12-2019

# LambdaNet: A Novel Architecture for Unstructured Change Detection

Bryan Matthew Blakeslee
bmb8610@rit.edu

# LambdaNet: A Novel Architecture for Unstructured Change Detection

Bryan Matthew Blakeslee

# LambdaNet: A Novel Architecture for Unstructured Change Detection

Bryan Matthew Blakeslee
December 2019

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | Kate Gleason
*College of* ENGINEERING

*Department of Computer Engineering*

# LambdaNet: A Novel Architecture for Unstructured Change Detection

Bryan Matthew Blakeslee

**Committee Approval:**

---

Dr. Andreas Savakis *Advisor* Date
Professor – Department of Computer Engineering

---

Dr. Raymond Ptucha Date
Associate Professor – Department of Computer Engineering

---

Dr. Cory Merkel Date
Assistant Professor – Department of Computer Engineering

# Acknowledgments

I first want to thank my advisor Dr. Andreas Savakis for all his support and understanding during my research and Master's Thesis. His discussions, guidance, and knowledge were instrumental in many aspects of my work. I would not be where I am now without his advice.

I also want to thank my committee members Dr. Raymond Ptucha and Dr. Cory Merkel for their willingness to work with me on this endeavor.

I am also grateful to my parents, Mary and Lorin Blakeslee, for giving me the launchpad to be able to pursue my dreams.

I also wish to acknowledge Rusty, Wil, and Kitware for their sponsorship of my research.

Next, I want to express my gratitude to everyone in the Real-Time Vision and Image Processing Lab. Our conversations were helpful, enlightening, and always entertaining.

Finally, I'd like to thank Sid Pendelberry and everyone at RIT Research Computing for teaching me to "use the Sporc".

*To Mom and Dad. The check is in the mail.*

# Abstract

The goal of this thesis is the development of LambdaNet, a new type of network architecture for the performance of unstructured change detection. LambdaNet combines concepts from Siamese and semantic segmentation architectures, and is capable of identifying and localizing the significant differences between image pairs while simultaneously disregarding background noise. Changes are marked at the pixel level, by interpreting change detection as a binary (change/no change) classification problem.

Development of this architecture began with an evaluation of several candidate models, inspired by other successful network architectures and layers, including VGG, ResNet, and the Res2Net layer. Once the best performing LambdaNet architecture was determined, it was extended to incorporate a multi-class version of change detection. Referred to as directional change, this technique allows segmentation-based output of change information in four different classes: No change, additive change, subtractive change, and exchange.

Lastly, change detection is not the only unstructured operation of interest. One of the most successful unstructured techniques is that of artistic style transfer. This method allows information from a style image to be merged into a supplied content image. In order to implement this technique, a new variant of LambdaNet was developed, called LambdaStyler. This network is capable of learning multiple artistic styles, which can then be selected for application to the desired content image.

# Contents

# List of Figures

# List of Tables

# Chapter 1

## Introduction

### 1.1 Motivation

Deep convolutional neural networks (CNNs) have proven to be useful for processing data with very high dimensionality [6] [8] [9]. They have become the backbone of many advanced applications, such as self-driving vehicles [10], image-based medical diagnostics software [9], and other image analysis tasks. One of the applications of CNNs is the ability to automatically identify changes between pairs of images [3]. This extremely time consuming activity was once relegated exclusively to humans, as it required a complex set of skills including object detection (finding objects of interest), contextual analysis (determining what changes are important), and semantic segmentation (separating relevant changes from the remainder of the image).

Existing approaches generally fall into one of two categories, either structured or unstructured techniques. Structured methods, such as those of Daudt, et al. [3], are designed to look for specific types of changes in image pairs, as indicated by labelled ground truth maps. Unstructured methods, like those of Amin, et al. [11], tend to focus on low-detail, large-scale changes and have little to no ground truth. LambdaNet attempts to bridge this gap, creating a compromise between structured, object specific change detection, and unstructured, general change detection.

LambdaNet is a new type of Siamese change detection network, capable of pro-

cessing directional change information and increasing accuracy through the use of the Res2Net multiscale layer [4]. First, it incorporates change "directionality" into the output segmentation map, in order to identify areas of an image where objects were added or removed, rather than marking them as a generic change. Second, the new Res2Net backbone layer can enhance the performance of other network architectures through the utilization of multiscale information. This thesis extends the Res2Net encoder into a Res2Decode network. Finally, in order to demonstrate LambdaNet's suitability for other unstructured operations, LambdaStyler was developed as a version of the network capable of performing artistic style transfer.

## 1.2 Related Work

Pixel-wise change detection is a critical topic of research, due to its wide variety of applications, ranging from agriculture to surveillance to national defense. However, when performed by a human, it is also extremely time consuming, since it requires a high degree of attention to detail. As a result, there have been many attempts to automate this process, with varying degrees of complexity.

The first and simplest technique for change detection is image subtraction. Changes between images can be seen by taking the present image and subtracting a reference image from it. Any non-zero pixel value is then assumed to be a change. However, in order to remove unimportant changes, possibly caused by lighting variation or shadows, a threshold is necessary. Manually selecting this value is difficult, as setting the threshold too low would result in background noise being marked as a change, while a too large threshold would cause important changes to be missed. Therefore, a series of automatic thresholding techniques were developed [12] based on image statistics to programmatically select an appropriate value.

A variation on image subtraction for change detection is image ratioing. Using this technique, the present image is divided by a reference image [13]. If the ratio

value is close to 1, then the two pixels are highly similar. If it is significantly larger or smaller, then it is likely that a change has occurred. However, this technique also suffers from the same issue that plagues image subtraction, in that a threshold for change/no change must still be selected.

Furthermore, both the subtractive and ratiometric techniques are susceptible to other forms of outside influence, the most significant being sensor variation. First, if the sensor is changed, even within the same make and model, the calibration difference may result in erroneous detections. Second, if sensor replacement is necessary, then it may not be possible for the new device to be located in exactly the same manner as the original. This would induce a shift in registration between images captured in the old configuration versus the new configuration. This shift would be interpreted as a change by either of these methods.

At the other end of the spectrum, deep learning-based techniques have also been used for unstructured change detection. In the work of Amin, et al. [11], an AlexNet [14] architecture pre-trained on ImageNet [15], was utilized for unstructured change detection. By passing pairs of images through the network and extracting the activation maps, it is possible to create a binary change map. This is accomplished by bilinearly upsampling the variously-sized maps and concatenating them together along the channel-wise dimension. A Euclidean distance measurement is then made between the two activation map stacks and thresholded in order to obtain a final binary change map.

However, while simple to implement, this method also has drawbacks. Since it relies purely on Euclidean distance and thresholding to determine a change, it is void of any semantic content. This means that it is not capable of determining whether the change that occurred was a car moving or the leaves of a tree blowing in the background. As a result, these types of techniques are more suited to extremely large scale changes, such as those seen in satellite imagery, as opposed more detailed

changes that would be of interest in a street-level scene.

In order to assess its efficacy, LambdaNet is compared against the top performing models on the Change Detection 2014 results database. The selected models include SuBSENSE (Self-Balanced SENsitivity SEgmenter) [16], CwisarDH [17], and IUTIS (In Unity There Is Strength) [18] variants. The first method, SuBSENSE, operates on a combination of RGB and local binary similarity patterns (LBSPs) [19]. LBSPs work by dividing the image into individual cells of a pre-determined resolution. The center of each cell is taken as the reference and compared to all surrounding pixels in the cell. If the reference is greater than the neighbor pixel, then a zero is recorded. In all other cases, a one is recorded. This yields a single number describing that location. By sampling known areas of background and foreground, a series of reference samples can be catalogued. By collecting additional LBSP descriptors and RGB pixels and comparing them to the reference set, a classifier can determine if that local region of the image experienced a change.

Next, the CwisarDH [17] method, also referred to as a weightless neural network, relies on the WiSARD (Wilkie, Stoneham, Aleksander Recognition Device), a hardware-based neural network that operates using RAM lookup tables. These lookup tables store the outputs of the trained classification functions, which are evaluated as memory accesses. By summing the responses from the individual lookup tables, a similarity score can be generated indicating how close the test sample is to the contents of the training set. CwisardDH itself combines this WiSARD module with a per-pixel history buffer, which stores information about prior observed frames. When a pixel changes significantly, the associated buffer is flushed. If the change remains long enough to fill the buffer, then the system is retrained on that data, effectively converting that change into part of the background representation.

Finally, IUTIS [18] relies on genetic algorithms to construct an optimal configuration of other change detection methodologies. These chosen methodologies include

SuBSENSE [16], CwisarDH [17], and others. This algorithm combines the output of these different change detection methodologies together via a series of operations, including erosion and dilation filtering, median filtering, logical operations, and majority voting. The system randomly combines and evaluates configurations, while pruning poorly performing models, until a top performing network of operations is found.

## 1.3 Contributions

The main contributions of this thesis are:

- A LambdaNet architecture that explores alternatives to VGG-based network architectures, including Res2Net and pre-trained encoders, to enhance the accuracy of unstructured change detection.
- Creation of a Res2Net-based decoder architecture for LambdaNet to better utilize multiscale information generated by the Siamese encoders.
- Four class change detection, consisting of the classes No Change (NC), Additive Change (AC), Subtractive Change (SC), and Exchange (EC). This encodes "direction" information into the network's output, identifying changes as additions, removals, or swaps.
- Conversion of the LambdaNet architecture into a LambdaStyler network, capable of performing artistic style transfer.

Publications:

- **B. Blakeslee**, A. Savakis, "LambdaNet: A Fully Convolutional Architecture for Directional Change Detection," in Electronic Imaging 2020, Jan 2020.

- **B. Blakeslee**, R. Ptucha, and A. Savakis, "Faster art-cnn: An extremely fast style transfer network," in 2018 IEEE Western New York Image and Signal Processing Workshop (WNYISPW) , Oct 2018, pp. 1–5.

## 1.4 Thesis Organization

This thesis is organized in six chapters. Brief information about each chapter is mentioned below:

- Chapter 1. Introduction: Contains the motivation for the thesis topic, a review of current literature regarding change detection and style transfer, and the thesis's primary contributions.

- Chapter 2. Background Work: Provides an overview of prior methods in change detection and style transfer, along with a description of the datasets utilized for training and evaluation.

- Chapter 3. Methodology – LambdaNet: Details the techniques utilized to enhance the selected change detection network architectures. Also covers the details of the evaluation metrics utilized to quantify performance of the networks.

- Chapter 4. Methodology – LambdaStyler: Details the conversion of LambdaNet to the LambdaStyler architecture.

- Chapter 5. Results: A comparison of results with the current state-of-the-art methodologies, with respect to the aforementioned performance metrics. Also contains qualitative analyses of any novel unstructured behavior.

- Chapter 6. Discussion: A summary of the work performed and associated results, with a brief discussion of future work opportunities.

# Chapter 2

## Background

This chapter details the associated datasets used in this work and the related topics that form the inspiration for LambdaNet. Section 2.1 describes the datasets used to analyze LambdaNet. Sections 2.2 and 2.3 cover the areas of semantic segmentation and Siamese networks, respectively. Section 2.4 provides an overview of Res2Net, a multi-scale network layer that aids in recognizing objects of different sizes. Section 2.5 gives a brief summary of the general goals of unstructured change detection. Finally, Section 2.6 describes the area of artistic neural style transfer.

### 2.1 Datasets

The primary dataset used in the development of LambdaNet was the Change Detection 2014 (CD2014) dataset [1], which consists of 53 videos split across 11 categories. These videos were captured at either street level or from an elevated position in a variety of resolutions and conditions, including heavy snowfall, day and night scenes, and locations with high levels of background noise. They also vary in the modalities they were captured with, including RGB and thermal imaging technologies.

Labelling these images takes the form of a set of semantic segmentation maps, with one map corresponding to each frame of the video. The labels themselves are class-agnostic, in the sense that each pixel is labelled as either a change or no-change event, rather than explicitly labelling each pixel with a class name. Additional secondary

information is also labeled in the ground truth change masks such as a shadow region, unknown region (due to motion blur around moving objects), or a "Don't Care" region. Figure 2.1 shows a variety of sample images from the dataset, along with their corresponding ground truth masks.

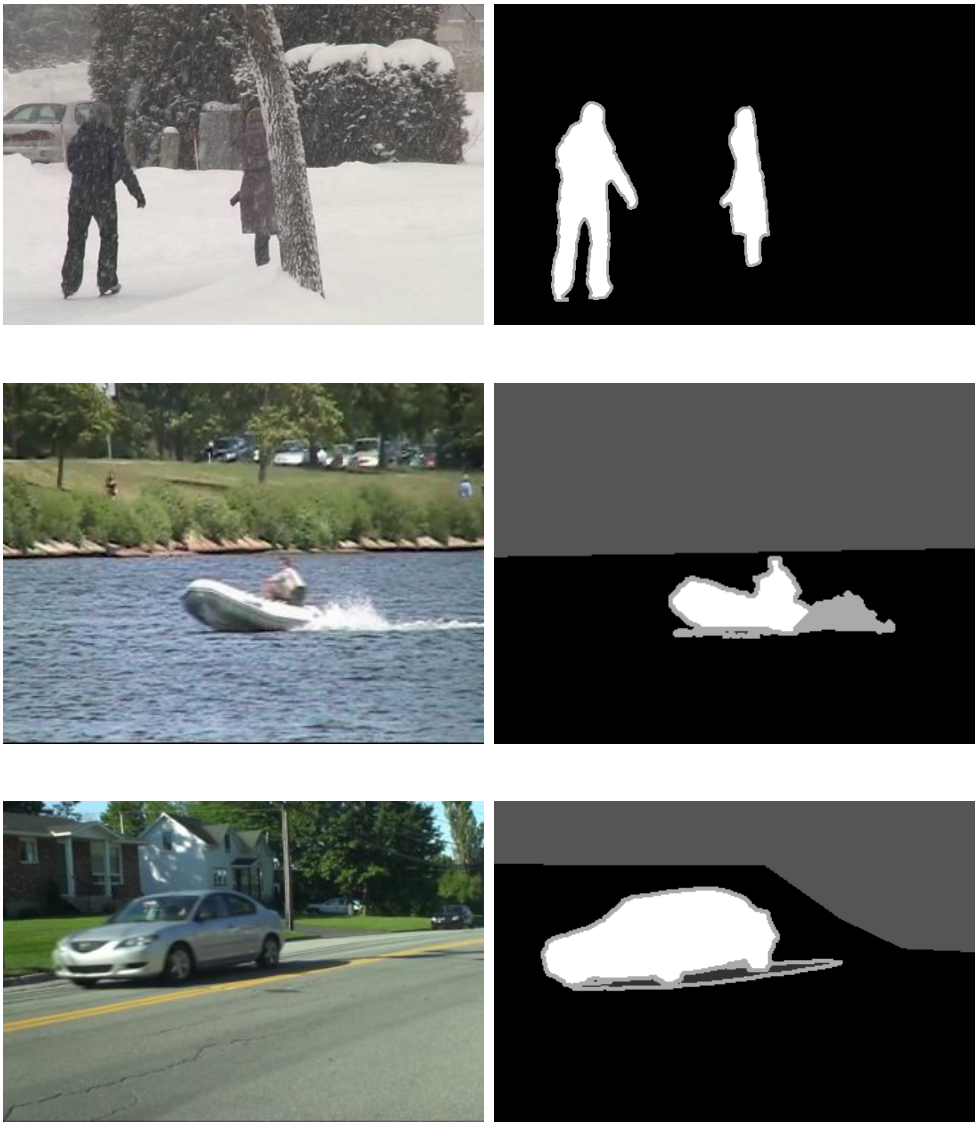


**Figure 2.1:** Sample images from the Change Detection 2014 [1] dataset. (Best viewed in color.) The left column contains sample images, while the right column consists of pixel level ground truth. Several other annotations are also present in the truth frames. In each of the truth frames, the target changes are labelled with white pixels. Surrounding the target pixels is a thin border of light grey, indicating a "Don't Care" region to account for motion blur. The truth frames in rows 2 and 3 also contain a large block of dark grey pixels, indicating a portion of the image outside the region of interest. Finally, in the row 3 truth image, the darkest grey region marks a shadow cast by the target object. This shadow also has a thin border of "Don't Care" light grey pixels.

## 2.2 Semantic Segmentation

Semantic segmentation is critical to the LambdaNet architecture, as it allows for the precise, pixel-level localization of changes between image pairs. One of the first deep learning-based semantic segmentation techniques is the deconvolutional network, introduced by Noh, et al. [2]. This network created the concept of a fully convolutional autoencoder. The encoder portion of the network, responsible for reducing the input image down to an intermediate form, is based on the VGG-16 architecture, developed by K. Simonyan, et al. [8]. However, rather than output a classification label for the image, the output of this encoder is then fed into a mirror image of the VGG-16 encoder. The decoder then expands the input back to the original dimensionality, assigning a class label for each pixel in the image. This process is shown graphically as Figure 2.2.

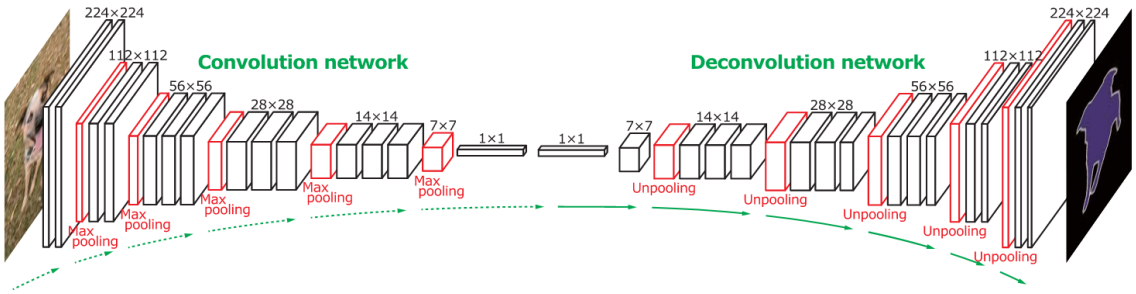



**Figure 2.2:** Architectural overview of the deconvolutional network. [2]

This network architecture is enabled by the use of two special operations: the unpooling and deconvolutional layers. While the unpooling layer is not utilized in LambdaNet due to architectural constraints, the deconvolutional layer plays a key role. This layer allows for the network to learn an upsampling function that expands a single pixel into a region of pixels via a learned filter. This operation is detailed graphically in Figure 2.3.

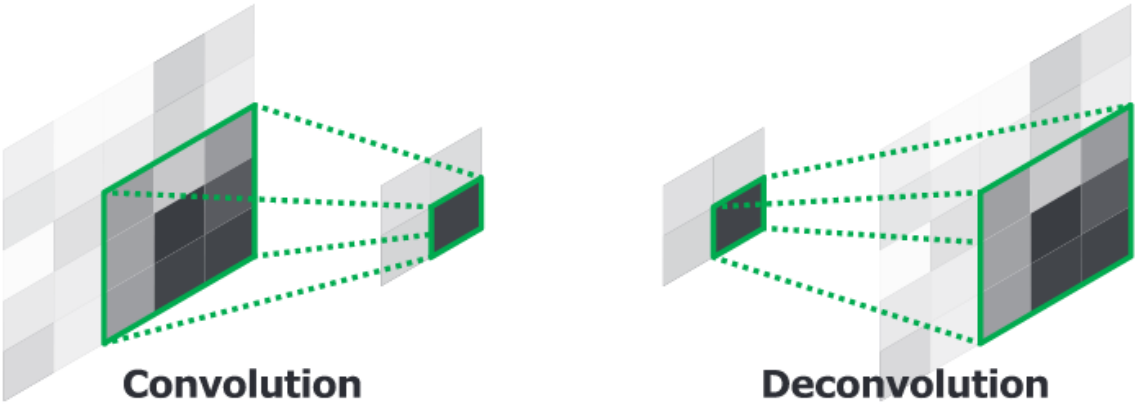



**Figure 2.3:** Comparison of convolution and deconvolution operations [2].

## 2.3 Siamese Networks

LambdaNet is also inspired by Siamese deep neural networks. Popularized by Koch, et al. [20] for one-shot learning, Siamese networks take their name from their symmetric construction, placing identical network components in parallel. This makes them well-suited for comparison-oriented applications. Operating by passing two data samples to their parallel structure, they generate a pair of intermediate outputs. These outputs can then be compared to each other. This comparison can take many forms, ranging from a simple Euclidean distance equation to a more complex trained decision network, such as in Rahman, et al. [21]. In the most extreme case, the decision network can be entirely replaced with a deconvolutional decoder, as in the case of the change detection network developed by Daudt, et al. [3]. Figure 2.4 shows a pair of example Siamese networks, with additional feedforward connections.

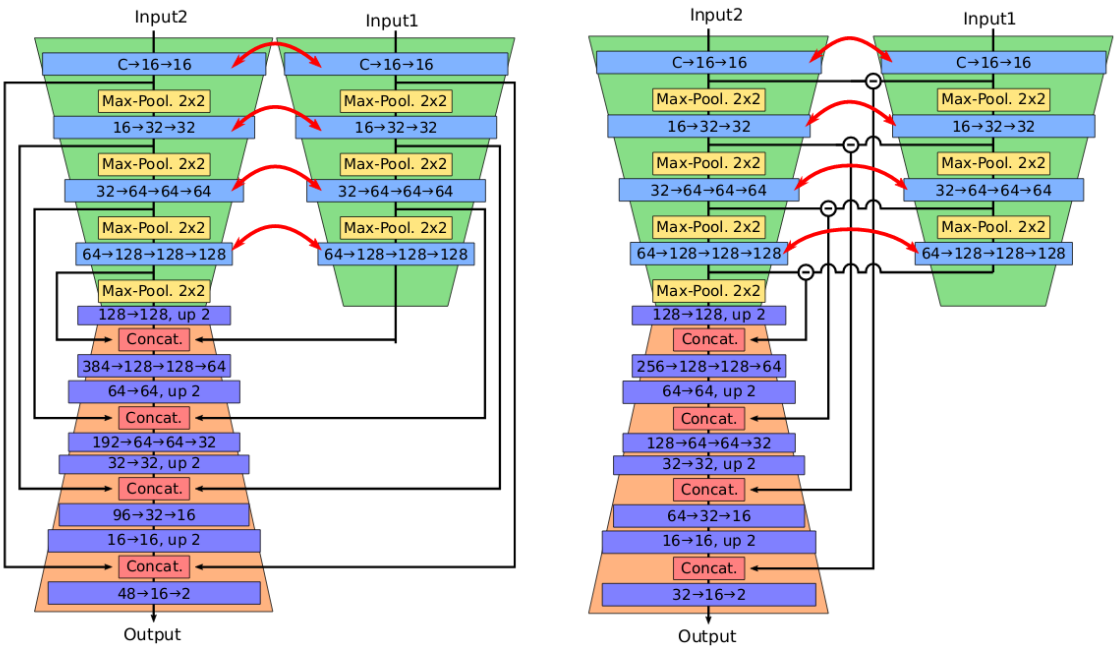



**Figure 2.4:** Example Siamese network architectures, featuring additional feedforward connections [3].

## 2.4 Res2Net Multi-Scale Network Layer

LambdaNet is a composite network, meaning it is constructed from pieces of other networks, such as VGG-16 [8]. Recently, a promising new backbone architecture was proposed: Res2Net [4]. This architecture allows for the improved integration and processing of multi-scale information. Multi-scale information allows for the network to learn to recognize a single object at many different sizes. This property is particularly useful for change detection, as changes between image pairs can be of any size. Qualitatively, this results in a nested set of receptive fields, shown in Figure 2.5, causing the network to be more sensitive to variations in the size of objects of interest.

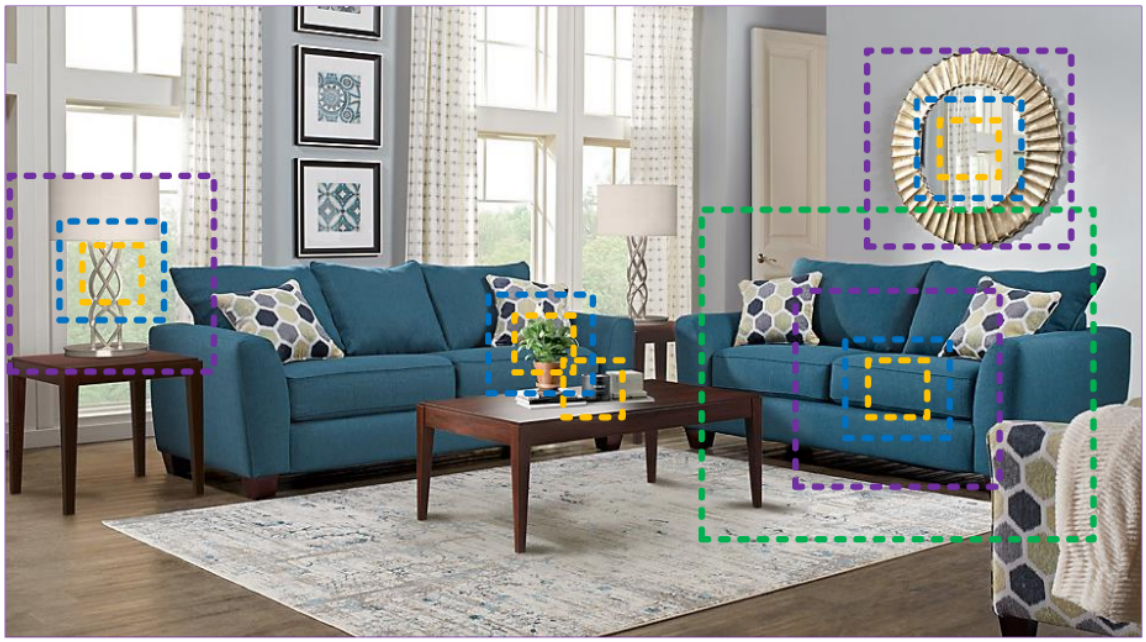


**Figure 2.5:** Nested receptive fields (multi-scale visual representation). Dotted lines of different colors represent different receptive field sizes [4].

This behavior is brought about through a series of operations, shown in Figure 2.6. First, the input activation map is convolved with a 1x1 set of filters to yield activation map X. Activation map X is then split into S (in this case four) groups, along the channel dimension. (E.g.: If S = 4 and the activation map has MxN elements and 64 channels, this operation results in four MxN blocks with 16 channels each.) The activation map Y represents an output buffer with the same dimensions as X for the multi-scale convolution represented by the kernels K. Block X1 is passed through to the corresponding Y1 slot. Block X2 is convolved with kernel K2 and placed in slot Y2. However, the left pointing arrow indicates that the output of this convolution (Y2) is also summed with block X3, before being convolved with kernel K3 and the output placed in slot Y3. This process repeats for slot Y4. The multiscale behavior originates from the forking connections after the individual kernels, labeled as K. Forking and chaining kernels together in this manner results in a series of receptive fields of multiple sizes sharing a common center point. The activation map Y is then convolved with a 1x1 set of filters to yield activation map Z. Finally, the activation

map Z is then summed with the input activation map to yield the output activation map O.

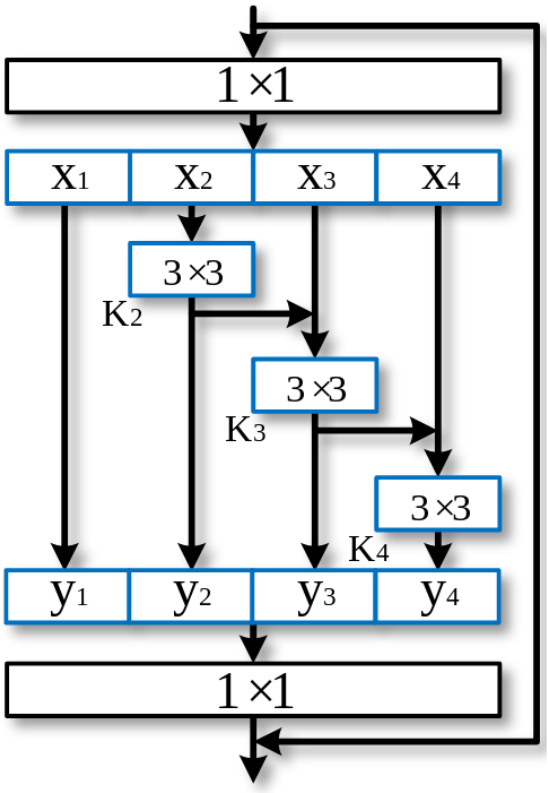



**Figure 2.6:** Detail view of Res2Net module [4].

## 2.5 Unstructured Change Detection

Change detection itself has also been an area of interest in deep learning research. Recent research in this area has focused heavily on leveraging Siamese networks. However, some of these approaches, such as that of Varghese, et al. [22], focus on class-based change detection. Their technique utilizes the VL-CMU-CD dataset [23], which is a semantic segmentation change detection dataset, labeled with 11 different object classes. Other techniques, such as Daudt, et al. [3], treat this domain as a binary classification problem, labeling each pixel as either "Change" or "No Change".

A subset of this research area is unstructured change detection. Under this

paradigm, the comparisions between images become more generalized. Rather than focusing on identifying specific change targets, such as cars or pedestrians, the goal is to identify any significant change in an image, regardless of whether it was explicitly labelled. These significant changes can take the form of objects, such as people or cars, or background structures, such as buildings. Simultaneously, it must be able to reject insignificant changes caused by background noise, such as the leaves of trees blowing in the wind. In the most extreme case, unstructured change detection can be applied to the landscape itself, searching for changes in geography or land cover, as in Chianucci, et al. [24].

## 2.6 Artistic Neural Style Transfer

Artistic neural style transfer is the act of taking a content image, usually a photograph, and a style image, usually a painting, and combining them into a single image containing the characteristics of both inputs. Qualitatively, if the content image is a portrait and the style image is an abstract texture, then the output of the network would be the portrait rendered in the style of the supplied texture.

This type of style transfer can be accomplished in two ways, either via an iterative technique or by a variety of feedforward methods [6] [25] [26]. The iterative method, developed by Gatys, et al. [5], treats style transfer as an optimization problem and is shown as Figure 2.7. First, the content and style images are fed through a pre-trained VGG16 network. However, rather than save the classification output, a set of selected activation maps are saved. In the case of the content image, this is usually a single layer from late in the loss network, so as to focus on the objects. For the style image, this is a set of maps from both shallow and deep levels of the loss network, in order to capture a wide variety of frequency content. A noise image is then passed into the network and the same set of activation maps are saved. These maps are then used to compute a weighted linear combination of the content and style loss functions. These

formulas are shown as Equations (2.1) to (2.5). Symbol definitions are provided in Table 2.1.

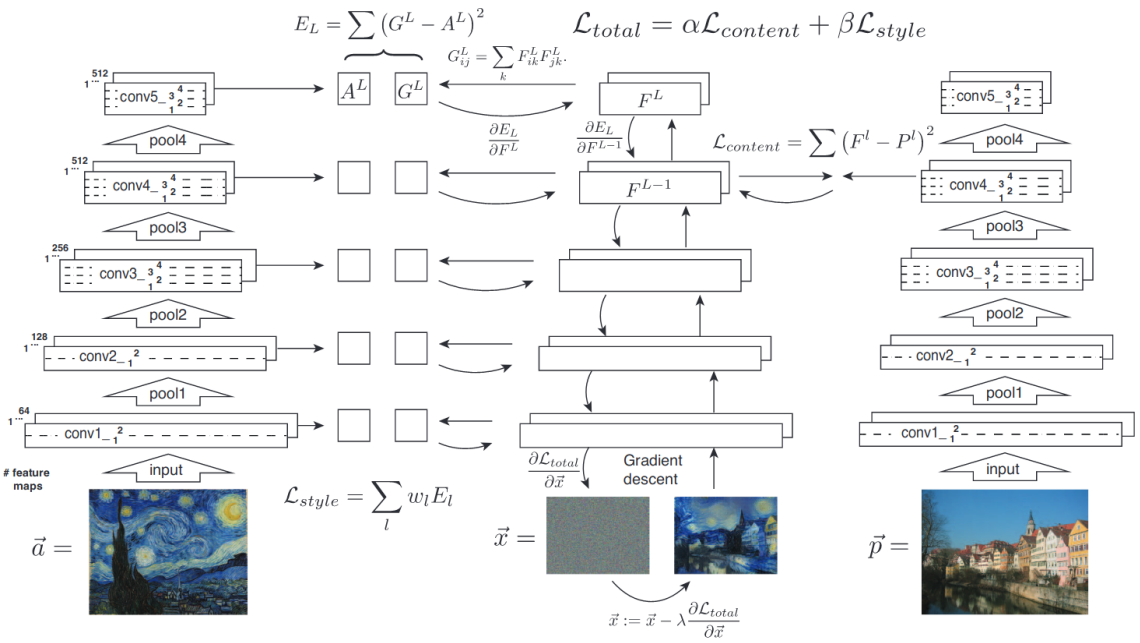



**Figure 2.7:** Illustration of iterative style transfer [5].

| Variable | Definition |
|---|---|
| $\mathcal{L}_{content}(\boldsymbol{P}, \boldsymbol{X}, l)$ | Content loss between the original image activation maps $\boldsymbol{P}$ and the generated image activation maps $\boldsymbol{X}$ at layer $l$ |
| $\boldsymbol{F}^l_{ij}$ | Activation map of $\boldsymbol{X}$ at layer $l$ |
| $\boldsymbol{P}^l_{ij}$ | Activation map of $\boldsymbol{P}$ at layer $l$ |
| $\boldsymbol{G}^l_{ij}$ | Gram matrix of generated image at layer $l$ |
| $E_l$ | Contribution of layer $l$ to style loss |
| $N_l$ | Number of feature maps at layer $l$ |
| $M_l$ | Number of elements in feature map at layer $l$ |
| $\boldsymbol{A}^l_{ij}$ | Gram matrix of original image at layer $l$ |
| $\mathcal{L}_{style}(\boldsymbol{A}, \boldsymbol{X})$ | Style loss between the original image $\boldsymbol{A}$ and the generated image $\boldsymbol{X}$ |
| $w_l$ | Weight of style loss at each layer $l$ |
| $\mathcal{L}_{total}(\boldsymbol{P}, \boldsymbol{A}, \boldsymbol{X})$ | Total loss function |
| $\alpha, \beta$ | Content loss weight and style loss weight, respectively |

**Table 2.1:** Definition of variables utilized in Equations (2.1) through (2.5).

$$[H]\mathcal{L}_{content}(\boldsymbol{P}, \boldsymbol{X}, l) = \frac{1}{2}\sum_{i,j}(\boldsymbol{F}^l_{ij} - \boldsymbol{P}^l_{ij})^2 \tag{2.1}$$

$$\boldsymbol{G}^l_{ij} = \sum_k \boldsymbol{F}^l_{ik}\boldsymbol{F}^l_{jk} \tag{2.2}$$

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}(\boldsymbol{G}^l_{ij} - \boldsymbol{A}^l_{ij})^2 \tag{2.3}$$

$$\mathcal{L}_{style}(\boldsymbol{A}, \boldsymbol{X}) = \sum_{l=0}^{L} w_l E_l \tag{2.4}$$

$$\mathcal{L}_{total}(\boldsymbol{P}, \boldsymbol{A}, \boldsymbol{X}) = \alpha\mathcal{L}_{content}(\boldsymbol{P}, \boldsymbol{X}) + \beta\mathcal{L}_{style}(\boldsymbol{A}, \boldsymbol{X}) \tag{2.5}$$

The content loss, shown as Equation (2.1), is effectively a mean squared error between the activation map of the content image and the activation map of the generated image. However, the style loss computation requires several steps. First, Equation (2.2) is used to compute the Gram matrix for both the style image and the generated image. This creates a matrix that correlates which style components activate together in the loss network. The mean squared error is again used in Equation (2.3) to minimize the differences in style between the style and generated images. Equation (2.4) is an optional term, which can be used to weight the style contributions of individual layers of the loss network. Finally, Equation (2.5) is the weighted linear combination of both the content and style losses.

A modified version of the standard backpropagation algorithm is then performed on the network. In this case, the weights of the network are frozen and the gradient updates are carried through all the way to the input image. This results in the slow refinement of the noise image into an image containing visual elements from both the content and style images.

Figure 2.8 shows the alternative method for stylization, based around a feedforward technique, created by Johnson, et al. [6]. This can vary based on the type

of network used, but generally it uses an architecture based around a residual autoencoder. During training, this network is connected to the same network used in the iterative technique of Gatys, et al. [5]. Training proceeds by first applying the content and style images to the loss network and saving their respective activation maps. The content image is then applied to the residual autoencoder, the output of which is then fed into the loss network. Again, the same activation maps are saved and the loss function is computed. The gradients are then backpropagated through the loss network, whose coefficients remain frozen, before flowing into the residual autoencoder. It is here that the coefficients are updated. After a sufficient number of training epochs, the residual autoencoder will have learned to impart the characteristics of the chosen style image into any image applied to its input.

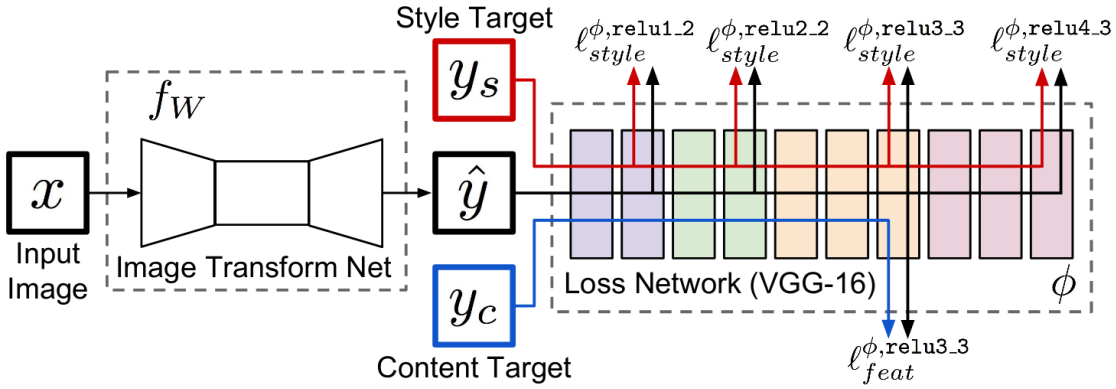



**Figure 2.8:** Illustration of feed-forward style transfer [6].

# Chapter 3

## Methodology – LambdaNet

This chapter provides greater detail on the architecture methodologies used to develop LambdaNet. Section 3.1 examines the initial networks, while Section 3.2 details the various configurations of the Change Detection 2014 [1] dataset that were used to evaluate the network. Section 3.3 gives a summary of the evaluation metrics used to quantify the network's performance. Section 3.4 covers the experimental process that was followed to determine the highest performing version of the network. Section 3.5 covers the modifications that were made in order to incorporate "directional" change into the network architecture.

### 3.1 Exploration of LambdaNet Architectures

Since LambdaNet is a new type of architecture, based around a fusion of elements from Siamese networks and autoencoders, the first step was to determine the architecture for the system. A generic view of the network is shown as Figure 3.1.

LambdaNet is composed of three different components, the encoders, the fusion node, and the decoder. In this architecture, the encoders have shared weights and are responsible for generating an intermediate feature map representation. Since the encoder output is taken from the final layer, this feature map representation will focus mostly on the "objectness" of targets in the input images and ignore the edge-based information.

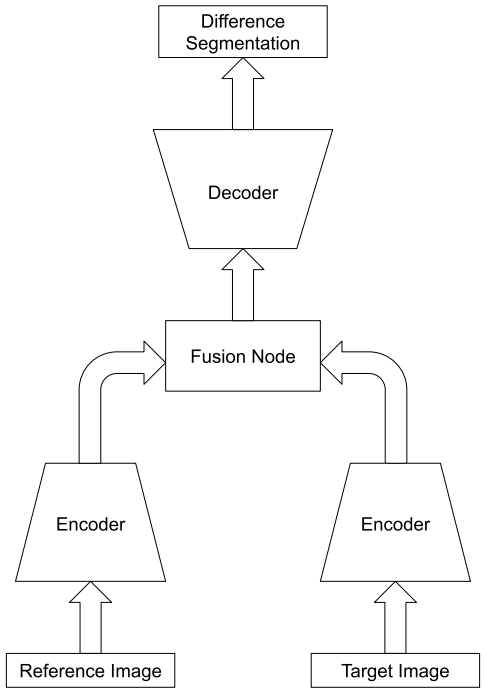
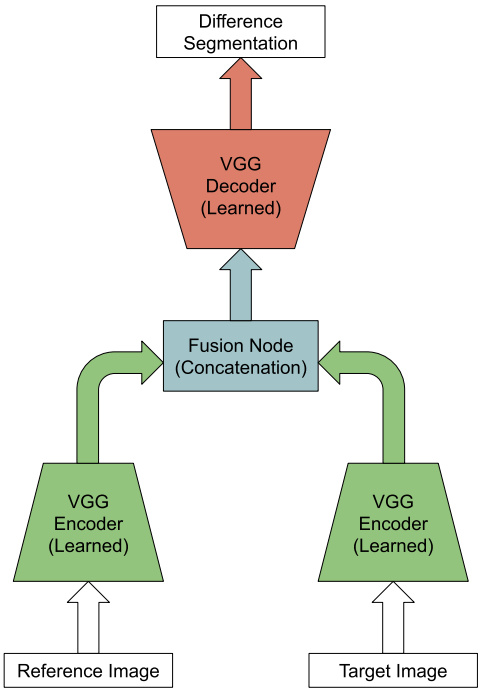


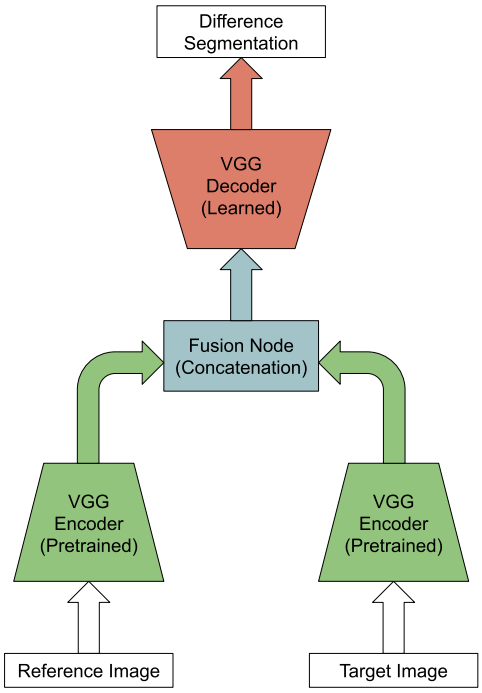

**Figure 3.1:** Abstracted view of the LambdaNet architecture.

This feature map pair is then passed to the fusion node, which is responsible for concatenating these maps into a single representation. This unified map is then passed to the decoder segment, which is responsible for separating the content of the unified map into a binary change map, indicating the differences between the images. Under this architecture, both the encoder and decoder may be learned, or pre-trained encoders may be used with a learned decoder. No learning takes place in the fusion node.

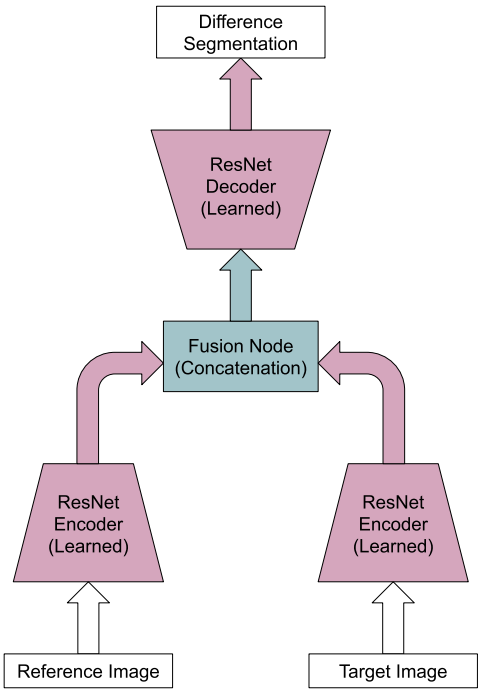
This abstracted architecture leads to several different possible concrete realizations. We focus on a set of the most common network archetypes, predominantly inspired by VGG16 and ResNet. Preliminary experimentation revealed that the quality of the change segmentation was dependent on the scale of the target in question. Thus, multiscale information was integrated into LambdaNet, using the Res2Net module. These six configurations are shown as Figure 3.2.



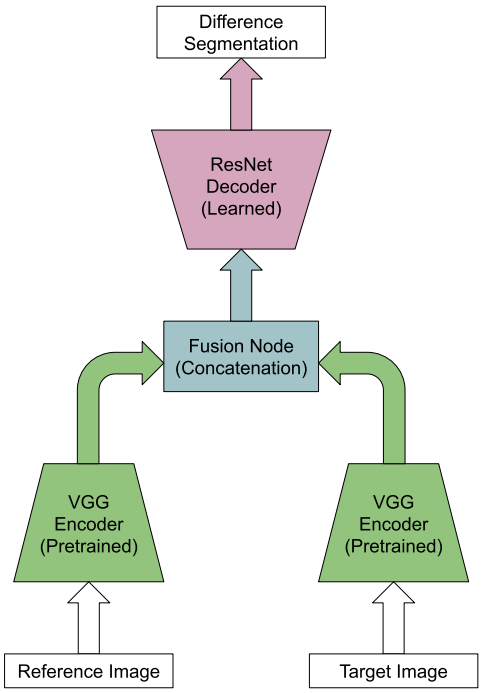

**(a)** Full VGG (FVGG): End-to-end trained version of LambdaNet, based on a VGG encoder/decoder structure.



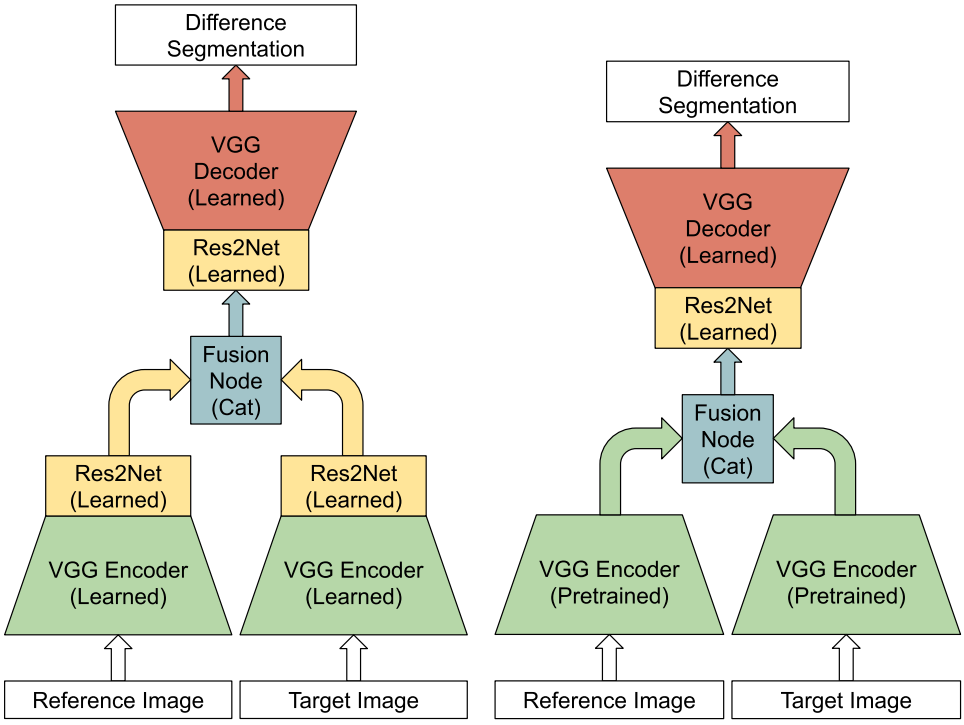

**(b)** Hybrid VGG (HVGG): Hybrid version of LambdaNet, based on a VGG encoder pre-trained on ImageNet and a learned VGG decoder.




**(c)** Full Residual (FRES): End-to-end trained version of LambdaNet, based on a residual encoder/decoder structure.




**(d)** Hybrid Residual (HRES): Hybrid version of LambdaNet, based on a VGG encoder pre-trained on ImageNet and a learned residual decoder.




**(e)** Full MultiScale (FMS): End-to-end trained version of LambdaNet, based on a VGG encoder/decoder structure, with internal learned multiscale layers.

**(f)** Hybrid MultiScale (HMS): Hybrid version of LambdaNet, based on a VGG encoder pre-trained on ImageNet, and a learned VGG decoder and multiscale layer.

**Figure 3.2:** Initial six concrete implementations of LambdaNet.

As the initial evaluation is framed as a binary classification problem, the binary cross-entropy loss function was chosen for training, as shown in Equation (3.1).

$$\mathcal{L}_n = -w_n[y_n \cdot \log(x_n) + (1 - y_n) \cdot \log(1 - x_n)] \tag{3.1}$$

One particular challenge in working with the Change Detection 2014 [1] dataset is that the distribution of change and no change classes is extremely unbalanced in favor of the no change class. Preliminary testing showed that the LambdaNet architecture learned the no change class much more effectively that it did the change class. In order to correct for this class skew, an additional weighting factor was used in the

loss function, which is detailed in Equation (3.2).

$$\mathcal{C}hangeWeight = \ln\left(\frac{TotalClassCount}{ChangeClassCount}\right) \tag{3.2}$$

Furthermore, an additional regularization method was employed to prevent overfitting of the network. This consisted of a random swap applied to the input images to ensure that the reference and target images were applied equally to both inputs. This mitigates overfitting in the decoder by preventing the concatenation from always passing the encoder's activation maps in the same order to the decoder segment. In the case of the target-target dataset configuration, it can also be viewed as a form of data augmentation, as the pattern of object removals and additions will be randomly reversed.

## 3.2 Dataset Configuration

The next step in developing LambdaNet was the configuration of the Change Detection 2014 [1] dataset. Three different setups were used, reference-target, target-target, and category holdout. Each of these setups all held the same set of image/truth pairs. However, each configuration possessed either a different pairing of input images or a different style of training/validation split, detailed below.

### 3.2.1 Reference-Target Splits

Under the reference-target training methodology, the data took the form of a triplet consisting of reference, target, and ground truth images. The reference image was selected to contain absolutely no marked change targets. In cases where this was not possible, a reference image was chosen with the change target almost completely out of frame. Meanwhile, the target image contained all objects of interest. As a

result, the ground truth image was the target image's corresponding ground truth map. Functionally, this treated the reference image as a conditional input, which influenced the semantic segmentation of the target image, producing a predicted change map with respect to the reference input.

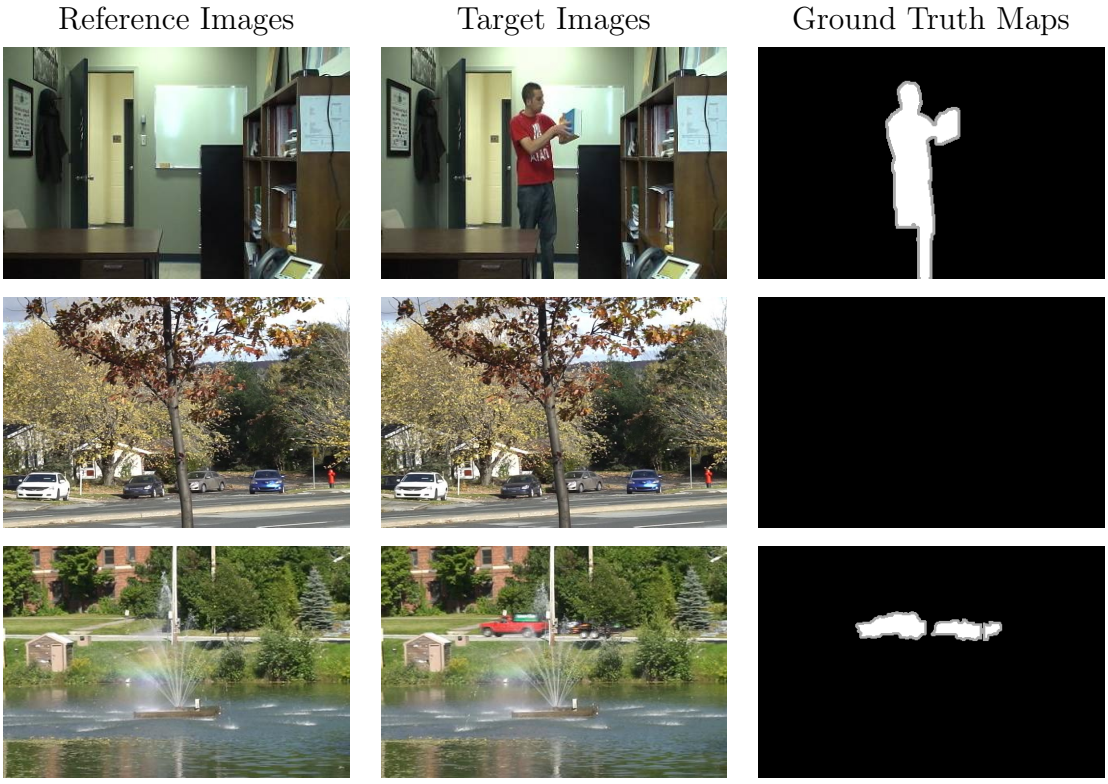



**Figure 3.3:** Sample triplet entries for the reference-target split configuration. The left column is the reference image, the center column is the target image, and the right column is the ground truth segmentation map. Note that rows 2 and 3 both contain dynamic background elements (trees, fountains) that are unlabelled.

In terms of the split configuration, all 53 videos from the dataset were placed into a single pool, configured for nine-fold cross validation. This entailed splitting the pool into nine random groups of six videos each. (Since 53 is a prime number the final validation group had five videos.) One group was then chosen as a validation set and all other videos placed in the training set. This process was repeated nine times, such that the entire dataset was represented in the total of the validation splits. This allowed nine different versions of the network to be trained, simulating the

ability of the network to be tested on every video in the dataset, while simultaneously guaranteeing that there was always a version of the network that had never seen a particular video before.

### 3.2.2 Target-Target Splits

The next dataset configuration created was the target-target split. This method of organization also took the form of a triplet, except in this case both input images contained marked change targets. This required an additional pre-processing step: The fusion of two ground truth maps into a single image via a logical OR operation. This resulted in a single truth image containing change targets from both input images. The dataset was then configured in the same nine-fold cross validation manner as the reference-target configuration.

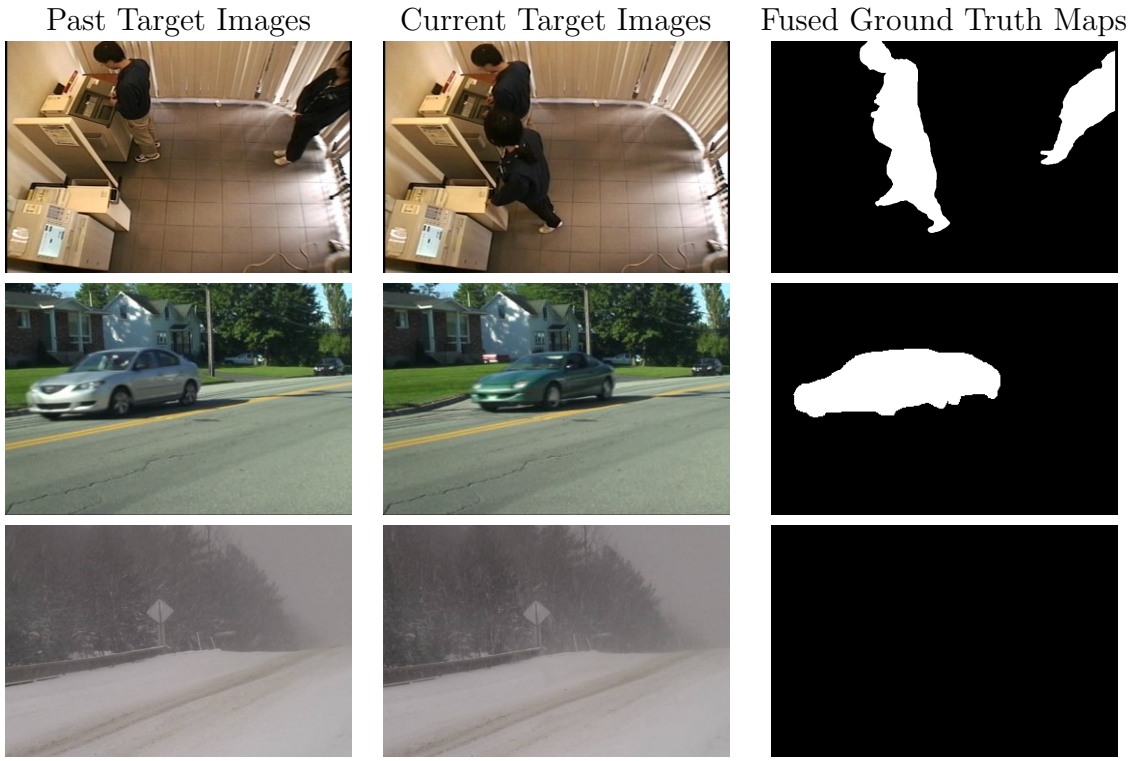



**Figure 3.4:** Sample triplet entries for the target-target split configuration. The left column is the prior target image, the center column is the present target image, and the right column is the fused ground truth segmentation map. Note that row 2 has overlapping targets and row 3 contains no change targets.

### 3.2.3 Category Holdouts

The final dataset configuration was that of a category holdout. In this case, the training triplet was structured in the same manner as the reference-target splits. However, rather than choose the same nine-fold cross validation setup, eleven-fold cross validation was used instead. This was because the Change Detection 2014 [1] dataset contains eleven categories of videos. For each split configuration, a single category was used as the validation set, while all other categories were placed in the training set. This ensured that there was always an entire class of videos that remained unseen by at least one version of the network.

## 3.3 Evaluation Metrics

This section covers the metrics that were chosen to evaluate the LambdaNet architecture. These metrics were selected based on their utilization in the Change Detection 2014 [1] contest, in order to enable direct comparison to existing methods. Intersection over union was also selected, as it provides a direct measure of the quality of the change segmentation. Table 3.1 contains the abbreviations and symbol definitions used in the following formulas.

| Name | Abbreviation | Definition |
|---|---|---|
| True Positive | TP | Correct change classification |
| True Negative | TN | Correct no change classification |
| False Positive | FP | Incorrect change classification |
| False Negative | FN | Incorrect no change classification |
| False Positive Rate | FPR | See Section 3.3.1 |
| False Negative Rate | FNR | See Section 3.3.2 |
| Percent Wrong Classification | PWC | See Section 3.3.3 |
| Intersection over Union | IoU | See Section 3.3.8 |

**Table 3.1:** Listing of abbreviations used in metric equations.

### 3.3.1 False Positive Rate

The False Positive Rate measures the fraction of no change pixels that were incorrectly marked as changes. The denominator consists of the total number of no change pixels, expressed as the sum of incorrect change classifications and correct no change classifications. This can be seen in Equation (3.3).

$$\mathcal{FPR} = \frac{FP}{FP + TN} \tag{3.3}$$

### 3.3.2 False Negative Rate

The False Negative Rate performs similarly to that of the False Positive Rate. In this case, the denominator contains the total number of change pixels, defined as the sum of correctly classified changes and changes that were erroneously marked as no change. This is shown as Equation (3.4).

$$\mathcal{FNR} = \frac{FN}{TP + FN} \tag{3.4}$$

### 3.3.3 Percent Wrong Classification

Percent Wrong Classification measures the total percentage of incorrectly classified pixels. The numerator consists of the total number of incorrectly classified pixels, while the denominator contains the total number of pixels. This is detailed in Equation (3.5).

$$\mathcal{PWC} = 100 * \frac{FN + FP}{TP + FN + FP + TN} \tag{3.5}$$

### 3.3.4 Specificity

Specificity is responsible for computing the fraction of no change pixels that were correctly classified as no change. In this case, the denominator is the total number of no change pixels. This is detailed in Equation (3.6).

$$\mathcal{S}pecificity = \frac{TN}{TN + FP} \tag{3.6}$$

### 3.3.5 Recall

Recall operates similarly to specificity, in that it measures the fraction of correctly marked change pixels. Here, the denominator is the total number of pixels marked as change. This can be seen in Equation (3.7).

$$\mathcal{R}ecall = \frac{TP}{TP + FN} \tag{3.7}$$

### 3.3.6 Precision

Precision calculates what fraction of pixels that were predicted to have changed were actually valid changes. For this case, the denominator is the total number of pixels marked as change, irrespective of that classification's correctness. This can be seen in Equation (3.8).

$$\mathcal{P}recision = \frac{TP}{TP + FP} \tag{3.8}$$

### 3.3.7 F1 Score

When the precision and recall are combined as a harmonic mean, it yields the F1 score. This metric distills the precision and recall metrics into a single ratio. The formula is shown as Equation (3.9).

$$\mathcal{F}1Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.9}$$

### 3.3.8 Intersection Over Union

The intersection over union (IoU) measures the amount of overlap between the predicted segmentation map and the ground truth segmentation map. The numerator quantifies the intersection where both segmentation maps indicate change. The denominator sums the correct change classification with both missed changes and wrongly marked changes to encompass the union of these two maps. This ratio is shown as Equation (3.10).

$$\mathcal{I}oU = \frac{TP}{TP + FN + FP} \tag{3.10}$$

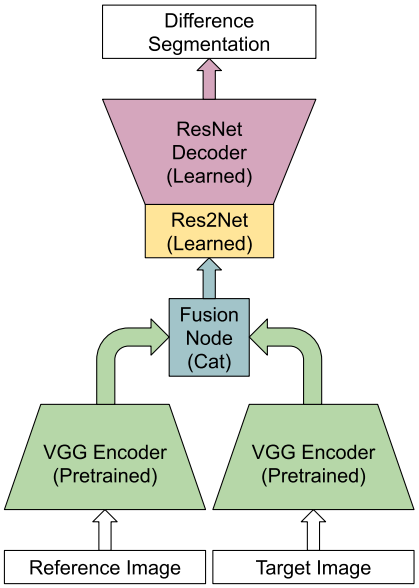
## 3.4 Evaluation of LambdaNet Architectures

The next step to determining the network architecture was to select the data splits that would be used for evaluation. The reference-target and target-target dataset configurations each contain nine splits, while the category holdout configuration contains eleven splits, for a total of twenty-nine possible dataset configurations. With six different architectures to evaluate, this would require training 174 models, which is computationally intractable.

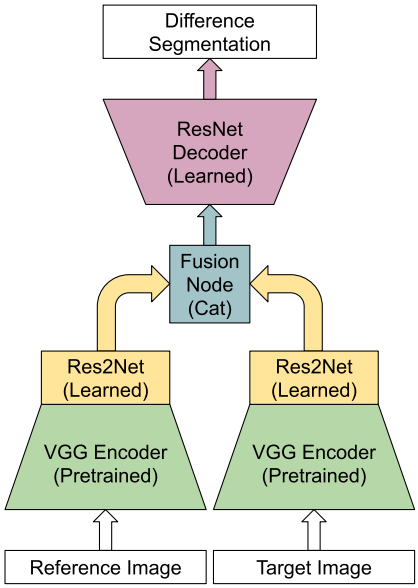
As a result, inspiration was drawn from the domain of genetic algorithms. In order to judge the most fit architecture, the F1-Score (Equation (3.9)) was chosen as the optimizing metric. The rationale behind this decision is that the F1-Score combines both precision (Equation (3.8)) and recall (Equation (3.7)) into a single metric, both of which measure the accuracy of the change segmentation. Precision measures the ratio of correctly marked changes to the total number of marked changes, while recall measures the ratio of correctly marked changes to the total number of changes in the ground truth.

Rather than evaluate all possible options, a single data split was chosen at random from the target-target configuration. This split was chosen, as it encompasses a middle ground between the reference-target and category holdout modes. It allows for testing network behavior under the conditions of additive, subtractive, and target exchange conditions, while simultaneously ensuring that the network is tested on videos it has not seen before. This split comprised stage 1 testing and was used to train and evaluate all six initial architecture configurations. These models were then ranked according to the F1-Score. The top 50% of models were then selected, yielding three architectures for stage 2 testing.

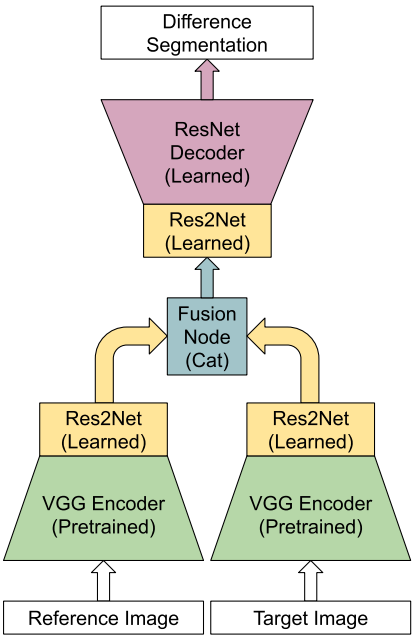
Stage 2 testing consisted of training and evaluation on a wider variety of data splits from all possible configurations, including reference-target, target-target, and category holdout modes. In each case, two splits were selected randomly from each mode. These six splits were then used to train and evaluate the top three models from stage 1 testing. The resulting models were then grouped based on the split they were trained on. For each of these six splits, the three models were then ranked by their F1-Score.




**(a)** Hybrid Single Multiscale Residual (HM1RES): Hybrid version of LambdaNet, based on a VGG encoder pre-trained on ImageNet, and a learned multiscale and residual decoder pair.




**(b)** Hybrid Double Multiscale Residual (HM2RES): Hybrid version of LambdaNet, based on a VGG encoder pre-trained on ImageNet, coupled to a learned multiscale layer and residual decoder.




**(c)** Hybrid Triple Multiscale Residual (HM3RES): Hybrid version of LambdaNet, based on a VGG encoder pre-trained on ImageNet, coupled to a learned multiscale layer and multiscale layer/residual decoder pair.

**Figure 3.5:** Final three candidate models for LambdaNet.

As the stage 2 testing ended with an effective tie between the hybrid ResNet and hybrid Res2Net architectures, a third stage of testing was performed using the stage 2 splits. In stage 3, the ResNet and Res2Net architectures were combined into three new candidate models, shown in Figure 3.5. Once again, the models were sorted by their F1-score, with the model that placed first most being selected. This model is shown as Figure 3.5c.

## 3.5 Directional Change Detection

Once the LambdaNet architecture of Figure 3.5c was finalized, it was then extended from a binary change detection system into a multi-class change detection system. Under this paradigm, four classes are used to denote change at the pixel level: No change, additive change, subtractive change, and exchange. These classes are derived from both the per-frame pixel classification, in addition to which frames, either reference or target, the change occurs in. This class labelling is detailed in Table 3.2.

The generation of the new ground truth data was accomplished via a logical-OR-like operation between pairs of binary change masks. One mask was designated as the reference mask, while the other was taken as the target mask. The operation detailed in Table 3.2 was then applied to obtain the fused multi-class ground truth. The source masks were then reversed and the same operation applied. This was necessary, as in the binary change classification case the inputs were randomly swapped with a 50% probability to ensure that the architecture would not overfit. However, in this case, swapping the inputs would lead to the input images being inconsistent with the ground truth, as the additive and subtractive change classes would be reversed. Sample training triplets are shown as Figure 3.6.

| Classification | Color | Definition |
|---|---|---|
| No Change | Black | Pixel is not a change target in either reference or target frame |
| Additive Change | Red | Pixel is not a change target in the reference frame, but is a change target in the target frame |
| Subtractive Change | Blue | Pixel is a change target in the reference frame, but is not a change target in the target frame |
| Exchange | Green | Pixel is a change target in both the reference and target frames |

**Table 3.2:** Definitions and color coding for multi-class change detection.

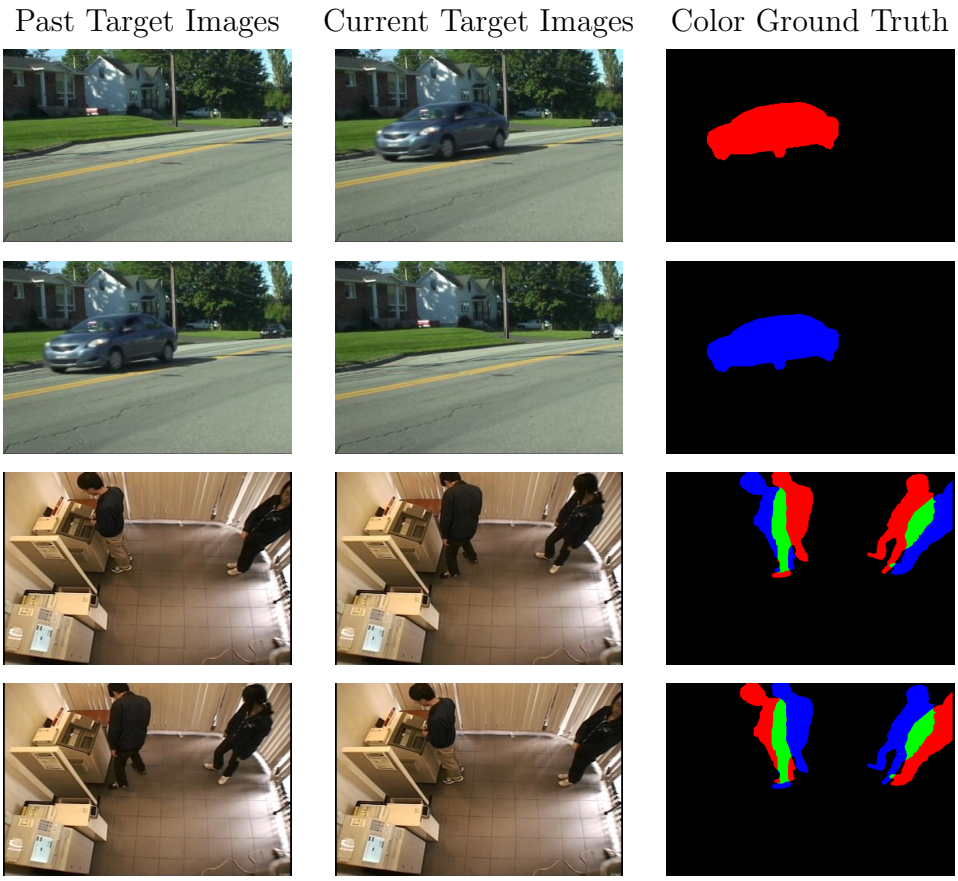



**Figure 3.6:** Sample colorized triplet entries for multi-class change detection. The left column is the prior target images, the center column is the current target images, and the right column is the fused ground truth segmentation map. Rows 1 and 2 use the same two swapped images to illustrate the directionality of the change. Red indicates additive change, while blue indicates subtractive change. Rows 3 and 4 are organized in the same manner, but are chosen to include the third exchange class in green. Note that depending on the order of the images, the additive and subtractive change regions are reversed, while exchange remains constant. Black indicates no change.

However, this reprocessed data contains the same class imbalances present in the binary classification case. As a result, a set of weighting factors was also computed for this configuration using Equation (3.2). This computation was performed on each of the three change classes, with the no change class being weighted as 1.

Finally, since this is no longer framed as a binary classification problem, the loss function was changed to a more general cross-entropy loss, shown as Equation (3.11). Training then proceeds normally, as in the binary classification case.

$$\mathcal{L}_{CE} = -\sum_{c=0}^{M-1} y_c \log(p_c) \tag{3.11}$$

# Chapter 4

## Methodology – LambdaStyler

### 4.1 Multiple Style Transfer with LambdaStyler

Given that the primary thrust of LambdaNet is to develop an architecture that is capable of performing unstructured change detection, this network was modified to determine its suitability for another unstructured technique: Fast style transfer. Based on the work of Johnson, et al. [6], it is known that a residual autoencoder is able to efficiently transfer a single style of artwork onto a provided content image. Drawing inspiration from this, an end-to-end trainable version of LambdaNet was constructed, called LambdaStyler. This architecture is shown as Figure 4.1.

The goal of this architecture is to perform multiple style transfer. Traditional fast style transfer [6] is restricted, such that the transformation network only learns a single artistic style. Under multiple style transfer, a single network is trained to impart one of many selectable styles to the content image. During inference, the reference input is designated as the content image input, while the target input is reserved for the style image. The network then evaluates imagery in the same manner as in the change detection case. The encoders reduce the input images to activation maps, which are then concatenated, and passed into the decoder section, which merges the maps into the final stylized image. By changing the style image, different styles can be imparted to the content image.

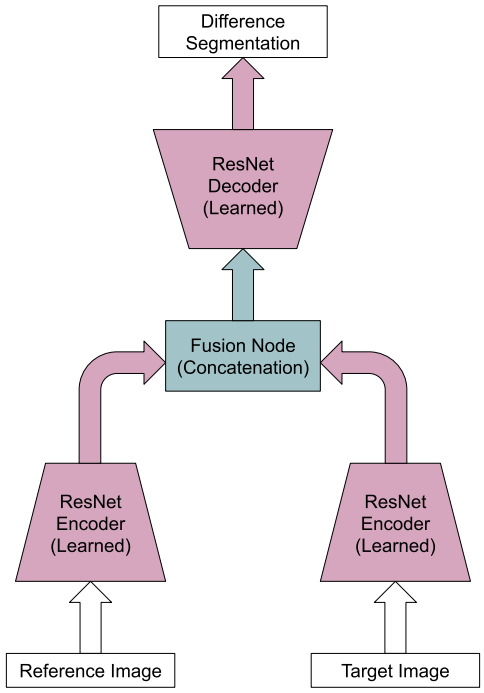



**Figure 4.1:** LambdaStyler: An end-to-end trainable architecture, based on a residual autoencoder.

Training is accomplished using a variation on the process used in the work of Johnson, et al. [6]. A VGG16 encoder trained on ImageNet is connected to the output of LambdaStyler as a perceptual loss function, while the same style and content loss functions detailed in Equations (2.1) to (2.5) were used. The difference is that instead of training on a single style image, multiple styles are used. Each pair of style and content images are first fed into the loss network, saving the respective activation maps. This same pair of images are then fed into the LambdaStyler network. This output is then passed through the VGG16 encoder and the style and content maps saved. These maps are then used to compute the total loss for the network. Backpropagation proceeds through the VGG16 loss network (whose coefficients are frozen) and into the LambdaStyler network, where the coefficients are updated. In order to prevent overfitting, the input images are randomly swapped. The result is a network that is able to merge two input images into a single stylized output. This modified training architecture is shown as Figure 4.2.

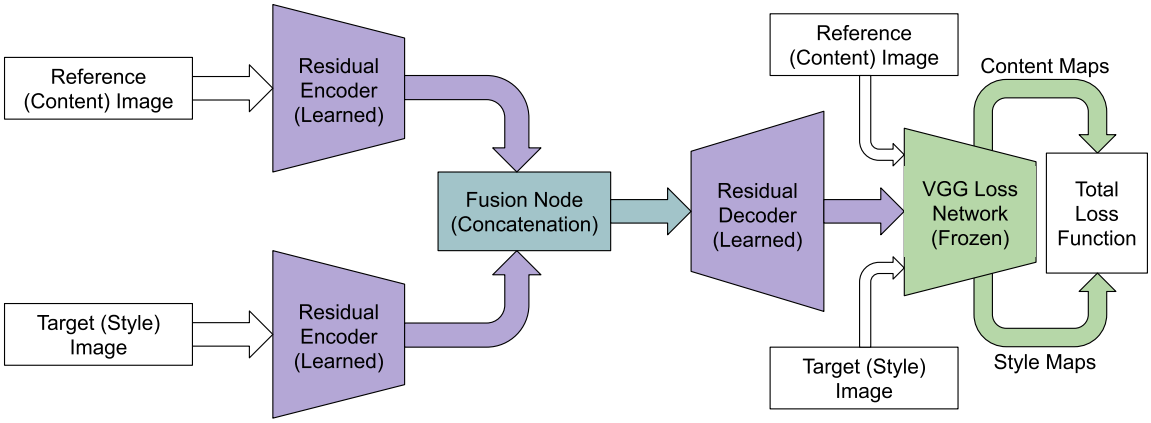



**Figure 4.2:** Training architecture for the LambdaStyler.

Training is accomplished using a selection of style images, in conjunction with the MS-COCO dataset [7], which form the content images. LambdaStyler was trained to impart one of four distinct styles to the provided content images, shown as Figure 4.3. Sample content images are shown as Figure 4.4.

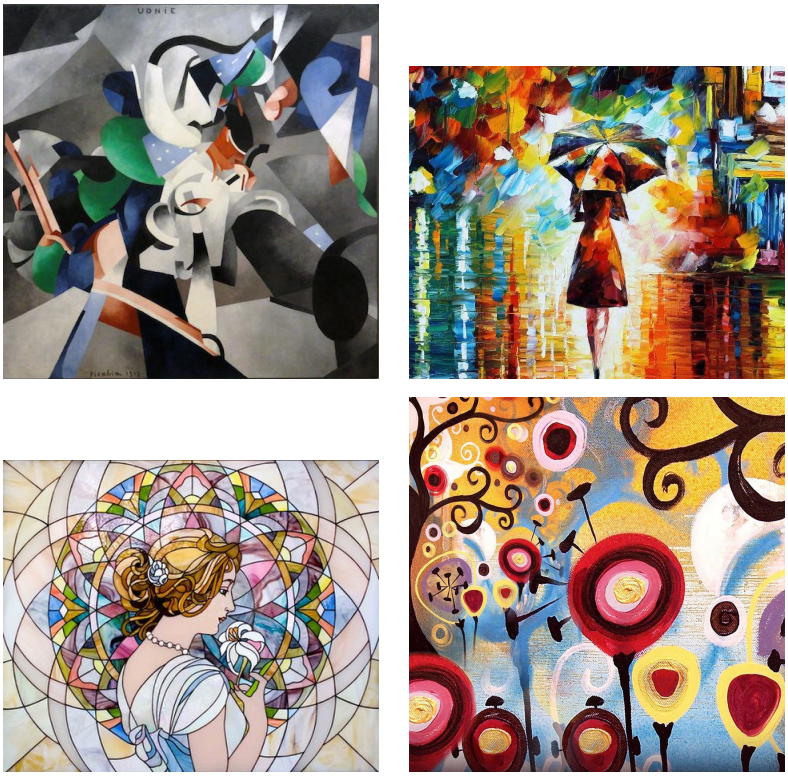


**Figure 4.3:** Style images used in training of LambdaStyler. Upper left is Udnie, by Francis Picabia. Upper right is Rain Princess, by Leonid Afremov. Lower left is Mosaic (artist unknown). Lower right is Candy (artist unknown).

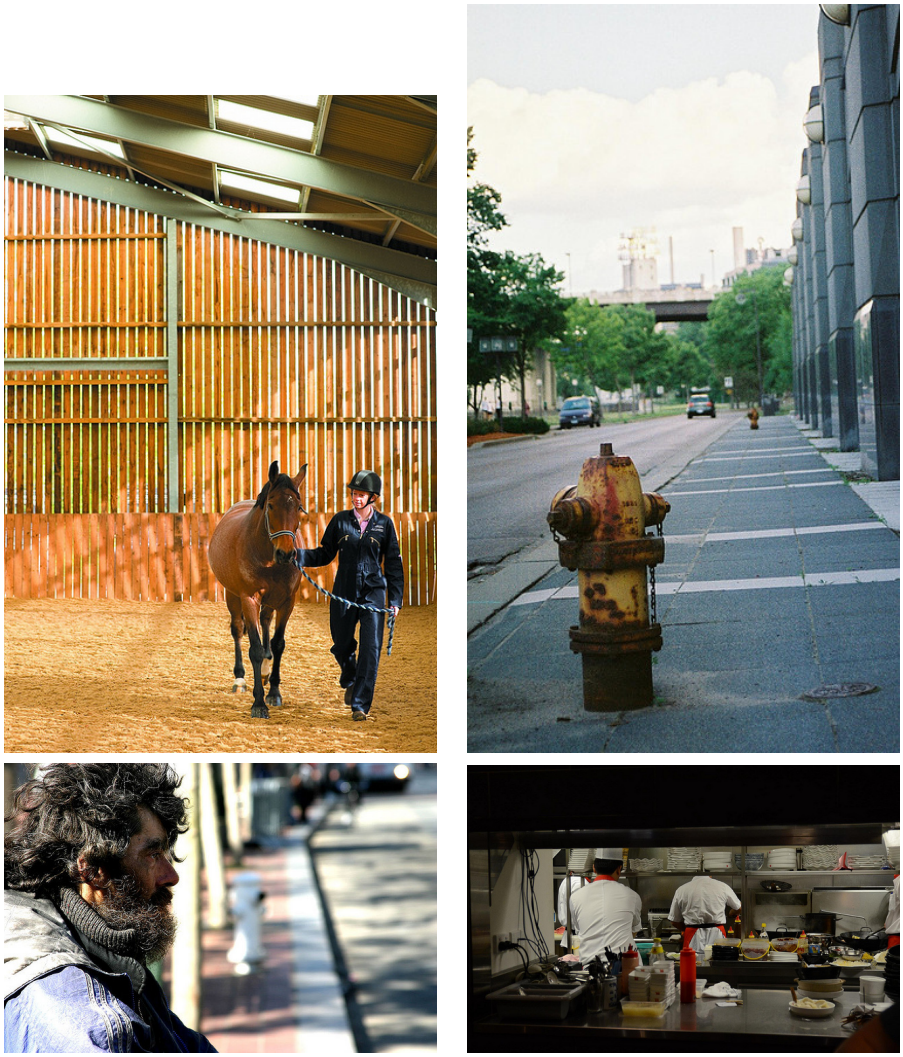


**Figure 4.4:** Sample content images from MS-COCO [7] used in training of LambdaStyler.

These images are paired together before being applied to the network inputs, where training occurs according to the above procedure. Once training is complete, the loss network is discarded. At this point, content images can be stylized by setting the desired style target and applying content images to the reference input.

# Chapter 5

## Results

This chapter provides information on the results obtained from LambdaNet and LambdaStyler. Section 5.1 provides the sequence of results, for both imagery and metrics, which detail the evolution of the network architecture. Section 5.2 details the overall performance of the selected LambdaNet model. Next, Section 5.3 presents the results for multi-class directional change detection, detailing both imagery and metrics. Finally, Section 5.4 presents the results of LambdaNet's conversion to LambdaStyler.

## 5.1 Finalized LambdaNet Architecture

Subsection 5.1.1 details the initial evaluation of all six of the candidate models. Subsection 5.1.2 shows the subsequent selection of the top performing models, along with results from more in-depth testing. Subsection 5.1.3 covers the final refinement steps and the associated results for the selected version of LambdaNet.

### 5.1.1 Stage 1 Testing

The first stage of testing was designed as a coarse evaluation of the individual implementation's performance on the Change Detection 2014 dataset [1], in order to quickly eliminate the lowest performing models. Due to the volume of results and the desire for a quantified measure of performance, the primary method of evaluation is

the F1 Score, presented as Table 5.3. The abbreviations used in the following results tables are given in Tables 5.1 and 5.2.

| Abbreviation | Definition | Description |
|---|---|---|
| FVGG | Full VGG | VGG-based network that contains learned encoders and decoders |
| FMS | Full Multi-Scale | Multi-scale network that contains learned encoders with Res2Net layers and learned decoder with Res2Net layer |
| FRES | Full ResNet | ResNet-based network that contains learned encoders and decoder |
| HVGG | Hybrid VGG | VGG-based network with pre-trained VGG encoders and learned decoder |
| HMS | Hybrid Multi-Scale | Multi-scale network that contains pre-trained VGG encoders and learned decoder with Res2Net layer |
| HRES | Hybrid ResNet | ResNet-based network that contains pre-trained VGG encoders and learned ResNet decoder |

**Table 5.1:** Listing of network abbreviations used in results tables.

| Abbreviation | Definition |
|---|---|
| Config | Configuration |
| Spec | Specificity |
| FPR | False Positive Rate |
| FNR | False Negative Rate |
| PWC | Percent Wrong Classification |
| F1 | F1-Score |
| Prec | Precision |
| mIoU | Mean Intersection Over Union |

**Table 5.2:** Listing of metric abbreviations used in results tables.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| FMS | 0.5984 | 0.9796 | 0.0204 | 0.4016 | 3.4896 | 0.5667 | 0.5382 | 0.3954 |
| FRES | 0.5815 | 0.9740 | 0.0260 | 0.4185 | 4.0954 | 0.5199 | 0.4701 | 0.3513 |
| FVGG | 0.6856 | 0.9650 | 0.0350 | 0.3144 | 4.5609 | 0.5341 | 0.4375 | 0.3644 |
| **HMS** | 0.6657 | 0.9864 | 0.0136 | 0.3343 | 2.5817 | **0.6629** | 0.6601 | 0.4958 |
| **HRES** | 0.7065 | 0.9833 | 0.0167 | 0.2935 | 2.7253 | **0.6641** | 0.6265 | 0.4971 |
| **HVGG** | 0.6323 | 0.9858 | 0.0142 | 0.3677 | 2.7701 | **0.6352** | 0.6380 | 0.4654 |

**Table 5.3:** Metric results for Stage 1 evaluation of LambdaNet. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

As seen in Table 5.3, the uniformly highest performing models were the hybrid variants of LambdaNet (HVGG, HMS, and HRES). These models combine encoders pre-trained on ImageNet, with VGG, residual, and multi-scale decoder types. The hybrid models exhibit improved performance, due to the wider sensitivity of the pre-trained encoders, which are capable of extracting information about a larger variety of objects. On the other hand, the fully end-to-end trained architectures (FVGG, FMS, and FRES) lack this increased sensitivity, as they only have access to the information contained within the dataset. This information is limited to only a few object types, including cars, pedestrians, and a handful of miscellaneous objects, such as boxes and signs.

Qualitatively, the results shown in Table 5.3 show that for the same test image the change segmentation is much more irregular for the end-to-end models than for the hybrid models. This is further reflected in the higher mean IoU scores for the hybrid models. This can be observed in the test images shown as Figure 5.1. Furthermore, it can be observed that unstructured change detection behavior is beginning to emerge in some of the architectures, particularly in the end-to-end VGG and multi-scale, and the hybrid residual network.

**(a)** Test image from end-to-end trained VGG-based network (Figure 3.2a).

**(b)** Test image from hybrid trained VGG-based network (Figure 3.2b).

**(c)** Test image from end-to-end trained residual network (3.2c).

**(d)** Test image from hybrid trained residual network (Figure 3.2d).

**(e)** Test image from end-to-end trained multi-scale network (Figure 3.2e).

**(f)** Test image from hybrid trained multi-scale network (Figure 3.2f).

**Figure 5.1:** Sample test images from evaluated architectures.

### 5.1.2 Stage 2 Testing

The second stage of testing was developed to more intensively evaluate the top performing architectures from Stage 1, these being the hybrid versions of the VGG,

residual, and multi-scale networks. Again, the F1 score was chosen as the primary metric for network evaluation, supplemented with a qualitative examination of selected test images.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| **HMS** | 0.8283 | 0.9712 | 0.0288 | 0.1717 | 3.6421 | **0.7085** | 0.6190 | 0.5486 |
| HRES | 0.7263 | 0.9789 | 0.0211 | 0.2737 | 3.4591 | 0.6917 | 0.6603 | 0.5287 |
| HVGG | 0.6795 | 0.9806 | 0.0194 | 0.3205 | 3.5532 | 0.6714 | 0.6636 | 0.5054 |

**Table 5.4:** Metric results for Stage 2 evaluation of LambdaNet, utilizing reference-target split C. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HMS | 0.4762 | 0.9944 | 0.0056 | 0.5238 | 1.3255 | 0.5140 | 0.5584 | 0.3459 |
| **HRES** | 0.4319 | 0.9967 | 0.0033 | 0.5681 | 1.1627 | **0.5224** | 0.6608 | 0.3535 |
| HVGG | 0.4774 | 0.9917 | 0.0083 | 0.5226 | 1.5827 | 0.4704 | 0.4636 | 0.3075 |

**Table 5.5:** Metric results for Stage 2 evaluation of LambdaNet, utilizing reference-target split G. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HMS | 0.7852 | 0.9803 | 0.0197 | 0.2148 | 4.5590 | 0.8207 | 0.8595 | 0.6959 |
| **HRES** | 0.8336 | 0.9833 | 0.0167 | 0.1664 | 3.6549 | **0.8584** | 0.8846 | 0.7519 |
| HVGG | 0.8175 | 0.9732 | 0.0268 | 0.1825 | 4.7459 | 0.8207 | 0.8239 | 0.6959 |

**Table 5.6:** Metric results for Stage 2 evaluation of LambdaNet, utilizing target-target split B. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HMS | 0.7858 | 0.9796 | 0.0204 | 0.2142 | 3.1486 | 0.7399 | 0.6992 | 0.5872 |
| HRES | 0.7038 | 0.9868 | 0.0132 | 0.2962 | 2.9332 | 0.7323 | 0.7632 | 0.5777 |
| **HVGG** | 0.8281 | 0.9757 | 0.0243 | 0.1719 | 3.2756 | **0.7424** | 0.6728 | 0.5903 |

**Table 5.7:** Metric results for Stage 2 evaluation of LambdaNet, utilizing target-target split D. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| **HMS** | 0.6470 | 0.9757 | 0.0243 | 0.3530 | 3.5610 | **0.5552** | 0.4862 | 0.3842 |
| HRES | 0.7867 | 0.9512 | 0.0488 | 0.2133 | 5.4458 | 0.4981 | 0.3644 | 0.3316 |
| HVGG | 0.7282 | 0.9632 | 0.0368 | 0.2718 | 4.4860 | 0.5272 | 0.4132 | 0.3580 |

**Table 5.8:** Metric results for Stage 2 evaluation of LambdaNet, utilizing category holdout split "Intermittent Object Motion". Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HMS | 0.7466 | 0.9898 | 0.0102 | 0.2534 | 1.9187 | 0.7411 | 0.7356 | 0.5886 |
| **HRES** | 0.7384 | 0.9930 | 0.0070 | 0.2616 | 1.6333 | **0.7688** | 0.8018 | 0.6244 |
| HVGG | 0.7445 | 0.9910 | 0.0090 | 0.2555 | 1.8072 | 0.7518 | 0.7594 | 0.6024 |

**Table 5.9:** Metric results for Stage 2 evaluation of LambdaNet, utilizing category holdout split "Shadow". Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

Examination of the results in Tables 5.4 through 5.9 show a diversity of top performing models. The hybrid multi-scale architecture places first in F1 Score twice, while the residual architecture places first three times, and the VGG architecture places first once. However, closer examination of Table 5.7 shows that the margin between the VGG-based model and its runner up, a multi-scale architecture, is only 0.0025 points difference. Compared to the other top performing models, which in nearly all cases best their runners up by an order of magnitude larger difference, this can be considered a tie. As a result, there are an equal number of first place finishes for the residual and multi-scale architectures.

### 5.1.3 Stage 3 Testing

Stage 3 testing evaluates the merging of residual and multi-scale architectures, according to Figure 3.5, utilizing the same data splits used in stage 2 testing. Metric results are shown in Tables 5.10 through 5.15.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| **HM1RES** | 0.7419 | 0.9795 | 0.0205 | 0.2581 | 3.3212 | **0.7048** | 0.6712 | 0.5441 |
| HM2RES | 0.7462 | 0.9705 | 0.0295 | 0.2538 | 4.1442 | 0.6580 | 0.5885 | 0.4904 |
| HM3RES | 0.7261 | 0.9796 | 0.0204 | 0.2739 | 3.3925 | 0.6958 | 0.6679 | 0.5335 |

**Table 5.10:** Metric results for Stage 3 evaluation of LambdaNet, utilizing random reference-target split C. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HM1RES | 0.4628 | 0.9968 | 0.0032 | 0.5372 | 1.1085 | 0.5514 | 0.6820 | 0.3807 |
| HM2RES | 0.4583 | 0.9963 | 0.0037 | 0.5417 | 1.1648 | 0.5367 | 0.6475 | 0.3668 |
| **HM3RES** | 0.5377 | 0.9970 | 0.0030 | 0.4623 | 0.9785 | **0.6180** | 0.7265 | 0.4472 |

**Table 5.11:** Metric results for Stage 3 evaluation of LambdaNet, utilizing random reference-target split G. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HM1RES | 0.8354 | 0.9814 | 0.0186 | 0.1646 | 3.7959 | 0.8540 | 0.8734 | 0.7452 |
| HM2RES | 0.8286 | 0.9826 | 0.0174 | 0.1714 | 3.7829 | 0.8534 | 0.8797 | 0.7443 |
| **HM3RES** | 0.8243 | 0.9864 | 0.0136 | 0.1757 | 3.5126 | **0.8618** | 0.9029 | 0.7572 |

**Table 5.12:** Metric results for Stage 3 evaluation of LambdaNet, utilizing random target-target split B. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HM1RES | 0.7551 | 0.9865 | 0.0135 | 0.2449 | 2.6711 | 0.7632 | 0.7715 | 0.6171 |
| HM2RES | 0.7192 | 0.9865 | 0.0135 | 0.2808 | 2.8737 | 0.7405 | 0.7631 | 0.5879 |
| **HM3RES** | 0.7979 | 0.9825 | 0.0175 | 0.2021 | 2.7978 | **0.7648** | 0.7343 | 0.6191 |

**Table 5.13:** Metric results for Stage 3 evaluation of LambdaNet, utilizing random target-target split D. Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HM1RES | 0.7831 | 0.9615 | 0.0385 | 0.2169 | 4.4671 | 0.5463 | 0.4195 | 0.3758 |
| **HM2RES** | 0.8024 | 0.9665 | 0.0335 | 0.1976 | 3.9133 | **0.5848** | 0.4601 | 0.4133 |
| HM3RES | 0.7939 | 0.9627 | 0.0373 | 0.2061 | 4.3084 | 0.5587 | 0.4310 | 0.3876 |

**Table 5.14:** Metric results for Stage 3 evaluation of LambdaNet, utilizing category holdout split "Intermittent Object Motion". Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

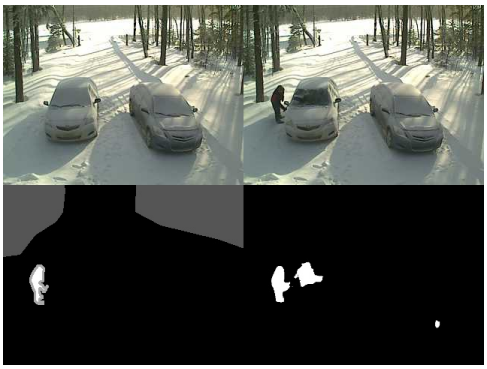
| Config | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| HM1RES | 0.7819 | 0.9886 | 0.0114 | 0.2181 | 1.8991 | 0.7517 | 0.7238 | 0.6022 |
| HM2RES | 0.7877 | 0.9882 | 0.0118 | 0.2123 | 1.9140 | 0.7517 | 0.7188 | 0.6021 |
| **HM3RES** | 0.7938 | 0.9920 | 0.0080 | 0.2062 | 1.5271 | **0.7927** | 0.7915 | 0.6565 |

**Table 5.15:** Metric results for Stage 3 evaluation of LambdaNet, utilizing category holdout split "Shadow". Top performing models by F1 Score are in boldface. Abbreviations can be found in Tables 5.1 and 5.2.

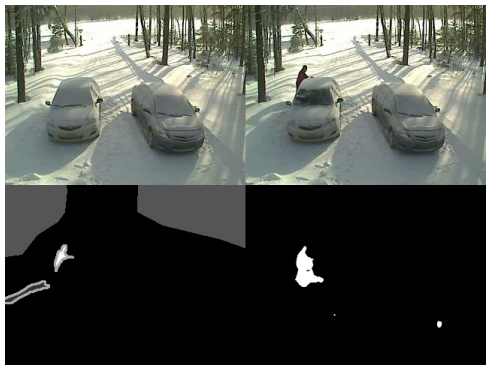
As can be seen in Tables 5.10 to 5.15, the highest performing architecture is the hybrid residual network with three multiscale layers (HM3RES), shown in Figure 3.5c. This particular architecture performed best in terms of F1 Score on four of the six chosen data splits. Furthermore, the Mean Intersection over Union (mIoU) metric is also positively correlated with the F1 Score. This indicates that not only is the change detection accuracy of the network improved, but the change segmentation is also more uniform and less prone to having voids, as seen in Figure 5.1.

## 5.2 Performance of Selected LambdaNet Model

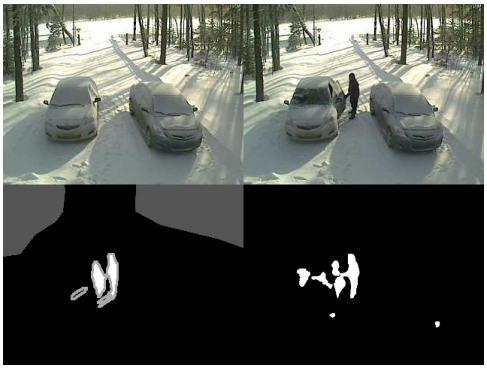
Qualitatively, there is also a wide array of significant behavior in both the structured and unstructured change domains. One of the most important behavioral observations is the concept of negative segmentation. Negative segmentation occurs when the network segments a region of an image, due to the absence of a particular object. Figure 5.2 shows a sequence of video frames from the reference-target split C of a man cleaning snow from his vehicle, including an example of negative segmentation. Each frame's caption contains a brief analysis of the predicted image.



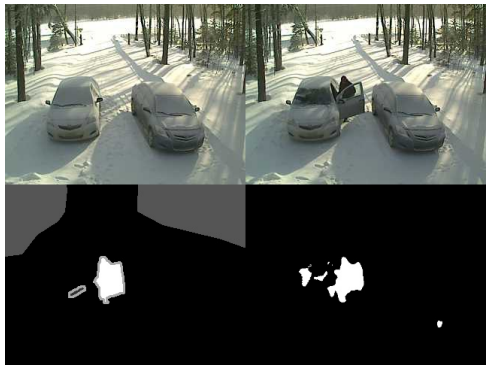
**(a)** This frame shows a man cleaning snow off his car. The person is a structured change, while the cleared windshield is an unstructured change.



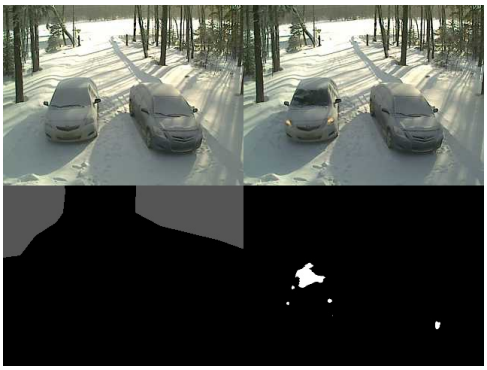
**(b)** Here, the man has moved behind his vehicle, but is still partially visible.



**(c)** In this case, the man is now opening his car door. As this change is labelled in the ground truth, it is considered structured.



**(d)** The man has now opened the door of his car. This change is considered a structured change.



**(e)** These change detections are purely unstructured. The car's windshield is clear and has been marked as a change. In addition, the two points below the windshield are marked from the car's activated headlights.

**(f)** This frame contains both structured and unstructured changes. The primary structured change is the detection from the car pulling away down the driveway. The vehicle's prior location is also segmented.

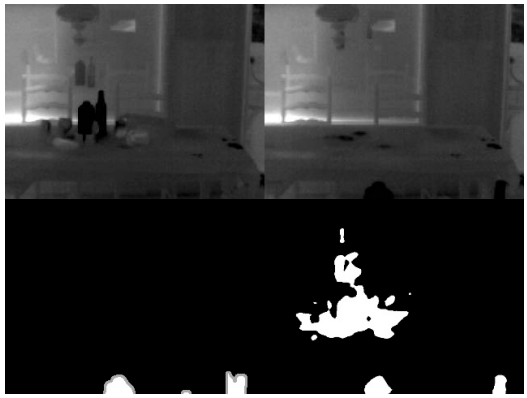
**Figure 5.2:** Sequence of video frames showing a man cleaning snow off his car, then driving away. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

This behavior is present in other videos, as well. Figure 5.3 shows a child clearing bottles off a table. This particular video was filmed using a thermal camera. Furthermore, this video was taken from a differently trained version of LambdaNet, where all thermal videos were withheld from the training set. Despite this, the network is not only still able to indicate structured changes in the image pairs, but is also able to mark unstructured changes.

**(a)** The child begins clearing the table. She is considered a structured change, as she is marked in the ground truth image.

**(b)** The child has begun to clear off the table. The network is beginning to register small unstructured changes as a result.

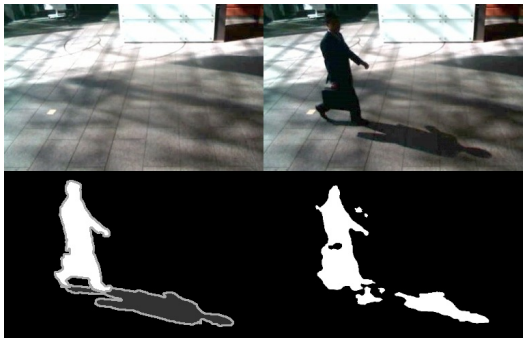
**(c)** The child has moved most of the bottles off the table, causing the unstructured change marks to grow in size. A number of bottles appear along the bottom row of the prediction as structured changes.



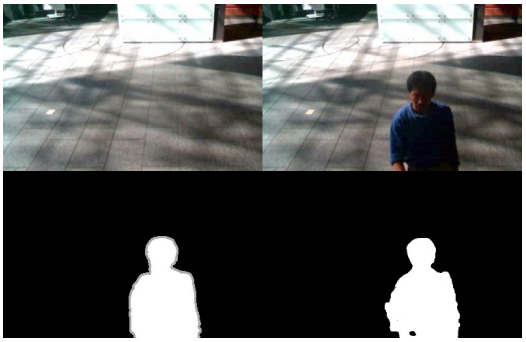
**(d)** The child has completed clearing the table. The new locations of the bottles are shown as structured changes along the bottom of the predicted image. The large change mark is a negative segmentation, indicating the removal of the bottles, which is an unstructured change.

**Figure 5.3:** Sequence of video frames showing a child clearing bottles from a table. Video is shot with a thermal camera. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

This unstructured behavior is a result of the Siamese-type architecture of the network, combined with a common decoder. Standard semantic segmentation networks usually operate only on a single image at a time and learn a function mapping regions of the input image into specific class domains. LambdaNet operates in a different manner, eschewing class-based ground truth in favor of a simpler binary labelling scheme of change or no change, reducing its dependence on semantic data. Since the change marks indicate differences between the image pairs, the common decoder in LambdaNet instead learns a separation function that ignores common elements between images and only highlights their differences. Further examples of this behavior are shown in Figures 5.4 to 5.7.

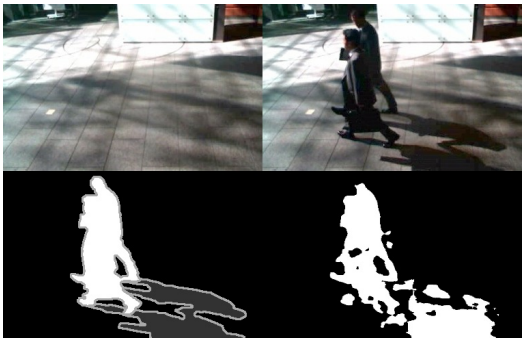


**(a)** A man walking with a briefcase. He is a structured change, while his shadow is unstructured.



**(b)** Example of LambdaNet marking a purely structured change.

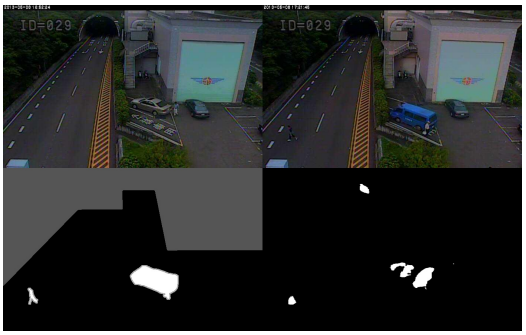
**(c)** Example of LambdaNet marking a purely structured change.



**(d)** Two men walking side-by-side. The change from the men is considered structured, while shadows are unstructured.

**Figure 5.4:** A video of people walking past a storefront. LambdaNet is not trained on shadows. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

**(a)** Note that even from an elevated angle, the "Don't Care" region still obscures many possible targets.

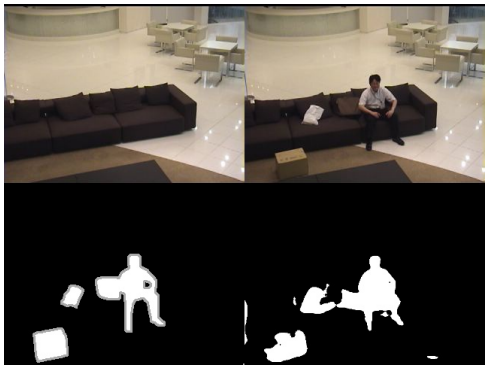


**(b)** This issue extends to objects inside the region of interest, such as the blue van.

**Figure 5.5:** Video of parked and moving cars taken at a low frame rate. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

**(a)** A man carries a bag to the sofa. All changes are structured.

**(b)** The man sits on the sofa. All changes are structured.



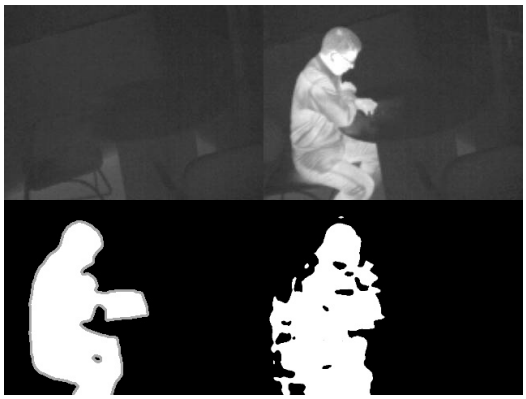
**(c)** Another man sits on the sofa next to a briefcase. All changes are structured.

**(d)** A man walks past the sofa with several objects on it. All changes are structured.

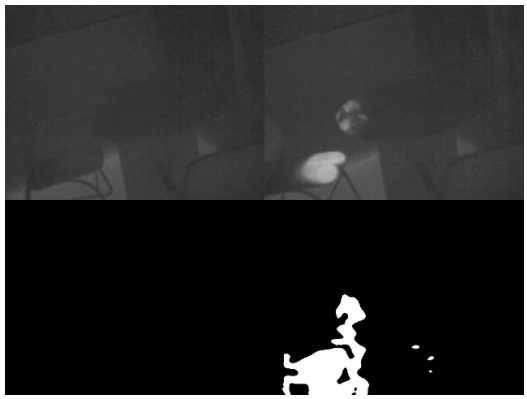
**Figure 5.6:** Video of people moving objects onto and off of a sofa. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

**(a)** An empty table and chair. No changes are present.

**(b)** A man sits down with a laptop. Both are marked as a structured change.



**(c)** The man shifts position. Both him and the laptop are marked as structured changes.



**(d)** The man has left the scene with the laptop. Residual thermal traces are marked as unstructured changes.

**Figure 5.7:** Video of a man working on a laptop at a table. Video shot on thermal camera. This trained architecture of LambdaNet was not trained on thermal videos. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

However, because the significant differences between images are indicated by a human labelling the training dataset, this also grants the network the ability to distinguish which changes are not important, such as background noise of leaves blowing on trees. An example of this type of insignificant change rejection is shown as Figure 5.8.

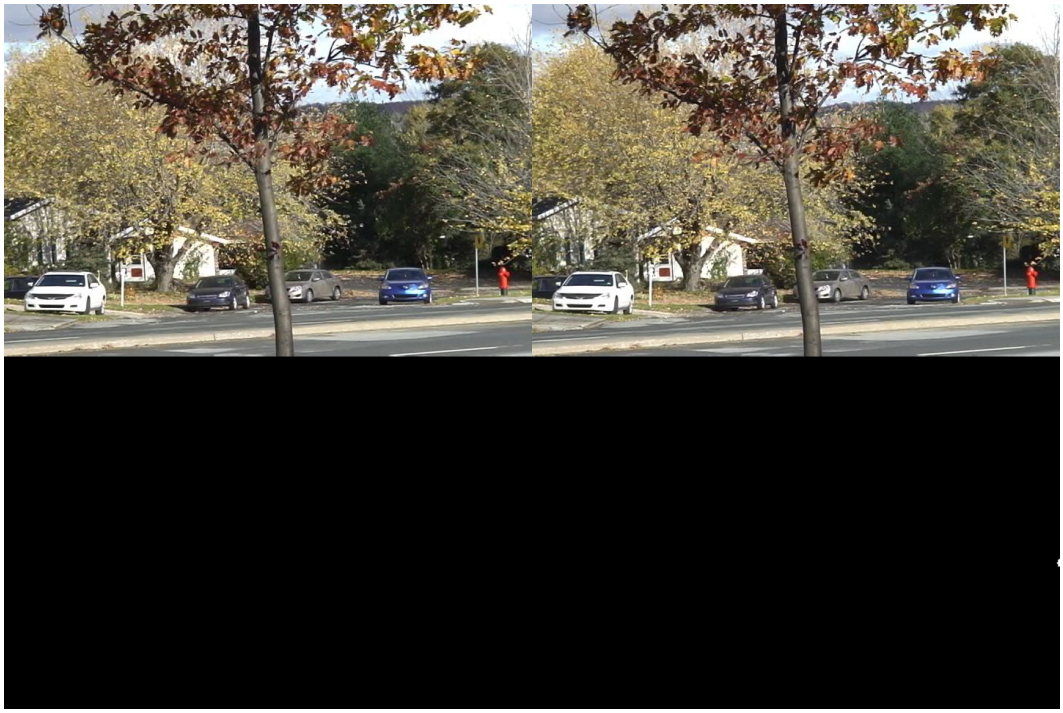
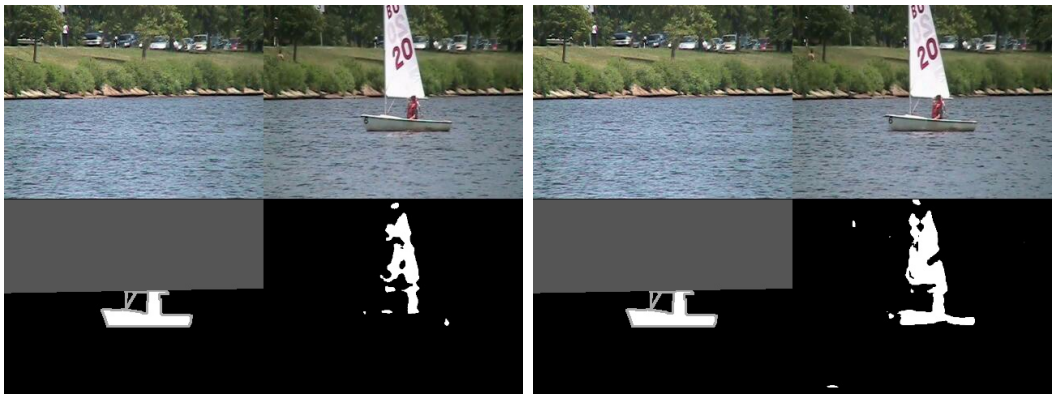


**Figure 5.8:** Example image with dynamic background of leaves blowing in the wind. Note that these changes are not marked as significant.

The quality of segmentation depends on how the network is trained. Figure 5.9 shows the same image from two differently trained versions of the network. Figure 5.9a was evaluated on a version of LambdaNet that had no boats in the training split, while Figure 5.9b was evaluated on a version of the network that had some example boats. However, neither network had seen this video before. In the case of Figure 5.9b, the segmentation quality is visibly improved, showing enhanced definition in the sail, along with the inclusion of the hull in the change segmentation. This is also reflected in the metrics for the two networks, indicating that mixing in a small amount of information on a specific desired target object with a larger body of unrelated change pairs can yield a significant improvement in the network's ability to recognize changes.



**(a)** Change pair images of a boat, evaluated with a LambdaNet not trained on images with a dynamic background. Most dynamic background images take place on water, and therefore feature boats.

**(b)** Change pair images of a boat, evaluated with a LambdaNet trained on images with a dynamic background. Since some images of boats are now included in the training set, segmentation quality is improved, specifically in regards to the boat's hull.

**Figure 5.9:** Identical change pairs evaluated on LambdaNets with different training splits.

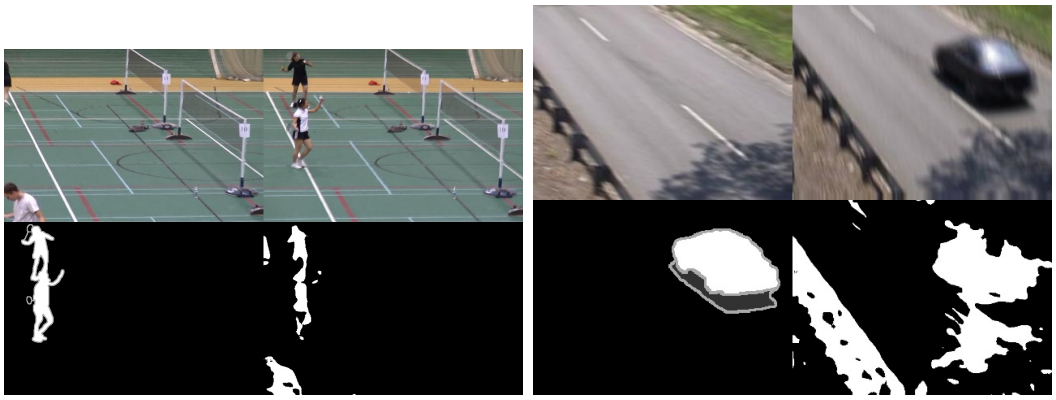
Finally, Table 5.16 lists the selected LambdaNet architecture metrics against competing change detection methodologies. This table was generated by evaluating the chosen LambdaNet architecture on all eleven category splits of the Change Detection 2014 dataset, via eleven-fold cross validation. The average of each metric was then computed and used to generate Table 5.16.

| Method | Recall | Spec | FPR | FNR | PWC | F1 | Prec |
|---|---|---|---|---|---|---|---|
| IUTIS-5 [18] | 0.7849 | 0.9948 | 0.0052 | 0.2151 | 1.1986 | 0.7717 | 0.8087 |
| SuBSENSE [16] | 0.8124 | 0.9904 | 0.0096 | 0.1876 | 0.6780 | 0.7408 | 0.7509 |
| CwisarDH [17] | 0.6608 | 0.9948 | 0.0052 | 0.3392 | 1.5273 | 0.6812 | 0.7725 |
| LambdaNet-HM3RES | 0.6088 | 0.9810 | 0.0190 | 0.3912 | 2.7590 | 0.5520 | 0.5578 |

**Table 5.16:** Overall metric results comparing LambdaNet to other change detection techniques. Abbreviations can be found in Tables 5.1 and 5.2.

LambdaNet does not achieve state of the art performance. This is likely due to a confluence of several factors. First is that the decreased performance is specific to only a few categories; however, those drops are significant. Table 5.17 shows the F1 Scores of all methods for each category, organized as a matrix. Due to the dataset

chosen, one category bears special consideration, "Camera Jitter". This category comprises videos where the camera is shaking. In one case, however, the shaking is so severe that it leads to the deregistration of the image. While LambdaNet does have some capacity to handle this issue, in the most extreme cases, it is unable to recover. This comparision is shown as Figure 5.10.



**(a)** Video of badminton players. Minor camera shaking is present between reference and target frames.

**(b)** Video of cars driving down the road. Note that the shaking is so severe that deregistration of the image occurs.

**Figure 5.10:** Comparison of LambdaNet's resiliency to a shaking camera.

| | IUTIS-5 | SuBSENSE | CwisarDH | LambdaNet |
|---|---|---|---|---|
| BW | 0.8248 | 0.8619 | 0.6837 | 0.7429 |
| BL | 0.9567 | 0.9503 | 0.9145 | 0.8284 |
| CJ | 0.8332 | 0.8152 | 0.7886 | **0.5443** |
| DB | 0.8902 | 0.8177 | 0.8274 | **0.4539** |
| IOM | 0.7296 | 0.8986 | 0.5753 | 0.5586 |
| LF | 0.7743 | 0.6445 | 0.6406 | **0.4459** |
| NV | 0.529 | 0.5599 | 0.3735 | 0.4850 |
| PTZ | 0.4282 | 0.3476 | 0.3218 | **0.1226** |
| SH | 0.9084 | 0.8986 | 0.8581 | 0.7926 |
| TH | 0.8303 | 0.8171 | 0.7866 | 0.7153 |
| TR | 0.7836 | 0.7792 | 0.7227 | **0.3823** |

**Table 5.17:** Table comparing F1 Scores for all methods for each video category. Significantly lower F1-Scores for LambdaNet are marked in boldface. (Legend: BW = BadWeather, BL = BaseLine, CJ = CameraJitter, DB = DynamicBackground, IOM = IntermittentObjectMotion, LF = LowFramerate, NV = NightVideos, PTZ = PanTiltZoom, SH = Shadows, TH = Thermal, TR = Turbulence)

Second, the Change Detection 2014 dataset is geared primarily towards structured change detection. As a result, the algorithms in Table 5.16 are all structured methods. LambdaNet operates in a mixed-mode between structured and unstructured methodologies. Due to this behavior, any structured change that LambdaNet detects will be scored correctly, but any unstructured change detection will cause the network to be penalized in the metric scores.

Third, there is a shortcoming in the labelling of the Change Detection 2014 dataset. Almost a quarter of the videos have associated ground truth that is marked with large "Don't Care" regions. In many of these regions, there are change targets, such as cars, that are present but not marked as a change. In other cases, a change target may be partially "obscured" by a "Don't Care" region, resulting in a change target that is only partially labelled. This is particularly problematic, as LambdaNet still learns classes intrinsically, even though all objects are only labelled as binary changes. This means that at least some of LambdaNet's learned object definitions may be of incomplete objects. If all change targets inside the "Don't Care" regions could be labelled, it may decrease the number of voids seen in the change maps. This is particularly true when comparing images of people and cars. People are often segmented with fewer voids than cars are, as seen in Figures 5.2 and 5.4. This causes a significant drop in accuracy for categories like "Dynamic Background," as they feature cars prominently. Examples of this type of labelling are shown in Figure 5.11, while an example failure from the "Dynamic Background" category is shown as Figure 5.12.

**(a)** Example of a car being marked as "No Change" due to its location in a gray "Don't Care" region.

**(b)** Example of a car fully in-frame, labelled as both "Change" and "No Change", due to the presence of a "Don't Care" region.

**Figure 5.11:** Examples of labelling issues caused by the presence of large "Don't Care" regions in the Change Detection 2014 ground truth.

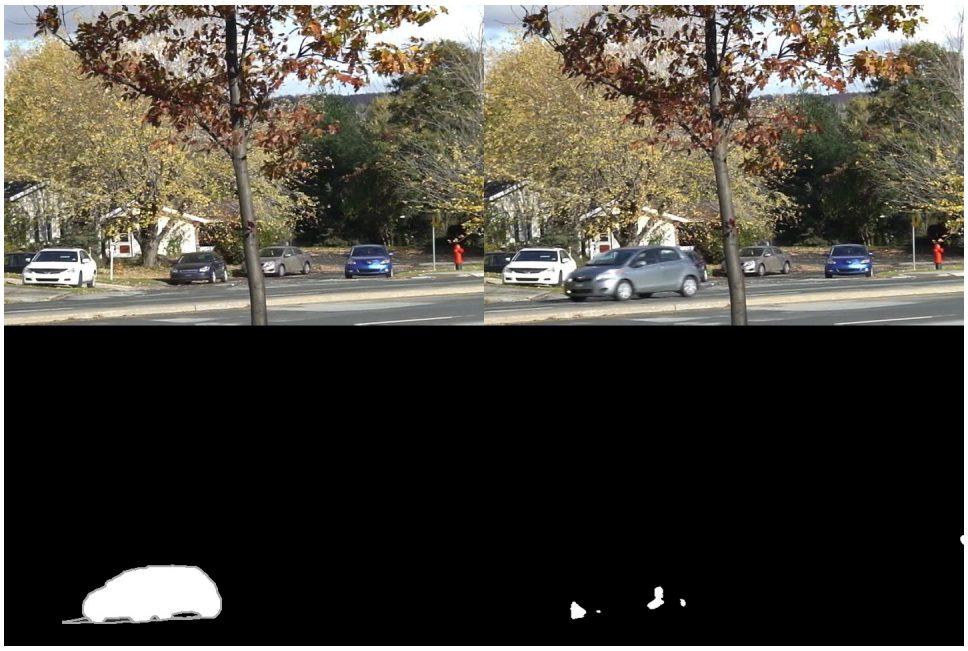


**Figure 5.12:** Change detection failure for cars in the "Dynamic Background" category.

Fourth, LambdaNet itself is sensitive to changes in scene illumination. This is likely related to the network's ability to detect unstructured changes, which appears to operate at some level on color differences. This can cause significant detections to occur solely due to lighting, as shown in Figure 5.13, where a change detection is marked due to a darker location in the target image. This occurs frequently in the "Night Videos" category, as many of these videos feature extreme glare from car headlights or streetlights and would account for the poor performance in this category.

**Figure 5.13:** Video frame from a highway exit ramp at night. Erroneous change detection was caused by extreme glare from the headlights of oncoming traffic.

Lastly, LambdaNet struggles to identify changes between objects that are extremely small and/or occluded, as shown in Figure 5.14. This is particularly concerning, as these are the situations that a human would also find the most difficult. This is likely related to issues with ground truth labelling. In particular, the "Low Framerate" and "Turbulence" video categories consist entirely of extremely small automotive change targets.

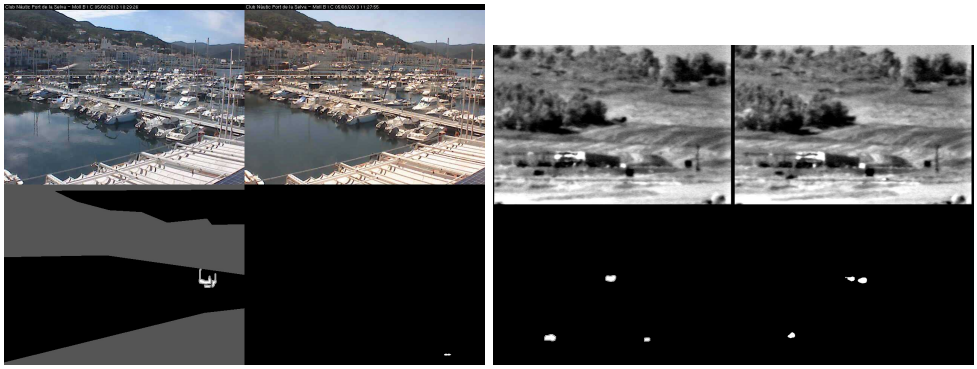


**Figure 5.14:** Samples where LambdaNet misses a change, due to small object size and partial occlusion.

However, there are two techniques which may be able to boost accuracy performance. First would be to generate custom reference background images in a preprocessing step. This would ensure that these images would be free from any target objects that could cause false detections. The second method involves the incorporation of an auxiliary network in parallel with LambdaNet. This network would be responsible for generating semantic segmentation maps of objects of interest, while LambdaNet focuses on identifying changes between the image pairs. The outputs of these networks could then be correlated, such that structured changes could be marked via a high resolution segmentation map. This would effectively suppress the unstructured portion of LambdaNet's behavior, allowing for a fairer comparison with the structured methods. Both of these operations could be performed with the work of Savakis, et al. [27].

More extreme changes to the dataset may also be helpful. A significant number of video frames in the Change Detection 2014 [1] dataset are unlabelled. One option may be to utilize a semantic segmentation network to label these frames automatically. This data could then be used as a training set for LambdaNet, which would fully eliminate the issues caused by the restrictive "Don't Care" regions during training. The already labeled images from the dataset would then be relegated to a validation set, maintaining the ability to compare results with existing change detection methods.

## 5.3 Directional Change Performance

This stage of testing focused on the selected architecture for LambdaNet, shown in Figure 3.5c, when extended to the directional change domain. Quantitative analysis is primarily driven by Table 5.18, which compares directional change detection performance with the overall performance of the binary change detection version of LambdaNet, trained on random target-target splits.

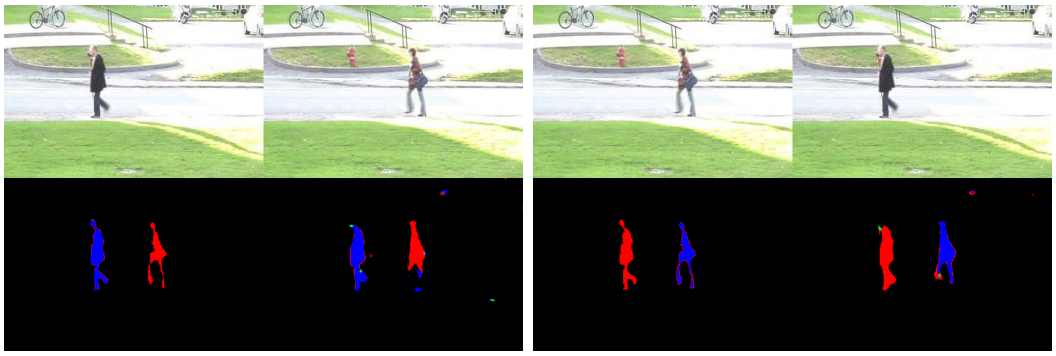
| Change Class | Recall | Spec | FPR | FNR | PWC | F1 | Prec | mIoU |
|---|---|---|---|---|---|---|---|---|
| Binary (BC) | 0.7292 | 0.9752 | 0.0248 | 0.2708 | 3.6564 | 0.6682 | 0.6395 | 0.5137 |
| Additive (AC) | 0.5940 | 0.9916 | 0.0084 | 0.4060 | 1.7904 | 0.6150 | 0.6447 | 0.4504 |
| Subtractive (SC) | 0.6329 | 0.9910 | 0.0090 | 0.3671 | 1.7296 | 0.6288 | 0.6399 | 0.4660 |
| Exchange (EC) | 0.4217 | 0.9888 | 0.0112 | 0.5783 | 0.3831 | 0.2452 | 0.1955 | 0.1556 |
| Average | 0.5495 | 0.9905 | 0.0095 | 0.4505 | 1.6343 | 0.4963 | 0.4934 | 0.3573 |

**Table 5.18:** Overall metric results comparing the directional change LambdaNet to binary change LambdaNet. Binary line indicates the baseline binary change detection version of LambdaNet, trained on the same random target-target dataset splits. Average line indicates the average metrics for additive change, subtractive, change, and exchange. Abbreviations can be found in Tables 5.1 and 5.2.

Table 5.18 was generated by first treating each directional change class as a one versus all binary classification problem. This means the Additive Change (AC) metrics were computed as the positive binary class versus the negative classifications of No Change (NC), Subtractive Change (SC), and Exchange (EC). This process was repeated for the subtractive and exchange classes. This method was chosen, as it allows for the direct comparison of each directional class with the baseline of the binary change version of LambdaNet.

Inspection of Table 5.18 shows that the AC and SC metrics compare favorably with the binary LambdaNet's version of change detection. This is especially true for the specificity, false positive rate, percent wrong classification, F1 score and precision. Somewhat concerning is the lowered recall and mean intersection over union (mIoU) and elevated false negative rate. This is indicative that there may be a larger number

of voids in the directional change segmentations than in the binary change segmentations. Furthermore, the similarity of the AC and SC metric scores indicate that LambdaNet has learned both of these directional classes equally well. Unfortunately, the EC class performs poorly across the board, indicating that the directional change scheme has severe difficulties identifying object swaps. Overall, though, the primary source of accuracy degradation is the EC class, due to unstructured changes. This is borne out qualitatively in the results images, shown in Figures 5.15 through 5.22. Figure 5.15 shows pedestrians on a sidewalk.

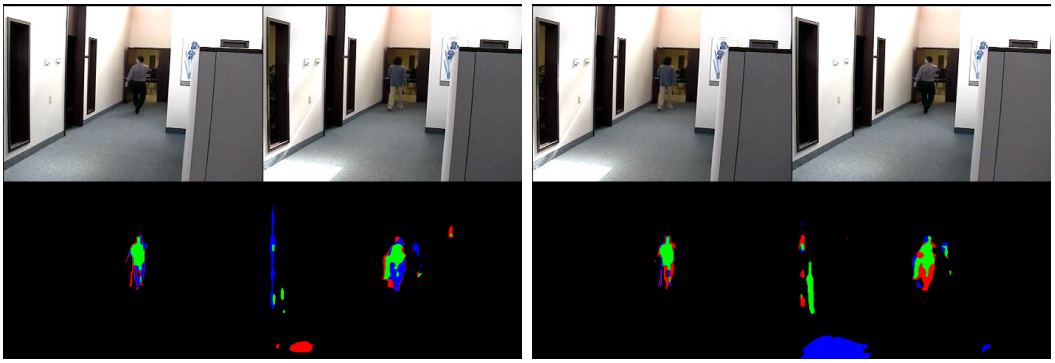


**(a)** Image pair showing a man in a suit marked as a blue subtractive change and a woman as a red additive change.

**(b)** Reversal of input images in Figure 5.15a. Note that the segmentations are similar, but the directionality is reversed.

**Figure 5.15:** Video of pedestrians walking and biking on pavement. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

Both the frames chosen in Subfigures 5.15a and 5.15b are the same; however, they are reversed between the two results images. This shows that the directional change scheme can correctly identify additions and removals from a pair of images. Furthermore, the segmentations of the man and woman are similar to their reversed counterparts. This provides further indication that LambdaNet is not biased in performance towards additive or subtractive change.



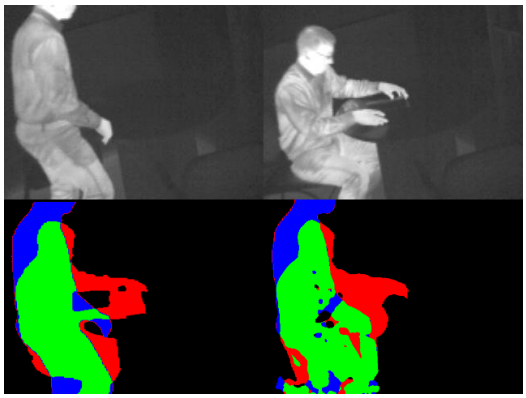
**(a)** The co-location of the two men is marked as a green exchange.

**(b)** Reversal of input images in Figure 5.16a. Note that the exchange marking is largely unchanged.

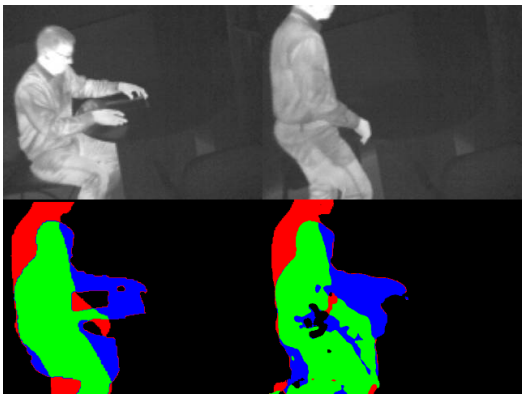
**Figure 5.16:** Video of two men walking down a hallway. Each one occupies the same spot at a different time. (Legend: Upper Left = Past Target Image, Upper Right = Current Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

Figure 5.16 shows an example of the Exchange class identifying an object swap. In the case of this exchange, the two change targets are located nearly on top of each other. As a result, this interaction is marked almost exclusively as an exchange of targets. Even when the target frames are reversed, this exchange region remains constant, which is consistent with the intuitive understanding of an object swap.

For the final case, Figure 5.17 shows an object exchange, where the two objects are offset from each other. In this case, the standing man's head is considered a subtractive change, while his extended arms in the sitting position are an additive change. However, where these two entities overlap is an exchange. When these target images are reversed, the additive and subtractive change markings reverse, which is consistent with Figure 5.15. Furthermore, the exchange region stays relatively constant, which is also consistent with Figure 5.16.



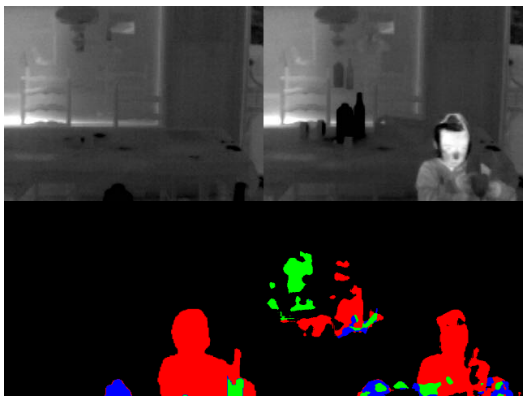
**(a)** A man sits down in a chair.



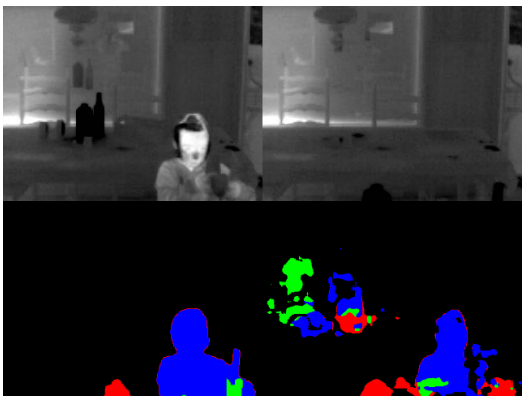
**(b)** Reversal of input images in Figure 5.17a.

**Figure 5.17:** Thermal video of a man sitting with a laptop in a library. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

Prior figures focused exclusively on tracking structured changes with the directional change methodology. The following image sets target unstructured directional changes in images analyzed with LambdaNet. Figure 5.18 revisits the video of a child clearing a table under the directional change paradigm.



**(a)** Child marked as structured additive change. Bottles are marked as an unstructured change, which is both additive and an exchange.

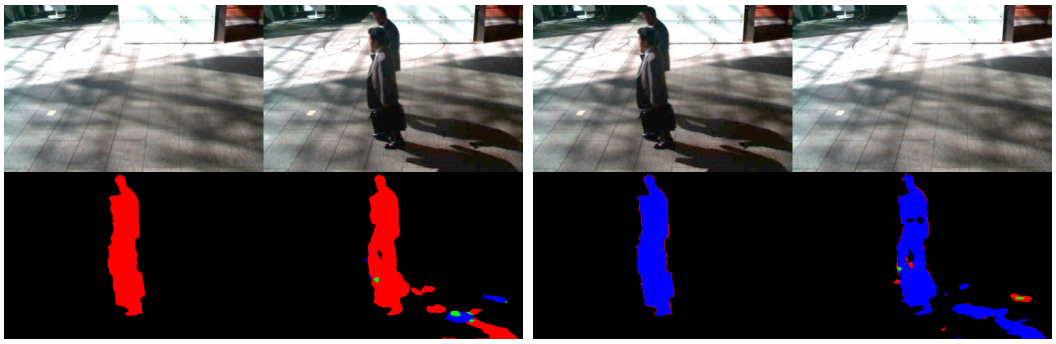


**(b)** Reversal of input images in Figure 5.18a.

**Figure 5.18:** Video of a child removing bottles from a table. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

Figure 5.18 illustrates the unstructured change detection behavior first exposed in

the binary classification form of LambdaNet is still present in the directional change version of the network. However, it also appears that while this desired behavior is still present, it is not strongly associated with a particular change class, as evidenced by the mix of classes used to mark the moved bottles. This can also be seen in Figure 5.19.



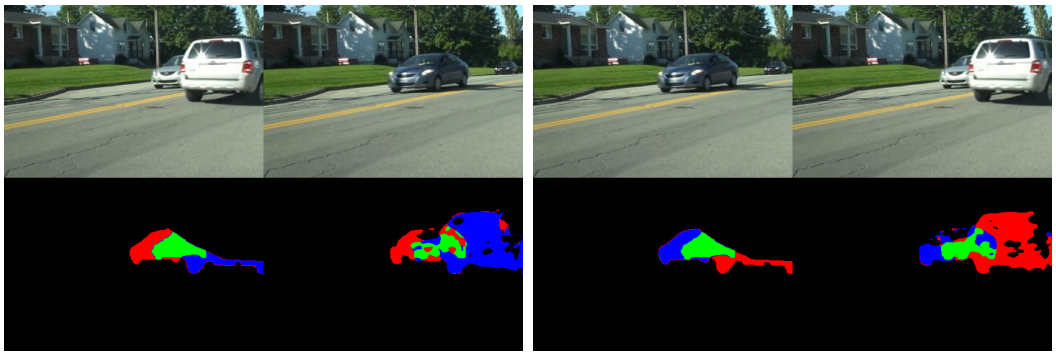
**(a)** Men marked as structured additive change, while their shadows are unstructured. Note all three classes are present in the unstructured change.

**(b)** Reversal of input images in Figure 5.19a.

**Figure 5.19:** Video of men walking down the sidewalk. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

The directional change version of LambdaNet also shares the same drawbacks as the binary change version. Its unstructured behavior is still penalized in the same manner as in the binary case, while the labelling issues with the "Don't Care" regions in the ground truth still lead to incomplete learned representations. This is particularly visible in the ground truth image in Figure 5.20 and yields a lower quality segmentation for the structured change of the vehicles.

Furthermore, the illumination sensitivity and small object issues are still present for the directional change version of LambdaNet. This can be seen in Figure 5.21.



**(a)** Cars passing each other at different times on the same street.

**(b)** Reversal of input images in Figure 5.20a.

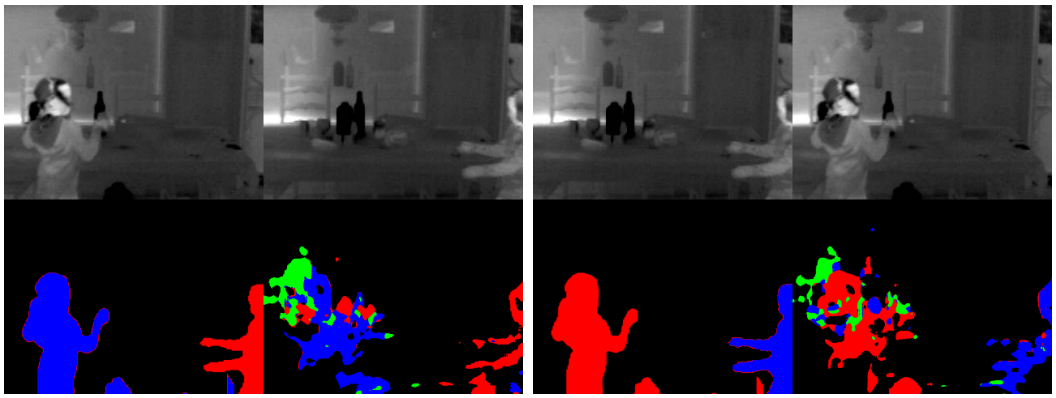
**Figure 5.20:** Cars driving down a two way street at different times. (Legend: Upper Left = Past Target Image, Upper Right = Current Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

**(a)** All cars are missed as change detections.

**(b)** Reversal of input images in Figure 5.21a. Note lighting creates a false additive change detection.

**Figure 5.21:** Cars driving off a highway exit at night. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

There is one additional issue with the directional change version of LambdaNet that is not present in the binary version, which involves the unstructured detections. In some cases, a structured change will occur in the same spatial region as an unstructured change. This is especially apparent in Figure 5.22.



**(a)** Inability of the network to process a structured change (child) and unstructured change (bottles) in the same location.

**(b)** Reversal of input images in Figure 5.22a.

**Figure 5.22:** Directional change is unable to track co-located structured and unstructured changes. (Legend: Upper Left = Reference Image, Upper Right = Target Image, Lower Left = Ground Truth, Lower Right = Change Prediction.)

In this situation, the network is unable to distinguish which pairs of changes belong with which classes. This is likely due to the complexity of the change taking place. LambdaNet is attempting to process this region as a single change, when in reality it is two different changes taking place concurrently. These two changes would be the girl's presence in front of the bottles and the absence of some of the bottles. This results in an extremely incoherent segmentation, as LambdaNet is trying to operate in both the structured and unstructured domains simultaneously. From a standpoint of novel behavior, this is desirable. However, due to its inconsistent operation, it is also detrimental to the collected performance metrics for structured change detection.

## 5.4 Performance of LambdaStyler

Due to the subjective nature of artistic style transfer, this section focuses on a qualitative analysis of LambdaStyler's output. Figure 5.23 shows a set of sample output triplets for both LambdaStyler and the Fast Residual technique [6]. These results were generated using the same style and content weights used for the Fast Residual case, which were $1e10$ and $1e5$, respectively. Figure 5.23 shows that the LambdaStyler

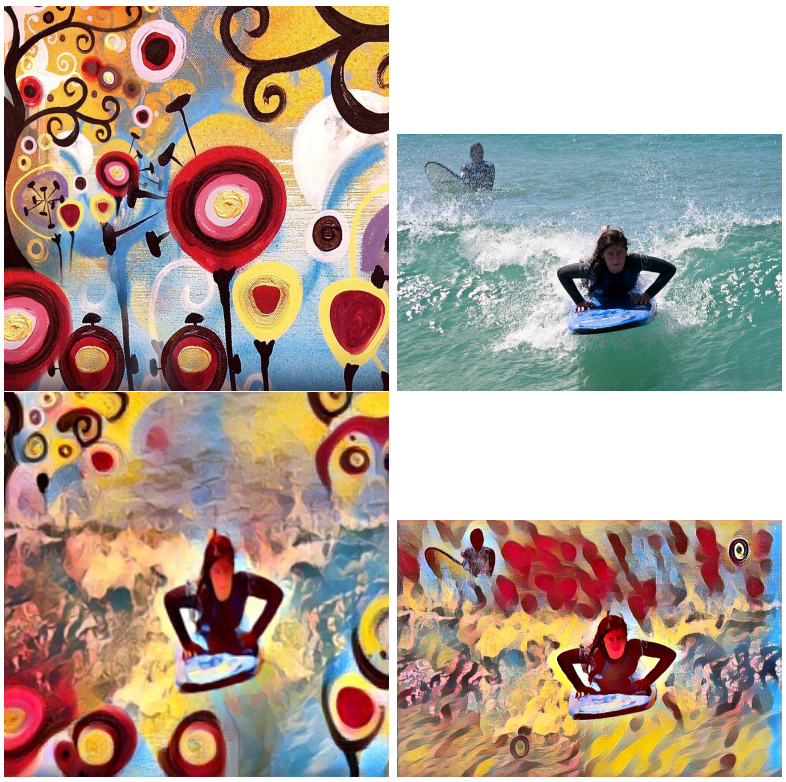
network effectively transfers both high and low frequency information, in addition to color, from the target style into the content image. In the case of the Udnie painting, which is largely composed of flat regions and pastel colors, the content image has had its colors muted, with most of the high frequency content of the content image also being removed. This manifests as a loss of detail in regions such as the blue blanket and the dog's fur. These regions have been rendered into pastel "panels", similar in structure to the large, flat regions in Udnie. This behavior is very similar to that found in the Candy and Rain Princess paintings.

The results of Mosaic are particularly interesting, due to the style elements that were transferred. Mosaic is painted in a style evocative of stained glass, with a focus on heavy, dark outlines around regions of bright, flat color. When used as a style image, the expected color and frequency information is applied to the content image. However, edge information was also utilized, in the form of thick black outlines that were applied to the objects in the content image. This was done in spite of the fact that there is no explicit edge information processed during training.

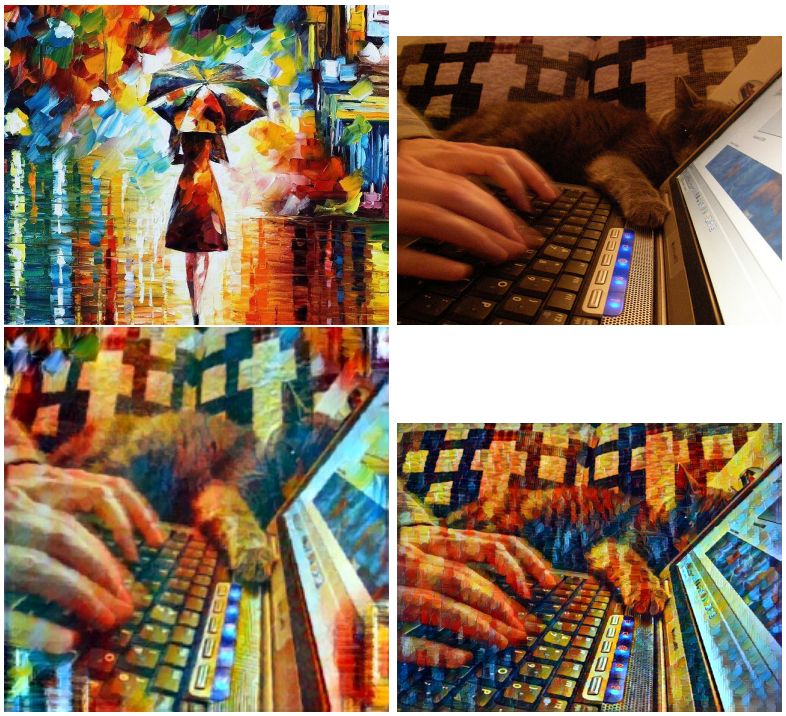
In general, the output of LambdaStyler compares favorably with that of the fast residual autoencoder technique of Johnson, et al. [6]. Figure 5.23 shows sample output from the fast residual technique, with the same content and style images utilized for the LambdaStyler evaluation. The transferred style for each network is qualitatively very similar, especially with respect to Udnie (Figure 5.23a) and Rain Princess (Figure 5.23c). However, there is one important difference in how these results were generated. In the case of the fast residual autoencoder, generating each of these four stylized images requires four independently trained stylization networks, as one network can only learn one style. With LambdaStyler, only one network needed to be trained to generate imagery using all four styles.

LambdaStyler also exhibits an additional unique behavior. In each of the stylized images shown in Figure 5.23, each one has a fixed style artifact present. Figure 5.24

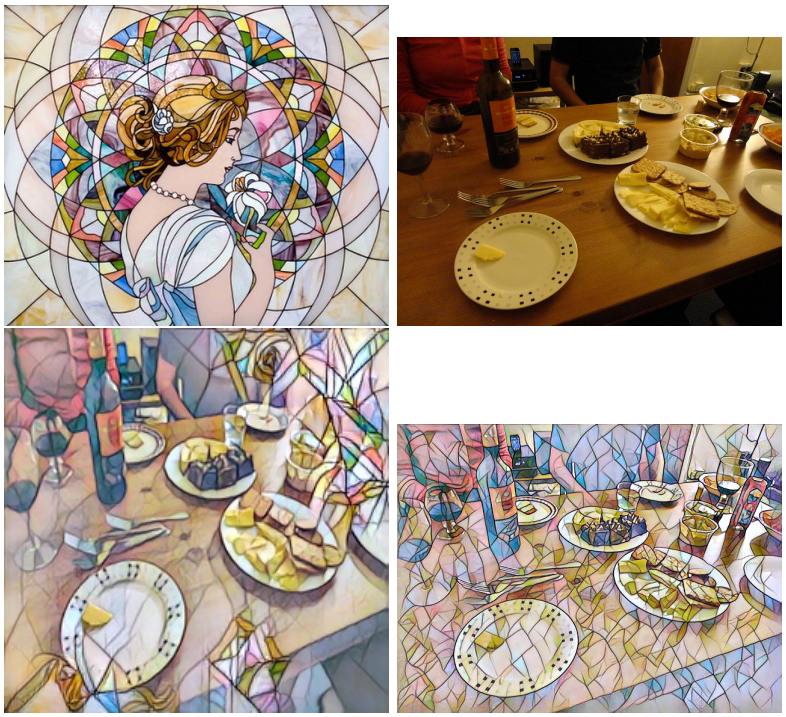
**(a)** Output of LambdaStyler and Fast Residual networks using the Udnie style image.



**(b)** Output of LambdaStyler and Fast Residual networks using the Candy style image.



**(c)** Output of LambdaStyler and Fast Residual networks using the Rain Princess style image.



**(d)** Output of LambdaStyler and Fast Residual networks using the Mosaic style image.

**Figure 5.23:** Sample output of LambdaStyler and Fast Residual networks. (Legend: Upper Left – Style Image, Upper Right – Content Image, Lower Left – LambdaStyler Output, Lower Right – Fast Residual Output)

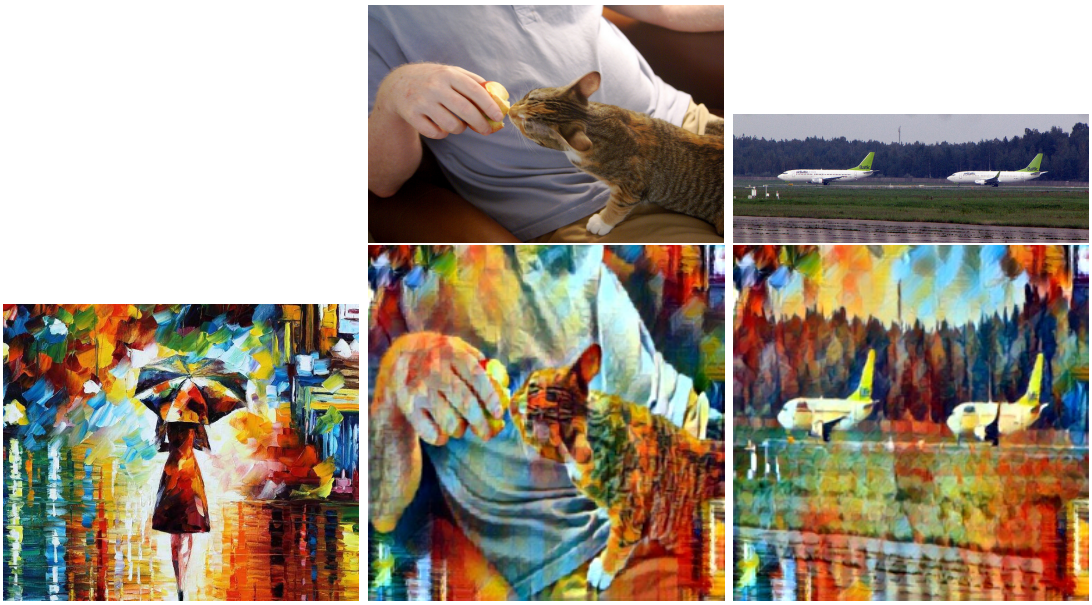
shows the image artifacts present for each style. In every case, except for Udnie, the output image accrues repeating elements from the style image in the corners of the output. (For Udnie, these artifacts take the form of a black and green set of blobs nearer the center.) This is particularly clear in the case of Mosaic, where the upper right corner of the image always contains the exact same pattern.

However, closer inspection shows that none of these inserted artifacts are actually present in the style image, meaning that they are generated by the network itself, and may not result from overfitting. This is readily apparent in the Candy output images, as elements such as the red curve in the upper right corner or the flowers along the bottom edge, do not appear in the style image. In an attempt to remove these artifacts, a hyperparameter search was undertaken. This search varied the style weight from $1e5$ to $1e10$ (content weight was kept fixed at $1e5$) and the number of epochs were swept from 2 to 10. While some tuning was able to eliminate the artifacting, specifically setting the style weight and content weight to be equal at $1e5$, this generated an output image which was almost identical to the content image.

As a result, this artifacting may be intrinsic to the LambdaStyler architecture. Under fast style transfer, such as that of Johnson, et al. [6], the information from the style image is imparted to the content image over the entire network. Under the LambdaStyler, the first half of the network is used to encode the content and style images down to a pair of intermediate representations, which are then merged at the fusion node. This leaves only half of the network weights available to merge the two representations together. The artifacting may be a result of this bottleneck fusion, as the remaining half of the network is insufficient to fully merge the two images together.

**(a)** Sample of LambdaStyler artifacting on Udnie style image (far left). Note black and white striped blob in upper right corner and green blob in lower left corner of the output images.

**(b)** Sample of LambdaStyler artifacting on Candy style image (far left). Note the presence of the same "flowers" in the upper corners and bottom edge of the output images.



**(c)** Sample of LambdaStyler artifacting on Rain Princess style image (far left). Note the common texture elements in the corners of the output images.

**(d)** Sample of LambdaStyler artifacting on Mosaic style image (far left). Note the common geometric patterns in the upper right corner and bottom edge of the output images.

**Figure 5.24:** Illustration of artifacting in LambdaStyler network. (Legend: Top Row – Content Images, Lower Left – Style Image, Lower Center and Lower Right – Stylized Outputs from LambdaStyler)

Finally, an untrained style image was used as an input to the LambdaStyler.

This image was The Starry Night by Van Gogh, shown in Figure 5.25. The results of this stylization are shown as Figure 5.26. As expected, LambdaStyler does not perform as well on untrained styles. While it does succeed in transferring some level of color information, it largely fails to incorporate any textures from the style image into the content image. This is especially apparent when compared to the stylization performed using the trained Udnie image. In the case of this image, a large amount of low frequency information is incorporated into the content image, flattening several high frequency regions, such as the dog's fur. When using the Starry Night painting, none of the high frequency brush strokes are incorporated into the content image. Qualitatively, it appears that LambdaStyler is shifting the mean of a region of the content image to the average of the corresponding area of the style image. This is readily apparent, as in the style image, there is a bright yellow moon in the top right corner. In both the stylized images shown in Figure 5.26, there is a bright yellow shape shown in that same location. As a result, it does not appear that LambdaStyler is currently able to transfer unknown styles.

**Figure 5.25:** The Starry Night by Van Gogh.


**Figure 5.26:** Sample outputs of LambdaStyler, when using an untrained style image.

# Chapter 6

## Discussion

Development of the proper network architecture for LambdaNet was critical to the performance of the final trained models. In order to arrive at an effective architecture, a set of base models were created featuring elements from other commonly used CNNs, including VGG (both untrained and pre-trained) and ResNet architectures. Since it was observed that the size of objects often played a significant role in the quality of the output segmentation map, multi-scale information was also included, in the form of the Res2Net layer. An approach, inspired by genetic algorithms, was then used to evaluate and remix these various network components into a new network configuration, capable of recognizing both structured and unstructured changes in image pairs at multiple scales.

Next, this finalized architecture was modified to perform directional change detection. Under this paradigm, a change is not only characterized by the presence or absence of an object in the image pair, but also in which frame, either reference or target, the object appears in. This results in four different possible classes: No change, additive change, subtractive change, and exchange. This classification scheme allows objects to be tracked based on how they enter or leave a scene.

Finally, to confirm LambdaNet's suitability for generally unstructured operations, a custom version of the architecture was developed for the purpose of performing artistic style transfer. Called LambdaStyler, this network is capable of applying a

variety of styles to a provided content image. Selection of the style is performed by using an image of a painting as the target image, while supplying a photograph as the reference image. Training is performed utilizing a perceptual loss network to balance the quantities of style and content information present in the output image. Once complete, this loss network is discarded and content images can be stylized by providing the appropriate artistic input.

Based on the demonstrated results, the following conclusions can be drawn:

- LambdaNet is a viable network architecture for performing binary change detection.
- Incorporation of encoders pre-trained on ImageNet, along with the addition of mutliscale elements into the network architecture, provide a significant accuracy boost.
- The Siamese portion of LambdaNet, with its parallel encoder and common decoder architecture, allow the network to extract additional unstructured, unlabelled change information from the image pairs.
- The combination of structured and unstructured change detections mean that LambdaNet operates in a mixed-mode, performing both structured and unstructured change detection.
- LambdaNet is also able to be effectively extended to perform directional change detection, where structured and unstructured changes are marked according to the order in which change targets are added or removed from the image pairs.
- LambdaNet can be easily modified into LambdaStyler to perform artistic style transfer, which is a fully unstructured operation.
- LambdaStyler is able to learn multiple unrelated artistic styles in a single network and effectively apply them to any content image.

## 6.1 Future Work

This work focused primarily on the development of a high-performing LambdaNet architecture for structured and unstructured change detection using labelled change maps. Further experimentation with the architecture revealed that it was also capable of performing other unstructured operations, such as artistic style transfer. In the case of style transfer, no actual labelled ground truth is required, due to its reliance on a loss network during training to balance the contributions from the content and style images. Future work may attempt to fuse these two domains, eliminating the need for labelled change pair training data by using a modified loss network to compute the relevant changes between input image pairs.

# Bibliography


[1] N. Goyette, P. Jodoin, F. Porikli, J. Konrad, and P. Ishwar, "Changedetection.net: A new change detection benchmark dataset," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, June 2012, pp. 1–8.

[2] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," *CoRR*, vol. abs/1505.04366, 2015. [Online]. Available: http://arxiv.org/abs/1505.04366

[3] R. Caye Daudt, B. Le Saux, and A. Boulch, "Fully convolutional siamese networks for change detection," in *IEEE International Conference on Image Processing (ICIP)*, October 2018.

[4] S. Gao, M. Cheng, K. Zhao, X. Zhang, M. Yang, and P. H. S. Torr, "Res2net: A new multi-scale backbone architecture," *CoRR*, vol. abs/1904.01169, 2019. [Online]. Available: http://arxiv.org/abs/1904.01169

[5] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, vol. abs/1508.06576, 2015. [Online]. Available: http://arxiv.org/abs/1508.06576

[6] J. Johnson, A. Alahi, and F. Li, "Perceptual losses for real-time style transfer and super-resolution," *CoRR*, vol. abs/1603.08155, 2016. [Online]. Available: http://arxiv.org/abs/1603.08155

[7] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[10] M. Teichmann, M. Weber, M. Zöllner, R. Cipolla, and R. Urtasun, "Multinet: Real-time joint semantic reasoning for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, June 2018, pp. 1013–1020.

[11] A. M. E. Amin, Q. Liu, and Y. Wang, "Convolutional neural network features based change detection in satellite images," in *First International Workshop on Pattern Recognition*, X. Jiang, G. Chen, G. Capi, and C. Ishll, Eds., vol. 10011,

International Society for Optics and Photonics. SPIE, 2016, pp. 181 – 186. [Online]. Available: https://doi.org/10.1117/12.2243798

[12] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 146 – 165, 2004. [Online]. Available: https://doi.org/10.1117/1.1631315

[13] D. L. C. author, P. Mausel, E. Brondízio, and E. Moran, "Change detection techniques," *International Journal of Remote Sensing*, vol. 25, no. 12, pp. 2365–2401, 2004. [Online]. Available: https://doi.org/10.1080/0143116031000139863

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[16] P.-L. St-Charles, G.-A. Bilodeau, and R. Bergevin, "Subsense : A universal change detection method with local adaptive sensitivity." *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 24, 12 2014.

[17] M. D. Gregorio and M. Giordano, "Change detection with weightless neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2014, pp. 409–413.

[18] S. Bianco, G. Ciocca, and R. Schettini, "How far can you get by combining change detection algorithms?" *CoRR*, vol. abs/1505.02921, 2015. [Online]. Available: http://arxiv.org/abs/1505.02921

[19] G. Bilodeau, J. Jodoin, and N. Saunier, "Change detection in feature space using local binary similarity patterns," in *2013 International Conference on Computer and Robot Vision*, May 2013, pp. 106–112.

[20] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," 2015.

[21] F. Rahman, B. Vasu, J. V. Cor, J. Kerekes, and A. Savakis, "Siamese network with multi-level features for patch-based change detection in satellite imagery," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Nov 2018, pp. 958–962.

[22] A. Varghese, J. Gubbi, A. Ramaswamy, and B. Purushothaman, *ChangeNet: A Deep Learning Architecture for Visual Change Detection: Subvolume B*, 01 2019, pp. 129–145.

[23] P. F. Alcantarilla, S. Stent, G. Ros, R. Arroyo, and R. Gherardi, "Street-view change detection with deconvolutional networks," *Auton. Robots*, vol. 42, no. 7, pp. 1301–1322, Oct. 2018. [Online]. Available: https://doi.org/10.1007/s10514-018-9734-5

[24] D. Chianucci and A. Savakis, "Unsupervised change detection using spatial transformer networks," in *2016 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)*, Nov 2016, pp. 1–5.

[25] B. Blakeslee, R. Ptucha, and A. Savakis, "Faster art-cnn: An extremely fast style transfer network," in *2018 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)*, Oct 2018, pp. 1–5.

[26] G. Ghiasi, H. Lee, M. Kudlur, V. Dumoulin, and J. Shlens, "Exploring the structure of a real-time, arbitrary neural artistic stylization network," *CoRR*, vol. abs/1705.06830, 2017. [Online]. Available: http://arxiv.org/abs/1705.06830

[27] A. Savakis and A. M. Shringarpure, "Semantic background estimation in video sequences," in *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*, Feb 2018, pp. 597–601.