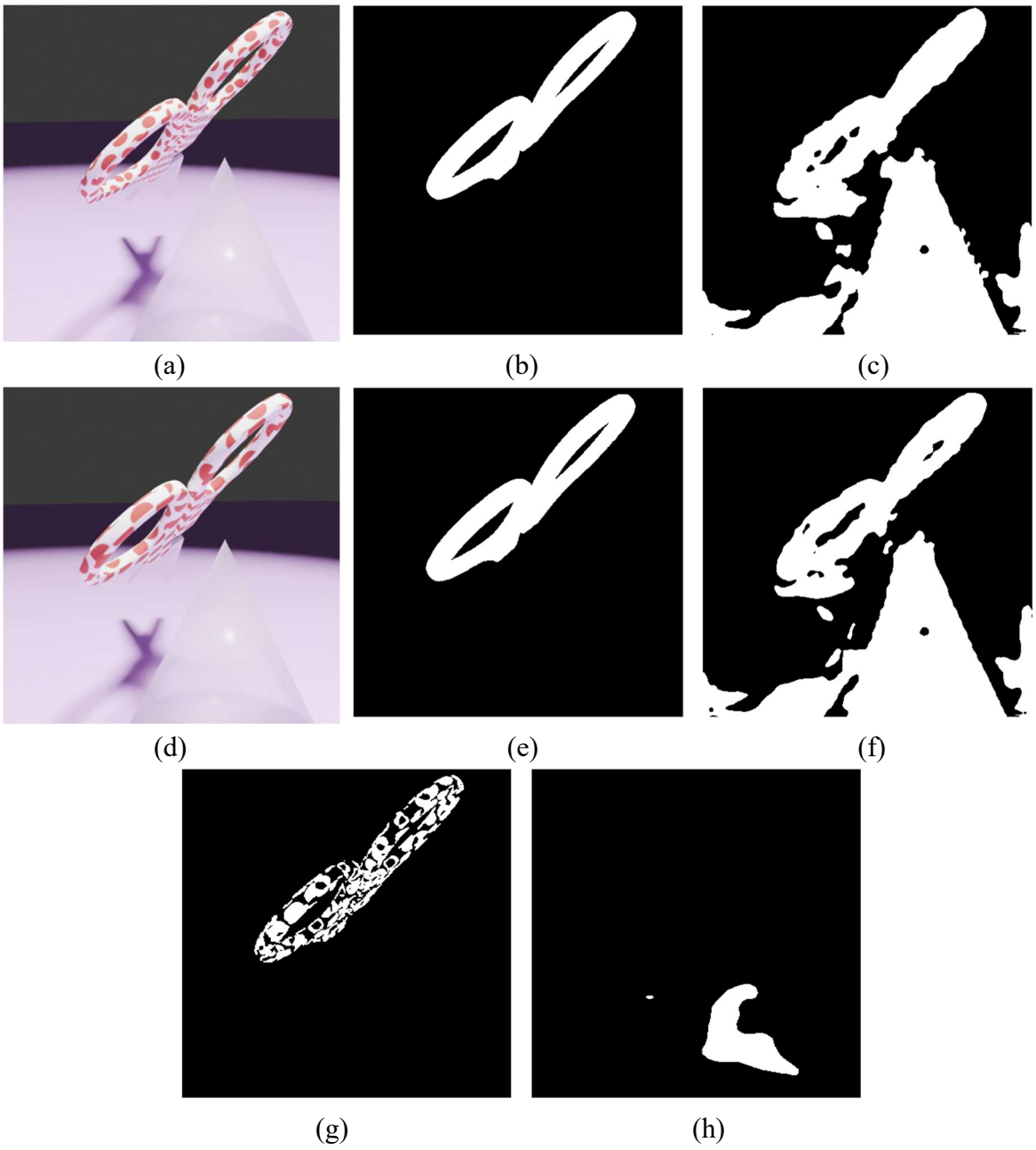


Figure 5.30: (a)-(b)-(c): Reference image - Texture GT – Texture Output, (d)-(e)-(f): Render image - Texture GT – Texture Output, (g)-(h): GT Change Map – Output Change Map

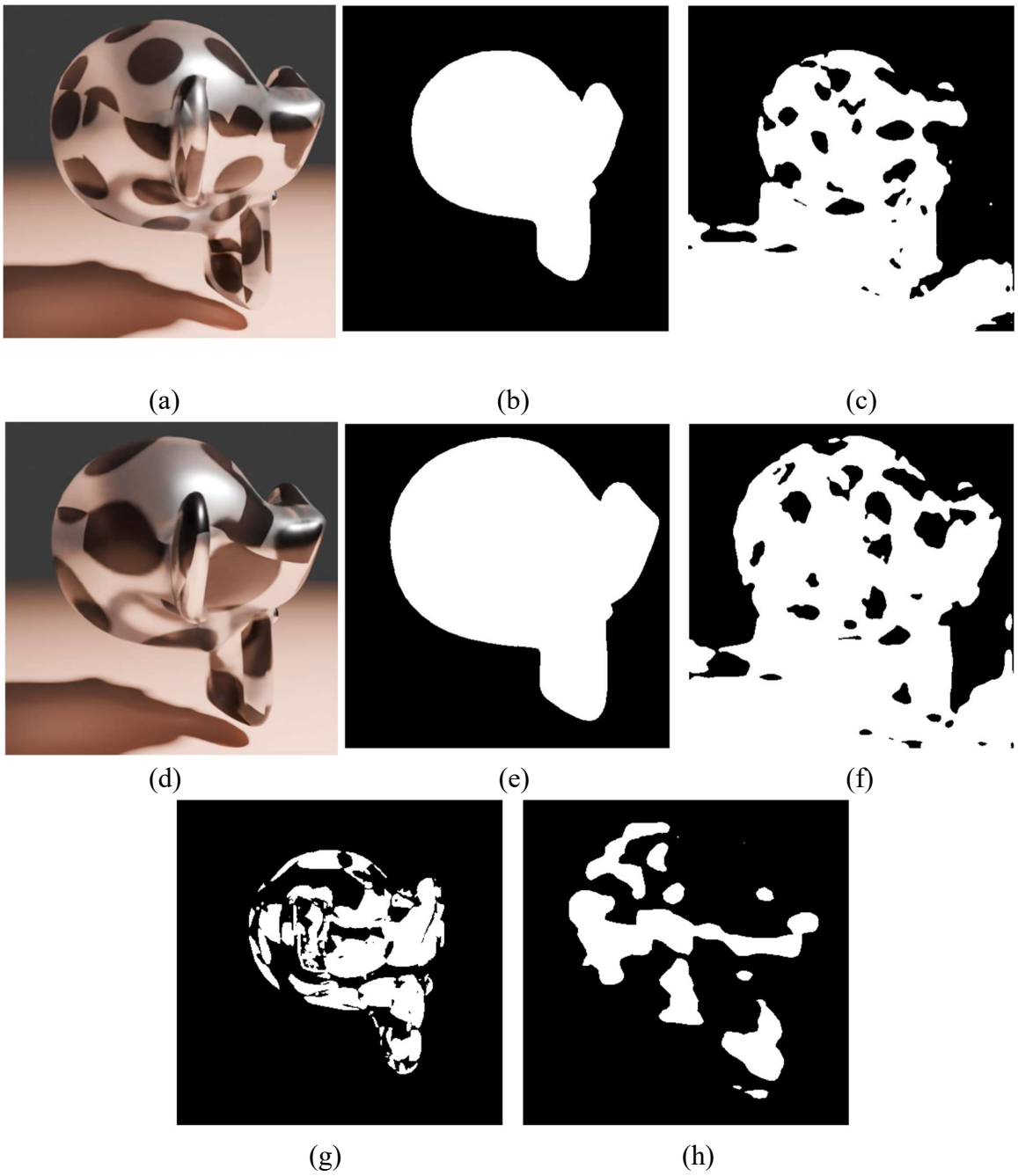


Figure 5.31: (a)-(b)-(c): Reference image - Texture GT – Texture Output, (d)-(e)-(f): Render image - Texture GT – Texture Output, (g)-(h): GT Change Map – Output Change Map

Starting from semantic segmentation, the network is not able to distinguish between the feature and the object: in the scissors, the texture detected are all the objects in the scene, even if no texture is placed on the cone. In the case of Suzanne, the task was easier, as there was only one object in the scene. However, the network wasn't able to understand that no texture was present on the floor. The change map was able to exclude the floor from the result, but on the object the result is imprecise. On scissors, even if the segmentation correctly detected the texture, change map is completely away from the GT.

Experiment 4 – Transparencies

The last experiment was done using Render and Reference pairs which had as only difference the transparencies [Figure 5.32, 5.33].

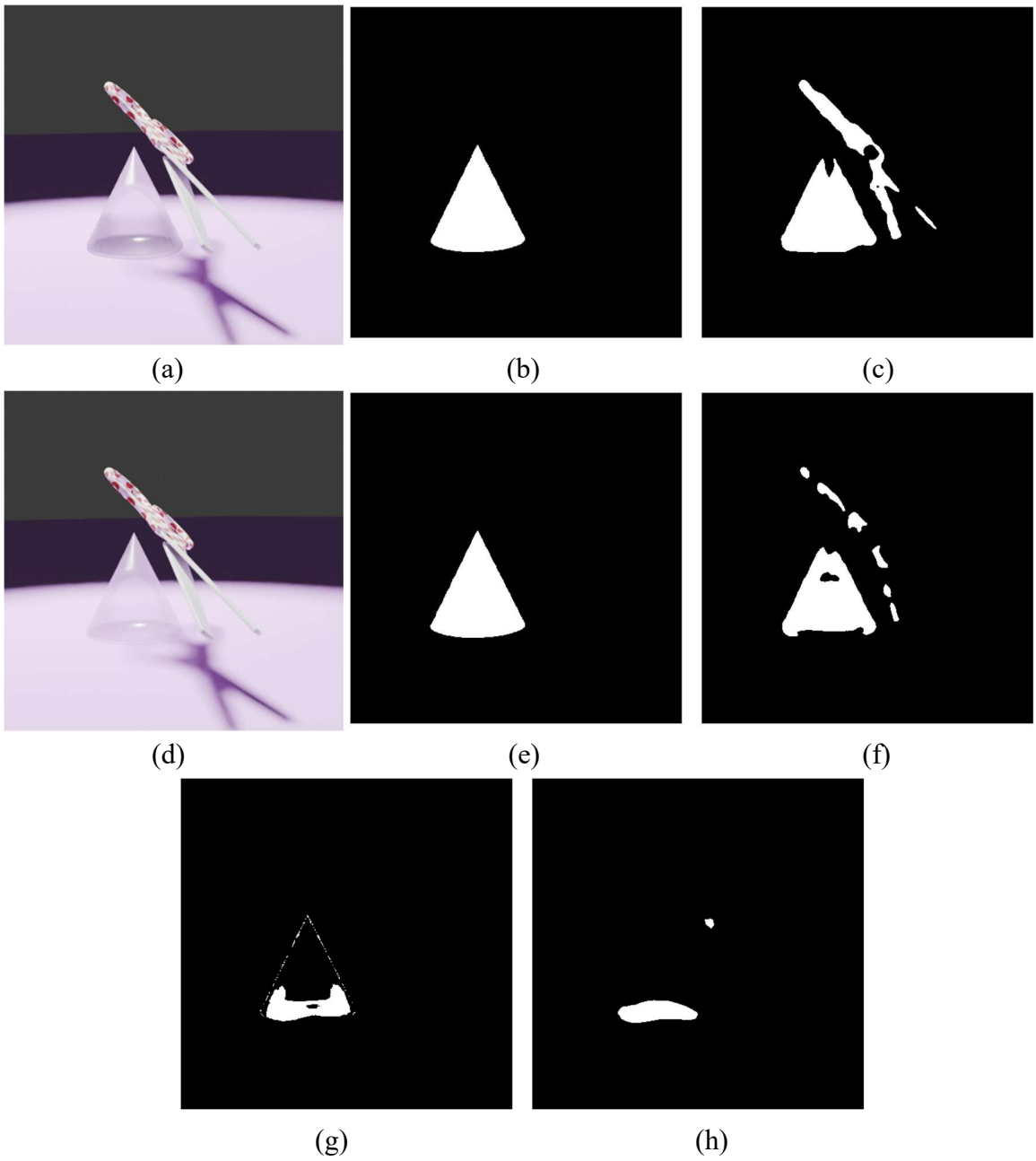


Figure 5.32: (a)-(b)-(c): Reference image - Transparency GT – Transparency Output, (d)-(e)-(f): Render image - Transparency GT – Transparency Output, (g)-(h): GT Change Map – Output Change Map

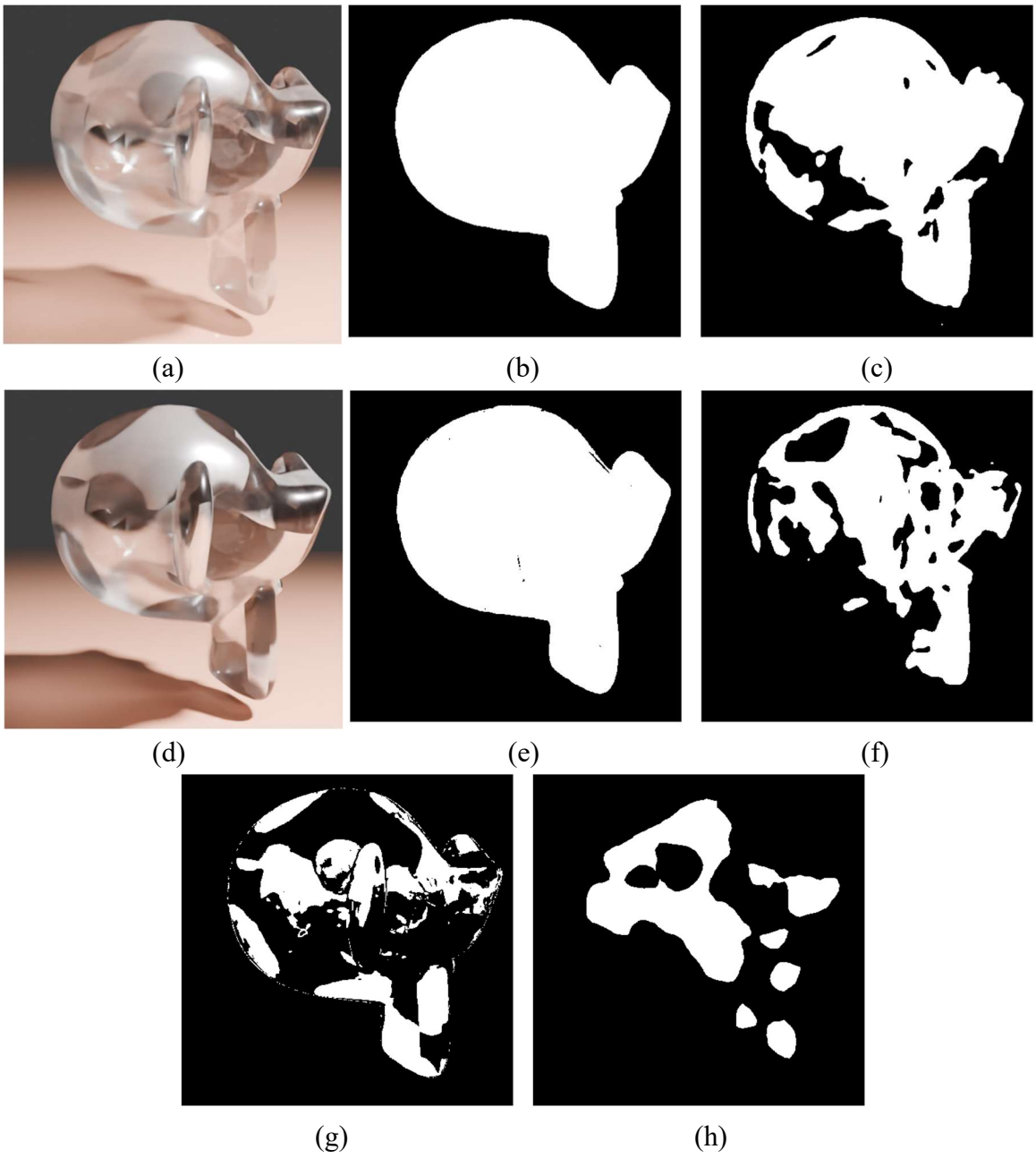


Figure 5.33: (a)-(b)-(c): Reference image - Transparency GT – Transparency Output, (d)-(e)-(f): Render image - Transparency GT – Transparency Output, (g)-(h): GT Change Map – Output Change Map

The network seem to have acquired the capability to detect transparencies, in Suzanne the annotation is quite precise, even if in scissors transparency is detected in both objects.

The test conducted on images with more features changes are presented in figure 5.34 and 5.35.

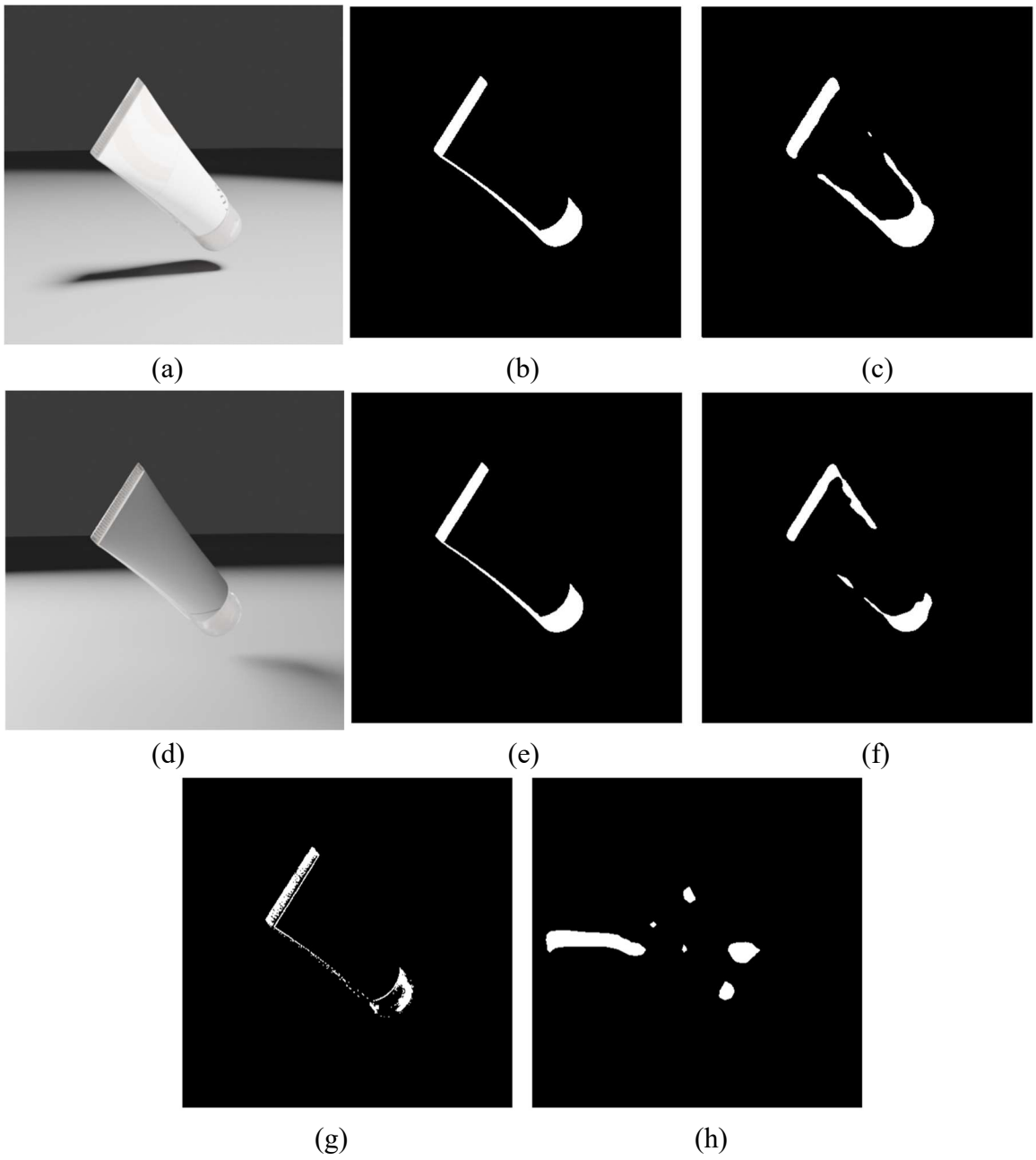


Figure 5.34: (a)-(b)-(c): Reference image - Transparency GT – Transparency Output, (d)-(e)-(f): Render image - Transparency GT – Transparency Output, (g)-(h): GT Change Map – Output Change Map

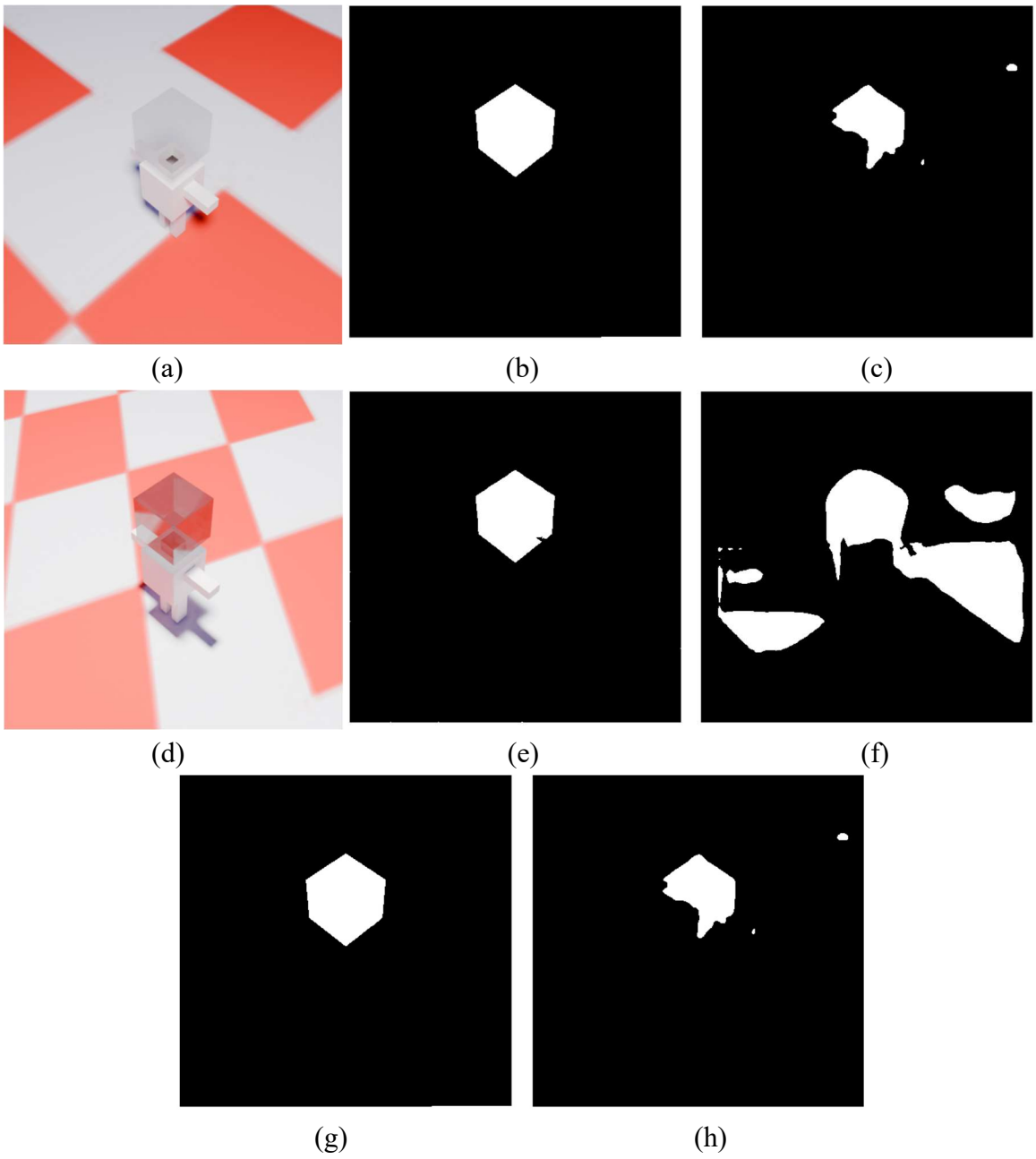


Figure 5.35: (a)-(b)-(c): Reference image - Transparency GT – Transparency Output, (d)-(e)-(f): Render image - Transparency GT – Transparency Output, (g)-(h): GT Change Map – Output Change Map

These tests demonstrate that the network can detect transparencies and the region where they are present. The segmentation in the tube is quite precise, but the change map detect also other part of the image. The humanoid segmentation in Render image is less precise, as it detect also a section of the texture, but the change map is able to discriminate changes in transparency.

5.2 Methods Comparison

This section aim is to compare the performances of the two main architectures utilized for Semantic Change Detection in this project, ChangeStar and FresUNet (Strategy 4). This comparison will take into account the metrics of the experiments and the visual results reported in the previous section. As metrics we consider precision, recall, intersection over Union, and F1-Score, and will be discussed separately for Change Detection and Semantic Segmentation.

The experiments conducted on shadows had the aim to determine if the illumination has changed between Reference and Render image. ChangeStar succeed in finding shadow, but struggle in isolating it from the object. It seems easier for the network to find strong shadow than soft shadows. FresUNet, instead, is more precise in the segmentation, even if it struggles in detecting soft shadow, and it is more precise in drawing the correct border of the feature [Figure 5.36].

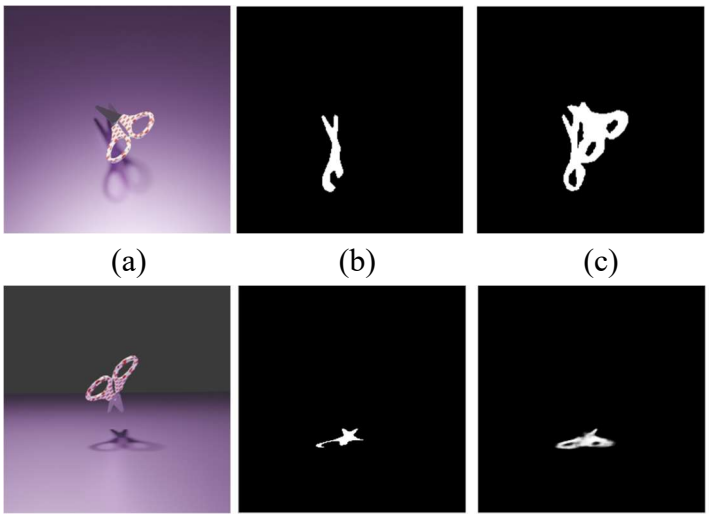


Figure 5.36: (a)-(b)-(c): Reference image - Shadow GT - ChangeStar Shadow Output. (d)-(e)-(f): Reference image - Shadow GT - FresUNet Shadow Output

The tests conducted on images with multiple features changes confirm the results; FresUNet consistently detect shadows also in scenario with more than one feature is alterated and is able to create an output change map similar to the Ground Truth (GT). While ChangeStar demonstrated the ability to detect shadows in these complex images, it frequently misclassified objects as shadows. Nevertheless, its change maps accurately identified the regions with the most visible changes.

In both architectures, the Change Detection module successfully excludes the annotated regions of the image that do not correspond to real shadows. It's possible to note that both models achieve better results when the shadows are more pronounced and exhibit a high contrast with the floor.

CD metrics - Shadows				
	Precision	**Recall**	**F1**	**IoU**
FresUNet	**0.45**	0.40	**0.42**	**0.28**
ChangeStar	0.19	**0.60**	0.28	0.18
Table 5.1				

Segmentation metrics - Shadows				
	Precision	**Recall**	**F1**	**IoU**
FresUNet	**0.36**	**0.21**	**0.23**	**0.18**
ChangeStar	0.14	0.05	0.06	0.03
Table 5.2				

The metrics in Tables 5.1 and 5.2 are in line with the visual results, demonstrating that higher metric values are associated with better qualitative performance. Among the metrics, ChangeStar achieved a higher recall than FresUNet in Change Detection metrics. However, in shadow detection, FresUNet had a better performance, particularly in defining boundaries and segmenting shadows more accurately. Both networks struggled with soft shadows, but FresUNet produced more precise change maps, with higher precision (0.45 vs. 0.19) and F1-score (0.42 vs. 0.28) in Change Detection. ChangeStar, while achieving better recall, often confuses objects as shadows, reflecting its tendency to over-detect.

For reflection detection, both networks struggle to identify reflective regions. In segmentation, FresUNet and ChangeStar detected significant section of the image, especially on the floor where the contribution of the light was more intense. However, FresUNet showed a better ability to follow the boundaries of reflective regions. This was particularly evident in the Suzanne case, where ChangeStar incorrectly detected the entire object together with its shadow, while FresUNet excluded the shadow in the segmentation and outlined the reflective region more accurately, even though some areas detected were not real reflections [Figure 5.37].

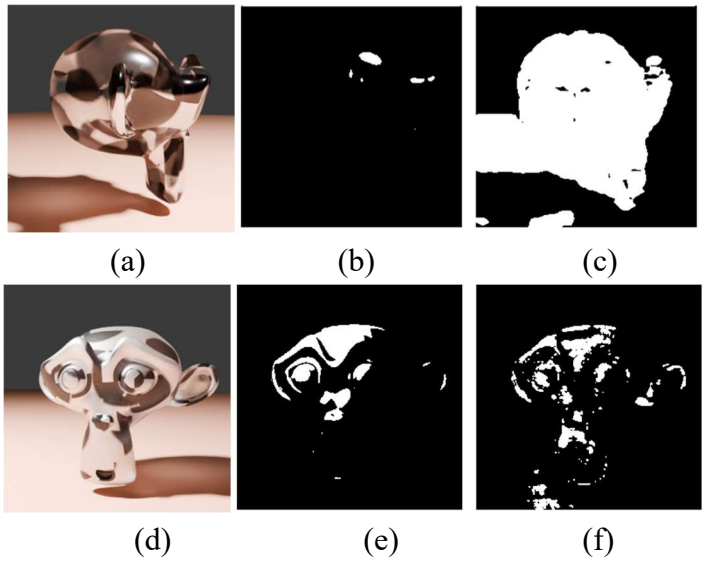


Figure 5.37: (a)-(b)-(c): Reference image - Reflections GT – ChangeStar Reflections Output. (d)-(e)-(f): Reference image - Reflections GT – FresUNet Reflections Output

In relation to Change Detection, both architectures acquired the ability to introduce a more discriminative factor. Despite inaccuracies in segmentation, Change Detection module

identifies where changes occurred. In both tests the final Change Map detected the areas of change. FresUNet prove its ability to draw more defined border on the reflections. However, both networks struggled to accurately annotate reflections in images with multiple overlapping changes, and in such cases, the Change Detection module was unable to effectively distinguish real reflections.

Segmentation metrics - Reflections				
	Precision	**Recall**	**F1**	**IoU**
FresUNet	0.1	**0.14**	0.1	0.05
ChangeStar	**0.42**	0.08	**0.13**	**0.075**

Table 5.3

CD metrics - Reflections				
	Precision	**Recall**	**F1**	**IoU**
FresUNet	**0.41**	**0.52**	**0.43**	**0.3**
ChangeStar	0.07	0.03	0.07	0.02

Table 5.4

As shown in Table 5.3, Segmentation Metrics are higher for ChangeStar, except for the recall, despite its inferior qualitative performance. A significant gap can be noted in Change Detection's metrics in Table 5.4, where FresUNet outperforms ChangeStar across all metrics. For reflections, both networks faced challenges, especially when multiple features were present. FresUNet showed a stronger ability to follow the contours of reflective regions, whereas ChangeStar frequently mistaken entire objects as reflections. This difference is evident in the Change Detection metrics, where FresUNet achieved a precision of 0.41 and an F1-score of 0.43, compared to ChangeStar's 0.07 and 0.07, respectively.

The test conducted on textures proof the best results above all features on FresUNet architecture. Segmentation output slightly overstimated the texture by assigning more pixels than the actual texture around the object, but produced outputs that closely resembled the ground truth (GT), including fine-grained details. In contrast, ChangeStar showed a tendency to label the entire scene, including objects and the floor, as texture, with only a few isolated regions correctly segmented. This is reflected in the Change Maps, where FresUNet's outputs closely align with the GT, while ChangeStar's maps correctly finds some texture changes but also introduce errors in other regions, resulting in outputs that deviate significantly from the GT. An example can be seen in figure 5.38.

(a) (b) (c) (d)

Figure 5.38: (a)-(b): ChangeStar CM GT Textures- ChangeStar CM Output Textures. (c)-(d): FresUNet CM GT Textures, FresUNet CM Output Textures

Test conducted on images with multiple features changes confirm the good performances of FresUNet in texture segmentation and change detection. On new images with changes in reflections, transparency and illumination, the network is able to both detect texture region and also to detect changes in texture's details. Metrics in Table 5.5 show that ChangeStar has better values in segmentation, while metrics presented in table 5.6 confirm that FresUNet outcome the performances of ChangeStar in all the metric considered.

Segmentation metrics - Textures				
	Precision	**Recall**	**F1**	**IoU**
FresUNet	0.1	**0.95**	0.19	0.1
ChangeStar	**0.2**	0.7	**0.31**	**0.19**

Table 5.5

CD metrics - Textures				
	Precision	**Recall**	**F1**	**IoU**
FresUNet	**0.94**	**0.95**	**0.95**	**0.9**
ChangeStar	0.26	0.11	0.16	0.1

Table 5.6

The texture detection experiments highlighted FresUNet's superior performance respect to ChangeStar. Its change maps closely matched the ground truth, capturing fine details and providing more accurate segmentation. ChangeStar, in contrast, tended to over-segment and label large regions, including entire objects, as textures. This difference is evident in the Change Detection F1-score, where FresUNet achieved 0.95 compared to ChangeStar's 0.16. Although ChangeStar showed better recall in segmentation, FresUNet's precision and IoU metrics indicate more accurate texture localization.

Regarding transparency, ChangeStar is able to make a fair transparency annotation, despite some occasional mistake in labelling of non-transparent object. However, it can adjust change map results focusing on changes within transparent regions. Conversely, when FresUNet is successful in detecting transparency is also able to generate an output change map that closely resemble GT. Otherwise, if Segmentation output deviates from GT, the change detection struggle to isolate transparency. In this case the changing in transparent area are close to GT.

In test conducted with this network on images with multiple features changes, ChangeStar achieve a good performance by detecting transparencies with some errors on the floor, while still isolating transparency changes effectively in the change map. Conversely, FresUNet, although less effective in initially detecting transparency and the corresponding changed areas, exhibited greater precision in identifying changes within the transparent regions once detected. An example can be seen in figure 5.39.

(a) (b) (c) (d)

Figure 5.39: (a)-(b): ChangeStar CM GT Transparency- ChangeStar CM Output Transparency. (c)-(d): FresUNet CM GT Transparency, FresUNet CM Output Transparency

Segmentation metrics - Transparencies				
	Precision	**Recall**	**F1**	**IoU**
FresUNet	**0.82**	**0.81**	**0.81**	**0.69**
ChangeStar	0.07	0.13	0.08	0.06
Table 5.7				

CD metrics - Transparencies				
	Precision	**Recall**	**F1**	**IoU**
FresUNet	**0.92**	**0.88**	**0.90**	**0.81**
ChangeStar	0.32	0.54	0.27	0.25
Table 5.8				

In table 5.7 and 5.8 metrics of this experiment are presented. All metrics are better for FresUNet even if qualitatively annotations and change map of both architectures presents defects. For transparency detection, FresUNet outperformed ChangeStar in both segmentation and change detection tasks. While ChangeStar managed to detect some transparent regions, it often misclassified non-transparent areas, leading to lower accuracy. FresUNet, despite occasional segmentation errors, produced more accurate change maps within transparent regions. This is supported by the metrics, with FresUNet achieving an F1-score of 0.90 in Change Detection compared to ChangeStar's 0.27.

In general, shadows and textures seems to be the feature on which architectures performs better. This can be attributed to the fact that these features typically have more well-defined edges compared to transparencies and reflections. Notably, even soft shadows are more challenging for the networks to detect. Reflections, on the other hand, can be partially obscured by textures, making them more difficult to identify. However, when considering images without textures and reflections are more defined, the network demonstrates good performance [Figure 5.40], successfully detecting the reflection even when it is broader and less distinct.

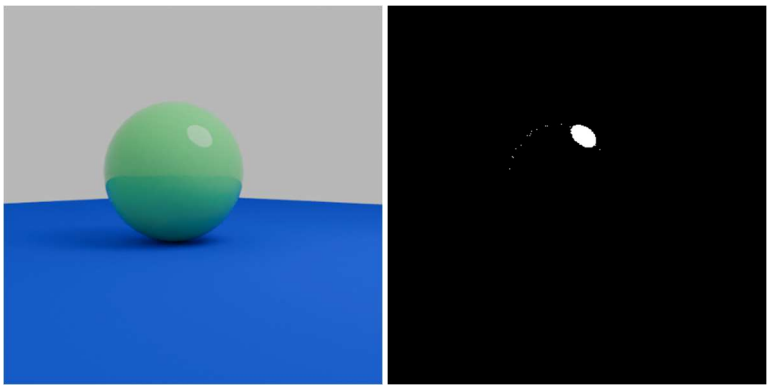


Figure 5.40: Reference image and reflection annotation with FresUNet

This section compared the performance of ChangeStar and FresUNet, for Semantic Change Detection, focusing on shadows, reflections, textures, and transparencies. The comparison analyzed both qualitative visual results and metrics as precision, recall, F1-score, and Intersection over Union (IoU), splitting the evaluation for Change Detection and Semantic Segmentation. FresUNet consistently showed better performances across all features, particularly in features that requires defined boundary segmentation. While ChangeStar shows a higher sensitivity in some cases, annotating more pixels than the GT, as reflected by its recall, it struggles with precision and often produces less reliable outputs. FresUNet, on the other hand, offers a more balanced and accurate approach, making it a more suitable choice for Semantic Change Detection in complex environments.

6.Conclusions, Limitations and Future Works

This study examined the creation, application, and assessment of two deep learning models for Semantic Change Detection. The primary aim was to achieve meaningful outcomes in Semantic Change Detection by using individual features separately, thus establishing a basis for understanding the potential and constraints of each model when using different features. The findings indicate that FresUNet generally performed better than ChangeStar in terms of precision, F1-score, and the production of change maps that closely matched the actual ground truth. Nonetheless, both models encountered difficulties in situations involving multiple feature changes. The results coming from the combination of two features confirm that the increasing of feature's number leads to an increasing difficulty in the model's training, and in the reaching of consistent results.

Several limitations emerged during the project. A significant constraint was the inability of Blender to render self-shadow, which refers to the shadows an object casts on itself, limiting the accuracy of labelling for this feature. Furthermore, the datasets used were relatively small and of low resolution due to time and resource limitation, which may have an impact on the results and the generalization capabilities of the models. The quality of change maps was influenced by other features, as different lighting conditions, that in some cases may have deviate the GT annotation from the real change. Similarly, reflections are highly sensitive to the direction of light, which made it difficult to detect changes in reflective surfaces as indicators of material roughness change. The process of creating annotations through thresholding might have caused some details to be missed or included areas where the feature wasn't visible, particularly in reflections and shadows, which could affect the accuracy of the segmentation ground truth. Additionally, the SAM feature extractor used with ChangeStar is a general segmentation model that classifies almost everything, and it isn't specifically designed for the feature we wanted to track. That may be the reason why even if ChangeStar was pre-trained model, its performances were less accurate that FresUNet which was fully trained. No segmentation networks are available for the recognition of all the features we wanted to monitor.

Future research may move in many directions, by exploring the integration of multiple features into a single neural network capable of addressing a wider range of changes without compromising accuracy. This would help the network to better generalize in different scenarios. Additionally, new features could be added to have a better evaluation of the changes between images. For example, the lighting feature could be divided into smaller components such as light direction, intensity, and radius, which would enable the network to distinguish more changes in lighting conditions.

Overall, this study has given us important information about the challenges and possibilities

in Semantic Change Detection, laying the groundwork for future progress in this area. By fixing the problems mentioned and exploring the suggested ideas for future work, we can create more accurate and flexible deep learning models that can handle more complicated and changing situations.

Bibliography

[1] S. Bako, T. Vogels, B. Mcwilliams, M. Meyer, J. Novák, A. Harvill, P. Sen, T. Derose and F. Rousselle, “Kernel-predicting convolutional networks for denoising Monte Carlo renderings,” *ACM Transactions on Graphics,* vol. 36, pp. 1-14, 2017.

[2] S. Fazekas, B. Budai, R. Stollmayer and P. Kaposi, “Artificial intelligence and neural networksin radiology –Basics that all radiologyresidents should know,” *Imaging,* vol. 14, pp. 73-81, 2022.

[3] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision,* vol. 60, pp. 91-110, 2004.

[4] H. Bay, T. Tuytelaars and L. Van Gool, “SURF: Speeded Up Robust Features,” *Computer Vision and Image Understanding - CVIU,* vol. 110, pp. 404-417, 2006.

[5] Z. Wang, A. Bovik, H. R. Sheikh and E. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *EEE Trans. Image Process.,* vol. 13, pp. 600-612, 2014.

[6] M. Cao, Q. Yan, Y. Yu, H. Zhao and Z. Lyu, “MLV: Robust Image Matching via Multi-layer Verification,” *IEEE Sensors Journal,* vol. PP, pp. 1-1, 2024.

[7] J. Bromley, J. Bentz, L. Bottou, I. Guyon, Y. Lecun, C. Moore, E. Sackinger and R. Shah, “Signature Verification using a "Siamese" Time Delay Neural Network,” *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 7, p. 25, 1993.

[8] A. Nandy, S. Haldar, S. Banerjee and S. Mitra, “A Survey on Applications of Siamese Neural Networks in Computer Vision,” pp. 1-5, 2020.

[9] “Image Segmentation Review”.

[10] N. Ibtehaz and M. Rahman, “MultiResUNet : Rethinking the U-Net architecture for multimodal biomedical image segmentation,” *Neural Networks,* vol. 121, 20192019.

[11] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. Berg, W.-Y. Lo, P. Dollár and R. Girshick, “Segment Anything,” *Imaging,* 2023.

[12] R. Daudt, B. Saux and A. Boulch, "Fully Convolutional Siamese Networks for Change Detection," 2018.

[13] E. Parelius, "A Review of Deep-Learning Methods for Change Detection in Multispectral Remote Sensing Images," *Remote Sensing,* vol. 15, p. 2092, 2023.

[14] K. S. Basavaraju , N. Sravya, S. Lal, J. Nalini, C. S. Reddy and F. Dell'Acqua, "UCDNet: A Deep Learning Model for Urban Change Detection From Bi-Temporal Multispectral Sentinel-2 Satellite Images," *IEEE Transactions on Geoscience and Remote Sensing,* vol. 60, pp. 1-1, 2022.

[15] T. Suzuki, M. Minoguchi, R. Suzuki, A. Nakamura, K. Iwata, Y. Satoh and H. Kataoka, "Semantic Change Detection," pp. 1785-1790, 2018.

[16] R. Daudt, B. Saux, A. Boulch and Y. Gousseau, "Multitask learning for large-scale semantic change detection," *Computer Vision and Image Understanding,* vol. 187, 2019.

[17] D. Peng, L. Bruzzone, H. Guan and P. He, "SCDNET: A novel convolutional network for semantic change detection in high resolution optical remote sensing imagery," *nternational Journal of Applied Earth Observation and Geoinformation,* vol. 103, p. 102465, 2021.

[18] Z. Zheng, A. Ma, L. Zhang and Y. Zhong, "Change is Everywhere: Single-Temporal Supervised Object Change Detection in Remote Sensing Imagery," 2021.

[19] L. Tan, X. Zuo and X. Cheng, "CGMNet: Semantic Change Detection via a Change-Aware Guided Multi-Task Network," *Remote Sensing,* vol. 16, p. 2436, 2024.

[20] "Blender," [Online]. Available: https://www.blender.org/.

[21] "Python," [Online]. Available: https://www.python.org/.

[22] "Pytorch," [Online]. Available: https://pytorch.org/.

[23] "OpenCV," [Online]. Available: https://opencv.org/.

[24] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," *IEEE transactions on pattern analysis and machine intelligence,* vol. PP, 2021.